DAMSAC
Dynamic Allocation Mid-Side Audio Codec

MUSIC 422 Final project
CCRMA, Stanford University

Andrea Baldioceda, Jan Stoltenberg, Vivian Chen

March 13, 2020

Source: `https://github.com/vivianschen/Audio_Codec`

Introduction

This paper introduces the *Dynamic Allocation Mid-Side Audio Codec (DAMSAC)*, an audio coder that significantly lowers the data rate of audio files while maintaining a high audio quality by making use of psychoacoustic properties. More specifically, it is aimed at coding dynamic audio content at low audio bit rates (e.g. 128kbps or less).

The baseline coder identifies masking frequencies in the signal and allocates the available bits in a manner such that quantization noise is shifted to less audible masked parts in the frequency spectrum.

DAMSAC achieves additional coding gain on top of the baseline coder by exploiting the fact that specific audio blocks require a higher number of allocation bits than other blocks in order to encode the signal at a certain quality level. By adding a bit reservoir to our existing base coder, we operate on a locally variable data rate while maintaining the overall data rate - a *Variable Bit Rate (VBR)* instead of a *Constant Bit Rate (CBR)*. With a VBR we are able to save a certain amount of bits when we don't need them, e.g. during quiet passages in the signal, and later allocate a larger amount of bits for more complex blocks.

Additionally, DAMSAC employs *Mid-Side stereo coding (MS)*, which further avoids redundancy by firstly computing correlations between the two channels and, if appropriate, separating the signal into a mid and a side channel, which gives an additional coding gain for most audio signals.

The larger the amount of bits we can use to represent an audio signal, the higher the audio quality we can achieve. However, applications that can't afford a high bit rate benefit from DAMSACs advanced bit allocation coding: The sparser bit budget is dynamically spent on audio passages that, based on the implemented psychoacoustic calculations, need more bits than others.

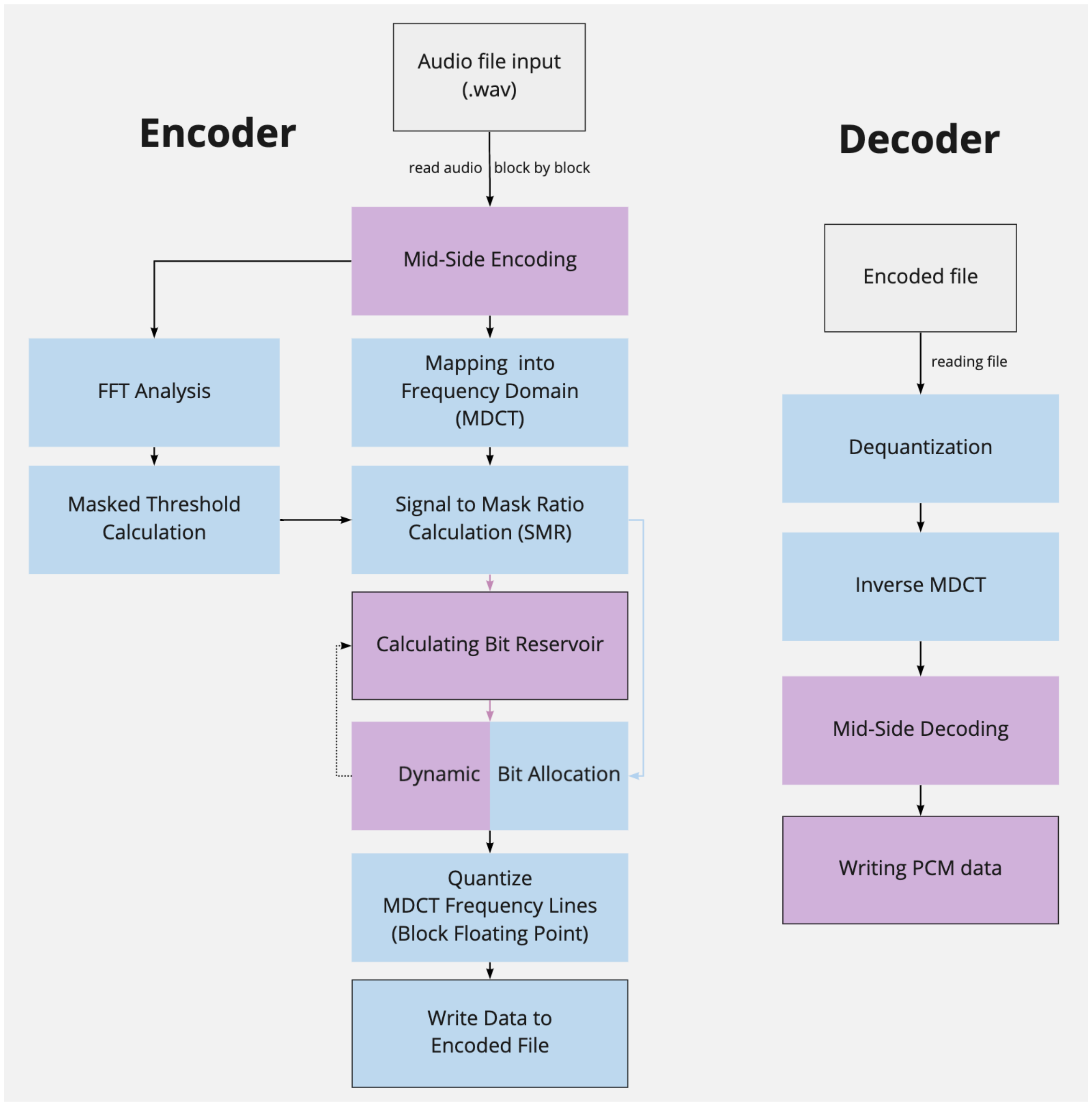
Encoder
Audio file input (.wav)
read audio
block by block
Mid-Side Encoding
FFT Analysis
Mapping into Frequency Domain (MDCT)
Masked Threshold Calculation
Signal to Mask Ratio Calculation (SMR)
Calculating Bit Reservoir
Dynamic Bit Allocation
Quantize MDCT Frequency Lines (Block Floating Point)
Write Data to Encoded File
Decoder
Encoded file
reading file
Dequantization
Inverse MDCT
Mid-Side Decoding
Writing PCM data

Figure 1: DAMSAC Encoder and Decoder
Elements of the baseline are colored blue, added features are colored violet

Methods

Baseline Coder Overview

Our baseline coder relies on psychoacoustic models to lower the data rate of a file while maintaining the quality as much as possible. The baseline coder is divided into two main parts, the encoder and the decoder. The encoder consists of two paths (shown in Figure 1 – in blue), the main path and the side path. The side path can be seen as an analysis path, while the main path is seen as the implementation path.

The side path consists of taking the Fast Fourier Transform (FFT) of the signal and then analyzing the result to compute the corresponding masking threshold. By using principles of psychoacoustic masking techniques, frequencies in our signal that are naturally masked (not perceived by our ear) are identified. Using this information, we can assign less bits to quantizing those masked frequencies. Following Zwicker and Fastl's [1] 25 critical bands scale, each critical band's maskers are identified and then a combined masked threshold is created by layering the threshold in quiet [2](p. 152) and all the identified maskers. This masked threshold is then fed to the SMR block of the main path.

The main path takes the Modified Discrete Cosine Transform (MDCT) of the input signal and uses the result along with the masked threshold to calculate the Signal to Mask ratio (SMR) of each critical band. To calculate the SMRs, firstly the MDCT of the windowed input signal is computed in Sound Pressure Level (SPL), followed by our masked threshold in SPL. Then, the difference is computed and the maximum SPL value of each critical band is taken. This leaves us with 25 SMR values, one for each critical band. The SMR values are then used to determine how many bits to allocate for the scale factor and mantissas of each critical band. To carry out the bit allocation, a water-filling algorithm is used, which allocates all the bits available for each block until the bit budget is exhausted. Finally, our encoder quantizes the MDCT signal using the specified scale factor and mantissa bits for each critical band. The bit allocation information is transmitted to the decoder.

The decoder takes the bit allocation information transmitted from the encoder to dequantize the quantized MDCT signal. It then takes the Inverse MDCT (IMDCT) of this dequantized signal, windows the result, and finally returns the resulting time domain output signal.

New Features and Implementation

1) Bit Reservoir

To achieve a higher coding gain we decided to implement a bit reservoir. This feature consists of having a variable bit rate for each block instead of a fixed bit rate. By doing this, a smaller amount of bits can be allocated to more quiet or silent blocks, saving the remaining bits to the bit reservoir. The bits from the reservoir can then be used on future blocks that are more complex and thus need more bits to achieve a certain quality level.

To implement the bit reservoir a threshold was created to determine when to stop allocating bits for each critical band. Previously in the base coder, all the available bits were allocated for each block, regardless of the SMR values of each block. In DAMSAC, the SMR values of each critical band are analyzed and bits are allocated until all the SMR values are lower than the threshold. This ensures that bits are allocated until the signal is masked, in which case there is no need to allocate further bits.

Another feature that was added to DAMSAC is the maxBitBudget value, which determines the maximum amount of saved bits that will be allocated to the bit reservoir. The extra bits that are not saved to the bit reservoir are then reallocated throughout the critical bands. This is a solution to the problem where, depending on the length and content of the audio signal, it may be desirable to more generously allocate bits to the signal rather than end up with a large bit reservoir that could have been spent on better encoding.

2) MS Stereo Coding

The motivation behind this feature is that most audio files are stereo instead of mono. In most stereo audio files, the two channels are highly correlated, which leads to redundancy. This presents an opportunity for our coder to save more bits to be allocated later. DAMSAC chooses whether a band will be coded with LR or MS coding depending on the correlation between the channels - using MS is best for highly correlated channels. The LR implementation remains the same as in our base coder. The MS implementation converts the LR signal to an MS signal and then for each critical band identifies which channel has a larger SMR . This channel's SMR value is then lowered by 6dB.

Correlation [3]
Stereo coding is advantageous when two channels are highly correlated. This correlation, e.g. used by MPEG-2 [2](Chapter 12), is calculated using the following equation:

$$\sum_{n=lower}^{upper} |(L_n^2 - R_n^2)| < 0.8 \sum_{n=lower}^{upper} |(L_n^2 + R_n^2)|$$

where L_n and R_n are the FFTs of the left and right channels, respectively. If the correlation is greater than a threshold (indicated by 0.8 in this case), then for that specific block MS coding is beneficial.

Encoding
During encoding, we transform time-domain LR channels to MS:

$$M = \frac{L+R}{\sqrt{2}}$$

$$S = \frac{L-R}{\sqrt{2}}$$

Decoding
In order to decode the signal, we assign the left and right channels to the sum and difference of the mid and side channels respectively:

$$L = \frac{\sqrt{2}}{2}(M+S)$$

$$R = \frac{\sqrt{2}}{2}(M-S)$$

When looking at figure 1, it should be noted that the MS signal is calculated both in the time-domain and frequency-domain based on the LR time domain and frequency domain channels. This is done in the beginning of our coder. Later at the end of the decoder, the MS channels are converted back to LR channels. As encoding and decoding a MS signal is an entirely lossless process, it makes sense to operate on the MS channels throughout the entire codec.

Fine Tuning

To fine tune the coder, additional changes to the coder were made by experimenting on different values for our main parameters. These changes include modifying the order in which bits are allocated, adjusting our maxBitBudget value, and determining whether to use a threshold value in our bit allocation function versus using a drop value in our psychoacoustic function.

To determine which parameters and values to use in the coder, three audio files with different content were used during the fine tuning process: castanet.wav (transient-rich castanets recording), intoxicated.wav

(excerpt of a pop song), and spfe.wav (British female speaker). Castanet.wav and spfe.wav feature more silences and a wider dynamic range than intoxicated.wav, which represents a typical pop song with low dynamic range and dense complexity.
In this process, our parameters were chosen based on the goal of optimizing our coder for a bit rate of 128kbps. We later modified some of the chosen parameters accordingly to optimize our coder for a bit rate of 96kbps.

1) Bit Allocation Order

When allocating bits for each block, the coder was modified to favor the lower frequency range of the signal when allocating bits for the quantization. This way, the coder prioritizes low-passing the signal instead of introducing distortion at lower frequency bands. To do this, the main bit allocation algorithm was divided into two parts, first allocating bits to the lower 12 critical bands, and then allocating bits to the remaining upper 12 critical bands.

2) Drop vs. Threshold

When first implementing our bit allocation algorithm, we used a threshold of zero to decide when to stop allocating bits. This means that once all the SMR values are negative, no more bits are allocated. This introduced a problem since not enough bits were being allocated and in result we were over-compressing the audio file. To resolve this issue we decided to experiment with the threshold value in the bit allocation algorithm, as well as the drop value in the psychoacoustic model.

While analyzing the two algorithms we realized that the main difference between modifying the threshold versus the drop is that the drops applies only to the maskers while the threshold applies to both maskers and the threshold in quiet. When using the drop [2](p. 180 ff.), the maskers are lowered while the threshold in quiet stays the same. Therefore, the higher frequency bands are already being masked by the threshold in quiet. On the other hand, when using the threshold, the SMR values are lowered for all frequency bands, which is ultimately equivalent to lowering both the maskers as well as the threshold in quiet.

To compare the impact of using the drop value versus the threshold value, several audio files were coded using different values for drop and threshold. Table 1 shows the differences in compression ratio when using drop versus threshold for a bit rate of 128kbps.

Table 1: Compression Ratios for Drop vs. Threshold (128kbps)

	drop = 16		threshold = 0	
	threshold = -6	threshold = -18	drop = 24	drop = 34
castanet	6.36	5.39	6.47	5.4
intoxicated	5.89	5.37	5.69	5.36
spfe	12.86	9.66	14.0	11.75

According to Table 1, most of the values do not vary much when using the threshold versus the drop. However, the spfe (British female speaker) has a higher compression ratio (14:1) when lowering the drop to 24 (8dB lower than the original 16dB in our baseline) than the compression ratio (12.86:1) when lowering the threshold to -6dB. A possible explanation could be that there are not many maskers at high frequencies for a speech signal (as the MDCT lines are densely packed and do not differ much at high frequencies of the human voice). Therefore, lowering the drop value does not have as great of an effect as lowering the threshold value. Moreover, we know that the threshold in quiet is also high for low frequencies, however this does not make as much of a difference since female vocals' fundamentals usually reside above 200Hz.

Because the goal is to get as close as possible to our desired compression ratio and also avoid over-compressing, we chose to lower the threshold value instead of lowering the drop value for our coder. Furthermore, we decided to use a threshold value of -18 to get as close to the desired compression ratio (5.51 for 128kbps). For similar reasons, when testing the values for a bit rate of 96kbps we decided to use a threshold value of -6. In both cases there were no significant differences in audio quality when using the threshold value vs the drop.

3) MaxBitBudget

To determine the maximum amount of bits to save to our bit reservoir for each block, we experimented with the maxBitBudget value and evaluated the results in terms of audio quality and compression ratio of the files. For a bit rate of 128kbps and using a threshold of -18, we compared the differences of using a maxBitBudget of 5000, 10000, and 100000.

Table 2: Compression Ratios for given MaxBitBudget (128kbps)

threshold = -18			
	maxBitBudget = 100k	maxBitBudget = 10k	maxBitBudget = 5k
castanet	5.39	5.39	5.39
intoxicated	5.36	5.36	5.36
spfe	5.66	5.42	5.40
harpsichord	5.7	5.42	5.40

Table 2 shows the effect of maxBitBudget on compression ratios for 128kbps. Our goal is to have a bit budget that does not constantly deplete yet is not overly large so that it is appropriately spent and dynamically varies in a certain range. A maxBitBudget of 10k was kept for 128kbps because it was the closest to the target compression ratio of 5.51. We performed the same test on 96kbps, and due to similar reasons chose 100k for its maxBitBudget. In terms of audio quality, we could hear whenever the coder was running out of bits, since there was a slight but noticeable drop in audio quality. Using larger values for maxBitBudget, just means that the drop in audio will happen at a later block in the signal. In our case, we decided to find a middle ground between when that audio quality drop happened and getting close to our target compression ratio.

Results

Listening Tests

Listening tests were conducted according to BS.1116 standards with 6 subjects for 4 files: castanet, harpsichord, spfe (British female speaker), and intoxicated. The SDG plots can be seen in Figures 2-5. The average values from our DAMSAC coder are closer to 0 than those from the baseline coder, proving that it performs better than the baseline at both 96kbps and 128kbps. As expected, intoxicated.wav is the hardest to differentiate.

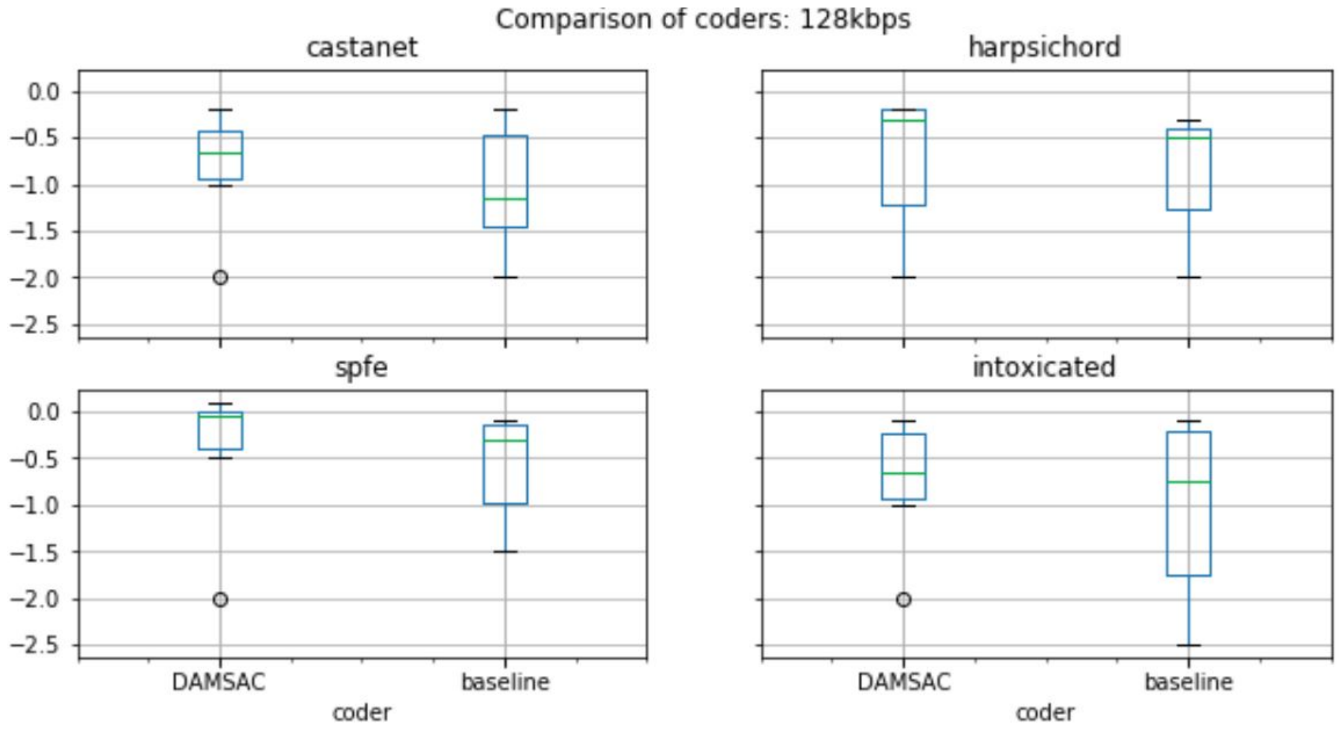

Figure 2: Boxplots grouped by coder at 128kbps

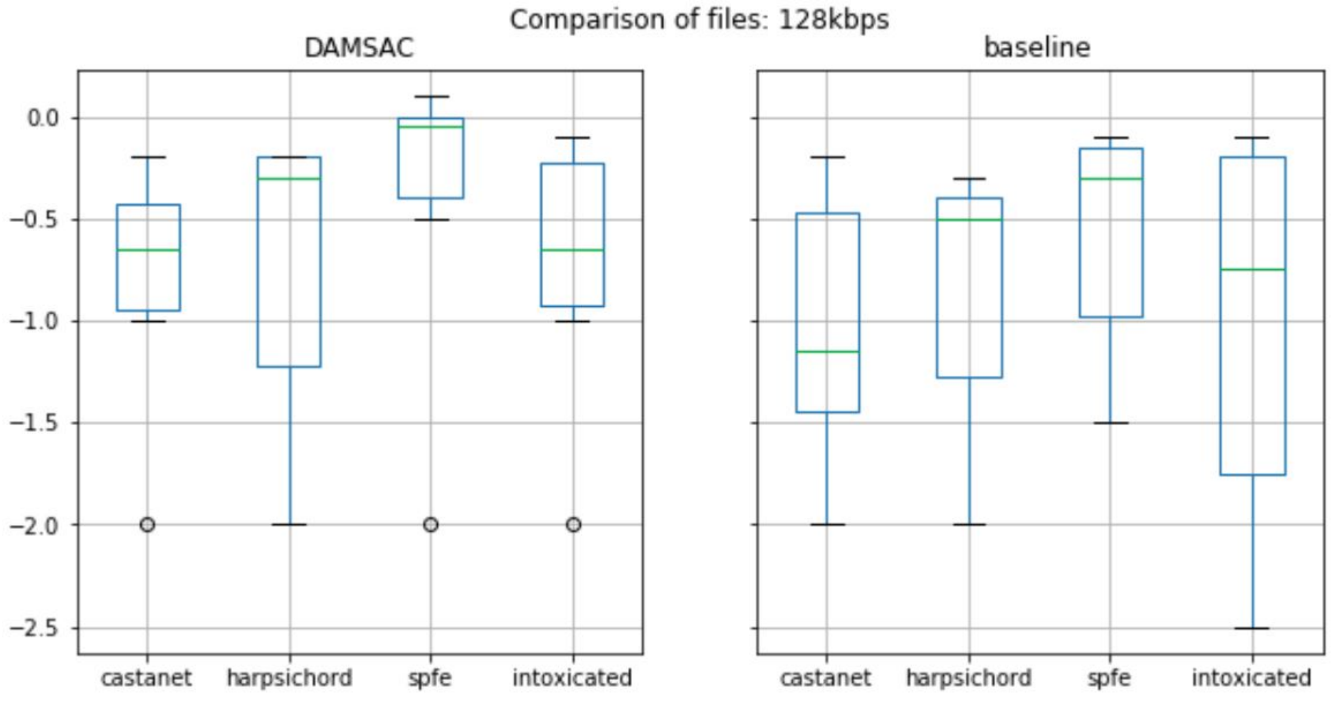

Figure 3: Boxplots grouped by file at 128kbps

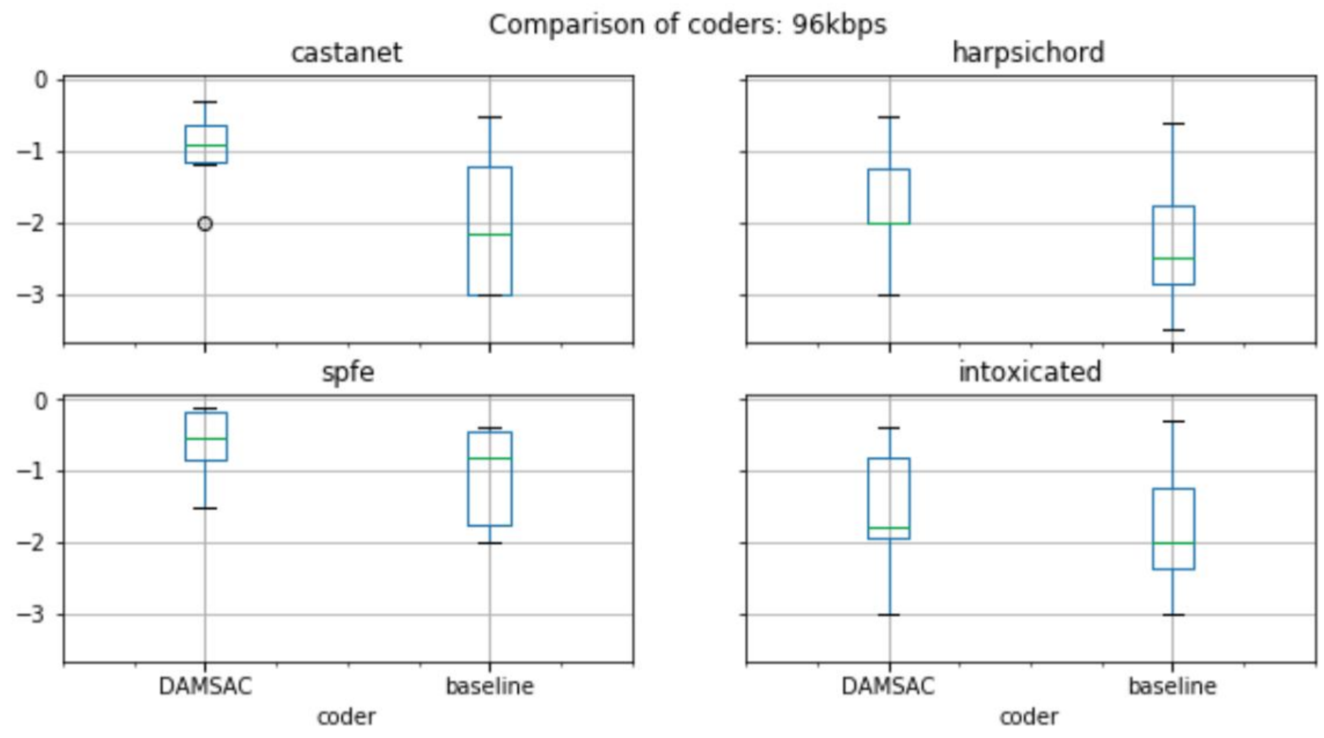

Figure 4: Boxplots grouped by coder at 96kbps

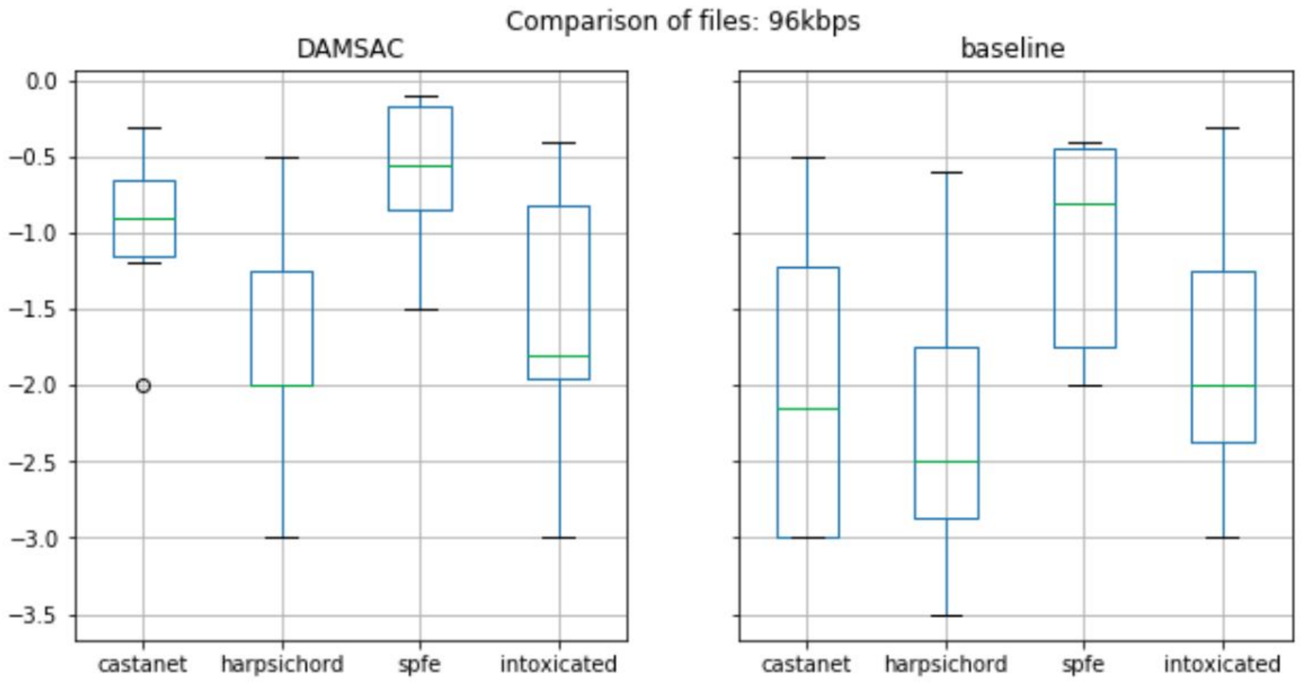

Figure 5: Boxplots grouped by file at 96kbps

Discussion

It can be inferred from our results that DAMSAC improves the perceived sound quality at 128kbps and also at a lower bit rate of 96kbps compared to our baseline coder. Additionally, the resulting compression ratios using using DAMSAC for each bit rate are close to the target ratio (5.51 for 128kbps and 7.35 for 96kbps).

By using a bit reservoir and tweaking the bit allocation algorithm, a significant amount of pre-echo is removed, which is most evident in the castanet audio file. At this point it should be emphasized that, while pre-echo naturally seems like a quantization error in time, the increased bit allocation in the MDCT frequency domain allows for an overall better reconstruction of the signal, decreasing frequency-domain quantization noise and thus also the block-size dependent pre-echo.

Due to the fact that bits are allocated to low frequencies first, lowering the bit rate to 96kbps audibly lowers the high frequencies, which makes the audio (particularly the castanets) sound slightly dampened. While this is a choice that might not fit every input signal, it is used in DAMSAC as it seems generally more acceptable to low pass the signal than to introduce more distortion in low and mid frequencies.

Due to the nature of the growing bit reservoir during quieter parts of the audio signal, the coder's bit reservoir would benefit from a larger maxBitBudget, if the audio signal was long enough and had relatively long quiet passages. Since, for the purpose of this report, only short audio files with a lot of continuous content were used, the maxBitBudget has been assigned a relatively low value. This way, the coder will distribute bits more generously after having reached this threshold, however in the future we might want to experiment further with this value. Depending on the application and content of the audio signal, this is one of DAMSAC's core advantages and can make a huge difference, if set appropriately for the right signals. Our coder also features MS coding. Based on the correlation formula, we identified when to use MS coding versus regular LR coding. Unfortunately, switching between MS and LR amongst blocks introduced clicks and pops in the audio. We believe this might be due to the speed in which the switching occurs in some instances - potentially switching every 4 milliseconds (512/128000). As a result, we resorted to using only MS coding, which ultimately still gives us a coding gain and avoids the clicks and pops in the audio.

The final step of our coder was to tweak certain parameters - threshold, drop, and maxBitBudget. Having a maxBitBudget of 100,000 bits for a bit rate of 128kbps sounds better for some signals (e.g. speech) as it saves more bits during quiet parts that can later on be allocated. However, we decided to use a maxBitBudget of 10,000 since we achieve a compression ratio closest to the desired one.
Generally, our coder does not work well for very compressed files such as pop music; for best performance, knowing what kind of audio file or genre is being encoded is advantageous.

Future Work

MS Coding

The rapid rate of switching between using LR and MS coding resulted in numerous clicks in the audio. In order to avoid such abrupt jumps when the reservoir runs out, a smoothing function could be implemented that less generously allocates the remaining bit budget as the bit budget runs low.

Huffman Coding

Huffman coding is an entropy coding method that achieves lossless data compression and is often used to compress video and audio formats. It provides this lossless compression by recovering the original symbol representation from Huffman tables, therefore maintaining the quality. Huffman coding can further lower the data rate or improve the sound quality at a fixed data rate by saving more bits to the bit reservoir and reallocating them to subsequent blocks.

With Huffman coding, bits can be efficiently allocated to represent symbols in the input data by exploiting the frequency of their occurrence, where symbols with high occurrence and therefore high probability being assigned fewer bits than the less common symbols. Since the Huffman code depends on the probabilities of each symbol, it gives coding gain when there are strong temporal correlations between consecutive amplitude values.

For the generation of codes, the first step is to calculate the frequency of input codes and produce Huffman tables, which we have done for one audio file (castanets). A portion of the table is shown below:

Table 3: Huffman Codes

Code	Probability (%)
0	29.3
1	7.6
10	7.5
11	2.2
100	8.3
101	1.5
110	0.9
111	0.6
1000	9.3
1001	2.5

Already, the most frequent code accounts for 29% of the occurrences, and the first 10 codes shown in the table make up the majority (65%) of all code occurrences. Since these frequently appearing codes can be represented by much fewer bits, there will be a significant coding gain.

References

[1] H. Fastl and Eberhard Zwicker, *Psychoacoustics: facts and models*, Number 22 in Springer series in information sciences. Springer, Berlin; New York, 3rd edition, 2007.

[2] Marina Bosi and Richard E. Goldberg, *Introduction to digital audio coding and standards*, Number SECS 721 in The Kluwer international series in engineering and computer science. Kluwer Academic Publishers, Boston, 2003.

[3] J.D. Johnston and Aníbal J.S. Feirreira, “Sum-difference stereo transform coding,” 1992.