Audio Compression using Entropy Coding and Perceptual Noise Substitution

Kapil Krishnamurthy and Tamer Deif
MUSIC 422
Stanford University - Winter 2009

1 Motivation/Background

When compressing an audio signal there are two main concerns, the compression rate and the quality of the decompressed audio. Tradeoffs occur according to the application in which the codec will be used. For example, if it is used for telephony systems compression rate is more important than quality, of course as long as the quality of the output speech is acceptable.

In our implementation of our codec we were concerned about achieving higher compression rates than our original (class) codec while, at least, maintaining the same quality as our original codec. In other words, for a given bit rate, the quality of the new codec should be better than the old one.

As we studied in this class, there are two main strategies to increase compression rate without deteriorating the quality; either lossless compression, which does not affect the quality of the signal at all, or lossy compression that groups its artifacts in non-perceivable ranges by the human auditory system. In the suggested codec the two strategies had been used. There are two main additions to our original codec; the first one is "Perceptual Noise Substitution" (PNS) and the second is "Entropy Coding" (more specifically, Huffman Coding).

PNS is a lossy compression technique that is based on the assumption that all white noise sounds similar to the human being. The technique is founded on detecting the "white-noise" like frequencies of a given audio signal and coding their power and frequency ranges instead of coding the original data. In fact, what PNS does is that it checks noisy bands in the signal. If a band is found to be noisy, then the whole band is coded by the power of the signal in it, which means only one parameter (noise power) describes the data in the whole band. This is also an introduction to parametric coding. On the other hand, on the decoder side, the flagged noisy bands are checked on a block-by-block basis and "white-noise" is generated and injected into the signal at those frequency ranges.

Clearly this technique saves bits in the case of noisy audio signals, however the more tonal the signal is, the less bits it saves. The number of bits saved, hence, vary according to the characteristics of the input signal, whether it is noisy or not and which bands are the noisy ones. In the following sections a more detailed implementation will be discussed.

The different motivations behind implementing PNS, apart from bit rate reduction, are that it takes advantage of the perception of the human auditory system, which find to be an interesting topic, and that it introduces us to the idea of parametric coding.

The definition for entropy is basically the measure of randomness of a system. Entropy coding is a form of coding that replaces a particular symbol in a given bit stream by a unique code word. The code word may be constructed in several different ways depending on the method employed. Similarly, by the same method we can obtain the original symbol back from the code word.

Entropy coding is done past the quantization stage prior to bit packing and transmitting the bit stream. The main motivation for using entropy coding is to achieve lossless compression. Using entropy coding, we basically replace frequently occurring symbols in the mantissa for each critical band with a code that is typically smaller than the original symbol. In this manner, we reduce the number of transmitted data bits, which gives us more compression of the stored audio file.

There are several types of entropy coding. Some of the commonly used ones are Huffman coding, Arithmetic coding and Rice coding. For our coder, we have used Huffman entropy coding.

2 System Overview

The overall system is very similar to the typical perceptual audio coder; figure 1 shows a block diagram of the system. On the encoder side, normally, there are two paths; one for the actual encoding and the other is for perceptual modeling. The perceptual modeling determines masking effects and bit allocation. In our implementation we added a third branch for prediction and noise detection. The output of those two blocks is the noise substitution flags for the frequency bands. These flags are then used by the mainstream coding steps to detect how to code the data (determine bit allocation for noisy bands as well as data to put in the scale factor). After the bit allocation, quantization takes place and then Huffman coding.

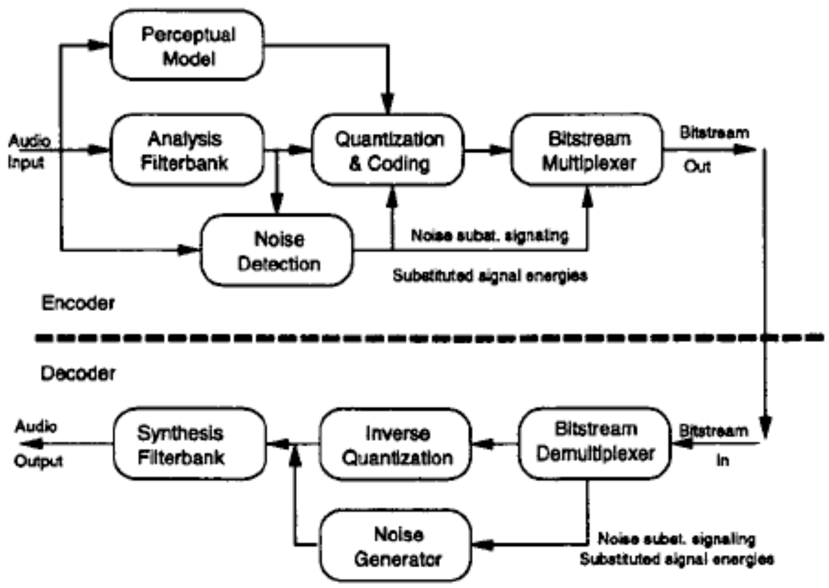


Figure 1: General System Block Diagram

The exact opposite of the mainstream coding steps take place on the decoder. Huffman decoding is done, followed by de-quantization of the non-noisy bands and then generating noise with the corresponding power for the noisy bands. Finally and time transform is run on the data to reconstruct the time domain signal.

2.1 PNS

The PNS algorithm is a simple one. The input signal is fed to a frequency transform stage, and then the spectrum of the signal is examined for noisy bands. One of the reasons for which the noise detection is done on band basis is that it will be simpler to encode/decode as will be shown later on in the implementation section. To detect noise, a prediction algorithm is used where the predictor tries to anticipate the following sample of the signal based on previous sampled from the same signal. The reason this works is that, basically, white noise is unpredictable, so whenever the predictor fails to follow the signal a noisy band is present (on a side note, this is the same for measuring tonality in a signal; the more accurately the predictor follows the signal, the higher its tonality index is).

A decision stage comes after the noise has been detected, where the system determines whether to substitute the noise in the detected bands or not. There are several issues to consider before substituting for the detected bands with noise. One of the issues is the risk that our detector might have detected a tonal band as noise. Of course, if a tonal band is substituted for by noise, then signal quality is immediately destroyed. In order to avoid this case, few measures are considered. The first one is the variation among different bands; if a noisy band is detected however its neighboring bands (immediate left and right neighbors) are not noisy as well, then it is considered tonal and noise substitution is not conducted for that band. Another measure is the power distribution within a noisy band, if the variance between the powers of the individual frequency lines in a noisy band is high (to be more precise, some measures like variance-to-mean ratio are conducted), then the content of this band is considered tonal and no noise substitution is done. Some other measures can be conducted, like variation of noise power with a group of three bands. In addition to the pervious issue, there are other problems that have to do with human perception that must be taken into account. For example, usually no noise substitution is done for frequencies below 5 KHz. Our auditory system is very sensitive in that frequency range, and therefore the ear can more distinctively detect noise. Another concern is the amount of noise injected into the signal, PNS can be potentially overused if we implement it for every noisy band detected. This in general would lead to degrading the quality of the signal. Hence, usually the noisy bands are sorted in terms of their noisiness and only few of them (20-40%) go through noise substitution (of course after passing the rest of the tests mentioned above).

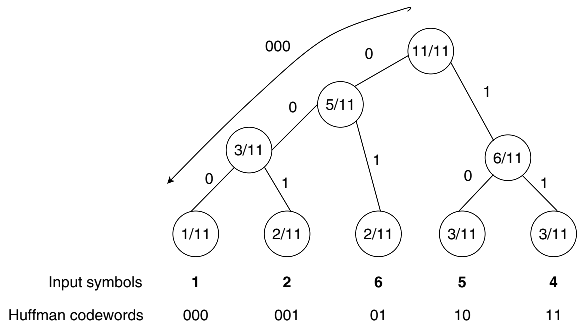
After examining the noisy bands, the power of the ones that pass the previous tests are measured and stored in the scale factor bits for the corresponding sub-band. The mantissa bits for those bands are free to be used by other "non-noisy" bands, which allows for higher compression rates. Finally, the noisy band has to be flagged somehow so that the decoder understands them. One possible way to do this is to out the flag information in the header of each block. Another possible way is to use a special code and allocate bits for that code to be sent for only noisy bands.

On the decoder side, the first thing that is done is checking the flags (regardless the way the are sent). Then for the corresponding bands, white noise is generated in the frequency domain with power equal to the values read from the noisy bands scale factors. Finally, an inverse frequency transform is conducted to have the signal back in time domain.

2.2 Huffman Coding

Huffman coding is regarded as one of the most successful compression techniques available today. It is used commonly for compression of both audio and images. For the generation of the codes based on the frequency of input symbols, the first step is to construct a Huffman table.

A typical Huffman table for the input stream X = [4 5 6 6 2 5 4 4 1 4 4] is shown in figure 2.



Input symbol	Probability	Huffman codeword
4	3/11	11
5	3/11	10
6	2/11	01
2	2/11	001
1	1/11	000

Figure 2: Huffman tree and corresponding table

Symbols are arranged in the decreasing order of frequency (increasing order of probability of occurrence). The main constituents of a Huffman tree are nodes and leaves. At each step, we compute the two leaves of lowest probability and then club them together to form a node. In this manner, the tree is constructed in a bottom up approach over N-1 steps where N is the number of symbols. To each left going path, a 0 is assigned and to each right going path, a 1 is assigned. In order to construct the code corresponding to a given symbol, move down the tree in a top down approach and build up the code for that symbol.

There are several variations of Huffman coding that can be implemented. The three main types of Huffman coding can be summarized as non-adaptive, semi – adaptive and adaptive Huffman coding.

The non-adaptive Huffman coding model is one that is typically used in audio coders. Basically, a number of Huffman tables are constructed for different frequency bands and also different genres of music. It is essential to have a reliable symbol to frequency mapping, so a lot of training data is used to prepare these tables.

Semi adaptive Huffman coding is also known as the 'two pass' encoding scheme. In the first pass, a Huffman code table is designed based on the input symbol statistics. In the second pass, entropy coding performed using the designed Huffman code table. In this scheme, the designed Huffman code tables must be transmitted along with the entropy coded bit stream.

Adaptive Huffman coding uses prediction techniques to compute the symbol frequencies based on the previous samples. The advantage to this is that the source can be coded in real time. However, a single error can lead to the loss of all the data.

The next section shows more details of the implementation of the described steps.

3 Implementation

Our implementation is built on the coded implemented in class, so in this section we will only discuss our implementation to the added features to the codec, namely PNS and Huffman Coding.

For the Noise detection stage of the PNS we implemented a predictor. The function of the predictor, as mentioned earlier, is to predict the signal from its past samples and depending on how successful it is we decide how noisy is our signal (remember that noise is impossible to predict). The function used for prediction is:

$$\hat{P}(n) = 2 * P(n-1) - P(n-2)$$

Where $\hat{P}(n)$ is the prediction of the current sample, and $P(n-1), P(n-2)$ are the actual values of the two previous samples. The prediction stage is conducted on the FFT of the original signal. FFT is chosen because it provides a higher frequency resolution than MDCT.

After prediction a noise index for each band is measured using the following equation:

$$N_j(m,n) = \frac{\sum_{i=m}^{n} \left| \hat{P}_j(i) - P_j(i) \right|}{\sum_{i=m}^{n} \left| P_j(i) \right|}$$

Where *j* is the index for the sub-band number, *i* is the index for the sample number in the j*th* sub-band, *P* is the same as before and *N* is the noise index. This function is just a normalized summation of the error of the predictor, which represents our noise index. For a sub-band to be noisy, its noise index must be above a certain threshold. After testing on a white noise signal and observing the average noise index for the sub-bands, a threshold of 0.5 was determined. More accurate thresholds can be obtained using different combinations of data; also an independent threshold can be used for each band. Figure 3 shows a graph of the noise index for a tonal signal vs. white noise.

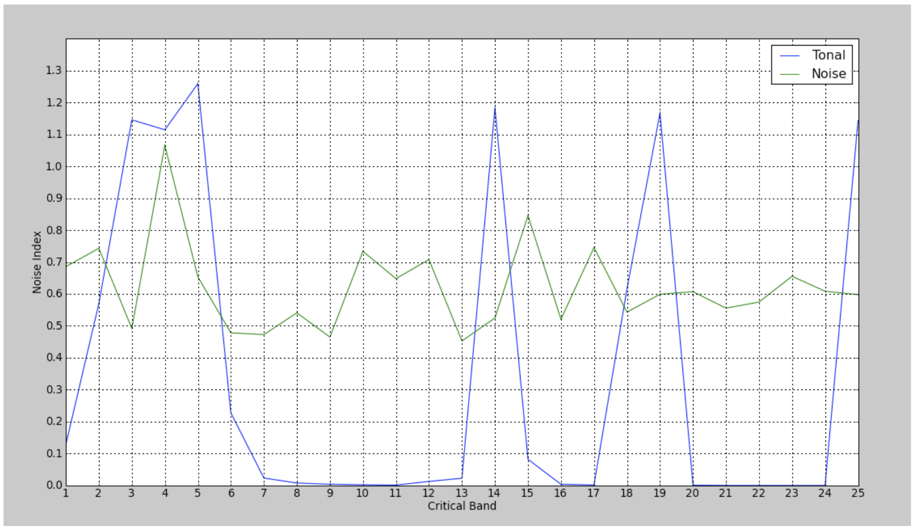


Figure 3: Comparison of Noise Index for White Noise vs Tonal signals

After calculating the noise index, the tests mentioned in the previous section are conducted to examine the system's confidence about the noise substitution decision.

Finally, the noise flags are set by the previous stage and accordingly no bits are allocated to flagged bands, the noise flags are added to the header of the block and sent to the decoder, and power is measured for those bands by summing the square of the spectrum amplitudes and coded in place of the scale factor of the corresponding bands by uniformly quantizing it.

On the decoder side, the header for each block is read and the noise substitution flags are extracted. For noisy bands, the scale factor is read and de-quantized to generate white noise with the corresponding power. The white noise is generator using a random number generator, and scaling it with the noise power in the band. This is added to the decoded MDCT values for the rest of the signal to reconstruct the whole frequency domain representation of the data before PNS. This is the input to the inverse MDCT to recover the time domain signal. Figure 4 shows a white noise input and the output of the noise generation function when it is all considered to be noisy vs. when the Noise detection algorithm is run on it.

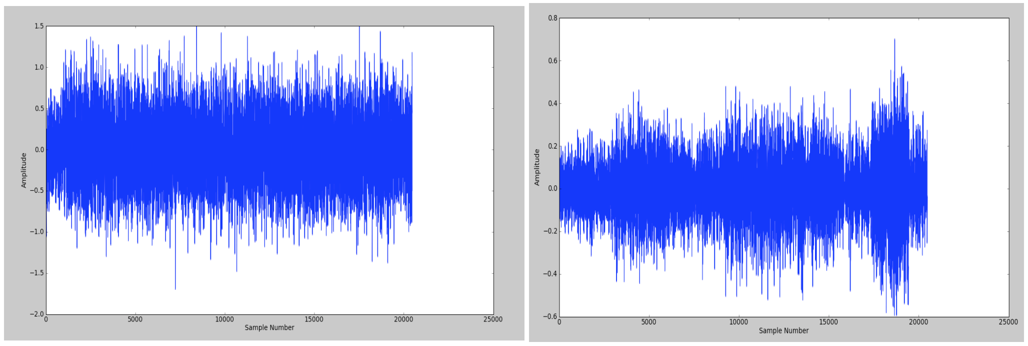


Figure 4: On the left is white noise generation when all input is not filtered. On the right is filtered white noise output.

In this project we have designed a non-adaptive Huffman coder, which involves having a static table at the encoder and decoder with the list of symbols and their respective codes. This list varies per critical frequency band and also with the number of mantissa bits taken into consideration. The lists were constructed by training the coder with different genres of music like pop, rock, electronic and classical.

The first step to the implementation involves constructing the reference tables for the different symbols and their respective codes. To do this, we turned off the bit allocation algorithm and allocated a fixed number of mantissa bits during block quantization. The number of mantissa bits spanned the range of 2 to 8 and for each value of mantissa bits, we calculated the frequency of occurrence of all the symbols for the 25 critical bands. Doing this step is useful because just by looking the tables, we can see that for many types of signals, the frequency of zeros at the highest critical bands is very high. This enables us to do things like club the lists for those bands together or even omit the mantissas of those bands. Figure 5 shows the frequency of 5-mantissa bit values for different music genres.

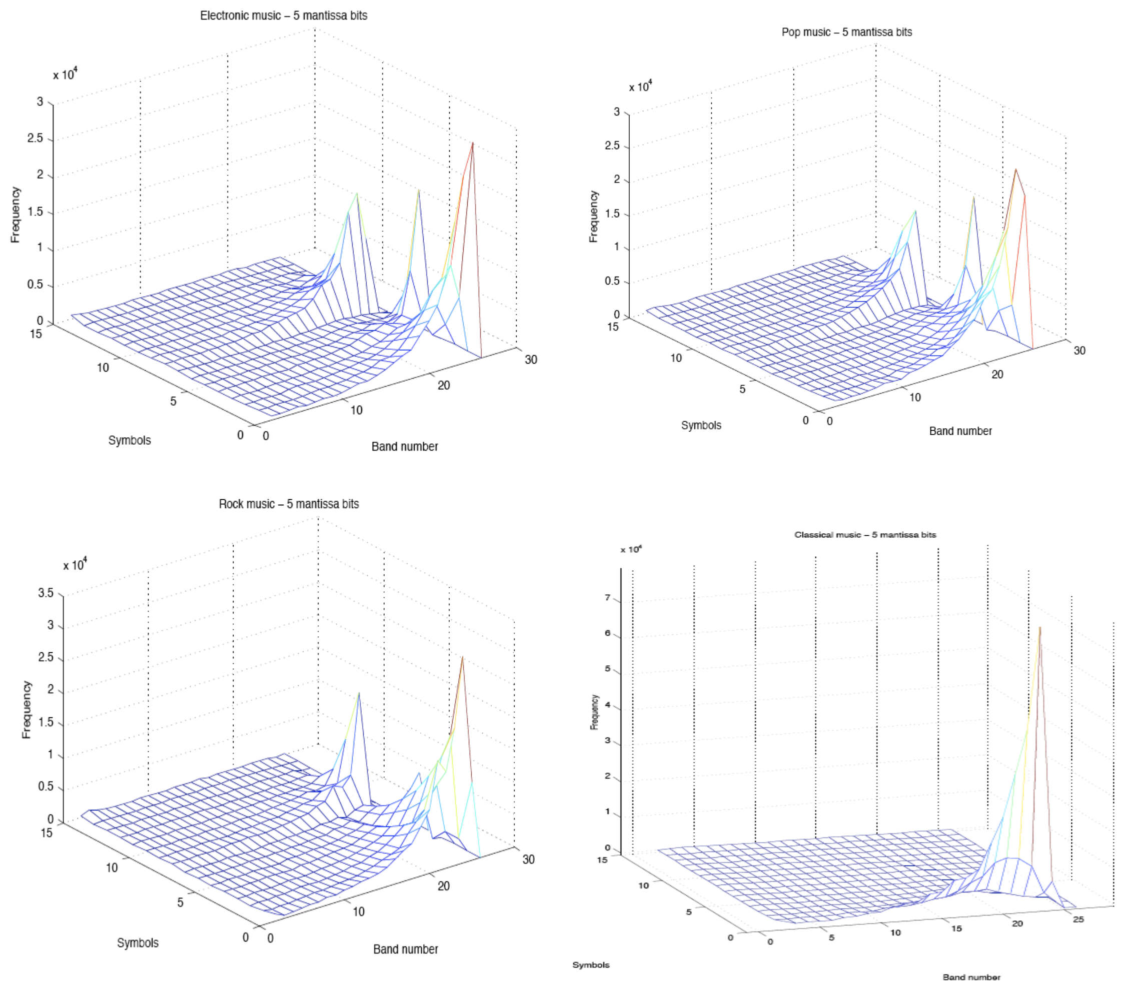


Figure 5: Comparison of 5-mantissa bit values among different music genre

The next step involves converting the frequencies of all the symbols to equivalent Huffman codes by constructing a Huffman table for each critical band. In our case, this was done with the help of Matlab. Using the Matlab file I/O options, we had the Huffman table in a reference able format.

At the encoder end, once the bit allocation algorithm is completed and certain number of bits have been allocated to each of the critical bands for block quantization, we use the Huffman functions to search for the equivalent code corresponding to the mantissa symbol. This code is written into the bit stream in place of the mantissa. The important factor to be considered here is that the mantissa is first separated into its magnitude and sign parts and they are coded in separately.

Similarly, at the decoder end, once all the usual parameters such as the scale factors and bit allocation information have been retrieved from the bit stream, we extract the Huffman code and the sign. Using this information, we can locate the original mantissa symbol from the table and apply its sign back to it.

4 Results

The following table shows a comparison between the compression rates of three different decoders. The first one is the normal 128 Kbps decoder, the other two are Huffman and Huffman with PNS. The results of this table are all based on 128Kbps target rate.

Codec	Classical	Rock	Dance	Pop	Castanets	Organ	Speech
128	5.514	5.505	5.580	5.358	5.368	5.530	5.281
Huffman	6.637	6.400	6.500	6.312	6.051	7.353	6.035
Huffman/PNS	6.700	6.650	6.908	6.583	6.163	7.353	6.260

It is obvious that Huffman gives us same quality, for a target bit rate, but with higher compression, for all kinds of audio. The most outstanding result was for the Organ, which is a highly tonal signal. Moreover, the PNS increases compression rate in most cases. However, it is noticeable that PNS did not add anything to the compression rate in the Organ signal, which is a very tonal signal and apparently no noise was detected in it.

The next figure summarizes the performance of the codecs in terms of quality of output. The tests are based on the ITU-R 5-grade impairment scale, double-blind triple-stimulus tests. Note that the maximum number of bits per mantissa was limited to 8 bits, due to the availability of Huffman tables only up to that number of mantissa bits. This would reduce the quality of the codecs in general, but relative to each other they should remain the same.

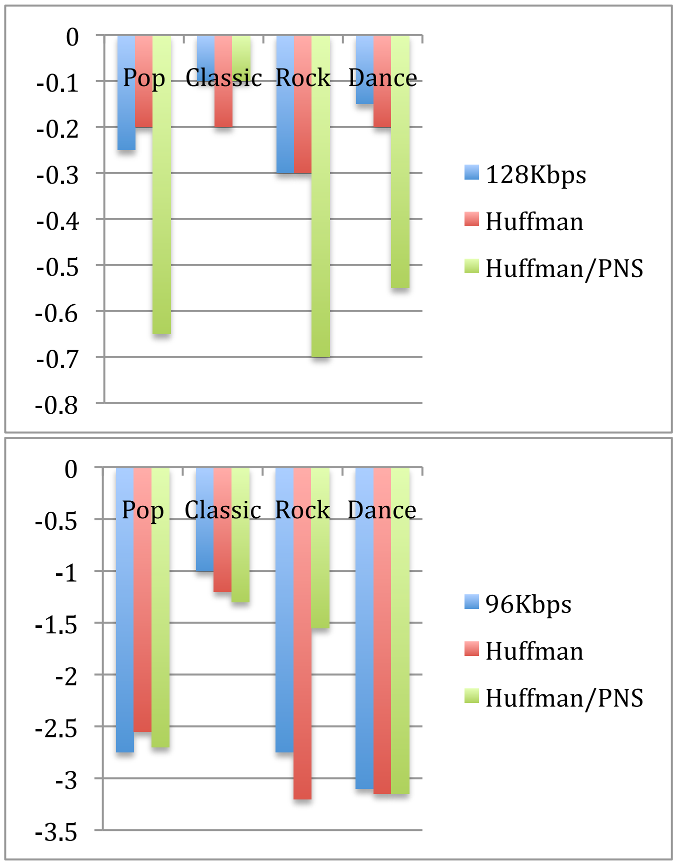


Figure 6: Comparison of quality test between coders for different genres of Music

Figure 6 shows that Huffman and non-Huffman coding are very close in performance while PNS differs according to the signal type. Figure 7 shows another summary of performance of the coders when applied to tonal, speech and impulsive signal (castanets).

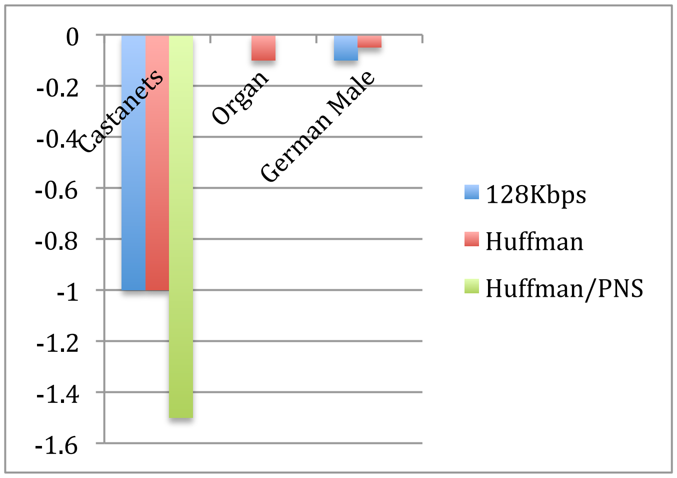


Figure 7: Comparison of quality test between coders for different types of signals

5 Conclusion/Future work

As the results showed, Huffman coding gives a great deal of compression gain. Of course the fact that this is lossless, makes it a very attractive option for any coder – high compression rate with no degradation in quality. The main disadvantage of Huffman coding is that it requires more computational power and time. This might be a problem for real-time codec (e.g. streaming applications), however with some code optimization and the continuous advancement in processors power it should be fine.

On the other hand, PNS is not as powerful as Huffman for few reasons. The first is that it is a lossy coding technique, so the coded signal can never be exactly reconstructed. Another reason is that it does not provide high compression rate, it helps but not too much. Although, both, Huffman and PNS provide variable compression rates that are dependent on the input to the codec, Huffman is more guaranteed to compress the signal. For example if the signal is not noisy, i.e. highly tonal, then theoretically no PNS takes place and hence no compression at all, however this is not the case with Huffman (it would only fail for abnormal signals with all values corresponding to the longer Huffman codes). Meanwhile, PNS would be a great addition for noisy signals.

It must be said that there is a huge amount of work that can be done to extend both PNS and Huffman. So for PNS, some work can be done on better noise detection algorithm, especially with coming up with more accurate thresholds. Also, more work to decide whether a band is noisy or not, for example some algorithms measure the variance in average power between groups of three bands and high variance means the signal is not noisy. Another issue to be taken care of is NS with stereo coding (which was not implemented here). In stereo coding, noise substitutions may lead to undesired correlation/non-correlation between the two channels that is not present in the original signal. This is one of the reasons for doing NS only for bands above 5 KHz, because below those frequencies is the range where the human auditory system is most sensitive to those artifacts.

For Huffman coding, we only implemented straightforward coding on a symbol-by-symbol basis. There are a lot of extensions that can be done and lead to higher compression rate. For example, vectorized Huffman coding for select critical bands is very famous (where you apply Huffman on several symbols instead of only one). This is beneficial in bands where there is a high repetition of certain symbols. Usually the last 2 – 3 critical bands have a high percentage of zeros because the input signal lacks high frequency content of that order. Using vectorized coding, several bits can be saved by grouping the repeating symbols together and replacing them by a single code. Vectorized quantization can also be applied to more than just a few bands if we have a higher probability of several symbols being grouped together. To achieve this, we can use the Burrow-Wheeler transform coupled with the Move to Front algorithm. These two modules are applied on the mantissa of the code and then sent to the vectorized Huffman routine.

Also, the entropy coding technique can be applied on the scale factor and header information for further compression. One more thing that can be implemented is coding of run streams of zeros or ones (kind of similar to vectorized Huffman).

6 References

1) ' Audio Signal Processing and Coding' by Andreas Spanias, Ted Painter and Venkatraman Atti

2) 'Sampled-Data Audio Signal Compression with Huffman Coding' by Shinjiro Ashida, Hironori Kakemizu, Masaaki Nagahara and Yutaka Yamamoto

3) 'Improving Audio Codecsby Noise Substitution' by Donald Schulz

4) 'The MPEG-4 General Audio Coder' by Bernhard Grill

5) 'A Tutorial on MPEG/Audio Compression ' by Davis Pan

6) 'Implementation of MPEG-4 Audio Components on Various Platforms' by Bernhard Grill, S tefan Geyersberger, Johannes Hilpert, and Bodo Teichmann

7) 'Extending the MPEG-4 AAC Codec by Perceptual Noise Substitution' by Jiirgen Herre, Donald Schulz

8) 'Advanced Software Implementation of MPEG-4 AAC Audio Encoder' by Danijel Domazet, Mario Kovac

9) 'An Overview of MPEG-4 Audio Version 2' by Heiko Purnhagen

10) 'An MDCT Domain Frame-Loss Concealment Technique For MPEG Advanced Audio Coding' by Sang-Uk Ryu and Kenneth Rose