# Low-Complexity Waveform Coding via Alphabet and Sample-Set Partitioning

**Amir Said***
Faculty of Electrical Engineering
State University of Campinas (UNICAMP), Campinas, SP 13081, Brazil
E-mail:said@ipl.rpi.edu

**William A. Pearlman**
Electrical, Computer and Systems Engineering Dept.
Rensselaer Polytechnic Institute, Troy, NY 12180, U.S.A.
E-mail:pearlman@ecse.rpi.edu

## ABSTRACT

We propose a new low-complexity entropy-coding method to be used for coding waveform signals. It is based on the combination of two schemes: (1) an alphabet partitioning method to reduce the complexity of the entropy-coding process; (2) a new recursive set partitioning entropy-coding process that achieves rates smaller than first order entropy even with fast Huffman adaptive codecs. Numerical results with its application for lossy and lossless image compression show the efficacy of the new method, comparable to the best known methods.

**Keywords:** entropy coding, lossless compression, image coding

## 1 INTRODUCTION

In practical applications of waveform coding, the common rule is that sources have characteristics that keep changing dramatically. These changes can be easy to identify, like activity in voice coding, or more subtle, like textures in images, or instrument timbre in music. The simplest coding methods are known to be ideal under conditions like stationarity, which may be too far from reality. On the other hand, much is known about the existence of methods for complex sources (e.g., nonstationary), but here the problem lies in the coding computational complexity. Frequently an exponential growth in the complexity makes any implementation impractical.

So, an important problem for waveform coding is to devise efficient coding schemes that are highly adaptive, yielding good results for whatever are the source characteristics at that moment, and which at the same time have complexity consistent with the type of application (e.g., real-time). Actually, some of the most effective algorithms appearing now in the literature follow this line by adding to the coding algorithms intelligent rules for detecting patterns in the source or some form of source state. However, those solutions are usually constrained to a single type of source.

In this paper we consider the problem in a more general perspective, and propose a scheme for combining complexity reduction and efficient entropy-coding based on an assumption common in several types of sources (usually waveforms after some transformation or prediction): the clustering of activity, i.e., the source statistics should be consistent inside regions in a proper signal space. For example, for subband transformed audio this space has both time and frequency components, and only noise would use the whole audible spectrum all the time. "Normal" sources use only some parts of the spectrum during some lengths of time.

*Currently with Iterated Systems, Inc., Atlanta, GA, U.S.A.

The first part of the paper presents an analysis of the alphabet partitioning method, which is used to reduce the complexity of the entropy-coding process. Then we present a new recursive set partitioning entropy-coding process that efficiently achieves rates smaller than first order entropy even with fast Huffman adaptive codecs. Finally, numerical results show the efficacy of the new method for lossy and lossless image compression. Even though the new approach can lead to both better compression and coding/decoding speed, our emphasis here is on speed. Notwithstanding, the results for image compression are among the best in the literature.

## 2 THE ALPHABET PARTITIONING METHOD

In practical waveform coding applications (e.g., image, video, audio) the data to be entropy-coded—obtained after an adequate signal transformation (e.g., linear prediction, DCT, wavelet, pyramids, etc.) and quantization—has a distribution that is simultaneously highly peaked and very long tailed. Typically, the values are concentrated near zero, but there is also a significant number of values with very large magnitude. This means that large compression ratios are possible, but the coding process must deal with a very large alphabet. This coding problem occurs for both lossy and lossless compression.

There are many practical difficulties when the data alphabet is large, ranging from the time required to design and implement the code, to the amount of memory necessary to store the codebooks or gather the data statistics (when the coding method is adaptive). These problems get much worse if we try to exploit the statistical dependence left between the source samples by, for example, coding several symbols together or designing conditional codes.

Several *ad hoc* methods have been devised to deal with the problem caused by large alphabets. For instance, a popular method uses an extra "overflow" symbol to indicate data that is to be coded with a special method. However, these methods are not very efficient and have limited application.

Here we analyze a method to reduce the coding complexity when the source alphabet is large, based on the following strategy:

- the source alphabet is partitioned into a relatively small number of sets;
- each symbol is coded in two steps: first the number of the set in which the symbol belongs (called *set number*) is coded; afterwards the number of that particular source symbol inside that set (the *set index*) is coded;
- when coding the pair (set number, set index) the set number is entropy-coded with a powerful and complex method, while the set index is coded with a very simple and fast method (which can be simply the binary representation of that index).

Figure 1 shows a diagram of such scheme when the set indexes are left uncoded. Note that that even though we frequently mention a waveform source, we are always implicitly assuming that the data source to be coded is the output of a stage where the signal is transformed and quantized.

The advantage of alphabet partitioning comes from the fact that it is normally possible to find partitions that allow large reductions in the coding complexity with a very small loss in the compression efficiency. Methods that fit in this general strategy had already been used in several compression schemes. For example, it is used inside the popular compression program `gzip`, and also in the JPEG still picture lossless compression standard,[2] in its variable-length-integer (VLI) representation. It has been used for complexity reduction in the context definitions for arithmetic coding,[3] and also in the arithmetic coding of waveforms.[5] However, it was not possible to identify who first proposed it, nor find any general description or the analysis of its performance, or even know if those who had been using special versions of it knew about its full power and generality.

Here we present an analysis of the trade-off between the coding/decoding complexity and the compression efficiency, and also propose and test an algorithm to find the best partitions for a given source distribution. With the complexity reduction achieved by the alphabet partitioning method, we show that it is feasible to use more powerful entropy-coding methods, and to exploit statistical dependencies which cannot be exploited by linear prediction methods. Numerical results make clear the advantages of the alphabet partitioning for practical problems

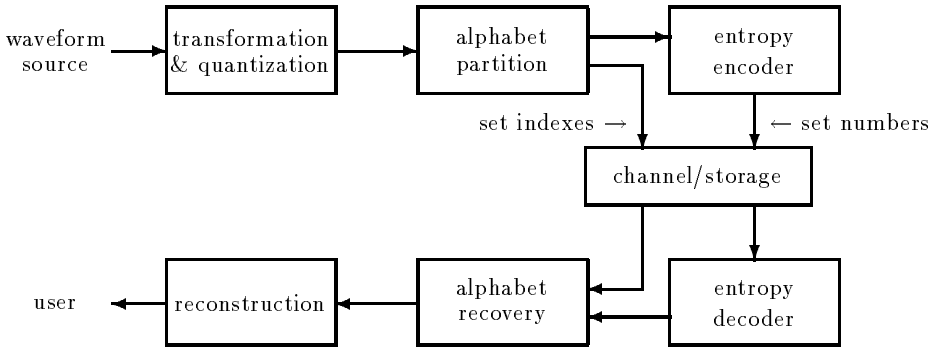


Figure 1: A system for wavelet compression using the alphabet partitioning method for complexity reduction.

Another practical advantage is related to error protection. An error in a variable-length code can have disastrous consequences. With the alphabet partitioning scheme the set-index data can be left uncoded, and an error in that data has very limited consequences (i.e., a single bit error will change only one reconstructed source sample).

## 3 ANALYSIS OF THE ALPHABET PARTITIONING

Let us assume an i.i.d. source with $M$ symbols, and where the symbol $i$ occurs with probability $p_i$. Its entropy is

$$\mathcal{H} = \sum_{i=1}^{M} p_i \log_2 \left( \frac{1}{p_i} \right) . \tag{1}$$

We partition the source symbols into nonempty sets $\mathcal{S}_n$, $n = 1, 2, \ldots, N$, and denote the number of elements in $\mathcal{S}_n$ by $M_n$.

Let us consider an ideal entropy-coding method which first codes the set in which the source symbol belongs (viz., the set number), and later codes the symbol conditioned that it belongs to that set (viz., the set index). The coding rate, here equal to the combined entropy, is

$$\begin{aligned} \mathcal{H}_1 &= \sum_{n=1}^{N} P_n \log_2 \left( \frac{1}{P_n} \right) + \sum_{n=1}^{N} P_n \sum_{i \in \mathcal{S}_n} \frac{p_i}{P_n} \log_2 \left( \frac{P_n}{p_i} \right) & (2) \\ &= \sum_{n=1}^{N} P_n \log_2 \left( \frac{1}{P_n} \right) + \sum_{n=1}^{N} \sum_{i \in \mathcal{S}_n} p_i \log_2 \left( \frac{P_n}{p_i} \right) , & (3) \end{aligned}$$

where $P_n = \sum_{i \in \mathcal{S}_n} p_i$ is the probability that a data symbol belongs to set $n$.

The first term in (3) is the rate to code the set, and the second term is the rate to conditionally code the symbol index in the set. Substitution of $P_n$ into (3) proves that $\mathcal{H}_1 = \mathcal{H}$, meaning that the combined rate is equal to the source entropy and that there is no loss in separating the coding process in two steps. This is intuitively reasonable, since the pair (set number, set index) is related to the symbol number by a one-to-one mapping.

To reduce the complexity we want to code all indexes of the set $\mathcal{S}_n$ with the same number of bits, $\log_2 M_n$. If $M_n$ is a power of two, then the set indexes can be coded using simply their binary representation. When arithmetic coding[8, 9] is used we still can have complexity reduction even if $M_n$ is not a power of two, because it is easier to code assuming an uniform distribution.

Of course, there is a penalty for assigning $\log_2 M_n$ bits for every symbol in the set $\mathcal{S}_n$, as the optimal number of bits to be used by symbol $i$ inside set $\mathcal{S}_n$ is $\log_2(P_n/p_i)$. To evaluate the loss we can calculate the combined rate of the simplified method, which is the entropy of the set numbers plus an average of the fixed rates used to code the

set indexes:

$$\mathcal{H}_2 = \sum_{n=1}^{N} P_n \left[ \log_2 \left( \frac{1}{P_n} \right) + \log_2 M_n \right] = \sum_{n=1}^{N} P_n \log_2 \left( \frac{1}{P_n} \right) + \sum_{n=1}^{N} \sum_{i \in \mathcal{S}_n} p_i \log_2 M_n . \quad (4)$$

From (3) and (4) it follows that the difference between $\mathcal{H}_2$ and $\mathcal{H}_1$ is

$$\Delta\mathcal{H} = \mathcal{H}_2 - \mathcal{H}_1 = \sum_{n=1}^{N} \sum_{i \in \mathcal{S}_n} \left[ p_i \log_2 M_n - p_i \log_2 \left( \frac{P_n}{p_i} \right) \right] = \sum_{n=1}^{N} \sum_{i \in \mathcal{S}_n} p_i \log_2 \left( \frac{M_n p_i}{P_n} \right) . \quad (5)$$

The analysis of equation (5) lead us to the conclusion that the relative loss $\Delta\mathcal{H}/\mathcal{H}$ can be made very small if we partition the alphabets in sets such that

1. $P_n \log_2 M_n / \mathcal{H}$ is very small, i.e., the relative contribution of $\mathcal{S}_n$ to $\mathcal{H}$ is very small.
2. $p_i \approx P_n / M_n, \forall i \in \mathcal{S}_n$, i.e., the distribution inside $\mathcal{S}_n$ is approximately uniform.

The first condition exploits the fact that some symbols can occur with very low probability—a property that has been used by several other methods. The second condition, on the other hand, can occur even for the most probable symbols. What makes the alphabet partitioning scheme really effective is the simultaneous exploitation of these two very different conditions.

Note that the analysis above can be easily extended to conditional entropy by simply replacing the probabilities with conditional probabilities.

## 4 OPTIMAL ALPHABET PARTITIONING ALGORITHM

In the last section we derived the conditions necessary to have good alphabet partitions. It is still necessary to define an efficient method to construct the optimal partitions. The objective is to minimize the complexity maintaining a relatively small loss in compression efficiency. Thus, we can set the optimization problem as: minimize the number of sets $N$, subject to $\Delta\mathcal{H} \leq \epsilon\ \mathcal{H}$, where $\epsilon$ is a small number (e.g., 0.01).

We can also add other practical constraints. For instance, if the set indexes are to be left uncoded, then we must require that the number of set elements, $M_n, n = 1, 2, \ldots, N$, be a power of two. When using an arithmetic code it may not be possible to separate the data streams with the set numbers and the set index, and the constraint above is not necessary.

The alphabet partitioning problem is a combinatorial problem. The optimization methods to solve that type of problem are quite complex and require a large computational effort to find the optimal solution. Another alternative is to use some heuristic methods, which have been successfully employed to solve combinatorial problems. They can find, in the majority of the cases, solutions that are very near the optimum. The disadvantage is that the user may not know with certainty the quality of the solution, but this may not be a problem for engineering applications.

So, we propose the following fast heuristic algorithm that can be used "on-the-fly" to find good partitions.

1. Set $\Delta\mathcal{H} = 0$; choose $N$ relatively large and the alphabet partitions $\mathcal{S}_n$ using an empirical rule (see comment below); choose the maximum relative loss $\epsilon$.
2. For every pair of sets with the same size, join the sets, calculate the entropy variation $\delta\mathcal{H}$, and select the smallest variation $\delta\mathcal{H}^*$.
3. If $\delta\mathcal{H}^* + \Delta\mathcal{H} > \epsilon\mathcal{H}$ then stop.
4. Otherwise
   (a) Set $N \leftarrow N - 1$, $\Delta\mathcal{H} \leftarrow \Delta\mathcal{H} + \delta\mathcal{H}^*$
   (b) replace the sets corresponding to $\delta\mathcal{H}^*$ by their union

| magnitude set (MS) | amplitude intervals | sign bit | magnitude-difference bits |
|---|---|---|---|
| 0 | [0] | no | 0 |
| 1 | [–1], [1] | yes | 0 |
| 2 | [–2], [2] | yes | 0 |
| 3 | [–3], [3] | yes | 0 |
| 3 | [–4], [4] | yes | 0 |
| 4 | [–6,–5], [5,6] | yes | 1 |
| 5 | [–8,–7], [7,8] | yes | 1 |
| 6 | [–12,–9], [9,10] | yes | 2 |
| 7 | [–16,–13], [13,16] | yes | 2 |
| ⋮ | ⋮ | ⋮ | ⋮ |

Table 1: Example of magnitude-sets to be used as initial solution in the alphabet-partitioning algorithm.

(c) go to step 2.

This algorithm belongs to a class of heuristics called "greedy," because its decisions do not take into account the future alternatives, and when a decision is chosen it cannot be undone.

In step 1 of the algorithm an initial solution is required. With some knowledge of the source and some experience with the algorithm it is easy to identify good initial solutions. For instance, with transformed waveforms we commonly have positive and negative values with the same probability. Thus, as shown by equation (5), there is absolutely no loss in joining the positive and negative values in the same sets. This is equivalent to say that there is no gain in attempting to entropy-code the sign bits. Furthermore, the difference in probabilities tend to get smaller for larger values. For example, the values 2 and 5 can have very different probabilities, while the values 82 and 85 should have practically the same probability, and thus can be put together in the same set.

An example of a good initial solution for image sources is shown in Table 1. In that example the value is decomposed in three parts: a magnitude set, which corresponds to the set number, and a group of sign plus magnitude-difference bits required to code the set index.

## 5 COMBINATION OF ALPHABET PARTITIONING WITH SAMPLE SET PARTITIONING

As explained above, with the alphabet partitioning scheme we only need to entropy code the set numbers. We assume that an alphabet partition was previously chosen and will not change during coding. So, hereafter when we say entropy coding, or mention source symbols, source samples, etc., we are always referring to the set numbers, which form a relatively small alphabet (e.g., 16 symbols or letters). Hereafter we use the term *source symbol* to indicate a general element that is an alphabet partition number. For example, if the number of sets is 16, then the new source alphabet can be $\{0, 1, 2, \ldots, 15\}$. On the other hand, we use the term *source sample or letter* to indicate the actual value of the partition number input to the encoder. In the last sections we discussed a partitioning scheme for alphabet symbols. Here we propose another partitioning method, but it applies to sets of source samples.

With the source alphabet reduced so dramatically by the alphabet partitioning method, we can try to exploit the dependence between several adjacent samples, and achieve compression ratios larger than that of the zero order entropy. In fact, lossy compression of images or video without an arithmetic encoder can only be efficient this way, as it is necessary to achieve rates like 0.1 bit/sample.

A straightforward solution is to aggregate several symbols (a technique called *source extension*) to form again a large alphabet. The extension order is constrained by complexity considerations, and even though we may have the same problems originally found with large alphabets, here the coding process is more efficient, as each extension symbol corresponds to several original source symbols, and it can yield rates below the zero-order entropy.

An equivalent approach is to use conditional coding. To keep the discussion simple (and include, for instance, Huffman codes) we discuss mostly the extension approach, but both are somewhat equivalent, and in our coding methods they are used together.

A practical problem when implementing the source extension is to find the number of symbols to be aggregated. The higher this number the better, but complexity limits this process, as the number of required extended-alphabet symbols increases exponentially. This problem is complicated by the fact that real applications (e.g., image, video, audio) correspond to sources with statistical properties that change drastically. A simple implementation of the alphabet extension must necessarily consider a worst-case scenario, which is surely not the best for the most common situations.

Here we propose adaptive schemes that address this problem. The objective is to separate the source letters according to their statistical distribution, and code each group accordingly. It is based on the fact that letters with the same distribution tend to occur in "clusters," which depend on the type of source. For example, with images we have two-dimensional regions. For audio it can represent a time range, or a two-dimensional region in a time-frequency range.

To simplify the exposition of the new coding method we begin with four examples, and then present a very general scheme, which can be extended and modified in many forms, and includes these four examples as special cases.

### Example 1: Groups of $4 \times 4$ Image Pixels

In an image coding example, let us suppose we want to code groups of $4 \times 4$ pixels, each one of which is a 16-letter source (the set numbers). We assume that the source letters are sorted according to their probabilities, with "0" representing the most common symbol, "1" the second most common, etc.

We may consider using a first extension, and code groups of two pixels with a 256-symbol alphabet, which is a reasonable size. If we decide to use a second or third extension, we need alphabets with 4,096 or 65,536 symbols, respectively, which may be too large. However, the vast majority of the extension symbols are expected to be all-"0", leading us to consider if it is possible to exploit this fact to use high order extensions, and still have alphabets with a small number of source symbols.

To achieve that goal we propose to code the data in that $4 \times 4$ group with the following algorithm:

1. Find the maximum value $v_m$ in the group of $4 \times 4$ pixels. (Here the values correspond to a set number.)
2. Entropy-code $v_m$; if $v_m = 0$ stop.
3. if $v_m$ is small use an $r$-order source extension to entropy-code the $16/r$ groups of $r$ pixels using a $(v_m+1)^r$-symbol alphabet; otherwise entropy-code the 16 pixel values with a $(v_m + 1)$-symbol alphabet.
4. Stop.

The stop conditions are defined at the moment that all values can be known by the decoder. For instance, if the maximum $v_m$ is zero then we can conclude that all 16 pixel values are zero. So in step 2 the encoder stops, but we should note that the decoder has to set the corresponding values (set numbers) to zero.

This simple scheme is very effective because it allows the proper adaptive selection of source extension and alphabet-sizes, according to the value distribution in that part of the image.

Indeed, in practice the most common case should be $v_m = 0$, followed by $v_m = 1$, $v_m = 2$, etc. Thus, if $v_m = 1$ or $v_m = 2$ we can, for example, choose in step 3 of the algorithm $r = 4$, and entropy code groups of $2 \times 2$ pixels using alphabets with 16 and 81 symbols, respectively. Similarly, if $v_m = 4$ or $v_m = 5$ we can choose $r = 2$, and entropy code groups of $2 \times 1$ pixels using alphabets with respectively 16 and 25 symbols. The choice of the best extensions depends on several practical factors. For example, adaptive encoders with large alphabets take a long time to gather reliable estimates of the source probabilities, and thus may not be suited when the image is small.

This entropy coding scheme has the following advantages.

1. In the most frequent situation the pixels in the $4 \times 4$ group are entropy coded with high-order extensions, which allows fully exploiting the statistical dependence between those adjacent pixels.
2. With the extensions it is possible to get very good results with simple and fast codes, which may have codewords with an integer number of bits. Indeed, in this example it is possible to obtain rates as low as 1/16 bit/symbol even with Huffman codes.
3. Because different alphabets are used for different values of $v_m$, this corresponds to entropy-coding the values *conditioned to* $v_m$, which is a measure of the activity in that $4 \times 4$ region. This property is the same employed by classifying encoders.

The algorithm cannot be arbitrarily extended to code any group of $n \times n$ pixels because we must also consider the overhead incurred by coding the maximum value $v_m$ (except when $v_m = 0$). Here the region size ($4 \times 4$) was chosen as a compromise between that overhead *versus* the gains obtained with extension and by exploiting the expected similarity inside the group. For a larger group (e.g., $8 \times 8$) the relative overhead (per pixel) may become insignificant, but the coding process is less adaptive. The opposite happens when the group is small (e.g., $2 \times 2$).

The example below shows a modification that can be used for coding sets with a small number of samples.

**Example 2: Groups of $2 \times 2$ Image Pixels**

Assuming the same conditions as in the example above, let us consider a more sophisticated scheme to code groups of $2 \times 2$ pixels. Note that we still may not be able to use source extension directly, as it yields an alphabet with 65,536 symbols.

When the set of samples (pixels) is small, we propose a coding scheme that is similar to the algorithm proposed in the previous example, but that reduces the overhead associated with coding the maximum values. The new proposed algorithm is

1. Find the maximum value $v_m$ in the group of $2 \times 2$ pixels.
2. Entropy-code $v_m$; if $v_m = 0$ stop.
3. Create a binary "mask" $\mu$ with 4 bits, each bit corresponding to a pixel in the $2 \times 2$ group. Each bit is set to 1 if the pixel value is equal to $v_m$ and otherwise set to 0.
4. Entropy-code $\mu$ using a 15-symbol alphabet (the case $\mu = 0$ never occurs). If $\mu = 15$ (binary [1111]) or $v_m = 1$ stop;
5. Let $r$ be the number of pixels with values smaller than $v_m$. If $v_m^r$ is sufficiently small then aggregate the $r$ symbols and entropy-code them with a $v_m^r$-symbol alphabet. Otherwise entropy-code each of the $r$ values with a $v_m$-symbol alphabet.
6. Stop.

Again, the stop conditions are defined by the knowledge available to the decoder. Here, for example, when $\mu = 15$ the decoder sets the four pixel values to $v_m$.

The overhead associated with coding the maximum value $v_m$ is minimized by the use of the binary mask because now only the pixel values smaller than $v_m$ need to be entropy-coded.

Again, the fact that different alphabets have to be used in different cases (values of $v_m$ and $\mu$) yields entropy-coding strongly conditioned to the data of the adjacent pixels. The condition above is an integral part of the method, but other extensions may also be advantageous. For instance, $\mu$ is always coded using a 15-symbol alphabet, but it may be better to code it conditioned to $v_m$.

Numerical results show that the compression does not degrade much if we do not aggregate the symbols in step 5. At the same time they show that the overhead of entropy-coding $v_m$ in step 2 can be still further reduced. In the next example we present a simple solution to this problem.

### Example 3: Recursive Coding of Groups of $2^n \times 2^n$ Image Pixels

Here we use basically the same algorithm proposed to code the group of $2 \times 2$ pixels. The difference is that coding is now done in an hierarchical and recursive manner. First the algorithm is used to code the maximum value in the group of $2^n \times 2^n$ pixels. Then a 4-bit mask is defined to indicate if the maximum in each of the four subgroups of $2^{n-1} \times 2^{n-1}$ pixels is equal to the maximum in the $2^n \times 2^n$ group. The mask is entropy-coded and later the maximum values in the four subgroups of $2^{n-1} \times 2^{n-1}$ pixels are also coded. This subdivision process is applied to code the values in gradually smaller blocks, until all the $2 \times 2$ subgroups are coded.

In other words, we use for the groups of $2^n \times 2^n$ pixels the same algorithm proposed to code the values in the $2 \times 2$ groups. The difference is that when $n > 1$ we code the maximum values in subgroups, and when $n = 1$ we code the actual pixel values.

Among the advantages of this extension of the coding method we can cite:

- it allows exploiting the statistical similarity in larger regions;
- extension alphabets with a small number of elements are automatically used in large regions where the values are uniformly small, extending the desired adaptive alphabet-selection;
- when there are large regions that have value 0, a single symbol can give information about a large number of pixels, allowing compression to rates much smaller than 1 bit/pixel, even when Huffman codes are used.

### Example 4: Spatial-Orientation Trees

The common feature in the previous examples is that basically we

1. define some sets of source samples,
2. entropy code the maximum value in each of those sets,
3. partition each set in a certain number of subsets,
4. possibly entropy code a binary mask indicating where is the maximum value to be found in the subset,
5. use the same basic algorithm to code the maximum in the subsets, until all single-sample values are coded.

Thus we have a great deal of flexibility in choosing the sets of samples, and implementing the coding method.

When the image is transformed using a wavelet transformation to construct a multiresolution pyramid,[6] there is a statistical dependence between the transformed pixel magnitudes following the pixels in different levels of the pyramid, but corresponding to the same spatial location. This property is exploited in another set-partitioning scheme.[11]

The same sets, and set partitioning rules defined in ref. 11 can be used in this new method to entropy-code wavelet-transformed images. A main difference between the method presented in ref. 11 and the method proposed here is that the first was designed for embedded coding, while here the methods are directly applied to an already quantized transformed image.

### General Scheme

A more general coding scheme, that includes the examples above, is:

1. Order the source symbols according to their probabilities, with "0" representing the most common symbol, "1" the second most common, etc.
2. Partition the source samples into a certain number of sets. (Here we refer to the samples put out by the source, the sets can be defined by image regions, a time range, etc.)
3. Create a list with the initial sets, find the maximum sample-value $v_m$ inside those sets and entropy-code those numbers.

4. For each set with $v_m > 0$ do

   (a) remove the set from the list and divide it into $n$ subsets;

   (b) entropy-code a binary "mask" with $n$ bits (one for each subset) such that a bit is set to 1 if the maximum in that subset, $v_{mi}$ is equal to $v_m$ and otherwise set to 0;

   (c) if $v_m > 1$ then entropy-code every $v_{mi} < v_m$, possibly aggregating them to create the proper source extensions;

   (d) add to the list of sets each subset with more than one element and with $v_{mi} > 0$.

5. If the set list is empty then stop; otherwise go to step 4.

Note that there is a great deal of freedom in the choice of the rules to partition the sets, the number of sets, the scheme to be used for entropy-coding the different type of data, etc. For instance, the binary "masks" were proposed to the cases when the number of subsets is small. However, when due to practical reasons, it is not possible to implement the recursive scheme, a direct implementation, like the one described in the first example can be employed. Also, in step 1 it is not always necessary to have the symbols strictly sorted. It may suffice to have the symbols partially sorted, according to some expected probabilities. (This avoids the effort of estimating the probabilities before or while coding.)

## 6 NUMERICAL RESULTS

### Alphabet Partitioning

Figure 2 shows how the individual contributions to total rate of the entropy of the set numbers and the rate of the set indexes change as the number of sets $N$ is reduced by the algorithm to find good partitions. The source used was a 12 bpp CT (tomography) medical image which was processed using the S+P transform,[12] which uses a small number of taps with rational values to produce integer transform coefficients without need for truncation.

Note that the entropy of the set numbers decreases as the number of sets is reduced, while the set index rate increases as more elements are added to the sets. The total rate given by the encoder is the sum of these two numbers. For large values of $N$ (number of sets nearly equal to the number of source symbols) we would have the set number entropy equal to the source entropy, and the set index rate equal to zero. In the example of Figure 2 this would occur for $N \approx 2^{14}$, and thus is not shown in the figure.

The important observation is that the sum is practically constant, and only increases notably when $N < 10$. Figure 3 shows the *relative* loss as a function of $N$. Note that we could choose for example $N = 11$ and have less than 2% relative loss. In conclusion, we initially had an alphabet with $2^{14}$ symbols, and have shown that it can be efficiently entropy-coded with a 11 symbol code, with only 2% compression loss.

Table 2 shows the results obtained with two medical images, but using the same partitions of the source alphabet. The two images have quite different activity levels. Nevertheless, the same partitioning scheme worked well for both. This shows that the partition choice is quite robust, i.e., we can choose one that can be effective for a wide range of similar sources, like medical images, audio, etc.

The use of the alphabet partitioning scheme with another entropy-coding method for lossless image compression, with several numerical results is presented in ref. 12. An example of those results is shown in Table 3, using the same medical images of Table 2. Note that the alphabet-partitioning permitted using quite sophisticated methods, which yield rates well below the zero order entropy, even when fast Huffman encoders are used.

### Sample Set Partitioning

In all image compression tests we used a single alphabet-partitioning scheme, which is shown in Table 4. The images used in the tests were the 512x512 8-bit monochrome Lena, Barbara, and Goldhill images. No training is required

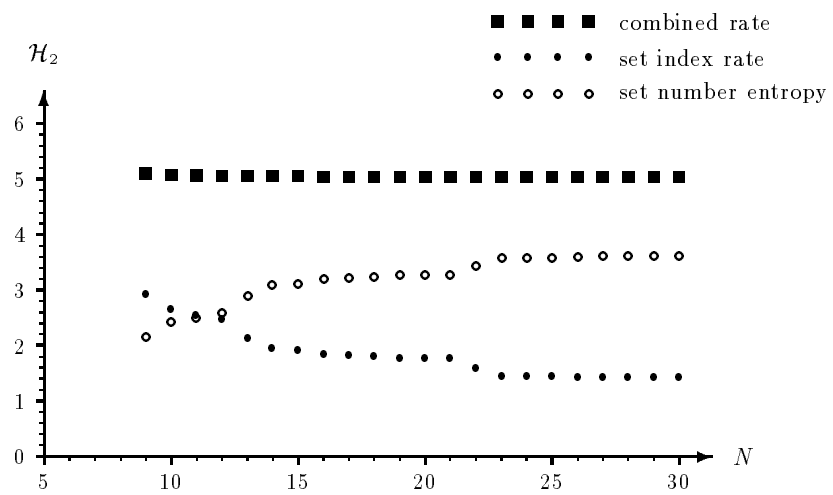


Figure 2: Variation of the contribution of the entropy of the set numbers plus the rate of the set indexes with the number of set $N$ (data source: transformed 12bpp medical image).

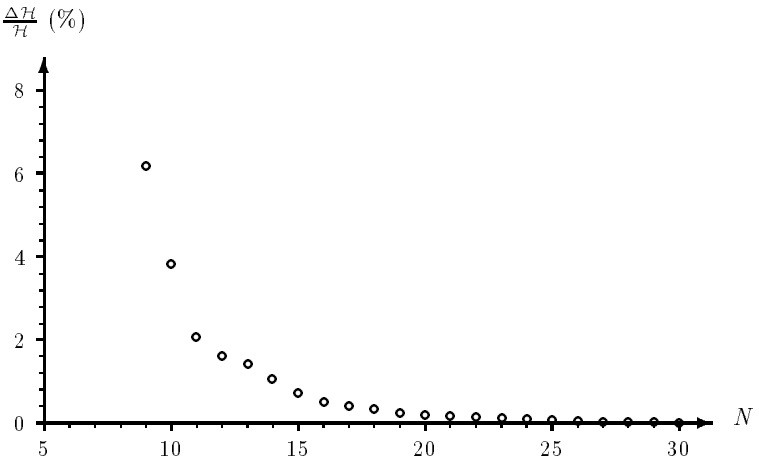


Figure 3: Relative loss in compression due to the reduction of the number of sets $N$.

| | direct method | | 13 sets | | | 23 sets | | |
|---|---|---|---|---|---|---|---|---|
| | values | entropy | set number entropy | set index rate | total | set number entropy | set index rate | total |
| image 1 | $\approx 2^{12}$ | 3.59 | 2.29 | 1.35 | 3.63 | 2.60 | 1.00 | 3.60 |
| image 2 | $\approx 2^{13}$ | 5.94 | 2.85 | 3.16 | 6.01 | 3.56 | 2.42 | 5.98 |

Table 2: Example of the rates (bpp) obtained with the alphabet partitioning method for lossless compression of medical images.

| image | entropy-coding method | |
|---|---|---|
| | arithmetic | Huffman |
| 1 | 2.80 | 2.89 |
| 2 | 5.26 | 5.30 |

Table 3: Lossless compression rates (bpp) obtained with an efficient entropy-coding method that uses the alphabet partitioning method complexity reduction (same images used in Table 2).

| magnitude set (MS) | amplitude intervals | sign bit | magnitude-difference bits |
|---|---|---|---|
| 0 | [0] | no | 0 |
| 1 | [–1], [1] | yes | 0 |
| 2 | [–2], [2] | yes | 0 |
| 3 | [–3], [3] | yes | 0 |
| 4 | [–5,–4], [4,5] | yes | 1 |
| 5 | [–7,–6], [6,7] | yes | 1 |
| 6 | [–11,–8], [8,11] | yes | 2 |
| 7 | [–15,–12], [12,15] | yes | 2 |
| 8 | [–23,–16], [16,23] | yes | 3 |
| 9 | [–31,–24], [24,31] | yes | 3 |
| 10 | [–47,–32], [32,47] | yes | 4 |
| 11 | [–63,–48], [48,63] | yes | 4 |
| 12 | [–127,–64], [64,127] | yes | 6 |
| 13 | [–255,–128], [128,255] | yes | 7 |
| 14 | [–511,–256], [256,511] | yes | 8 |
| 15 | [–1023,–512], [512,1023] | yes | 9 |
| ⋮ | ⋮ | ⋮ | ⋮ |

Table 4: Definition of the magnitude-sets used in the new image compression method.

in any of the tests. The software implementation is a full image coder and decoder, so that cited bit rates are scaled values of compressed file sizes.

The sample-set partitioning entropy-coding method was applied to images processed with the wavelet transform[6] and also the discrete cosine transform, each properly quantized (uniform quantization). The methods corresponding to examples 3 and 4 of Section 5 were implemented, using only Huffman codes for the data that require entropy-coding. Table 5 shows the combination of transforms and set partitioning schemes used for the tests, and their corresponding names.

Note that the methods that do not try to exploit the statistical dependence between subbands (called QT) yield results very similar to the methods that try (WV), using spatial orientation trees. Thus, in a parallel implementation, this allows efficiently coding the subbands independently, or spatial orientation trees independently.

The results for lossy compression are shown in Table 6. The wavelet-based are very similar to those obtained with the SPIHT[12] embedded coding method, but the best SPIHT results were obtained with an arithmetic encoder. The execution speed is similar to that of binary-uncoded SPIHT.

Table 7 shows the results for lossless compression. The best results were obtained with the method called LSS, which employs basically the scheme of example 3, but depending on the data it may not use a mask for all partitioned sets.

Those results are only slightly larger that those obtained with the best known lossless image compression methods. However, the important fact here is that the methods were equally efficient for both lossy and lossless compression,

| Method | Image transform | Sample-set partitioning | Entropy coding |
|---|---|---|---|
| WV-9/7 | wavelet pyramid - 9/7 tap filters[14] | example 4 | adaptive Huffman |
| WV-10/18 | wavelet pyramid - 10/18 tap filters[14] | example 4 | adaptive Huffman |
| QT-10/18 | wavelet pyramid - 10/18 tap filters[14] | example 3 | adaptive Huffman |
| QT-S+P | S+P pyramid[12] (lossless) | example 3 | adaptive Huffman |
| WV-S+P | S+P pyramid[12] (lossless) | example 4 | adaptive Huffman |
| LSS | S+P pyramid[12] (lossless) | similar to ex. 3 | adaptive Huffman |
| DCT-8 | $8 \times 8$ discrete cosine | example 4 | adaptive Huffman |
| DCT-16 | $16 \times 16$ discrete cosine | example 4 | adaptive Huffman |

Table 5: Methods implemented to test the proposed coding schemes.

| Image | Rate | Coding Method | | | | |
|---|---|---|---|---|---|---|
| | (bpp) | DCT-8 | DCT-16 | QT-10/18 | WV-9/7 | WV-10/18 |
| Lena | 0.25 | 32.51 | 33.46 | 34.16 | 34.10 | 34.25 |
| | 0.50 | 36.18 | 36.82 | 37.23 | 37.21 | 37.30 |
| | 0.75 | 38.27 | 38.70 | 39.01 | 39.00 | 39.04 |
| | 1.00 | 39.81 | 40.13 | 40.38 | 40.38 | 40.45 |
| Barbara | 0.25 | 27.36 | 28.93 | 28.73 | 28.45 | 28.67 |
| | 0.50 | 31.58 | 32.98 | 32.87 | 32.25 | 32.74 |
| | 0.75 | 34.67 | 35.89 | 35.86 | 35.20 | 35.77 |
| | 1.00 | 37.09 | 38.13 | 38.16 | 37.54 | 38.09 |
| Goldhill | 0.25 | 29.89 | 30.41 | 30.50 | 30.53 | 30.60 |
| | 0.50 | 32.69 | 33.11 | 33.17 | 33.13 | 33.17 |
| | 0.75 | 34.64 | 34.98 | 35.08 | 34.94 | 35.13 |
| | 1.00 | 36.28 | 36.56 | 36.67 | 36.53 | 36.67 |

Table 6: Peak signal to noise ratios (PSNR, in dB) obtained with entropy-coding methods that use the alphabet and sample-set partitioning scheme.

covering a very wide range of bit rates (e.g., from 0.25 to 4 bpp), and using only fast Huffman codes.

## 7 CONCLUSIONS

We proposed two new methods for efficient compression of waveform sources: (a) an alphabet partitioning scheme that allows dramatic reduction in the complexity of coding those sources; and (b) a new recursive set-partitioning entropy-coding method for blocks of samples which is shown to be as efficient as the best known methods.

| Method | Lena | Barbara | Goldhill |
|---|---|---|---|
| WV-S+P | 4.28 | 4.65 | 4.83 |
| QT-S+P | 4.27 | 4.63 | 4.82 |
| LSS | 4.25 | 4.61 | 4.81 |

Table 7: Bit rates (bpp) for lossless recovery when the proposed entropy-coding methods were applied to the S+P lossless multiresolution transform.[12]

## REFERENCES


1. M. Rabbani and P.W. Jones, *Digital Image Compression Techniques,* SPIE Opt. Eng. Press, Bellingham, Washington, 1991.
2. G.K. Wallace, "The JPEG still picture compression standard," *Comm. ACM*, vol. 34, pp. 30–44, April 1991.
3. S. Todd, G.G. Langdon, Jr., and J. Rissanen, "Parameter reduction and context selection for compression of gray-scale images," *IBM J. Res. Develop.,* vol. 29, pp. 188–193, March 1985.
4. M. Rabbani and P.W. Melnychuck, "Conditioning contexts for the arithmetic coding of bit planes," *IEEE Trans. Signal Processing*, vol. 40, pp. 232–236, Jan. 1992.
5. S.D. Stearns, "Arithmetic coding in lossless waveform compression," *IEEE Trans. Signal Processing*, vol. 43, pp. 1874–1879, Aug. 1995.
6. J.W. Woods, ed., *Subband Image Coding,* Kluwer Academic Publishers, Boston, MA, 1991.
7. A. Said and W.A. Pearlman,"Reversible image compression via multiresolution representation and predictive coding," *Proc. SPIE* vol. 2094: Visual Commun. and Image Processing, pp. 664–674, Nov. 1993.
8. I.H. Witten, R.M. Neal, and J.G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, pp. 520–540, June 1987.
9. W.B. Pennebaker, J.L. Mitchell, G.G. Langdon Jr., and R.B. Arps, "An overview of the basic principles of the Q-coder adaptive binary arithmetic coder," *IBM J. Res. Develop.,* vol. 32, pp. 717–726, Nov. 1988.
10. A. Said and W.A. Pearlman, "Reduced-Complexity Waveform Coding via Alphabet Partitioning," *IEEE Int. Symposium on Information Theory,* Whistler, B.C., Canada, p. 373, Sept. 1995.
11. A. Said and W.A. Pearlman, "A new fast and efficient codec based on set partitioning in hierarchical trees," *IEEE Trans. on Circuits and Systems for Video Technology,* vol. 6, pp. 243–250, June 1996.
12. A. Said and W.A. Pearlman, "An image multiresolution representation for lossless and lossy compression," *IEEE Trans. on Image Processing,* vol. 5, pp. 1303–1310, Sept. 1996.
13. M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. Image Processing,* vol. 1, pp. 205–220, April 1992.
14. M.J. Tsai, J.D. Villasenor, and F. Chen "Stack-run image coding," *IEEE Trans. on Circuits and Systems for Video Technology,* vol. 6, pp. 519–521, Oct. 1996.