Priority-Based Arithmetic Coding for Wavelets

Dane K. Barney*

June 10, 2005

Abstract

JPEG-2000, the successor to JPEG, is emerging as an attractive candidate for the future de facto image compression standard. However, while it offers superior results over most other image compression techniques, it comes with a long list of features and options, and consequently a highly complex algorithm. We have developed an image coder that can compete with the performance of JPEG-2000 while offering a much simpler design. *Priority-Based Arithmetic Coding for Wavelets (PACW)* involves a wavelet transform followed by arithmetic coding of bit-planes in a prioritized order, achieving lossy compression of grayscale images that can match, and occasionally outperform, JPEG-2000.

1 Introduction

The PACW project began in the summer of 2003. The original intent was to develop an image coder that could perform at the level of other wavelet coders, like JPEG-2000, yet would offer a more elegant and straightforward algorithm than JPEG-2000, making it more suitable for teaching in an undergraduate data compression course. JPEG-2000 was a common part of the undergraduate curriculum because it introduced students to many of the important concepts underlying the latest generation of wavelet coders, such as the discrete wavelet transform, binary arithmetic bit-plane coding, and Trellis coded quantization. However, since JPEG-2000 has become a commercial coder rather than a research coder, it contains countless bells and whistles. Thus, while it may be used to introduce important general concepts of wavelet-based image compression, to get a full picture of JPEG-2000, students must also be exposed to a torrent of extra features like progressive transmission by quality, resolution, component, or spatial locality; random spatial access to the bitstream; dyadic or packet decomposition; pan and zoom; rotation and cropping; and region of interest coding by progression. The more streamlined PACW was therefore intended to replace JPEG-2000 in the curriculum. It would include many of JPEG-2000's important aspects but exclude

*Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195. dane@cs.washington.edu. Research supported by Mary Gates Endowment at the University of Washington and by NSF Grant No. CCR-0104800 through an NSF Research Experiences for Undergraduates (REU) award.

most of its frills, yielding a more user-friendly coder for the introduction of wavelet coding concepts to an undergraduate class.

The starting point for PACW was the Group Testing for Wavelets (GTW) coder [1] developed by Hong and Ladner in 2000. GTW, like JPEG-2000, is an embedded wavelet coder, meaning it employs bit-plane coding of wavelet coefficients. However, its novelty lies in its use of Hwang's group testing algorithm [2] as a means of entropy coding, and with it a method of handling coefficients in the significance pass that is more suited to group testing. The PACW coder is essentially GTW with the group testing stages removed and replaced with arithmetic coding, making it more similar to JPEG-2000. Along with this replacement of the entropy coder, certain features of GTW were dropped, such as pattern type, while other features were added, such as PACW's novel use of a priority queue to dictate the order in which coefficients are coded. The result was a coder which maintained the simplistic nature of the GTW research coder, and provided a compression performance that could effectively compete with JPEG-2000, yielding results at or above JPEG-2000's level in compressing several different images.

Having met our original goal, a new goal was set in the summer of 2004 to build upon and improve the original PACW coder through various experiments in *context modeling*, in an attempt to establish a more intelligent means of handling coefficients through contexts during the arithmetic coding stage. Our aim is to push the performance of PACW to its limit without sacrificing its simplicity. The original version, which did not consider all optimizations, is dubbed Version 1, and this new version, which does consider optimizations, will be called Version 2. It is this move from Version 1 to Version 2 which is the primary subject of this paper.

The remainder of this paper is arranged as follows. After a brief explanation of rate-distortion curves, Section 2 gives an overview of the encoding phase of PACW, focusing primarily on the theory behind the methods involved at each stage and the integration of progressive stages. Section 3 is then devoted to the research involved in working from Version 1 to Version 2 of PACW, detailing each study in context modeling that was undertaken, and following the sequential changes in compression performance with each optimization. Finally, Section 4 gives a brief synopsis of related work in wavelet coding, followed by a rate-distortion comparison of the overall performance of PACW Version 2 vis-a-vis other wavelet coders.

1.1 Measuring Compression Performance

The standard method of measuring compression performance uses a *rate-distortion curve*. Throughout this paper, rate-distortion curves will be shown to visually depict the relative performance of multiple compression schemes. A rate-distortion curve is a plot of the quality of a reconstructed image, measured at all possible file sizes for the compressed image. The size of the compressed image, along the x-axis, is given as a *bit rate*, which is simply the total number of bits taken up by the compressed image divided by the total number of pixels in the

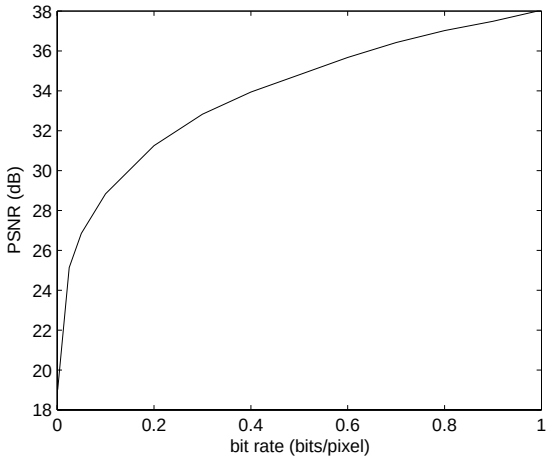


Figure 1: A typical rate-distortion curve.

image, resulting in units of bits per pixel. For example, an uncompressed grayscale image, which contains 256 possible shades of gray for each pixel, has a bit rate of exactly 8 bits/pixel.

Along the y-axis is the image quality, which is described in terms of *peak signal to noise ratio (PSNR)*, measured in decibels (dB). Since we are dealing specifically with lossy compression, the reconstructed image is not necessarily identical to the original image; some of the data is lost in the compression process. PSNR is intended to measure the perceptible quality of an image by mathematically describing the pixel difference, or distortion, between the original image and the reconstructed image. Specifically, PSNR is a logarithmic function of the inverse *mean squared error (MSE)* between the pixels of the original and reconstructed images. If $X = (x_1, x_2, ..., x_n)$ is the set of values for the n pixels of an image and $X' = (x'_1, x'_2, ..., x'_n)$ is the set of pixel values for the reconstructed image, then the MSE, σ^2, is defined as

$$\sigma^2 = \sum_{i=1}^{n} (x_i - x'_i)^2 / n \tag{1}$$

PSNR can then be defined in terms of MSE with the following equation:

$$10 \log_{10} \left(\frac{M^2}{\sigma^2} \right) \tag{2}$$

where M denotes the maximum possible value for any pixel. In the case of grayscale images, this value is 255. As PSNR increases, the quality of an image increases. An image with a PSNR of about 40 dB is of very high quality and visually indistinguishable from the original image.

Figure 1 shows an example of a rate-distortion curve. As can be seen, the PSNR increases as the bit rate increases. However, the increase in PSNR is steepest at bit rates near zero, and begins leveling out as the bit rate gets higher. This is because the very first bits of an

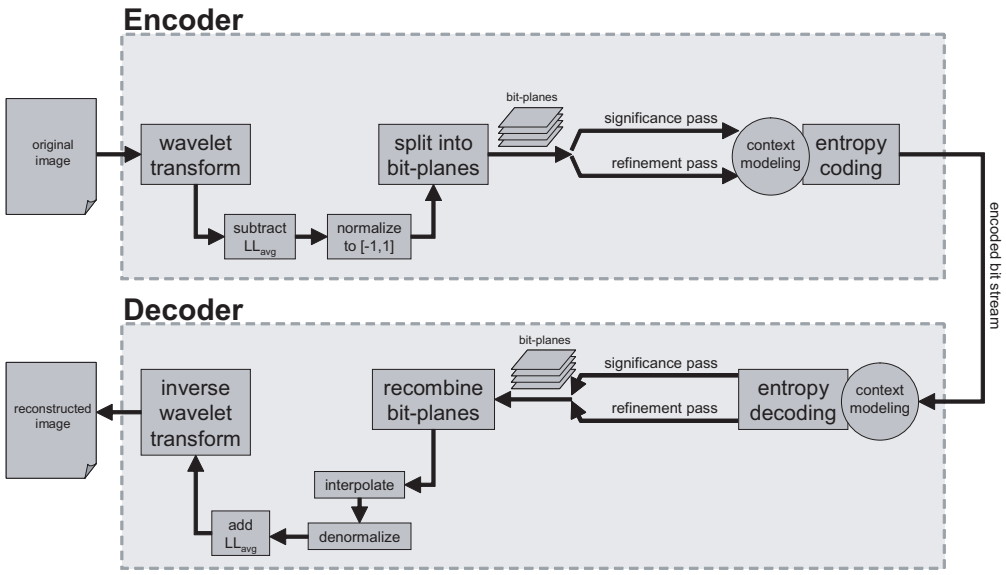


Figure 2: Block diagram of both the encoding and decoding phases of the PACW algorithm.

embedded bitstream represent the most important information about the overall characteristics of an image, hence they have a much greater impact on the reconstruction quality than do the bits farther down in the embedded bitstream. In general, the goal for any coder is to have the rate-distortion curve as far up and to the left as possible. This indicates that the coder is able to compress images to an extremely small file size and still maintain a high level of quality in the reconstructed image.

2 The PACW Architecture

The general algorithm involved in wavelet-based image compression is fairly standardized. Compressing—or encoding—an image involves a wavelet transform, followed by an optional quantization step, and finally bit-plane transmission of the wavelet coefficients to some type of entropy coder. While PACW follows this scheme, its uniqueness among other wavelet coders lies in:

- Its treatment of coefficients in a significance and refinement pass
- The manner in which coefficients with similar properties are grouped into contexts
- The use of a priority queue to determine the order in which bits are entropy coded

For a full discussion of the details behind PACW's priority queue, refer to Section 3.6. In this section, we provide a brief overview of the PACW architecture, walking through each stage of the encoding procedure. Decompressing—or decoding—an image is simply the inverse function of encoding, in which each step performed in the encoder is undone in reverse order. The block diagram of PACW, shown in Figure 2, illustrates this point.

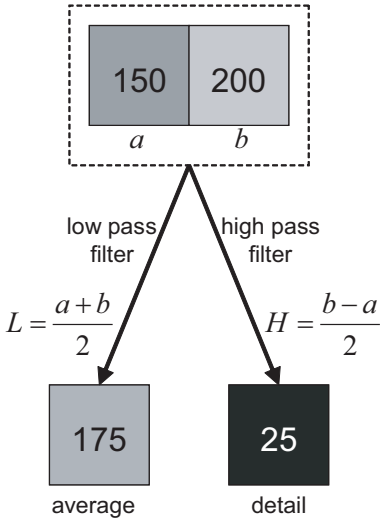


Figure 3: One-dimensional average transform on two samples.

2.1 Wavelet Transform

The first stage of the PACW algorithm is a wavelet transform. This is a special kind of linear transform which decomposes an image into a low resolution version with detail data. Specifically, the image pixels are run through a set of two filters, one which creates a down-sampled version of the original pixel values, and another which creates complementary detail pixels. Figure 3 shows a simple one-dimensional average transform. Two adjacent samples are filtered through a low pass filter (L), which generates the average of the two samples (the low resolution version), and a high pass filter (H), which generates the difference between this average and the actual values (the detail data).

This averaging filter can be easily extended to two dimensions by first filtering samples horizontally and then vertically. This process is illustrated in Figure 4. An image—a two-dimensional matrix of samples—is first horizontally transformed, creating two half-width pieces, or *subbands*: (1) the low resolution subband from the low pass filter (L), containing the average of every two neighboring pixels, and (2) the detail subband from the high pass filter (H), containing all the difference values. These two subbands can then be treated together as a matrix of samples and further decomposed through a vertical transform, resulting in four total subbands. The upper-left subband now contains a quarter-sized averaging of the original image, and is referred to as the LL subband since it has gone through two low pass filters. The other three subbands are considered detail subbands since they have all been filtered through at least one high pass filter. This two-dimensional transform can then be repeated on only the LL subband, decomposing it into four subbands, and so on. Each iteration creates a new *subband level* consisting of an LL, LH, HL, and HH subband. This recursive process can continue until the coder instructs it to halt, which in PACW happens once the width or height of the LL subband becomes smaller than 16 pixels in size. The end result is a very small version of the image in the final LL subband, and several levels of

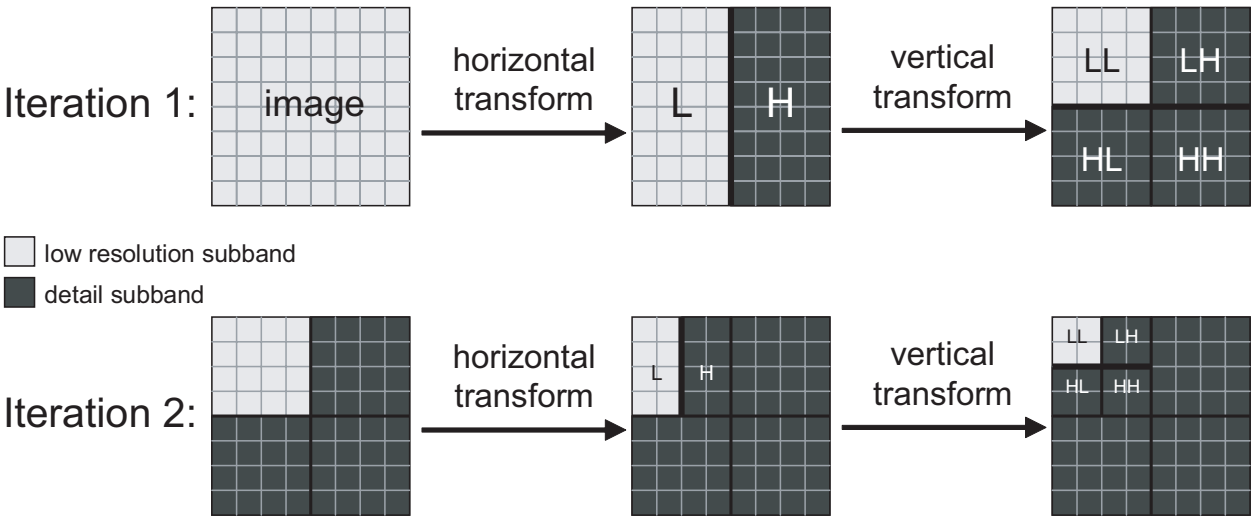


Figure 4: Two-dimensional wavelet transform on an image, showing two iterations.

detail subbands.

The purpose of a wavelet transform is to condense the image energy into the LL subband, leaving the majority of the spatial area to contain less important detail data. Thus, the LL subband contains values much larger in magnitude than the detail subbands, which are largely comprised of values at or near 0. The LL subband can then be treated with more importance than the detail subbands, and transmitted first. In PACW, we further decrease the magnitude of coefficients in the LL subband by subtracting the average value from all of its members. This value, which we refer to as LL_{avg}, must then be sent as side information to the decoder.

Additionally, rather than use the simple averaging filter shown in these examples, PACW employs a more sophisticated set of filters known as Daubechies 9/7 [3]. This is the same set of wavelet filters used by JPEG-2000 and GTW, among other wavelet coders. Since the Daubechies 9/7 filters contain floating-point values, the wavelet coefficients in the transformed image are all floating-point numbers. These numbers must be normalized between -1 and 1 in preparation for bit-plane coding. This normalization is done by dividing every value by the largest absolute magnitude, which must be sent as side information so that the decoder will know the value by which to multiply all coefficients.

2.2 Bit-Planes

The next stage in our algorithm is the splitting of coefficients into bit-planes. We now have an $m \times n$ matrix of normalized wavelet coefficients and by treating each value in binary, we can split up the values into separate bits and form $m \times n$ bit-planes which contain only one bit from each value. Thus, the first bit-plane contains the most significant bit of every coefficient, the second bit-plane contains the next most significant bit, and so on. This concept

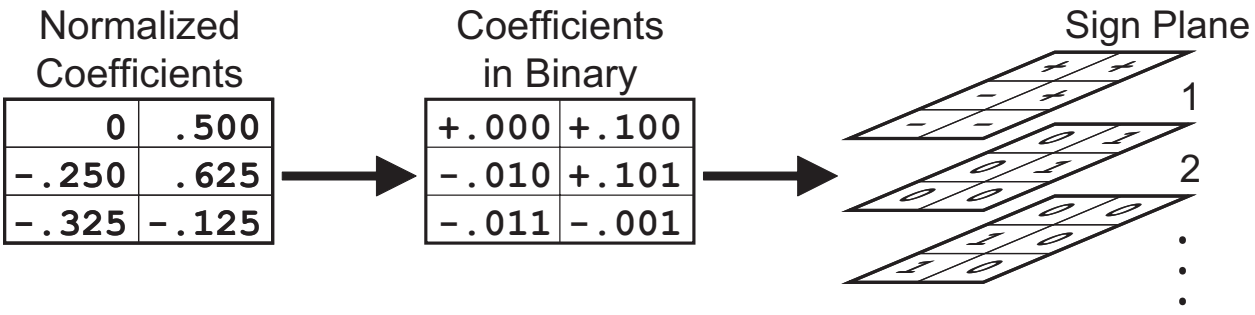


Figure 5: Bit-plane coding of normalized wavelet coefficients.

is illustrated in Figure 5. Additionally, the sign of each coefficient is extracted and used to form its own bit-plane called the sign plane. We can then transmit the data on a bit-plane by bit-plane basis, beginning with the most significant bit-plane. Each subsequent bit-plane provides the decoder with an additional level of refinement by appending one more bit onto each value. Consequently, we can abort the transmission of the bit-planes prematurely and the decoder will still have a full reconstruction of the image, just at a coarser level of detail. This is what is known as an *embedded bitstream*, wherein data is progressively transmitted and can be stopped at any point, yielding different approximations of the original image.

The idea behind bit-plane transmission is that the first several bit-planes will contain mostly "0" bits, given that most wavelet coefficients are equal or close to 0. Segmenting the data into largely uniform partitions like this greatly enhances the compression performance attained by the entropy coder in the next stage. However, we can further partition the data by transmitting each bit-plane in two separate passes: a significance pass and a refinement pass. During the transmission of bit-planes, each coefficient is labeled *insignificant* until its first "1" bit is reached. Once a coefficient becomes significant, all of its subsequent bits are *refinement bits*. Figure 6 shows a list of coefficients in binary with the refinement bits shaded. We see that at bit-plane 3 coefficients 1, 7, and 8 are already significant; coefficients 2 and 9 are becoming significant; and all other coefficients are still insignificant.

For each bit-plane, we first transmit all coefficients which are still insignificant in the *significance pass*. Amidst all of the insignificant coefficients transmitted, some will likely become significant in the current bit-plane. In such a case, the coefficient's "1" bit is transmitted to the entropy coder followed by the sign bit for that coefficient, to indicate whether the value for the coefficient is to interpreted as positive or negative. Despite these rare occurrences in which a coefficient becomes significant in the significance pass, the majority of bits encountered during this pass will be 0. This is especially true during the first few bit-planes when only the largest-magnitude coefficients in the LL subband are becoming significant. In PACW, we attempt to further optimize the level of compression attainable in the significance pass by transmitting the bits in a prioritized order. The order is based on how likely a coefficient is to become significant at the current bit-planes, such that the coefficients which are to become significant at the current bit-plane will be transmitted before the coefficients which

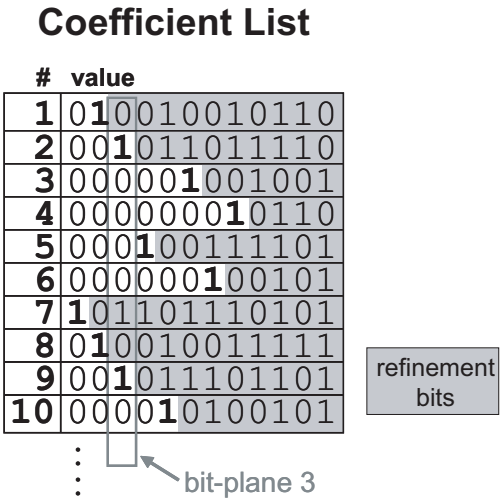


Figure 6: The refinement bits in a list of binary coefficients.

will remain insignificant. This prioritized ordering is explained in further detail in Section 3.6.

Following the significance pass is the refinement pass, in which all coefficients which have previously become significant in a past bit-plane send a bit-plane of refinement bits to the entropy coder. Unlike the bits encountered in the significance pass, which are mostly 0's, the refinement bits exhibit a more random distribution of 0's and 1's. Since there is much less structure within the refinement, it is much more difficult to compress. By far the largest proportion of an embedded bitstream belongs to refinement bits.

2.3 Entropy Coding with Context Modeling

The final stage of the encoding process is entropy coding of the bit-planes. An *entropy coder* is a general purpose lossless compression algorithm. It makes no inference on the type of data represented, it simply maps input symbols to bit codes with the aid of a probability model. Thus, an entropy coder takes as input a sequence of source symbols together with their respective probabilities, and is able to output a compressed binary string by using the probabilities as a means of prediction. There are many entropy coders of varying complexity, but the one we chose for PACW is *arithmetic coding*. Arithmetic coding is also the entropy coder used in JPEG-2000 and many other wavelet coders. The details of the inner-workings of arithmetic coding are too involved to explain in this paper; what is most important to discuss is the specific type of arithmetic coding employed in PACW and its properties.

The type of arithmetic coding used in PACW is called *context-based adaptive binary arithmetic coding (CABAC)*. "Binary" refers to the fact that there are only two different input symbols handled: 0 and 1. "Adaptive" refers to the fact that the probability model changes as each input symbol is encoded. Rather than use fixed values as the probability

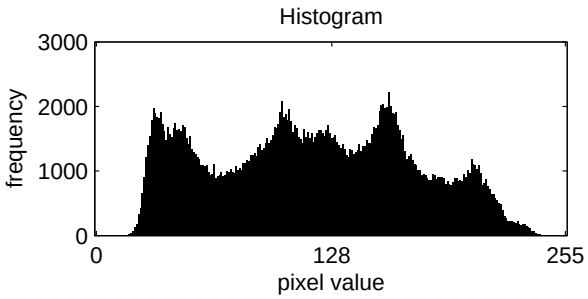
of each input symbol, the values are constantly refreshed and updated to reflect the actual data being encoded. Thus, we begin with the equal probability model—or alternatively, a weighted probability model based on empirical data—and upon coding each new input symbol, we increment its frequency by 1, thereby updating its probability to more accurately represent the true symbol frequencies in the input data.

Finally, "context-based" means that there is more than one probability model used by the entropy coder. Each represents a different context within the data, based on past input symbols. For example, there could be a context corresponding to the event in which the most recently coded symbols were both 1's, and another context for the event in which they were both 0's. Then, instead of only one probability model containing the probabilities $P(0)$ and $P(1)$, we would have a probability model containing $P(0|00)$ and $P(1|00)$, another containing $P(0|11)$ and $P(1|11)$, and so forth. By maintaining unique probability models for each of these different situations, the probabilities are consequently more accurate because they are custom tailored to a specific situation; they are uncorrupted by potentially conflicting probabilities contained in some other context. Now, the question of what to base these contexts on and how many to use is a strategy called *context modeling*. This is a gray area in the sense that it is open to ingenuity and experimentation, and can have a dramatic impact on the overall compression performance of a coder. For this reason, we devote the next entire section of this paper to our experimentations with context modeling in PACW, as we seek to optimize PACW's performance before comparing it with JPEG-2000 and other competitors.

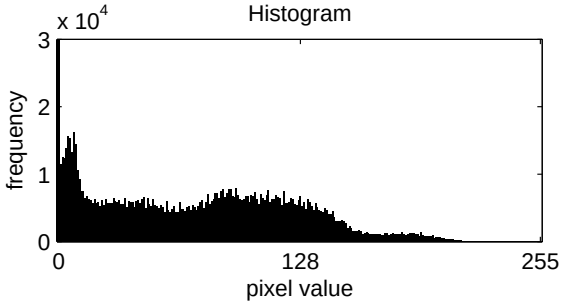
3 Experiments in Context Modeling

We now present the details of the various studies performed on PACW in our attempts to move from Version 1 to a more optimized Version 2. Virtually every study we undertook with PACW dealt in some way with context modeling, given that it is the one area of the PACW architecture with the most room to play while still leaving the general procedure intact. Context modeling itself occurs during the significance pass. All the coefficients are separated into different contexts based upon certain criteria, and each context is assigned a unique probability model for the entropy coder to use in the encoding of that context's coefficients. In grouping together coefficients with similar properties, the aim is to improve the prediction accuracy of a coefficient's significance, and in so doing increase the compression. Determining which criteria to use for classification is the key to the amount of compression gain achieved. Our initial method of coefficient classification was based upon two criteria: (1) the subband level and (2) the number of significant coefficients around it. From this rudimentary setup, we experiment with several variations in an effort to locate an optimal point of compression performance. This is measured through comparative rate-distortion curves from several different test images. The images selected as our test set are shown and described in Figure 7.

"Barbara"
512×512 pixels



"Man"
1024×1024 pixels



"Zebra"
400×561 pixels

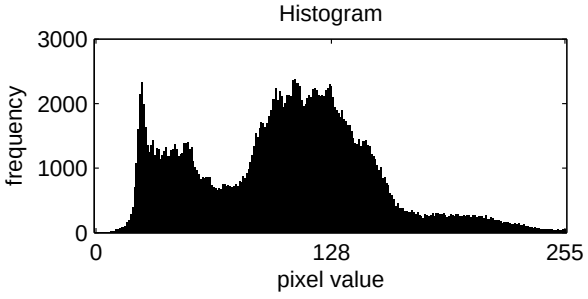
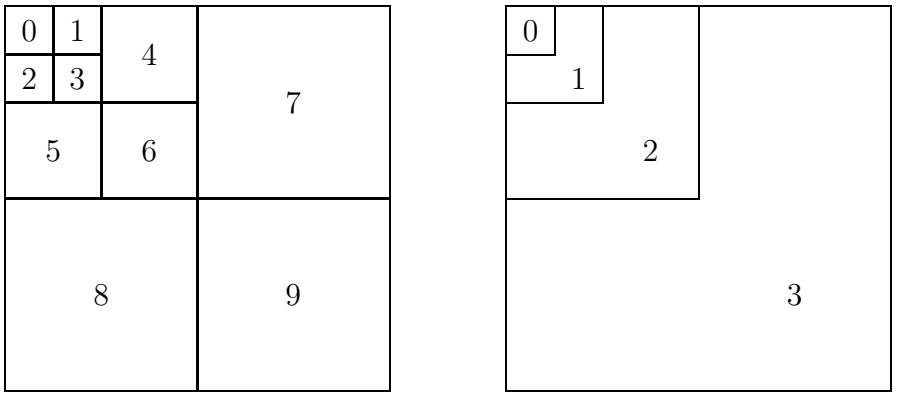


Figure 7: Test set of images used in the experiments.



(a) Subbands (b) Subband levels

Figure 8: Decomposition of an image into subbands, and the corresponding layout of subband levels.

3.1 Subband Levels

As mentioned previously, the effect of the wavelet transform is to condense the energy of the image into the upper-left corner of the image space. This means the coefficients in the lower frequency subbands are of much larger magnitude than those in the higher frequency subbands. Because of this large difference in coefficient magnitude, it follows that classifying coefficients based on their subband would be advantageous. We seek to verify this theory by comparing the compression performance yielded from three different context models: (1) classifying coefficients based upon their subband number, (2) classifying based upon their subband level, and (3) ignoring subbands entirely during classification. A *subband level* is simply the group of three subbands at the same detail resolution, as shown in Figure 8. Subband 0 is the one exception to this, as it constitutes an entire level alone. Our original hypothesis was that grouping coefficients by subband level would yield better results than grouping by subband number, because it would capture the inherent differences in coefficient significance among the different subbands yet would avoid the effects of *context dilution* which might result from treating each subband individually. Context dilution occurs when such a large number of contexts are used that the data is spread too thin amongst them and its characteristics are not accurately captured, resulting in a degradation of compression efficiency.

However, the results, shown in Figure 9, reveal that this is not case. Treating each subband individually actually yields slightly better compression than combining them into levels. This is true for all three test images. Yet this difference is negligible, as we see that all three context models—even the model which ignores subbands completely—give virtually the same results. For this reason, the Version 2 of PACW keeps the original model of classifying by subband level.

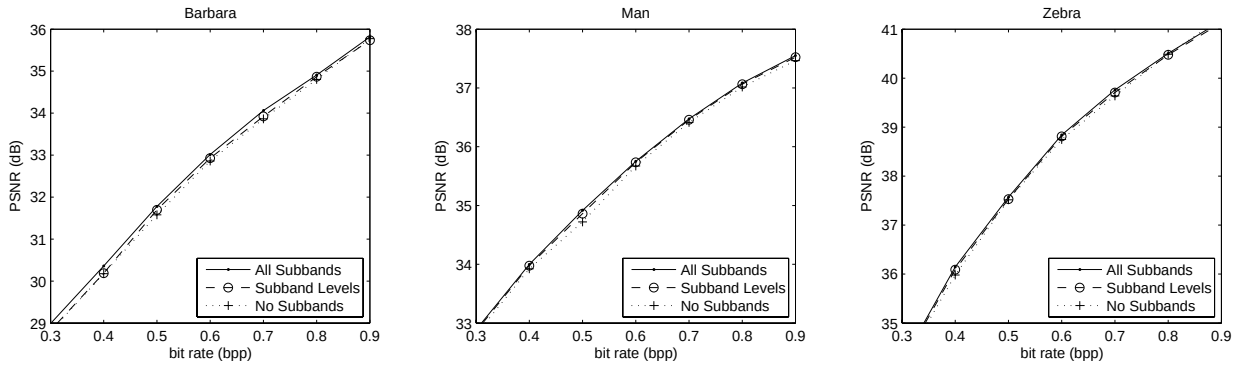


Figure 9: Rate-distortion curves of three images which compare three ways of treating subbands during classification: (1) treating each subband individually, (2) grouping them into subband levels, and (3) ignoring them altogether.

3.2 Significant Neighbors

The second criterion used for coefficient classification in Version 1 of PACW is the number of significant neighbors surrounding each coefficient. If a given coefficient is surrounded by many coefficients which have already become significant, it is likely that it will soon become significant itself, more so than a coefficient which is amongst a patch of insignificant coefficients. Thus we should treat these different situations with different contexts. This idea is a very common feature implemented in image coders today, yet it is an idea which involves a number of variables, such as how many significant neighbors to consider and where these neighbors should be located relative to the current coefficient. It is these questions which provide the basis for the next set of studies.

Initially, the significant neighbor metric implemented in PACW was taken directly from GTW [1]. The GTW significant neighbor metric checks 15 different neighbors for significance during the classification of a coefficient. These 15 neighbors are specially located based upon the correlation that exists among coefficients in a wavelet-transformed image. As Figure 10 shows, the set of neighbors include the eight spatially adjacent coefficients immediately surrounding the coefficient in the same subband, the two spatially identical coefficients in the same local positions of the other subbands at the same level, the parent coefficient in the corresponding position of the next lower subband, and the four child coefficients in the corresponding positions of the next higher subband. In counting the significant neighbors, each of the eight spatially adjacent, two spatially identical, and one parent contribute individually to the total, but the four children together only count as one significant neighbor if any of the four is significant. This is because these four children taken together represent one single spatially equivalent position in the next higher subband level. Thus, the maximum count of significant neighbors is 12 even though there are 15 total neighbors. However, this does not mean that 12 different contexts are created based upon this count. Rather, in GTW, there are only four different contexts resulting from the significant neighbor count, which correspond to 0, 1, 2, and ≥ 3 significant neighbors. Grouping all coefficients with ≥ 3

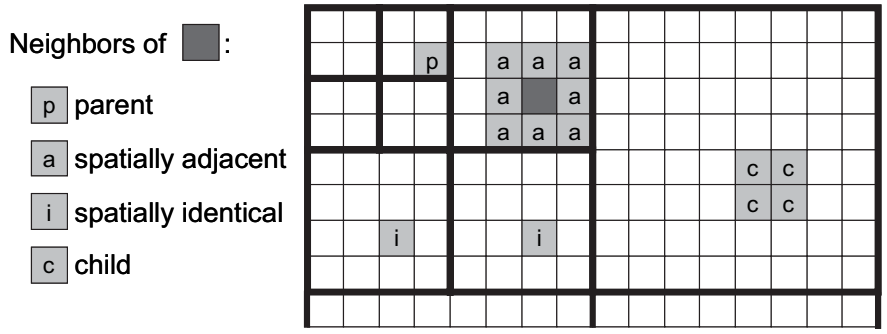


Figure 10: Neighbors of a coefficient in a wavelet-transformed image, as defined by the GTW algorithm.

significant neighbors into the same context is based upon the observation that contexts with that many significant neighbors tend to have very similar properties, and in reducing the number of contexts to four, possible context dilution is avoided.

With this metric used as a starting point, we attempt to find an optimal level for PACW by making various adjustments to the GTW metric and comparing the compression performance. Four different models are compared: Model 1 looks at only the spatially adjacent neighbors, model 2 looks at the spatially adjacent and spatially identical neighbors, model 3 looks at the spatially adjacent and parent/child neighbors, and model 4 is the original GTW metric. Additionally, for each of these four models we study the effect of incrementally increasing the number of the significant neighbor contexts. With only one context, all coefficients are grouped together based on having ≥ 0 significant neighbors. With two contexts, coefficients are classified based on whether they have 0 or ≥ 1 significant neighbors. Three contexts corresponds to 0, 1, ≥ 2 significant neighbors, and so on. The results of this study are shown in Figure 11, with the performance of the four models at a single bit rate (0.50 bits/pixel) plotted against the number of contexts.

The relative performance of the four models varies depending on which image is being compressed. "Barbara" exhibits the most regularity and consistency, whereas with "Man" and "Zebra" the top-performing model changes with the number of contexts. If there is one clear thing that can be concluded from all three plots, it is that the inclusion of spatially identical neighbors in fact hurts the compression performance. If we say models 1 and 2 are counterparts and models 3 and 4 are counterparts, in the sense that they differ only in the inclusion or exclusion of spatially identical neighbors, then we can see from the plots that the models which exclude spatially identical neighbors almost always outperform their counterpart. So, with spatially identical neighbors safely out of the picture, only models 1 and 3 need be considered, and the question becomes whether or not the parent and child neighbors are worth including. For "Barbara" the answer is no, since model 1 is consistently better than model 3, but for "Man" and "Zebra" the answer is yes. Now, since model 3 is the best for "Man" and "Zebra" and the second best for "Barbara," whereas model 1 is the best for "Barbara" but one of the worst for "Man" and "Zebra," it seems that the best

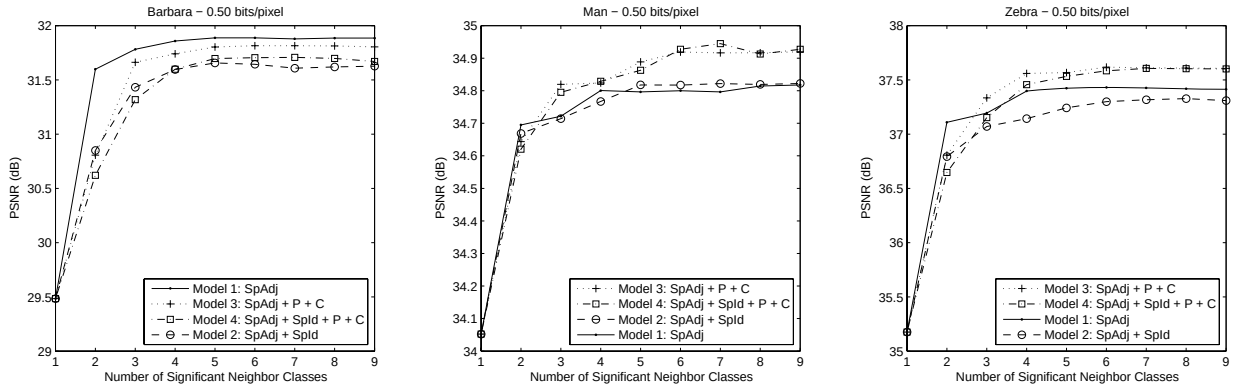


Figure 11: Rate-distortion curves of three images which compare four different context models, who look at different combinations of neighbor types, across an incrementing number of significant neighbor contexts. "SpAdj" stands for eight spatially adjacent neighbors, "SpId" stands for the two spatially identical neighbors, "P" is the parent neighbor, and "C" represents the four children neighbors.

concession we can make is to use model 3. Therefore, for our optimized version of PACW we adopt model 3, which looks at the spatially adjacent, parent, and child neighbors, but not the spatially identical neighbors.

In terms of the number of significant neighbor contexts, the plots show that the compression performance tends to flatten out once we get up to around four, five, or six contexts, depending on the image. Table 1 shows an alternative way of representing this information. It lists the total count of significant coefficients and insignificant coefficients classified as having 0 through 8+ significant neighbors, and shows the percentage of significant coefficients for each context. Note that model 3 was used to generate this table, so only the spatially adjacent, parent, and child neighbors were considered. The significant and insignificant counts were accumulated across eight bit-planes, and all neighbors which had not yet been encoded were treated as insignificant. This table reveals that the proportion of significant coefficients approaches 50% as the number significant neighbors increases, with the last few contexts exhibiting very similar properties. For example, in the table for "Barbara," the first several rows have very different proportions of significant coefficients. This difference tapers off once we get up to five significant neighbors, where we see that the difference in proportion between four and five significant neighbors is 8.8% compared to the difference between five and six significant neighbors, which is only -1.6%. Similarly, in the table for "Zebra," the large changes in proportion diminish from 7.8% between four and five significant neighbors, to 1.9% between five and six significant neighbors. The fact that the last few contexts exhibit the same behavior, with roughly 50% significant coefficients, means that these contexts can safely be grouped into one context. Furthermore, the cut-off point at which to create this final catch-all context seems to be at five significant neighbors, giving us a total of six contexts (0, 1, 2, 3, 4, or ≥ 5 significant neighbors). This is the metric we adopt for Version 2. In contrast, recall that GTW uses four contexts (0, 1, 2, ≥ 3 significant neighbors).

Sig Nbrs	Barbara			Man			Zebra		
	# Sig	# Insig	% Sig	# Sig	# Insig	% Sig	# Sig	# Insig	% Sig
0	1696	1990124	0.09%	2025	8279185	0.02%	899	1669804	0.05%
1	3817	54002	6.60%	3533	61752	5.41%	2620	63225	3.98%
2	4268	11364	27.3%	3192	13922	18.7%	3141	12525	20.0%
3	3204	6462	33.2%	2225	6231	26.3%	2977	7011	29.8%
4	2536	3307	43.4%	1459	2968	33.0%	2591	3932	39.7%
5	1568	1438	52.2%	844	1295	39.5%	1851	2041	47.6%
6	801	784	50.5%	479	605	44.2%	1098	1122	49.5%
7	460	465	49.7%	244	286	46.0%	682	639	51.6%
≥ 8	219	270	44.8%	225	235	48.9%	706	707	50.0%

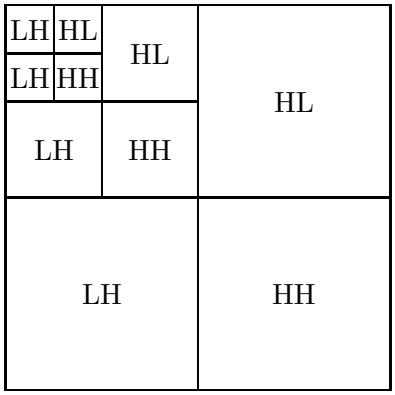
Table 1: The number of significant coefficients and insignificant coefficients which have 0 through ≥ 8 significant neighbors. The counts were accumulated across eight bit-planes, and only spatially adjacent, parent, and child neighbors were looked at.

3.3 PACW Classification vs. EBCOT Classification

We now compare the newly-optimized method of classification in PACW to the method used in Taubman's EBCOT coder [4]. EBCOT (Embedded Block Coding with Optimized Truncation) is an image compression algorithm which was adopted for use in the JPEG-2000 standard and effectively serves as its primary foundation. Essentially everything which follows the wavelet transform and quantization phases of JPEG-2000 is the EBCOT algorithm. Included among the elements JPEG-2000 takes from EBCOT is the metric used for classifying coefficients. Thus, by implementing the EBCOT metric into PACW and comparing its performance with our own metric, we are comparing the way coefficients are classified in JPEG-2000 to the way they are in PACW.

Just like PACW, EBCOT classifies coefficients based upon two criteria. But whereas PACW classifies first by the subband level of the coefficient, EBCOT classifies by the *orientation type* of the subband. There are three different orientation types: LH, HH, or HL, as shown in Figure 12a. A subband is of type LH if it is the result of the low pass from a horizontal transform and the high pass from a vertical transform. Likewise, a subband is of type HL if it is the horizontal low pass and the vertical high pass component, and is of type HH if it is the horizontal high pass and vertical high pass component. Refer to Section 2.1 for a discussion of low and high pass filters in a wavelet transform. The special exception here is subband 0, the LL subband, which EBCOT classifies as type LH for its purposes.

The second criterion EBCOT uses for classification is the number of significant neighbors surrounding the coefficient. It only looks at the eight spatially adjacent neighbors, however it divides them into four diagonal (d), two vertical (v), and two horizontal (h) neighbors. These three types of spatially adjacent neighbors are weighted differently, depending upon the orientation type of the current subband, as described in Figure 12b. The table shows



(a) Orientation type of wavelet subbands.

Assigned label	LH subband			HL subband			HH subband	
	h	v	d	h	v	d	$h+v$	d
0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	1	0
2	0	0	≥ 2	0	0	≥ 2	≥ 2	0
3	0	1	≥ 0	1	0	≥ 0	0	1
4	0	2	≥ 0	2	0	≥ 0	1	1
5	1	0	0	0	1	0	≥ 2	1
6	1	0	≥ 1	0	1	≥ 1	0	2
7	1	≥ 1	≥ 0	≥ 1	1	≥ 0	≥ 1	2
8	2	≥ 0	≥ 0	≥ 0	2	≥ 0	≥ 0	≥ 3

(b) Context label assigned to coefficients of LH, HL, and HH subbands based upon h, v, and d significant neighbor counts.

Figure 12: EBCOT's method of coefficient classification based upon the subband orientation type and the number of horizontal, vertical, and diagonal significant neighbors.

how the h, v, and d neighbors are counted up and used to assign a context label. Since an LH subband contains the low pass data from the horizontal transform, it values h neighbors more than v neighbors, since there is greater spatial correlation horizontally than vertically. Likewise, in HL subbands the v neighbors are counted as more important than the h neighbors since there is greater spatial correlation vertically than horizontally. Finally, HH subbands value d neighbors more than either h or v neighbors, since such subbands contain only high pass data and have almost no spatial correlation either horizontally or vertically.

With three different orientation types and nine possible context labels for each orientation type, there are a total of 27 contexts used in EBCOT. By comparison, the number of contexts in PACW varies depending upon the size of the image being compressed, due to the fact that PACW classifies by subband level. For each subband level, there are six contexts (see Section 3.2). For an image such as "Barbara," which is 512×512 pixels, there are seven subband levels and thus 42 contexts. "Man" is four times the size of "Barbara," but has only one additional subband level and hence six more contexts, making a total of 48. We now compare the compression performance of EBCOT's classification scheme versus PACW's classification scheme, both implemented within PACW. Rate-distortion curves of this study are presented in Figure 13. The results show that, despite EBCOT's attempts to exploit the horizontal and vertical spatial correlation patterns within each subband orientation type, its performance is actually slightly worse than the PACW classification method. Moreover, the fact that PACW is able to outperform EBCOT, albeit negligibly, shows that PACW's tendency to create many more contexts than EBCOT's fixed number of 27 does not actually result in context dilution.

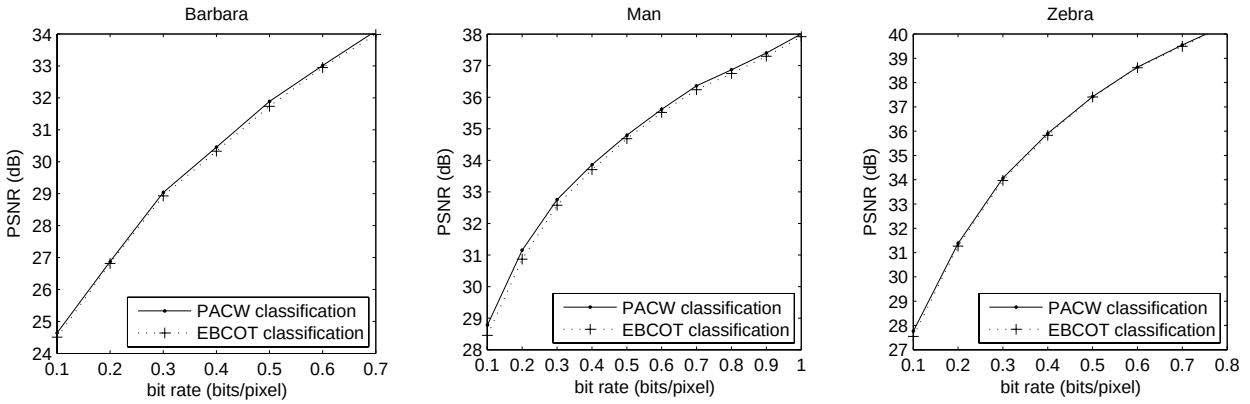


Figure 13: Rate-distortion curves of three images which compare the performance of EBCOT's method of classification to PACW's method of classification, both implemented within PACW.

	Barbara			Man			Zebra		
	0's	1's	% 1's	0's	1's	% 1's	0's	1's	% 1's
Refin Bit 1	65421	36277	64.3%	122906	56656	68.4%	66185	35253	65.2%
Refin Bit 2	36326	26300	58.0%	47147	32167	59.4%	30919	21077	59.5%
Refin Bit 3	19637	16807	53.9%	19513	15722	55.4%	15267	12885	54.2%
Refin Bit 4	9711	8858	52.3%	7404	6822	52.0%	8660	7905	52.3%
Refin Bit 5	3824	3613	51.4%	2642	2509	51.3%	4738	4550	51.0%
Refin Bit 6	1000	916	52.2%	879	909	49.2%	2355	2331	50.3%
Refin Bit 7+	488	526	48.1%	465	499	48.2%	1821	1834	49.8%

Table 2: The frequency of 0's and 1's among refinement bits which were accumulated across 12 bit-planes. Each row corresponds to the relative bit position of the refinement bit for a given coefficient (i.e. Refin Bit 1 = the position of the first refinement bit in each individual coefficient).

3.4 Refinement Bit Contexts

Up to this point, our discussion of contexts in PACW has pertained solely to the significance contexts—the contexts used to classify coefficients during the significance pass. However, some type of context is also needed for the arithmetic coding of refinement bits during the refinement pass. Prediction of refinement bits is much trickier because once a coefficient becomes significant, the values of its subsequent refinement bits tend to be fairly random. For this reason, PACW Version 1 used only a single context for all refinement bits.

However, looking at Table 2, we see that there is actually some structure present in the refinement data. The first couple of refinement bits in each coefficient have a higher frequency of 1's than 0's and it is not until the later refinement bits that the data starts to become truly random, with an equal probability of 1's and 0's. Based upon this information, we use

two refinement contexts for Version 2, rather than one. The first context is used for only the first refinement bit of each coefficient, and the second context encapsulates all following refinement bits. This way, the slight bias toward bit 1 in the first refinement bit will be more accurately captured and predicted by its own context, and we avoid it becoming muddied by the more even distribution of 0's and 1's which occurs in the subsequent refinement bits.

3.5 Inter-plane Prediction

Each context is responsible for keeping track of the behavior of the coefficients associated with it. This is done by maintaining frequency counts for the number of 0's and 1's encountered so far, and using these counts to form probabilities for future 1's and 0's, so that it is able to predict future bits and thus enhance compression. However, because PACW encodes the data on a bit-plane by bit-plane basis, there is the question of whether or not to use *inter-plane prediction*, in which the frequency counts of previous bit-planes are carried over for the prediction of the current bit-plane. On one hand, different bit-planes contain different structure and therefore the behavior of previous bit-planes may not apply to the current bit-plane. On the other hand, it takes a number of bits for a context to learn the behavior of its data, and starting the learning process over again from scratch for each bit-plane may hurt the compression performance. One option is to divide the frequency counts of each context by a certain factor at the start of each new bit-plane. In so doing, a fraction of the counts will be remembered and applied to the current bit-plane, but the sheer size of these accumulated counts will be diminished so that it is easier for the current bit-plane to influence them in its own favor. At the start of a new bit-plane p_{i+1}, we compute $F(p_{i+1})$, the context's frequency count at that bit-plane, using a multiplying factor α, according to the following equation:

$$F(p_{i+1}) = \max\left(1, \lceil \alpha \times F(p_i) \rceil\right) \quad \text{where} \quad 0 \leq \alpha \leq 1 \tag{3}$$

Note that the use of the max function is needed to avoid the situation in which $F(p_{i+1}) = 0$; the frequency count must always be at least 1 in order for the entropy coder to handle it properly. Thus, according to the above equation, $\alpha = 0$ corresponds to completely resetting the counts back to their initial value of 1, $\alpha = 1$ corresponds to completely remembering the counts, and values in between 0 and 1 correspond to remembering a percentage of the counts. Note that this computation occurs only on the significance contexts; the sign and refinement contexts' frequency counts are always carried across bit-planes.

Up to this point, all tests run on PACW have had $\alpha = 0$. However, in order to determine which value of α is the best choice, we now study the effect of α on the compression performance. The results are shown in Figure 14, with PSNR plotted against α at a single bit rate (0.50 bits/pixel). In all three images, the trend is a gradual decline in compression performance as α increases to 1. However, all three images also show a big jump when moving from 0 to the next closest data point. Thus we can conclude that neither completely resetting nor completely remembering frequency counts is the best choice. Rather, we should remember

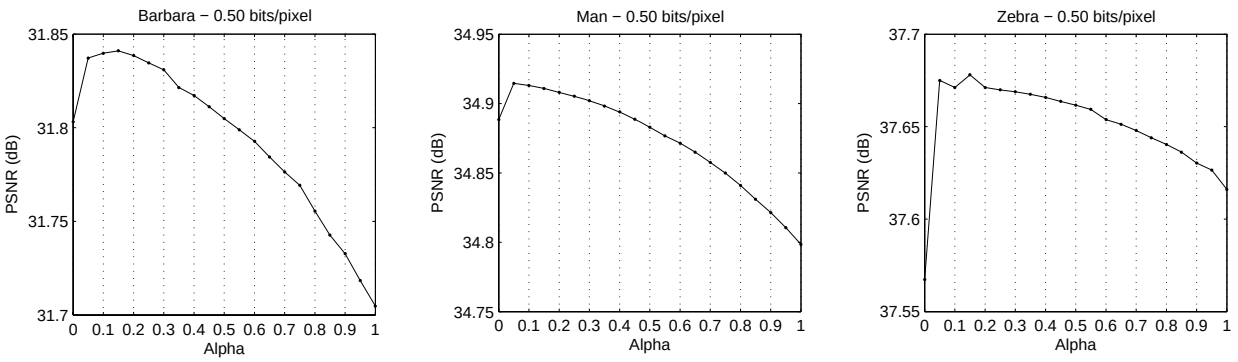


Figure 14: PSNR plotted against the value of α at a single bit rate.

a very small percentage. The true peak in compression performance varies slightly among these three plots, and as a compromise we settle upon $\alpha = 0.15$ for Version 2 of PACW.

3.6 The Priority Queue

PACW Version 1 includes the use of a priority queue during the significance pass to dictate the order in which coefficients are coded, in an attempt to get the coefficients which become significant in the current bit-plane sent to the encoder before those that remain insignificant. A priority key is assigned to each context rather than to each individual coefficient, however. This priority key is equal to the proportion of significant coefficients already encountered within that context. If we let $P(C_k)$ be the priority of some context C_k, and $F(S_k)$ and $F(I_k)$ be the frequency counts for the significant and insignificant coefficients, respectively, within C_k, then:

$$P(C_k) = \frac{F(S_k)}{F(S_k) + F(I_k)} \tag{4}$$

By using this value as the priority key of each context, those contexts which have a high probability of containing significant coefficients are given precedence over others. Once a context is selected from the priority queue, an arbitrary coefficient belonging to that context is encoded.

However, the use of this priority queue adds overhead to the run-time of both encoding and decoding, and it was originally unknown whether or not an improvement in compression was even achieved. Therefore, we now seek to validate our use of a priority queue by comparing its performance to a version of PACW with the priority queue removed. Without a priority queue, coefficients are simply coded in subband order, from lowest to highest, and in straight raster order within each subband. The performance comparison is shown in Figure 15. The advantages of the priority queue are most evident for "Barbara" where it is able to consistently outperform the non-priority queue version. There are a few bit rates in the coding of "Man" and "Zebra" where the non-priority queue version actually does better, but the difference is extremely small. Based on the overall trend seen in these plots, we deem the priority queue a beneficial asset to PACW, and keep it intact for Version 2.

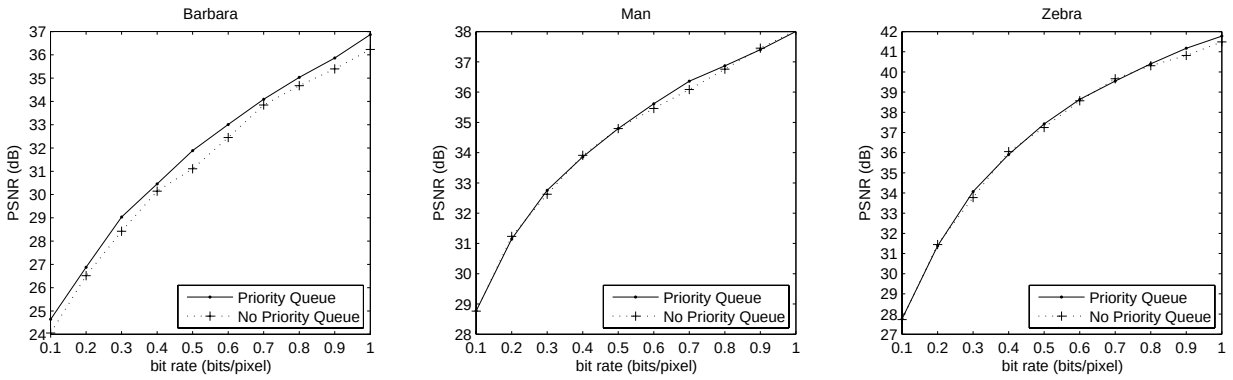


Figure 15: Rate-distortion curves for three images comparing the compression performance of PACW with and without the use of a priority queue to dictate the order coding coefficients.

	GTW	PACW Version 1	PACW Version 2
Entropy coder	group testing	arithmetic coding	arithmetic coding
Classification criteria	1. subband level 2. sig neighbor metric 3. pattern type	1. subband level 2. sig neighbor metric	1. subband level 2. sig neighbor metric
Sig neighbors	8 spatially adjacent, 1 parent, 4 children, 2 spatially identical	8 spatially adjacent, 1 parent, 4 children, 2 spatially identical	8 spatially adjacent, 1 parent, 4 children
# Sig contexts	4 (0, 1, 2, ≥ 3)	4 (0, 1, 2, ≥ 3)	6 (0, 1, 2, 3, 4, ≥ 5)
# Refin contexts	1	1	2
α	0	0	0.15
Priority queue	no	yes	yes

Table 3: An overview of the differences between GTW, PACW Version 1, and PACW Version 2.

3.7 Version 1 vs. Version 2

As an overview of all of the changes that have been made to our original coder, we now present a performance comparison between Version 1 and Version 2 of PACW. In addition, we include GTW in our comparisons. GTW can be thought of as Version 0 of PACW since it was the precursor to PACW and the only major differences between the two coders are the replacement of group testing with arithmetic coding and the addition of a priority queue. The details of these three coders are outlined in Table 3. A rate-distortion comparison is shown in Figure 16 and the corresponding PSNR values are given in Table 4 as a clearer representation of the results. The good news from these results is that, with all of the changes combined together in the optimal version, performance is improved across the board; PACW Version 2 comes out over both Version 1 and GTW for every bit rate. The improvement from Version 1 to Version 2 is certainly nothing drastic, but we should not expect it to be,

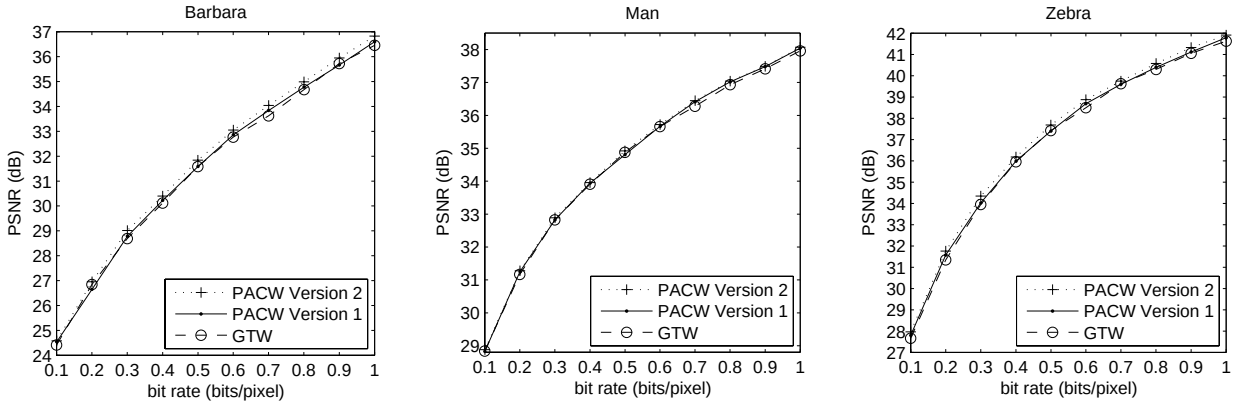


Figure 16: Rate-distortion curves on three images comparing the compression performance of GTW, PACW Version 1, and PACW Version 2. See Table 4 for the PSNR values.

bit	Barbara			Man			Zebra		
rate	GTW	Ver 1	Ver 2	GTW	Ver 1	Ver 2	GTW	Ver 1	Ver 2
0.1	24.41	24.54	**24.58**	28.84	28.83	**28.88**	27.66	27.80	**27.96**
0.2	26.85	26.65	**26.96**	31.17	31.25	**31.28**	31.34	31.58	**31.76**
0.3	28.69	28.78	**29.02**	32.82	32.83	**32.89**	33.95	34.03	**34.34**
0.4	30.12	30.22	**30.40**	33.91	33.94	**33.96**	35.96	35.98	**36.18**
0.5	31.58	31.58	**31.84**	34.88	34.81	**34.92**	37.42	37.40	**37.68**
0.6	32.76	32.86	**33.05**	35.66	35.67	**35.72**	38.50	38.69	**38.87**
0.7	33.62	33.84	**34.04**	36.27	36.42	**36.45**	39.62	39.60	**39.75**
0.8	34.67	34.78	**35.00**	36.93	37.02	**37.04**	40.30	40.39	**40.58**
0.9	35.73	35.67	**35.95**	37.42	37.48	**37.49**	41.05	41.11	**41.33**
1.0	36.46	36.60	**36.83**	37.96	38.05	**38.07**	41.62	41.79	**41.91**

Table 4: PSNR (dB) values corresponding to the rate-distortion curves of Figure 16. The top performer at each bit rate is shown in bold.

since we are merely fine-tuning the context modeling details of our coder at this stage. What is most important is that we have settled upon a set of values for all of the variables in our coder which yields a consistent dominance over the original set of values.

4 Related Work

PACW belongs to a family of image coders known as *wavelet coders*, named in reference to their use of a wavelet transform at the beginning of their algorithm. Other notable wavelet coders include EZW, SPIHT, ECECOW, JPEG-2000, and GTW. All of these coders are similar in that they all transmit wavelet coefficients bit-plane by bit-plane, with the most significant bit-plane first. The primary difference between them is the method they use to encode the bit-planes.

The Embedded Zerotree Wavelet (EZW) coder [5] was introduced by Shapiro in 1993, as a new technique for image compression which offered a very effective and computationally simple algorithm. Using bit-plane coding of wavelet coefficients, it generated an embedded bitstream wherein bits were coded using zerotrees during a significance pass and remaining bits were coded in a refinement pass. Both passes used an adaptive arithmetic coder for the entropy coding.

This algorithm was improved upon with Said and Pearlman's Set Partitioning in Hierarchical Trees (SPIHT) coder [6] in 1996. SPIHT built upon the original concepts of EZW, but used a different wavelet, a different method of organizing the coefficients into zerotrees, and a different type of adaptive arithmetic coder which used several contexts. The result was an improvement in rate-distortion performance and an even faster runtime than EZW.

In 1997, Wu's Embedded Conditional Entropy Coding of Wavelet Coefficients (ECECOW) algorithm [7] was introduced. Rather than using zerotrees to code each bit-plane, as done in EZW and SPIHT, ECECOW focused instead on a more sophisticated and intelligent system of adaptive context modeling to push the limit of rate-distortion performance. The contexts were based upon the significance level of surrounding neighbors of a given coefficient, and a separate adaptive arithmetic coder was used to entropy encode each context. Additionally, to avoid context dilution, ECECOW used training data to identify contexts with similar properties and merge those contexts together. The rate-distortion results of ECECOW were the best to date, with improvements of 0.4 to 1.3 dB over SPIHT. However, the downside to ECECOW was the need for training data, as well as an increase in computational complexity resulting in slower runtime performance.

In 2000, Taubman introduced the EBCOT coder [4], which featured several new concepts such as resolution scalability, embeddedness, and spatial accessibility through the use of an identifiable bitstream. Constructing such a bitstream involved several sub-divisions of the image. First, before even doing the wavelet transform, the image was split into rectangles called "tiles," each of which underwent the wavelet transform separately. Following the

transform, each subband was then divided into smaller rectangles called "packet partitions," and the three spatially related rectangles of the same subband level together comprised a single "packet partition location" which was further divided into smaller rectangles called "code-blocks." It was these code-blocks that served as the fundamental unit of data sent as input to the entropy coder.

In addition to the novelty of an identifiable bitstream, EBCOT brought a slight improvement in compression performance over SPIHT, ranging anywhere from 0.0 dB to 1.1 dB, with an average improvement of about 0.4 dB. Eventually EBCOT was adopted as the foundation for the JPEG-2000 standard. JPEG-2000 is essentially a superset of EBCOT, with the main difference being improvements in runtime speed at the expense of a slight loss in PSNR performance. In addition, JPEG-2000 offers many more options and settings than EBCOT. For example, the wavelet transform can use one of many possible filters, even user-defined filters, and can operate in either floating point or integer space based on user specification. A variety of quantization methods are offered: Trellis coded quantization (TCQ) using either a Lagrangian rate allocation procedure or a simpler fixed quantization table, or even scalar quantization. The structure of the embedded bitstream allows the extraction of different resolutions, pixel fidelities, regions of interest, components, and more. In short, JPEG-2000 took the EBCOT algorithm and added a slew of extra user-customizable features, increasing the flexibility (and complexity) of the coder.

4.1 Rate-Distortion Comparison

We now present rate-distortion curves comparing the overall compression performance of PACW Version 2 against other wavelet coders. In addition to the images "Barbara," "Man," and "Zebra" (Figure 7), we include three more images in our comparisons to see how well PACW can compete in compressing images outside of its own test set. These three additional images are "Boat," "Couple," and "Lena," which all come from the USC Image Database [8].

We compare our new PACW with results from JPEG-2000, SPIHT, and ECECOW. The particular version of JPEG-2000 used to generate these results comes from the implementation found in the Java Advanced Imaging API v1.1.2_01 [9]. All default options were in effect for the execution of the encoder. The SPIHT program used was the C++ version 8.01 of Said and Pearlman's CODETREE/DECDTREE executables [10]. For ECECOW, PSNR results were taken from [7] because an executable of the program is not publicly available. As a result, we are only able to present the six PSNR values given for ECECOW in [7], corresponding to three bit rates from the "Barbara" image and three bit rates from the "Lena" image.

The PSNR results are represented as both rate-distortion curves in Figure 17 and as numbers in Table 5 for additional clarity. First off, we see that at every bit rate available for ECECOW, it yields the highest PSNR values among all coders. From this it is probably safe to infer that ECECOW is the top performer among the four coders being compared here.

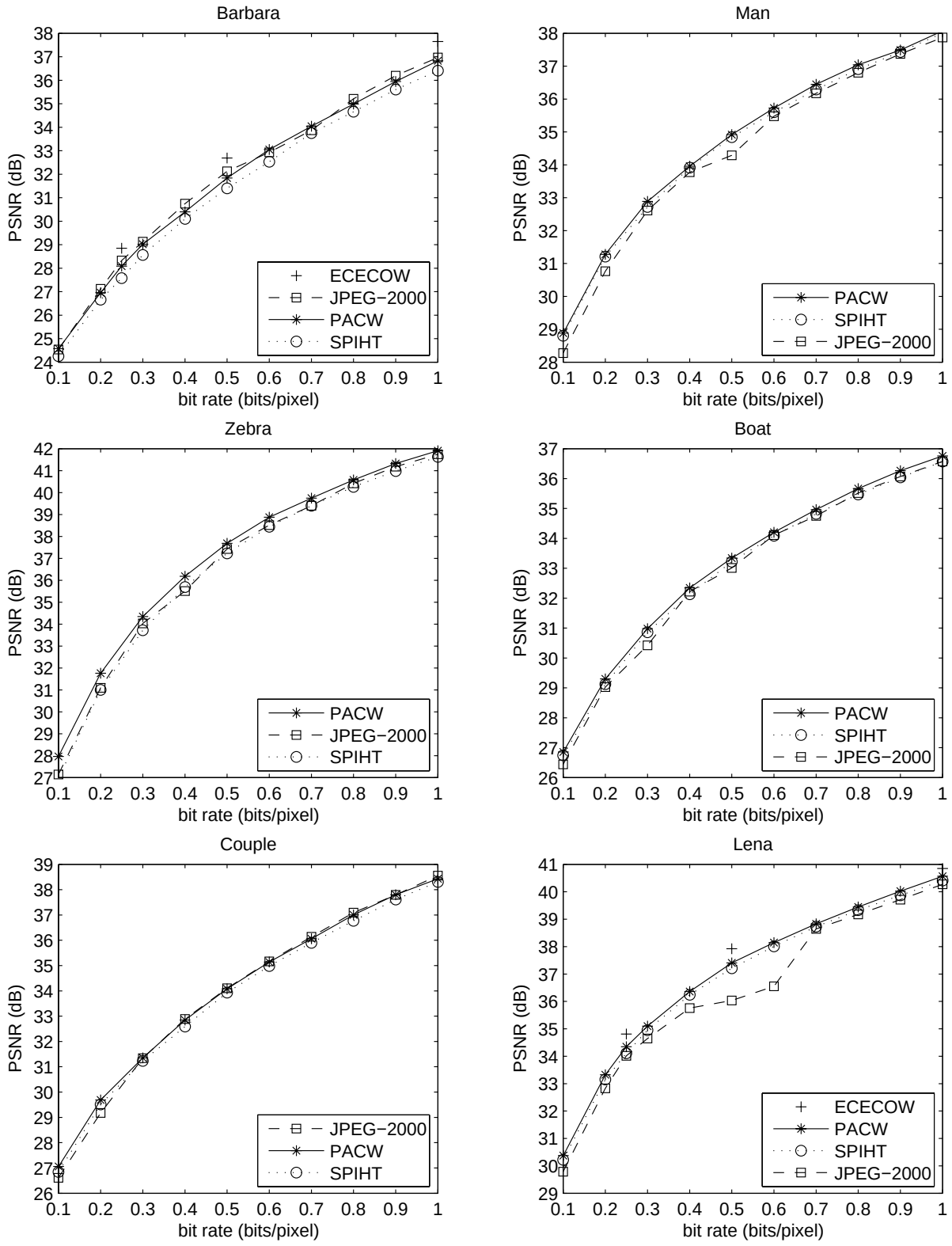


Figure 17: Rate-distortion curves of six images comparing Version 2 of PACW with SPIHT, JPEG-2000, and (where known) ECECOW.

(a) Barbara

bit rate	PACW	J2K	SPIHT	ECECOW
0.1	**24.58**	24.54	24.26	
0.2	26.96	**27.12**	26.66	
0.25	28.08	28.33	27.58	**28.85**
0.3	29.02	**29.12**	28.56	
0.4	30.40	**30.74**	30.10	
0.5	31.84	32.12	31.40	**32.69**
0.6	**33.05**	32.93	32.53	
0.7	**34.04**	33.87	33.75	
0.8	35.00	**35.21**	34.66	
0.9	35.95	**36.20**	35.60	
1.0	36.83	36.97	36.41	**37.65**

(b) Man

bit rate	PACW	J2K	SPIHT
0.1	**28.88**	28.27	28.80
0.2	**31.28**	30.77	31.21
0.3	**32.89**	32.61	32.72
0.4	**33.96**	33.77	33.91
0.5	**34.92**	34.29	34.84
0.6	**35.72**	35.48	35.60
0.7	**36.45**	36.17	36.29
0.8	**37.04**	36.80	36.91
0.9	**37.49**	37.37	37.43
1.0	**38.07**	37.87	38.01

(c) Zebra

bit rate	PACW	J2K	SPIHT
0.1	**27.96**	27.14	26.99
0.2	**31.76**	31.09	31.00
0.3	**34.34**	34.03	33.72
0.4	**36.18**	35.51	35.69
0.5	**37.68**	37.43	37.23
0.6	**38.87**	38.52	38.44
0.7	**39.75**	39.40	39.40
0.8	**40.58**	40.42	40.26
0.9	**41.33**	41.18	40.98
1.0	**41.91**	41.75	41.63

(d) Boat

bit rate	PACW	J2K	SPIHT
0.1	**26.85**	26.44	26.74
0.2	**29.30**	29.03	29.12
0.3	**30.98**	30.42	30.86
0.4	**32.34**	32.22	32.13
0.5	**33.34**	33.01	33.22
0.6	**34.20**	34.11	34.09
0.7	**34.96**	34.75	34.82
0.8	**35.67**	35.52	35.47
0.9	**36.27**	36.07	36.04
1.0	**36.76**	36.57	36.57

(e) Couple

bit rate	PACW	J2K	SPIHT
0.1	**27.06**	26.61	26.84
0.2	**29.69**	29.17	29.51
0.3	**31.35**	31.34	31.23
0.4	32.85	**32.89**	32.59
0.5	34.08	**34.10**	33.93
0.6	35.14	**35.17**	34.98
0.7	36.06	**36.14**	35.90
0.8	37.01	**37.09**	36.77
0.9	**37.80**	37.79	37.61
1.0	38.43	**38.56**	38.31

(f) Lena

bit rate	PACW	J2K	SPIHT	ECECOW
0.1	**30.38**	29.79	30.22	
0.2	**33.32**	32.83	33.15	
0.25	34.35	34.02	34.11	**34.81**
0.3	**35.10**	34.65	34.95	
0.4	**36.36**	35.75	36.24	
0.5	37.40	36.03	37.21	**37.92**
0.6	**38.14**	36.55	38.01	
0.7	**38.83**	38.65	38.73	
0.8	**39.45**	39.18	39.33	
0.9	**40.02**	39.71	39.87	
1.0	40.56	40.28	40.41	**40.85**

Table 5: PSNR (dB) results on six images compressed by PACW Version 2, JPEG-2000 (J2K), SPIHT, and (where known) ECECOW.

However, for our purposes, what is most important to see is that PACW is able to outperform JPEG-2000 at almost every single bit rate, except in the cases of the "Barbara" and "Couple" images which seem to be the anomalies in this set of images. Thus, although the PSNR improvements we saw from Version 1 to Version 2 in Section 3.7 were unsubstantial, we see in Table 5 how extremely close the relative performance of each wavelet coder is, and the importance of the extra PSNR boost given to Version 2 through our various studies is made more significant.

5 Conclusion

In this paper we have introduced PACW, a new wavelet-based image compression algorithm based on a priority-based arithmetic coding of bit-planes. The original goal of the coder was to provide a simpler and more elegant alternative to JPEG-2000, such that it could replace JPEG-2000 in an undergraduate course as a means of introduction to advanced wavelet-based image coding concepts. We have shown that such a coder can exist; without implementing any of the frills of JPEG-2000, PACW is still able to perform at the level of JPEG-2000. In fact, after investigating and optimizing the context modeling details of PACW, we are able to push its PSNR performance to a point at which it can fairly consistently outperform JPEG-2000. Furthermore, avoiding all the complexities of JPEG-2000 involved in the production of an identifiable, spatially accessible bitstream, PACW is a much cleaner and more teachable image coder.

6 Acknowledgments

I would like to thank Richard Ladner and Eve Riskin for help in the conception of the original PACW algorithm. Thanks to Amanda Askew for co-authoring the PACW coder from scratch in Java.

References

[1] E. S. Hong and R. E. Ladner, "Group testing for image compression," *IEEE Transactions on Image Processing*, vol. 35, pp. 901–911, August 2002.

[2] D. Du and F. Hwang, *Combinatorial Group Testing*. Singapore: World Scientific, 1993.

[3] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transforms," *IEEE Transactions on Image Processing*, vol. 1, pp. 205–220, April 1992.

[4] D. Taubman, "High performance scalable image compression with ebcot," *IEEE Transactions on Image Processing*, vol. 9, pp. 1158–1170, July 2000.

[5] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3445–3462, December 1993.

[6] A. Said and W. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 243–250, June 1996.

[7] X. Wu, "High-order context modeling and embedded conditional entropy coding of wavelet coefficients for image compression," in *Proceedings of the 31st Asilomar Conference on Signals, Systems and Computers*, vol. 23, pp. 1378–1382, 1998.

[8] USC Signal and Image Processing Institute, "The USC-SIPI Image Database." http://sipi.usc.edu/services/database/.

[9] Sun Developer Network, "Java Advanced Imaging (JAI) API." http://java.sun.com/products/java-media/jai/.

[10] A. Said and W. A. Pearlman, "Image Compression Programs." http://www.cipr.rpi.edu/research/SPIHT/spiht3.html.