Selective Use Of Multiple Entropy Models In Audio Coding

Sanjeev Mehrotra, Wei-ge Chen

Microsoft Corporation
One Microsoft Way, Redmond, WA 98052
{sanjeevm,wchen}@microsoft.com

***Abstract*—The use of multiple entropy models for Huffman or arithmetic coding is widely used to improve the compression efficiency of many algorithms when the source probability distribution varies. However, the use of multiple entropy models increases the memory requirements of both the encoder and decoder significantly. In this paper, we present an algorithm which maintains almost all of the compression gains of multiple entropy models for only a very small increase in memory over one which uses a single entropy model. This can be used for any entropy coding scheme such as Huffman or arithmetic coding. This is accomplished by employing multiple entropy models only for the most probable symbols and using fewer entropy models for the less probable symbols. We show that this algorithm reduces the audio coding bitrate by 5%-8% over an existing algorithm which uses the same amount of table memory by allowing effective switching of the entropy model being used as source statistics change over an audio transform block.**

I. Introduction

Adaptive entropy coding often provides significant savings over static entropy coding methods as source statistics are usually not stationary. Adaptive entropy coding has often been used in audio coding [1], image coding [2], [3], and video coding [4]. There are various types of adaptive entropy coding algorithms. One technique is for the encoder to try a multiple number of entropy tables and pick the best one and send the index of the table to the decoder [5], [6]. Such schemes when employed in audio and video coding can provide significant entropy coding gains. Entropy tables can also be switched within a frame for video or audio coding at certain fixed points, and if the savings from switching tables offsets the additional bits to signal a new table, the overall bitrate can be reduced.

Although the use of multiple entropy tables can reduce bitrate, without any additional memory constraints the amount of memory required to support multiple entropy models grows linearly with the number of models both at the encoder and decoder end. This can sometimes be prohibitively large. Except for the memory increase, multiple entropy tables do not increase decoder side complexity and only increase encoder side complexity as more tables have to be tried to see how to minimize rate. There has been limited work to address the memory increase due to multiple entropy models [7], [8], [9]. Memory constrained use of multiple entropy models reduces the optimality of the code.

We measure the *performance* of memory constrained codes by looking at the difference between the optimal average bit length by using all the entropy models in full and the average bit length obtained by the actual code.

Suppose we know that the source X can be in one of M possible states, where the probability mass function for X is different for each of the possible states. Then, it is good for the encoder to have M entropy models to choose from, and for a given set of data try all M entropy models, pick the best one, and use it to code the data. If the alphabet size is K, then the use of M possible entropy models results in memory requirements proportional to MK. If M is large, then the memory requirements may be too large for the encoder or decoder. There are two known ways to reduce the memory footprint in this situation [9]. One is to only use $N < M$ entropy models, and then each of the M states maps to one of these N models. The other is to reduce the alphabet size for each of the N entropy models by only accurately representing the probability mass function for the top $L[n] \leq K$ of the symbols for model n. That is, each of the N entropy models can have a different number of symbols for which the probability mass function is accurately represented. For symbols which are not represented in the entropy model, a special symbol, typically called an "escape" symbol in practice can be used followed by a fixed length code or a table-less code. The memory needed for this can be given by $\sum_{n=0}^{N-1}(L[n]+1)$, where one additional element is needed for each of the entropy tables to store the escape symbol.

The work presented in [9] tries to find the optimal value for N, the optimal value for each $L[n]$, and the optimal N probability distributions by clustering the M original distributions to minimize the average bitrate. The algorithm presented starts with a fixed distribution for a given N, with $L[n] = 0$ for all n. It then iterates between finding the N best distributions and increasing $L[n]$ by one for one of the N distributions until the memory constraint is reached. It also searches over all N using a hierarchical search to find the best value for N. Finding the best N distributions for given values of $L[n]$ and N itself is an iterative operation which iterates between finding the optimal mapping from the M distributions to N and then finding the best distribution for each of the N models by centroiding. This is essentially clustering using the Generalized Lloyd Algorithm (GLA) [10] with Kullback Leibler divergence (relative entropy) being used as the distortion "metric".

This algorithm is very complex and may not perform well in

many cases. It has to balance reducing the number of entropy models with reducing the alphabet size for each of the models to achieve the best bitrate. By reducing the number of entropy models, the compression efficiency suffers, especially due its effect on the most probable symbols. For the less probable symbols, only a few of the entropy models have an accurate representation of the true distribution since the alphabet size is reduced. In addition, such an algorithm may not be robust to state model mismatch between what is used for training and the actual data set. For example, if a particular state has a small probability in the training set than it will get absorbed into one with a larger probability. If for some reason that state is being used more often in actual data, then there can be a large penalty due to state mismatch.

To alleviate these issues, we propose an alternative solution which maintains a large (all the way up to M) number of entropy models for the most probable symbols taking full advantage of the multiple entropy models for these symbols, and successively reducing the number of models for the less probable symbols. This even allows less probable symbols to use a code which represents a true probability distribution rather than just using a uniform one. However, the distribution for the less probable ones is not as accurate since it clusters multiple models into a single one.

In section II we give a description of this algorithm. Section III is an example of the algorithm and how a sample Huffman table is constructed. Section IV gives some experimental results of the use of this algorithm and comparisons to existing work and the optimal solution.

II. Selective Use of Multiple Entropy Models

Let S be the random variable which represents the current state of the distribution with a value between 0 and $M-1$. Assume that the K sized alphabet has a probability mass function P_m when in state m, so that for example $P_1[2]$ is the probability that the symbol $X = 2$ when the state $S = 1$. Also, let R be the probability mass function for the states so that for example $R[1]$ is the probability that the state $S = 1$.

Here we propose a solution which tries to maintain true distributions for more probable elements but only approximate distributions for less probable ones, thus partitioning them into sets. We propose to divide the K elements of the alphabet into L partitions of size K_l, $l = 0, 1, \ldots, L-1$, so that $\sum_l K_l = K$. The elements of the partition are assumed to be contiguous symbols. Let V_l be the starting point for partition l given by $V_l = \sum_{i=0}^{l-1} K_i$, and let $\sigma[k]$ be the partition to which symbol k belongs. For example, $\sigma[k] = 0$ for $k = 0, 1, \ldots, K_0 - 1$. For the lth partition, we use N_l entropy models. We create a code to code $K_l + 1$ symbols for partitions $0, 1, \ldots, L-2$, the one additional symbol being used to represent a special symbol to indicate that the symbol value is larger than the maximum value in the partition (called the "escape" symbol). Partition $L-1$ (the last partition) has a code to code K_{L-1} symbols since no escape symbol is needed for the last partition. For symbols $V_l, V_l+1, \ldots, V_l+K_l-1$, we use the code created for the lth partition to code it. If the symbol value is larger than the maximum value for partition l, then we code an escape symbol $V_l + K_l$ followed by coding that symbol using the code for partition $l+1$. This process is continued until the symbol is actually coded.

For example, if $K = 10$, $K_0 = 4$, $K_1 = 4$, and $K_2 = 2$, partition 0 would have symbols $\{0, 1, 2, 3, 4\}$, partition 1 would have symbols $\{4, 5, 6, 7, 8\}$, and partition 2 would have symbols $\{8, 9\}$. Then to code symbol $X = 8$, we would code the escape symbol 4 for partition 0, the escape symbol 8 for partition 1, and the symbol 8 for partition 2.

Let $Q_{n,l}$ be the probability distribution of the nth entropy model of partition l, so that $Q_{n,l}[k]$ is the probability that the symbol $X = k$, when entropy model n is being used and l is the partition to which k belongs, $k = V_l, V_l+1, \ldots, V_l+K_l-1, V_l+K_l$. Again the last symbol is for the escape symbol for the given partition. Let μ_l be the mapping from the M states to one of the N_l entropy models in partition l, so that state m uses the distribution of the $\mu_l[m]$ entropy model. The average bit length is then given by

$$B = \sum_{m=0}^{M-1} R[m] \sum_{k=0}^{K-1} P_m[k] B_m[k], \quad (1)$$

where the bit length of the kth symbol in state m is given by

$$B_m[k] = -\log_2 \left(Q_{\mu_{\sigma[k]}[m],\sigma[k]}[k] \right) - \sum_{l=0}^{\sigma[k]-1} \log_2 \left(Q_{\mu_l[m],l}[V_l + K_l] \right). \quad (2)$$

The first term is the approximate number of bits it takes to code the symbol using the code for the partition to which it belongs. The second term is the sum of the approximate number of bits taken to code the escape symbol for the previous partitions. Instead of simply using a uniform distribution for the less probable symbols, the true probability distribution is being approximated where one can think of the approximated distribution as $P'_m[k] = 2^{-B_m[k]}$.

The total memory needed can be approximately given by

$$\alpha = K_{L-1} N_{L-1} + \sum_{l=0}^{L-2} (K_l + 1) N_l, \quad (3)$$

where one additional element is needed for the escape symbol for all partitions except the last. In addition, if one of the partitions (let's say the last) is simply using a table-less code (e.g. fixed length coding), then that term can be removed from the memory giving $\alpha = \sum_{l=0}^{L-2} (K_l + 1) N_l$. Suppose we have a memory constraint of $\alpha \leq T$.

Given the memory constraint $\alpha \leq T$, to find the optimal solution one would have to find the following to minimize the average bit length B.

- Optimal value for L, the number of partitions.
- Optimal partitioning, that is the optimal values for K_l which would also give the optimal value for $\sigma[k]$.
- Optimal number of entropy models for a given partition N_l.

- Optimal probability distributions for the entropy models $Q_{n,l}$ and the corresponding mapping function $\mu_l[m]$.

Given a partitioning (L, K_l) and number of entropy models for each partition N_l, it is fairly straightforward to derive $Q_{n,l}$ and $\mu_l[m]$. This can be done by clustering using GLA with relative entropy as the distance metric. Finding the optimal mapping $\mu_l[m]$ for each m simply uses the relative entropy as distortion to find the closest matching distribution,

$$\mu_l[m] = \arg\min_n \sum_{k=V_l}^{V_l+K_l} P_{m|l}[k] \log_2 \frac{P_{m|l}[k]}{Q_{n,l}[k]}, \tag{4}$$

where $P_{m|l}$ is the conditional distribution when the symbol belongs to partition l given by

$$P_{m|l}[k] = \frac{P_m[k]}{\sum_{i=V_l}^{K-1} P_m[i]}, \tag{5}$$

for $k = V_l, V_l+1, \ldots, V_l+K_l-1$ and

$$P_{m|l}[V_l+K_l] = \frac{\sum_{i=V_l+K_l}^{K-1} P_m[i]}{\sum_{i=V_l}^{K-1} P_m[i]}, \tag{6}$$

for the escape symbol. For a given mapping, finding the optimal distribution $Q_{n,l}$ can be done by centroiding as

$$Q_{n,l}[k] = \frac{\sum_{j\in\theta_{n,l}} R[j] P_j[k]}{\sum_{j\in\theta_{n,l}} R[j] \sum_{i=V_l}^{i=K-1} P_j[i]}, \tag{7}$$

for $k = V_l, V_l+1, \ldots, V_l+K_l-1$, and

$$Q_{n,l}[V_l+K_l] = \frac{\sum_{j\in\theta_{n,l}} R[j] \sum_{i=V_l+K_l}^{K-1} P_j[i]}{\sum_{j\in\theta_{n,l}} R[j] \sum_{i=V_l}^{i=K-1} P_j[i]}, \tag{8}$$

for the escape symbol. $\theta_{n,l}$ is the set of j for which the mapping from the M states to the N_l models is n for partition l, i.e. $\theta_{n,l} = \{j : \mu_l[j] = n\}$. Centroiding finds the conditional distribution of the symbols for a given partition and entropy model. Starting with a given mapping from M states to N_l models, we find the optimal distribution using equation (7) and (8), then for a given distribution for models we find the optimal mapping using equation (4), and then repeat until convergence. Similar to many iterative algorithms, the solution is not guaranteed to find the global minimum, it will find a locally optimal solution. However, given a good initial condition, the certain knowledge of global optimal solution (such as the lower bound of the average number of bits per symbol) and sufficient computational resources, there can be many practical means to ensure a reasonable solution is reached at the end. Such implementation details, however, are beyond the scope of this paper.

Now to find the partitioning (L, K_l), and number of models for each partition N_l adds significant additional complexity. This is a fairly complicated problem which can be attempted to be solved by iterating over each of the parameters one by one, fixing the others and optimizing just that one parameter. The optimization of the one parameter can be done by exhaustively searching all possible values, or starting with the value from the previous iteration and then doing a bisection search.

A. Simplified Practical Solution

Here we propose a much simplified solution which works well in practice without having to explicitly optimize the parameters L, K_l, and N_l. We propose to divide the symbol space into two or three partitions, one partition which uses M entropy models and is an accurate representation of all probabilities for the most probable symbols, one partition which uses one entropy model and is an accurate representation of probabilities within that partition, and the third partition which uses one model with a fixed length code if needed for the least probable symbols. The third partition is only needed in certain cases as will be explained later. That is we use

$$N_0 = M,\ N_1 = 1,\ N_2 = 1. \tag{9}$$

Since the third partition $l = 2$ is a fixed length code, it does not add to the table memory. The total memory needed is given by

$$\alpha = (K_0+1)M + (K_1+1) \tag{10}$$

if three partitions are being used, and

$$\alpha = (K_0+1)M + K_1 \tag{11}$$

if only two partitions are needed.

For the first partition since the number of entropy models is same as the number of states ($N_0 = M$), finding the distribution for each of the symbols in the partition is trivial. For each entropy model $n = m$, we can write

$$Q_{m,0}[k] = P_m[k], \tag{12}$$

for $k = 0, 1, \ldots, K_0-1$, and

$$Q_{m,0}[K_0] = \sum_{i=K_0}^{K-1} P_m[i], \tag{13}$$

for the escape symbol $k = K_0$.

For the second partition, finding the distribution for the one entropy model is also trivial since $\mu_1[m] = 0$ for all m. We can write

$$Q_{0,1}[k] = \frac{\sum_{j=0}^{M-1} R[j] P_j[k]}{\sum_{j=0}^{M-1} R[j] \sum_{i=K_0}^{K-1} P_j[i]}, \tag{14}$$

for $k = K_0, K_0+1, \ldots, K_0+K_1-1$, and

$$Q_{0,1}[K_0+K_1] = \frac{\sum_{j=0}^{M-1} R[j] \sum_{i=K_0+K_1}^{K-1} P_j[i]}{\sum_{j=0}^{M-1} R[j] \sum_{i=K_0}^{K-1} P_j[i]}, \tag{15}$$

for the escape symbol if it is needed. The escape symbols is only needed if $K_0+K_1 < K$.

If the third partition is needed, since it is just a uniform distribution we can write

$$Q_{0,2}[k] = \frac{1}{K-K_0-K_1}, \tag{16}$$

for $k = K_0+K_1, K_0+K_1+1, \ldots, K-1$. The third partition although valid if $K_0+K_1 < K$ is useless unless it has at least two symbols which means $K_0+K_1 < K-1$ is the

condition for having a third partition. Thus the calculation of the distributions for all the partitions and entropy models is trivial if the number of entropy models is constrained using equation (9).

Using the distributions from equations (12)-(16), the average bit length for this simplified solution is given by

$$B = \sum_{m=0}^{M-1} -R[m] \left[\sum_{k=0}^{K_0-1} P_m[k] \log_2 Q_{m,0}[k] + \sum_{k=K_0}^{K_0+K_1-1} P_m[k](\log_2 Q_{m,0}[K_0] + \log_2 Q_{0,1}[k]) + \sum_{k=K_0+K_1}^{K-1} P_m[k](\log_2 Q_{m,0}[K_0] + \log_2 Q_{0,1}[K_0+K_1] + \log_2 Q_{0,2}[k]) \right]. \quad (17)$$

The approximate bit length for each symbol k in state m is given by $B_m[k]$, where

$$B_m[k] = \begin{cases} -\log_2 Q_{m,0}[k], \\ \text{ if } 0 \le k < K_0 \\ -\log_2 Q_{m,0}[K_0] - \log_2 Q_{0,1}[k], \\ \text{ if } K_0 \le k < K_0 + K_1 - 1 \\ -\log_2 Q_{m,0}[K_0] - \log_2 Q_{0,1}[K_0+K_1] \\ -\log_2 Q_{0,2}[k], \\ \text{ if } K_0 + K_1 \le k < K \end{cases} \quad (18)$$

Since we wish to use the entire T amount of memory available, for a given K_0, we can compute

$$K_1 = \begin{cases} \min(K - K_0, T - (K_0+1)M), \\ \text{ if } T - (K_0+1)M + K_0 \ge K \\ T - 1 - (K_0+1)M, \\ \text{ if } T - (K_0+1)M + K_0 < K \end{cases} \quad (19)$$

The values for K_1 are computed for two cases using equations (10) and (11). The first one is if only two partitions are used, and the second one is if three partitions are used. Note that if $T - (K_0+1)M + K_0 = K - 1$, then $K_0 + K_1 = K - 2$ which ensures that there will at least be two symbols in the third partition if it is being used.

Also, since $K_2 = K - K_0 - K_1$, the value for K_2 can be written as

$$K_2 = \begin{cases} 0, \\ \text{ if } T - (K_0+1)M + K_0 \ge K \\ K - (T - 1 - (K_0+1)M + K_0), \\ \text{ if } T - (K_0+1)M + K_0 < K \end{cases} \quad (20)$$

Again the first case is if only two partitions are used, the second is if three partitions are used.

The remaining problem to solve is the partitioning to find K_0, K_1, and K_2 to satisfy the memory constraints to minimize the average bit length in equation (17). Since both K_1 and K_2 can be written as functions of K_0 and the total memory constraint T using equations (19) and (20), there is only one degree of freedom, K_0. To find the value for K_0, we first start with K_0 as large as it can be such that $K_0 + K_1 = K$ (only two partitions are used) and so that all the memory is used. This gives

$$K_0 = \max\left(\left\lfloor \frac{T - M - K}{M - 1} \right\rfloor, 0 \right). \quad (21)$$

For this, we compute K_1 and K_2 using equations (19) and (20) and then compute the average bit length using equation (17). Then, we increase K_0 by one and recompute the average bit length. This causes the third partition to be used. If the bit length reduces, we keep increasing K_0 by one until the bit length starts increasing. The values that are used for K_0, K_1, and K_2 are given by the solution which gives the minimal average bit length. Although this is not guaranteed to be the optimal value for K_0 since we do not search over all possibilities, in practice it gives good performance for very low complexity.

Note that throughout this description, we have assumed that the first K_0 symbols are using M entropy models. However, any arbitrary partitioning can be used so that the most probable K_0 symbols use the M entropy models.

TABLE I
EXAMPLE PROBABILITY MASS FUNCTIONS FOR VARIOUS STATES.

	$P_0[k]$	$P_1[k]$	$P_2[k]$
$k=0$	0.400	0.700	0.100
$k=1$	0.400	0.200	0.600
$k=2$	0.089	0.055	0.165
$k=3$	0.039	0.030	0.090
$k=4$	0.072	0.015	0.045

III. EXAMPLE

Suppose we have three states $M = 3$, with probabilities of the states being $R[0] = 0.5$, $R[1] = 0.2$, and $R[3] = 0.3$. We have an alphabet size of $K = 5$, with the probability mass function of the symbol in each of those states as shown in table I.

Suppose the memory constraint is $T = 12$. Assume that we want $K_0 + K_1 = K = 5$ with K_0 as large as possible to satisfy the memory constraint. Then, we get $K_0 = \lfloor (12 - 3 - 5)/(3-1) \rfloor = 2$, $K_1 = 3$, $K_2 = 0$. Thus we only use two partitions.

From this we obtain the probability mass functions for each of the partitions. For partition $l = 0$, we have three entropy models with distributions as shown in table II(a). For partition $l = 1$, we have only one model with distribution as shown in table II(b). These are obtained using equations (12)-(16). Using this, the original probability mass function is being approximated using the approximate probability mass function shown in table III which is obtained as $P'_m[k] = 2^{-B_m[k]}$, where $B_m[k]$ is the average bit length from equation (18). The loss in performance comes from this approximation. As can be seen, approximating the distribution is better than simply removing some of the states or replacing portions of the distribution with a uniform distribution which is what happens when the alphabet size is reduced for some states.

TABLE II
PROBABILITY MASS FUNCTIONS FOR PARTITIONS $l = 0$ AND $l = 1$.

	$Q_{0,0}[k]$	$Q_{1,0}[k]$	$Q_{2,0}[k]$
$k = 0$	0.400	0.700	0.100
$k = 1$	0.400	0.200	0.600
$k = 2$	0.200	0.100	0.300

(a) $l = 0$

	$Q_{0,1}[k]$
$k = 2$	0.500
$k = 3$	0.250
$k = 4$	0.250

(b) $l = 1$

TABLE III
APPROXIMATED PROBABILITY MASS FUNCTIONS FOR VARIOUS STATES.

	$P_0[k]$	$P_1[k]$	$P_2[k]$
$k = 0$	0.400	0.700	0.100
$k = 1$	0.400	0.200	0.600
$k = 2$	0.100	0.050	0.150
$k = 3$	0.050	0.025	0.075
$k = 4$	0.050	0.025	0.075

TABLE IV
HUFFMAN CODE FOR APPROXIMATED PROBABILITY MASS FUNCTIONS.

	Code $m = 0$	Code $m = 1$	Code $m = 2$
$k = 0$	0	0	11
$k = 1$	10	10	0
$k = 2$	11*0*	11*0*	10*0*
$k = 3$	11*10*	11*10*	10*10*
$k = 4$	11*11*	11*11*	10*11*

TABLE V
HUFFMAN CODE TABLES FOR PARTITIONS $l = 0$ AND $l = 1$.

	$m = 0$	$m = 1$	$m = 2$
$k = 0$	0	0	11
$k = 1$	10	10	0
$k = 2$	11	11	10

(a) $l = 0$

	$m = 0$
$k = 2$	0
$k = 3$	10
$k = 4$	11

(b) $l = 1$

Although the example shown here is a very simple one and it does not have a large reduction in memory (going down from 15 elements to 12), it still shows how selective use of multiple entropy models is implemented and used.

A. Huffman Code Tree Representation

In this section, we view the effects of selectively using multiple entropy models when used for Huffman coding. For the example approximated probability mass function given in table III, the Huffman code can be as shown in table IV. From this table, we can see that all symbols $k \geq 2$, have a common suffix across all states which is shown in bold. Also, we can see that all symbols $k \geq 2$ for a given state have a common prefix shown in underline. We can also construct a Huffman coding tree from the Huffman code tables as shown in figure 1 for the three states. Each of the three states has its own unique tree, however, due to all symbols $k \geq 2$ having a common suffix, there exists a common branch in all three of the trees as shown in the dashed lines in the trees.

We can obtain the same results when creating the Huffman codes for the probability mass functions for the individual partitions. For the distributions in table II for the two partitions, we obtain the Huffman codes shown in table V. The common prefix for all symbols $k \geq 2$ in table IV for a given state is the coding of the escape symbol shown in table V(a). The common suffix for symbols $k \geq 2$ across all states is the Huffman code for the symbols in partition $l = 1$ shown in table V(b).

The reduction in memory comes from the fact that instead of using table IV as the Huffman code, we can instead use tables V(a)-(b) which contain the same information.

IV. EXPERIMENTAL RESULTS

We compare the performance of our algorithm with previous work and the lower achievable bound. First we use create a source with an alphabet size of 256 where the source distribution can be in one of eight different states. Each state has a distribution which is approximately Laplacian given by

$$P_m[k] = \frac{e^{\frac{-k}{2+10m}}}{\sum_{i=0}^{255} e^{\frac{-i}{2+10m}}}, \tag{22}$$

for $k = 0, 1, \ldots, 255$, where $m = 0, 1, \ldots, 7$ is the state. The probability of being in each state is assumed to be uniform, i.e. $R[m] = \frac{1}{8}$ for all m.

In figure 2, we compare our solution with Lee et al's algorithm from [9] and the optimal solution. The optimal solution will require 2048 elements of memory and will achieve an average bit length of approximately 6.0 bits. A single table performs about 7% worse than this and gives 6.5 bits. With our solution, even with only 256 elements of memory, we are only 3% worse than the optimal, and after about 500 elements are within 1% of the optimal.

Except for very low memory constraints, our algorithm exceeds the performance of [9] and achieves the optimal value with only about one-fourth of the memory of the optimal solution. In addition, the algorithm from [9] seems to achieve a local minimum and does not improve performance beyond a certain amount of memory. The only reason the quality is better for very small memory sizes (memory sizes smaller than the alphabet size K) for [9] is that for memory constraints smaller than K our algorithm will not attempt to find a reduced number of entropy models, instead it will continue to use M entropy models for most probable symbols. We could also attempt to use more complicated methods for very small memory constraints to improve the performance, but we are more interested in memory constraints which are at least the size of one table (K elements).

A. Application to Audio Coding

For the second experiment, we use this algorithm to code spectral coefficients in an audio codec. The codec works by performing a modulated discrete cosine transform (MDCT) on the time domain samples. The spectral coefficients are weighted using a psychoacoustic model and transformed to have redundancy across the channels removed. These coefficients are then quantized and entropy coded using a hybrid vector Huffman code to code the lower coefficients and run-length and Huffman coding for the higher spectral coefficients. Huffman tables are used for coding the vector symbols as well as the run-length pairs. We compare two methods of Huffman

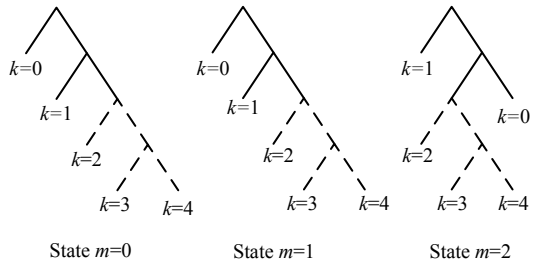

Fig. 1. Huffman Code Tree.

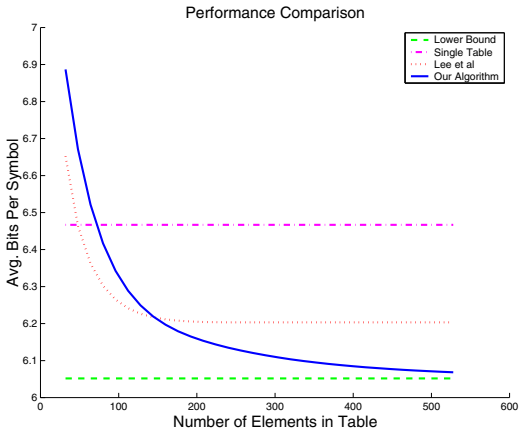

Fig. 2. Comparison

coding, one is to use two Huffman tables with probability distributions accurately represented for all symbols, another is with eight Huffman tables which are created using the algorithm presented in this paper and switching is allowed on per bark band (critical band) basis for the eight table case. The eight table and the two table methods both use approximately the same amount of memory and are trained using a set of music clips.

In order to create the Huffman tables, the spectral coefficients are quantized over a range of quality levels and the symbols for the vectors and run-length pairs are obtained. The probability mass function for these symbols are clustered into either two or eight clusters using the GLA algorithm.

We encode a large number of 44.1kHz, stereo clips (outside the training set) consisting of various genres of music (rock, jazz, instrumental, electronica) and some speech of various languages (English, German, Japanese) which total approximately one hundred minutes in length. The encoding is done using a constant quality throughout the file. For quality, we use a measure of noise power to signal power, where the signal power is computed after using a psychoacoustic masking model. We call this measure the noise-to-mask ratio (NMR). The higher the NMR value, the poorer the audio quality.

We compare the average bitrate of the file in both cases. The results are shown in table VI. We see that we obtain a 5%-8% reduction in average bitrate while approximately using the same amount of encoder and decoder memory. The increase

TABLE VI
AUDIO CODING RESULTS.

NMR	Two tables (bits/sec)	Eight tables (bits/sec)	Improvement (%)
0.005	284349	270829	4.99
0.010	222655	212778	4.64
0.020	185676	176495	5.20
0.050	107740	101381	6.27
0.070	74506	69635	7.00
0.100	47158	43711	7.89

in improvement with decreasing rate is due to the fact that the two table case was trained using higher bitrate data. However, even at higher rates, there is still a significant improvement with no increase in memory.

V. CONCLUSION

We have shown an improved method by which almost all entropy coding gains of a system using multiple entropy models can be obtained using only a fraction of the memory. The system uses multiple entropy models with the number of models being equal to the number of states for the most probable symbols and then only one entropy model for the less probable symbols. The construction of the entropy tables is very easy requiring a fraction of the time of that of existing work. The performance is as good as the existing work when the memory constraint is approximately that of the size of a single table and is almost as good as that of the full multiple table system when the memory constraint is approximately only twice the size of a single table. When used in an audio codec, it gives significant bitrate reduction when compared to a codec with similar memory requirements.

REFERENCES

[1] K. Brandenburg, J. Herre, J. D. Johnston, Y. Mahieux, and E. F. Schroeder, "Aspec-Adaptive spectral entropy coding of high quality music signals," in *AES Convention 90*. Audio Engineering Society, Feb. 1991, paper 3011.

[2] Y. Kim and J. Modestino, "Adaptive entropy coded subband coding of images," *IEEE Trans. Image Processing*, vol. 1, pp. 31–48, Jan. 1992.

[3] H. Malvar, "Fast adaptive encoder for bi-level images," in *Proc. Data Compression Conference*. IEEE, Mar. 2001, pp. 253–262.

[4] D. Marpe, H. Schwarz, G. Blattermann, G. Heising, and T. Wiegand, "Context-based adaptive binary arithmetic coding in JVT/H.26L," in *Proc. Int'l Conf. Image Processing*, vol. 2. IEEE, Sep. 2002, pp. 513–516.

[5] G. Lakhani, "Using multiple huffman code tables for optimal coding of dct blocks," in *Proc. Data Compression Conference*. Snowbird, UT: IEEE Computer Society, Apr. 2002, p. 460.

[6] J. Lervik and T. Ramstad, "Optimality of multiple entropy coder systems for nonstationary sources modeled by a mixture distribution," in *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing*, vol. 4. IEEE, May 1996, pp. 1874–1877.

[7] S. Lee, K. Yang, and C. Lee, "Optimal sharing of huffman tables for memory-constrained variable length coding of multiple sources," *Electronics Letters*, vol. 31, pp. 1657–1658, Sep. 1995.

[8] ——, "Memory allocation for huffman coding of multiple sources," *Electronics Letters*, vol. 32, pp. 15–16, Jan. 1996.

[9] S. J. Lee and C. W. Lee, "Determination of huffman parameters for memory-constrained entropy coding of multiple sources," *IEEE Signal Process. Lett.*, vol. 4, pp. 65–67, Mar. 1997.

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