

Lossless Sound Compression using the Discrete Wavelet Transform

Ankit Patel
abpatel@fas.harvard.edu

Mark Tonkelowitz
mtonkel@fas.harvard.edu

Mike Vernal
vernal@fas.harvard.edu

January 14, 2002

Abstract

We explore the performance of the Discrete Wavelet Transform (*DWT*) as applied to the lossless compression of sampled waveform data. Specifically, we have developed a parameterizable algorithm that uses the *DWT* to losslessly compress RIFF WAVE audio files. Our algorithm uses the *DWT* as an approximation to our original waveform, and we use a variety of entropy coders to store the difference between the original waveform and our approximation.

We explore the performance of our algorithm under a number of different parameters, including different types of music, entropy encoders, and wavelet bases. Despite our optimizations, we find that our algorithm achieves compression that is inferior to existing lossless codecs. This leads us to conclude that wavelets are not the most appropriate models for complex sound data.

1 Introduction

With the advent of Napster, Gnutella, and other popular file sharing services, compressed

sound data has become one of the most popular forms of information transferred over the Internet. At its peak, various universities and colleges estimated that Napster traffic alone was consuming 60-80% of their total bandwidth.¹ The sheer amount of sound data shared daily underscores the importance of effective audio compression algorithms.²

The vast majority of sound data on the Internet is compressed using some form of lossy coding, including the extremely popular MPEG Layer III (MP3), Windows Media Archive (WMA), and Real Media (RM) formats. These algorithms can generally achieve compression ratios of over 10 : 1 by using a combination of signal processing techniques, psychoacoustics, and entropy coding. For the average listener on standard computer equipment, these lossy compression algorithms are often indistinguishable from the original sound data. For more demanding applications or distinguishing users, lossy compression is often unacceptable. To that

¹Numerous universities, including Indiana University, the University of Vermont, and Drew University, have made statements to this effect in the news media.

²One author of this paper has had nearly five terabytes worth of (legal) sound files downloaded from his personal computer in 18 months.

end, we explore the use of the Discrete Wavelet Transform (*DWT*) in the lossless compression of sound data.

While not as popular as their lossy brethren, there are a number of publically-available tools for lossless sound compression, including Shorten, Monkey’s Audio, RKAU, and others. While details of the compression algorithm were not available for every file format we examined, we found the same general approach repeatedly used. First, some form of mathematical approximation of the waveform was performed, followed by the entropy encoding of the difference between the original waveform and the approximation. In our approach, we use the *DWT* to approximate the sound waveform, and we use a variety of entropy encoding techniques to efficiently store the residual data.

We found that our technique yielded acceptable compression performance, both in terms of time and space, but that our techniques were inferior to simpler techniques like Linear Predictive Coding (LPC). In general, we found that we could achieve compression ratios of 60-80% for more tonally-complex modern music (e.g. rock, pop, etc.), while we could achieve compression ratios of 40-60% for classical and choral music.

Looking ahead, we describe some commonly-used audio compression algorithms and other related work in section 2. In section 3, we summarize some of the mathematical background necessary for this paper, including a look at wavelets and the entropy coding schemes that we used. We describe the design and implementation of our compression algorithm in section 4, and we describe some of our performance results in section 5. We discuss our results and describe possible future work in section 6, and we conclude in section 7.

2 Related Work

As previously mentioned, most popular attention has been focused on lossy compression schemes like MP3, WMA, and Ogg Vorbis. In general, these schemes perform some variant of either the Fast Fourier Transform (FFT) or Discrete Cosine Transformation (DCT) to get a frequency-based representation of the sound waveform. Lossy algorithms generally take advantage of a branch of psychophysiology known as psychoacoustics that describes the ways in which humans perceive sound. By removing tones and frequencies that humans should not be able to hear, lossy algorithms can greatly simplify the nature of the data which they need to encode. By removing excess minor frequencies, the frequency representation of the sound data can now be efficiently compressed using any number of entropy coding techniques.

Most lossless audio compression schemes are structured in the same way. First, the compression scheme performs some form of predictive coding – it uses some contextual information about the waveform to generate an approximation that is compressible. The difference between the original and approximation waveforms – the residual waveform – is then losslessly encoded using some form of entropy coding. Clearly, the better the approximation, the smaller the residual encoding.

Most of the lossless compression schemes we found were commercial in nature, lacking both source code and an academic explanation of their operation. The one notable exception also seemed to be the most popular scheme, Shorten [7]. Shorten was originally developed by Tony Robinson at the University of Cambridge to deal with the efficient compression of speech samples

and other digitized waveforms. The algorithm predicts the waveform using a linear combination of past samples:

$$\hat{s}(t) = \sum_{i=1}^p a_i s(t-i)$$

Because the waveform is likely not stationary, the LPC is performed on a small window of the waveform in the hopes of exploiting some locally-stationary behavior. Shorten attempts to fit a p^{th} -order polynomial to the last p data points – if it fails, it chooses from one of four hardwired polynomial predictors. After Shorten chooses its predictive model, it encodes the residuals using a special type of entropy coding called Rice coding (see Section 3.2.1).

3 Mathematical Background

As previously described, our algorithm involves two main components – the predictive code, for which we use the *DWT*, and an entropy encoder, which we use to compress both the quantized coefficients of the *DWT* and the residual values. We offer a brief background on both wavelets and entropy encoders for the unfamiliar reader.

3.1 Wavelets

We can express a function $f(t)$ as a (possibly infinite) linear combination of a set of basis functions combined with a set of scalar coefficients. For example, $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$ is the expression of the sine function in terms of the set of basis functions $\{f_n(x) = x^n : n \in \mathbb{Z}^+\}$ (this is more formally known as the Taylor Series of

$\sin(x)$ centered at $x = 0$.) While this is an infinite series in theory, requiring an infinite number of terms for exact evaluation of the function, we only need to consider the dominating terms of the series in practice.

As opposed to the Taylor basis, the Fourier basis is composed of sine and cosine functions: $\{\sin(nx), \cos(nx) : n \in \mathbb{Z}\}$. This implies that an arbitrary function can be represented in terms of a series of sines and cosines. The Fourier basis functions have a property known as *global support*, which implies that each basis function has “influence” over the entire parameter domain. Formally, the basis functions are non-zero over the reals. This means that if the curve were to be modified slightly, the effect of this small change would be propagated over the entire domain (i.e., many, if not all, the scalar coefficients would change). Global support is undesirable for our application – in our approximation of the waveform, if some part of the curve changes slightly, we do not want that change to propagate to our entire set of scalar coefficients. Instead, we desire *compact support*, where the basis functions have only local influence. This is accomplished by functions that are non-zero over only a finite interval of the domain.

The Fourier basis is also of interest because it allows to perform fine-grained frequency decomposition. In a Fourier basis, sines and cosines of high frequencies correspond to scalar coefficients that describe the “contribution” of that frequency to the entire function or signal. Hence, we can extract both high and low resolution information from a signal by simply examining the high and low frequency scalar coefficients of the corresponding Fourier basis functions. Frequency decomposition can be especially useful in audio signal processing, as recorded music tends to be fundamentally tonal, so the frequency rep-

representation of an audio signal may have lower entropy than an amplitude representation. Unfortunately, as we previously mentioned, the Fourier basis has the unfortunate property of global support, so that a set of coefficients that describes a function $f(t)$ could be completely different from a set of coefficients that describes another function $g(t)$, even if $f(t) \approx g(t)$. We desire a set of basis functions with compact support and the ability to extract information from a signal at different frequencies.

Wavelets are a family of mathematical basis functions that have the properties we desire; they provide frequency decomposition and a compact representation. Unlike the aforementioned bases in which each function is unique, a wavelet basis consists of translations and scalings of the same function.

3.1.1 Haar Basis

The simplest and oldest example of a wavelet basis is the Haar basis. It consists of a single box function and many step functions, each of successively higher resolution (i.e., the width of the step). Consider two adjacent step functions $\phi(l, 0)$ and $\phi(l, 1)$. They can be replaced by the basis consisting of a box function twice as wide $\phi(l - 1, 0)$ and a step function (called the Haar function) $\psi(l - 1, 0)$. This process can be repeated indefinitely to produce a basis with as fine a resolution as needed. In essence, this is the idea behind a hierarchical basis – one that captures information at varying scales. Without delving further into the mathematics of wavelets, we list some of the properties of the Haar basis that are important for our purposes:

- Multiresolution: the ability to extract information at any scale or resolution

- Hierarchical Representation: the ability to economically represent a function (constant areas of little detail can be removed without significant loss)
- Efficient Coefficient Calculation: transformations between the box and Haar bases are computationally efficient.

3.1.2 Daubechies Basis

The Haar Basis, despite its useful properties, is still a discontinuous basis. It is not well-suited for approximating continuous signals. What we need is a wavelet basis that is at least C^0 continuous (though not necessarily C^1). The Daubechies Wavelet is both continuous and has better resolution than Haar for approximating smoothly varying time signals. The tradeoff, of course is that the Daubechies wavelet coefficients are more expensive to calculate than the Haar coefficients. Figure 3.1.2 shows the Daubechies mother wavelet.

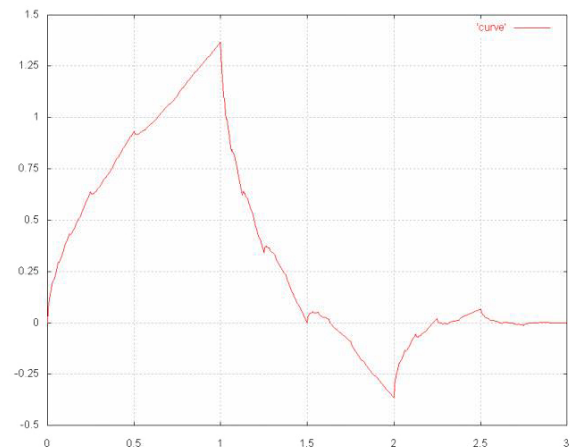


Figure 1: The Daubechies Mother Wavelet

3.1.3 Wavelet Transforms using Lifting Schemes

The next step in the theory of Wavelets is to generalize the notion of a wavelet beyond the scaling and dilation of a single mother wavelet, or what we refer to as classical wavelets. We use what is called a Lifting Scheme to generalize wavelets to more abstract frameworks. The Lifting Scheme consists of three main steps:

- Split - subsample the original signal data into odd and even sets
- Predict - find wavelet coefficients as the failure to predict the odd set based on the even set
- Update - update the even set using the wavelet coefficients to compute the scaling function coefficients

This algorithm begins with an initial wavelet called a Lazy Wavelet (in general, we could use any wavelet as our initial wavelet (e.g., Haar). We try this in our implementation along with the Lazy wavelet). The Lazy Wavelet function has all the properties of a wavelet but does not have any computing power. From this Lazy Wavelet, we construct a wavelet family for which the split, predict, and update phases of the above algorithm are specifically “tuned” to our signal data – our own custom wavelet. The ability to adapt the wavelet to the data at hand has made the lifting approach very useful, especially in image processing. In our algorithm, we apply this scheme to audio signals, using the software library LIFTPACK [3]. The Fernandez *et al* paper has a much more in-depth discussion of the Lifting Scheme, as well as further details about the LIFTPACK library itself.

3.2 Entropy Coding

Every lossless audio compression scheme we studied involves the computation and encoding of residuals, i.e., the difference between our original waveform and our approximation. We describe some of the more commonly used variable-length coding scheme for integers, with emphasis on those schemes that we used with our algorithm.

3.2.1 Golomb/Rice Codes

Each different scheme for variable-length integer coding is best suited for a particular distribution. Golomb codes treat the problem by dividing integers into two parts, the low-order bits and the high-order bits. The low-order bits are assumed to be uniformly random, implying that compression of them would be useless. As such, we encode them directly. We presume the high-order bits to have lower entropy, allowing them to be compressed with some kind of coding technique. We examine one of the oldest and simplest methods, known as Rice coding. Rice coding transmits the high order bits by treating them as integers and representing them in unary notation using zeroes, followed by a trailing one. For example, if the high-order bits of our input were 101 (5 in decimal), we would encode and transmit them as the binary string 000001.

The optimal boundary between high-order and low-order bits can be determined if we assume a certain distribution on the input data (e.g., Laplacian, Power, or Normal). Thus, we can theoretically derive an estimator for the boundary, as is done in the Shorten paper [7]. We note that it can be proven that Rice Codes are optimal for the Laplacian/Exponential distribution, where $Pr[\text{residual} = x]$ is proportional to

$\frac{1}{2^x}$. We can reason this by observing that because Rice uses approximately $B = \frac{n}{2^k} + k + 1$ bits to represent the integer n (with boundary k), we know from information theory that the optimal distribution would associate a probability of $2^{-B} \propto 2^{-n}$ with the integer n .

3.2.2 Elias-Gamma Codes

Elias-Gamma codes are another set of codes that happen to be more efficient than Rice codes for certain probability distributions. The Elias-Gamma code follows the same scheme of separating an integer in to two parts, directly coding the low-order bits (again, the high/low-order boundary is determined from the data) and encoding the high-order bits in an optimal way. The high-order bits are prefixed with $(l-1)$ zeroes, where l is the number of significant high-order bits (i.e., ignore leading zeroes in the binary representation).

For example, if the high-order bits were 100, we would encode this as $0^2100 = 00100$. Note that this does not include an encoding for 0 – we resolve this by introducing Biased Elias-Gamma coding, wherein we simply code the integer n (where n in binary represents high-order bits of the residual) by the Original Elias-Gamma Encoding of $n + 1$ (a simple shift). For an integer n with boundary k , this requires $B = \log(n) + \log(n) - k - 1 = 2\log(n) - k - 1$ bits. Thus, an optimal distribution for this encoding scheme is $Pr[\text{residual} = n] = 2^{-B} \propto \frac{1}{n^2}$, i.e., a power distribution. Again, we must calculate the high-order/low-order boundary. In this case, it happens to be the point at which $\frac{2}{3}$ of the samples can be represented without having to encode the high-order bits (i.e., the samples all have $\leq k$ significant bits). The derivation of this is in Robin Whittle’s guide to Lossless Audio

Compression [9]. For the interested reader, Fenwick explores integer coding in deeper detail in [2].

3.2.3 Delta Coding

Delta coding is not an entropy coder *per se*, but we used it in our algorithm in conjunction with the arithmetic coder. A delta coder stores an initial value s_0 , and for every subsequent value $s_i, i > 0$, it stores $s_i - s_{i-1}$ instead of s_i . For structured waveform data, this coding scheme offers us the possibility of limiting the distribution of output values, as we do not expect a smooth, frequently-sampled waveform to make large inter-sample jumps.

Combined with a straight arithmetic coder, we found that this scheme was occasionally helpful in the encoding process.

4 Compression Scheme

Our algorithm proceeded as the other lossless encoding schemes we have described. We produced a model of the waveform by using the lifting wavelet described in section 3.1.3, with the option of two different types of initial wavelets, the Haar wavelet and the Lazy wavelet. The residual waveform is stored using one of four different entropy coding schemes, including Rice, Elias-Gamma, Delta and Arithmetic coding.

The utility that we developed was called `sndcomp`, and its encoding scheme was configurable using a number of command-line options. Given a 44.1 KHz stereo RIFF WAVE file, our utility reads in a block of samples, separates the left and right channels, and performs the *DWT* on each channel [1]. The wavelet coefficients are

then quantized and compressed using an entropy coder. The DWT is inverted and the residual waveform is computed by taking the difference of the input samples and the DWT^{-1} . The residuals are then encoded using the specified entropy coder.

To perform the DWT , we used the LIFTPACK Fast Lifting Wavelet Transform (FLWT) library, version 1.0. LIFTPACK supported both Haar and Lazy wavelet transforms. It was parameterized by three variables (N, \tilde{N}, L) . N is the order of the predictive step in the Lifting scheme algorithm, \tilde{N} is the order of the updating step, and L is the number of levels in the wavelet transform. These three parameters allow one to create generalized wavelet transforms that are hopefully customized for a particular data set. After extensive experimentation, we found that we achieved optimal compression when $N = 4, \tilde{N} = 2, L = 1$.

We experimented with a number of quantization of factors before settling on 256, both for compression and implementation reasons. We found that 8-bit coefficients struck an optimal balance between compression of the DWT coefficients and the residual samples. Smaller coefficients significantly hurt our compression of the residual samples, while larger coefficients served to decrease our overall compression ratio. Our experience with wavelets has led us to believe that saving the low-order bits of the quantization coefficients is more expensive than saving the coarser residuals caused by a rougher approximation function.

After the quantization is inverted, we apply the inverse DWT and construct our approximation signal. The approximation signal is then subtracted from the original signal, yielding the residual waveform. The residuals are then entropy-coded by either Arithmetic, Rice, Elias-

Gamma or Delta coding. For arithmetic coding, a range coder called `szip` is used to achieve near-arithmetic coding ratios at a much faster speed than a normal arithmetic coder. The other entropy coders are implemented as described in section 3.2.

5 Performance

We conducted performance tests for each of the aforementioned options on a variety of sound data. Our performance tests were conducted on a 1 GHz Pentium III running FreeBSD 4.4. Our test machine was equipped with 256 MB of RAM and a 10,000 RPM 10 GB SCSI drive. For our testing purposes, we used `shorten` version 3.2 and LIFTPACK version 1.0, and our program was compiled with `gcc-2.95.3`.

A number of seemingly arbitrary decisions needed to be made before we could perform meaningful analysis. In each case, we tried to find empirically optimal values to justify our decisions. Our first such case involved finding the optimal block size. Most of the schemes we investigated suggested a block size of 256. After compressing the same file for a variety of block sizes, we found that we also achieved optimal compression for a block size of 256 (Figure 5).

We suspect a block size of 256 yielded the best compression because the complexity of the sound data over that period is similar to the complexity of our wavelet functions. It is complex enough that we can achieve a fair amount of correlation, but not so complex as to introduce too much variation into the signal.

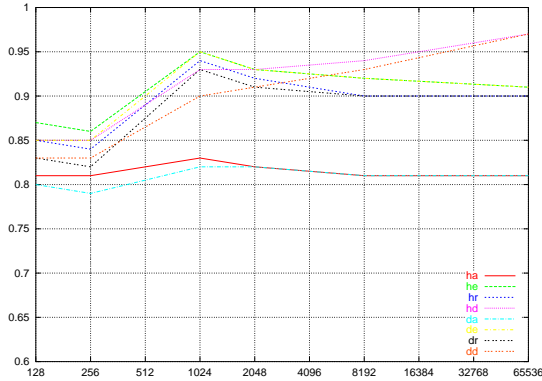


Figure 2: Block Size vs. Compression Ratio

After determining the optimal block size, we began to examine our algorithm’s performance on different genres of music. Our test files consisted of a mixture of classical music, studio rock, live rock, heavy metal, and artificially-generated noise. We compared the sizes of the compressed files produced by all eight of our encoding schemes with those produced by Shorten and `szip`. We discovered that amongst our schemes, FLWT with arithmetic coding was consistently one of the best performers, producing file sizes within 5-10% of Shorten. We were comforted by the fact `szip` was the worst performer – clearly our wavelet analysis was bearing some fruit.

6 Discussion

We were both surprised and a bit disappointed with our results. With the exception of artificial

Input File	Shorten	<code>szip</code>	FLWT/ <code>szip</code>
berlioz.wav	0.39	0.70	0.49
bowa.wav	0.68	0.90	0.75
complex.wav	0.61	0.09	0.23
crash.wav	0.72	0.92	0.79
fuel.wav	0.81	0.98	0.90
howie.wav	0.71	0.93	0.78
newyork.wav	0.56	0.82	0.64
noise.wav	1.03	1.01	1.08
pmb.wav	0.61	0.91	0.67
sine.wav	0.64	0.03	0.09

Table 1: Compression Ratios

Input File	Shorten	<code>szip</code>	FLWT/ <code>szip</code>
berlioz.wav	11.8	109.1	110.6
bowa.wav	17.2	164.4	155.4
complex.wav	0.5	2.1	2.8
crash.wav	14.3	122.5	114.4
fuel.wav	12.7	111.4	102.7
howie.wav	9.9	84.1	74.6
newyork.wav	15.6	127.8	120.7
noise.wav	0.3	2.2	2.1
pmb.wav	2.8	23.5	20.1
sine.wav	0.3	0.8	1.2

Table 2: Compression Times (seconds)

input data (complex.wav and sin.wav), Shorten’s compression ratio always outperformed our own by at least 5%. It also took a tenth of the time to execute, though this seems to be a result of the slow execution time of `szip` (see Table 2). Nevertheless, clearly wavelet transforms are able to decorrelate the sound signal more than just the pure entropy coding.

We were pleased that our results on different music genres were consistent with those found by the First Principles studies [9]. Quiet classical music (berlioz.wav) had the most compression

(49%), while Heavy Metal (fuel.wav) showed almost none (90%). Performance for rock music, both live (bowa.wav, howie.wav, pmb.wav) and studio (crash.wav, newyork.wav) fell in the middle, as expected.

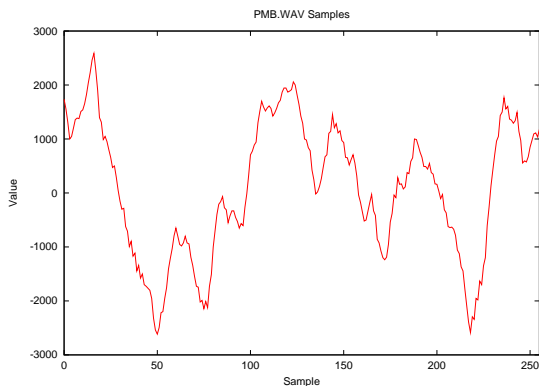


Figure 3: Input Samples

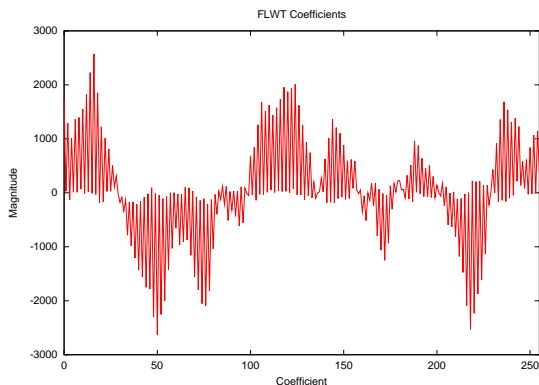


Figure 4: Input before Quantization

One of the biggest surprises to us is that the FLWT performs best when configured with $N = 4$, $\tilde{N} = 2$, and $L = 1$ (see Figure 6). This configuration represents a single-level cubic predictive Lazy-based wavelet. (Note that FLWT wavelets are generalizations of classical wavelets, and so they are not simple translations and dilations of a single mother wavelet.) $N = 4$ (cubic prediction) worked better than $N = 2$ (piecewise linear prediction), presumably because it allows for smooth interpolation in the prediction step. $L = 1$ implies that the multi-resolution power of wavelets was not really needed to compress, perhaps because audio signals can change so quickly that such analysis is not always beneficial. Furthermore, higher levels of analysis using the FLWT produce smaller average coefficients, which would be affected much more by our current quantization scheme.

Another interesting but expected observation is that Lazy-based FLWT's almost always achieved higher compression rates than Haar-based FLWT's. Again, we attribute this to the discontinuity in the Haar basis, which impedes it from approximating a continuous signal well. We had hoped to try our tests with Daubechies wavelets, but the Daubechies-4 transform that we used was not perfectly invertible, yielding larger than expected residuals, even when not quantized.

As for the entropy encoders, we got the best compression rates with `gzip`, followed by Rice, and then Elias-Gamma. We believe this is mainly due to the fact that the residual distributions (as seen in Figure 6) are neither Laplacian/Exponential (optimal for Rice) nor Inverse

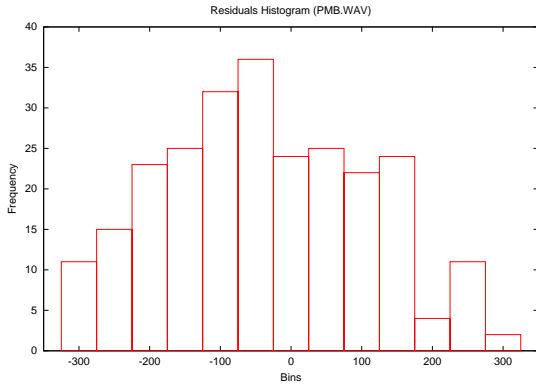


Figure 5: Example Block of Residuals

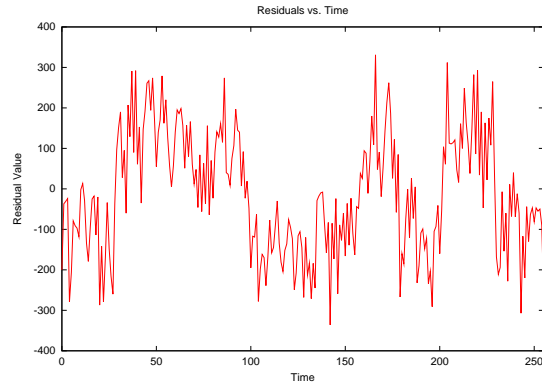


Figure 6: Input Block Residuals

Square (optimal for Elias-Gamma). The histograms appear Laplacian in nature, offering one possible explanation for Rice’s outperformance of Elias-Gamma. On the other hand, arithmetic coding is not optimized for a specific distribution – it “discovers” the best distribution for the data by partitioning the interval $[0, 1]$ over \mathbb{R} . This is perhaps the strongest explanation for the arithmetic coder’s superior compression performance, however, it is also in general twice as slow as the other codes. We explain the strong performance of Delta coding by noting that the residuals tend to still resemble a coherent waveform, limiting the inter-sample amplitude distance (see Figure 6).

We spent some time exploring the value in exploiting inter-channel correlation between the left and right stereo channels. We found that it improved performance in some cases, but we could not come up with a clean way to incorporate into our current scheme, as it seemed to penalize compression as often as it aided it.

7 Conclusion

Ultimately, our experiences suggest that wavelets may not be the right paradigm for lossless sound compression. While we were able to achieve reasonable performance, simpler compression schemes like LPC were able to achieve superior results. We suspect this is because LPC makes fewer assumptions about the nature of the data to be compressed.

While we originally believed that wavelets were the correct basis because it exhibited compact support, our small block size may nullify this advantage. On such a small block size, the difference between compact and global support seems minor. In the future, we would be interested to see how a DCT-like transform performed as a predictive modeler. In light our experiences with wavelets, we suspect that it may be more successful at modeling the tonal nature of music.

References

- [1] Microsoft Corporation. Riff tagged file format. Technical report, Microsoft Corporation, 1992.
- [2] P. M. Fenwick. Punctured elias codes for variable-length coding of the integers. Technical Report Report Number 137, The University of Auckland, Department of Computer Science, 1996.
- [3] Gabriel Fernandez, Senthil Periaswamy, and Wim Sweldens. Liftpack: A software package for wavelet transforms using lifting. In *Wavelet Applications in Signal and Image Processing IV*. SPIE Conference, 1996.
- [4] Ciprian Doru Giurcaneanu, Ioan Tabus, and Jaakko Astola. Integer wavelet transform based lossless audio compression, 1999.
- [5] Amara Graps. An introduction to wavelets. In *IEEE Computational Science and Engineering*, volume 2 of 2, 1995.
- [6] Ankit Patel and Mark Tonkelowitz. Whasupp: A novel approach to query-by-sketch using wavelet coefficients and color histograms. Technical report, Division of Engineering and Applied Sciences, Harvard University, 2000.
- [7] Tony Robinson. Shorten: Simple lossless and near-lossless waveform compression. Technical report, Cambridge University Engineering Department, 1994. Technical Report CUED/F-INFENG/TR.156.
- [8] George Tzanetakis, Georg Essl, and Perry Cook. Audio analysis using the discrete wavelet transform. In *WSES International Conference on Acoustics and Music: Theory and Applications*, 2001.
- [9] Robin Whittle. First principles: Lossy and lossless compression of audio, 2001. <http://www.firstpr.com.au/audiocomp/>.