Predictive Coding

João Branquinho
76543
University of Aveiro
joaobranquinho@ua.pt

Luís Silva
76585
University of Aveiro
lfssilva@ua.pt

Tiago Ramalho
76718
University of Aveiro
t.ramalho22@ua.pt

***Abstract*—Nowadays, the amount of data generated is tremendous. Compression Algorithms are ubiquitous and necessary to maintain an acceptable size of data stored in today's databases and servers spread around the world. Some approaches to the compression problem were already explored by the authors in the past, for instance, the lossy compression algorithms like uniform scalar quantization and vector quantization compression. However, lossy methods are not always an option. In this report we discuss such methods and other hybrid solutions.**

I. Introduction

In the previous work, we explored compression solutions based on lossy methods. Namely, we discuss Uniform Scalar Quantization and Vector Quantization. These methods being lossy result in the loss of information with the process of encoding and decoding. Data that is compressed using lossy techniques generally cannot be recovered or reconstructed exactly. However, this methods enable higher compression rates. There is a tradeoff between size and quality.

On the other side of the spectrum, there are the lossless techniques. With these techniques as the name imply, there is no loss in the data. These methods are used when the difference between original and reconstructed data. Examples of these applications are, for instance, medical records, in them, we cannot lose details because they might be important for a diagnostic. When we are dealing with music production environments, a music studio cannot store their artists' recordings with a lossy compression method, as it will compromise the quality of the song. These methods, however, will never attain a very high compression rate. Again, there is a tradeoff between size and quality.

First, we will talk about the *Golomb Code*, this code is very useful in lossless compression, we will discuss how to calculate it and why is useful in this application.

Next, we discuss our lossless audio codec, the CAVLAC. We discuss how it was implemented, the architectural choices we made, why we make them, our difficulties and the final results.

We also developed a lossy option, based on the residual quantization. We will also discuss our methods and our process of implementing it.

We end with a conclusion where we confront results and give our opinions.

II. Golomb Code

The purpose of using a code is to minimize the average length of the messages encoding more frequent symbols with fewer bits.The Golomb code is a prefix code meaning that once a certain bit pattern has been assigned as the code of a symbol, no other codes should start with that pattern.

Next, we will see how to encode numbers using the *Golomb code*. Let's begin with the assumption that the data we are trying to encode are integers and that the higher the integer the lowest is the probability of appearing.

For this type of data distribution, the simplest code to use is the unary code in which for the integer N the code is N ones followed by one zero.

However, the Golomb codes take this one step further. The Golomb Code is a family of codes with a parameter $m > 0$ and where we represent an integer $n > 0$ using two numbers q and r where:

$$q = \left\lfloor \frac{n}{m} \right\rfloor \tag{1}$$

and

$$r = n - qm \tag{2}$$

Resuming, in a Golomb code there is:

n is the number to be coded
m is a predefined value to divide the number n into two parts.
q is the first part, coded in unary, and is the quotient of the division (equation 1)
r is the second part, coded in binary, and is the rest of the division (equation 2)

A class was created that deals with the coding and decoding of numbers to and from Golomb.

A. Encode

When we start developing the Golomb encoder we found some constraints that he had to overcome. The first challenge that we faced was developing a class that also enabled the encoding of signed integers. As stated above, Golomb was developed to code positive integers.

The solution found was coding the negative numbers on the odd numbers and the positive numbers on pair numbers, like the snippet code under this text.

```
if(number >= 0)
    new_number = number * 2;
else
    new_number = -2*number-1;
```

There was another problem encountered, the writing of the remainder. If m was a power of two the process is straightforward we just write the binary representation.However, if m is not a power of two we have two options we set $b = \lceil \log_2(M) \rceil$ and if $r < 2^(b) - M$ we code r as plain binary using $b - 1$ bits. If $r >= 2b - M$ we code the number $r + 2b - M$ in plain binary representation using b bits.

Finally our class used the class used in the previous work to write and read bitwise to files.

III. Predictive Coding

When we talk about predictive coding we use a predictive model to predict the values being used and then only encode the difference between the predicted value and the real one.

Next, is discussed the creation of a lossless codec and after that the creation of a lossy one, that is also based on predictive coding.

A. Lossless Codec

Lossless compression basic principle is to remove the redundancy of the data. Our approach consists of 3 main sections:

- Framing - Dividing the raw input into frames that will be taken care separately
- Intra/Inter Channel Decorrelation - This is the process of removing the redundancy
- Entropy Coding - After the redundancy is removed we have no other option than to encode the differences
- Decompression - Restoring the original data

Next, we will explain our approach to the development of these sections.

1) Framing: By framing, we mean the process of dividing the original data into blocks and analyze them independently. The entire original audio is divided into frames of fixed size specify at encode time. Each of this frames starts with a header that will give the decoder some metadata about himself and how the decoder should decode its contents.

This number (size of the blocks) can be any integer, however, there should be some consideration when choosing one. Choosing a very small number will increase the overhead of the header that is appended to the beginning of the block. Choosing a very high number will compromise the performance of the predictor, making it more difficult to find a suitable and efficient predictor. A compromise must be found. From the literature analyzed there is a consensus about using a multiple of 192. This number came to be because of the size of the frames in AES/EBU, standards of audio data transmission.[1] Effects of the block size in the compression ration can be seen in I

In our solution, the block size is chosen by the user at encode time. Our frame is composed of a header and the residuals (more on this later).

The header has the metadata needed for the decoding of the frame and occupies 8 bits.

[1]FLAC (Free Lossless Audio Codec) also uses this approach, of choosing the frame size in multiples of 192.

TABLE I
Effect of the block size in Compression Ratios

Sample	192	576	1152	2304	4608
Sample 1	1,42	1,43	1,43	1,43	1,42
Sample 2	1,52	1.53	1.53	1.52	1.51
Sample 5	2,03	2,05	2,05	2,05	2,05

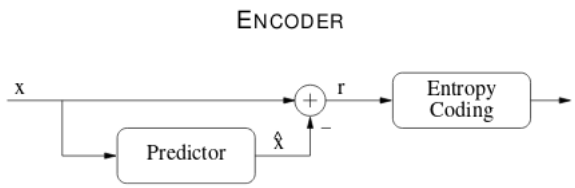

Fig. 1. Schematic of the Lossless Encoder

It has the following fields:

```
2 bits - Constant frame or not
2 bits - The predictor used
4 bits - M of the Golomb Code
```

Each field will be explained ahead when we explain their necessity.

2) Intra/Inter Channel Decorrelation and Entropy Coding: This section is arguably the most important section in the development of the lossless codec. The basic process is, use a predictor model (hence the name predictor coding) to predict the value of the next sample. And then encode the difference, called residual, between the predicted value and the real one. With this in mind, two important things to explain is how this prediction model works and how are the residuals encoded. Let's start with the prediction model.

The predictor method used was a simple polynomial predictor based on this four equation:

$$\begin{aligned} \hat{x_n}^{(0)} &= 0 \\ \hat{x_n}^{(1)} &= x_{n-1} \\ \hat{x_n}^{(2)} &= 2x_{n-1} - x_{n-2} \\ \hat{x_n}^{(3)} &= 3x_{n-1} - 3x_{n-2} - x_{n-3} \end{aligned} \tag{3}$$

The use of this polynomials with these fixed coefficients is purely based on Literature. There is an argument that the use of this simple predictors is good because they don't cause much overhead in the computation. We don't lose time creating the optimal polynomial that fits the Block neither choosing optimal coefficients. This because in other works there is an inclination to believe that a perfect fit is not very important neither is attainable in a useful time. So the argument is that it is better to store the residuals efficiently rather than loose much time finding the perfect predictor.

These polynomials also have a very interesting property. They allow us to compute the residuals very efficiently without

the need of computing the predicted values. The resulting residuals can be calculated in the following way:

$$\begin{aligned} \hat{r_n}^{(0)} &= x_n \\ \hat{r_n}^{(1)} &= r_n^{(0)} - r_{n-1}^{(0)} \\ \hat{r_n}^{(2)} &= r_n^{(1)} - r_{n-1}^{(1)} \\ \hat{r_n}^{(3)} &= r_n^{(2)} - r_{n-1}^{(2)} \end{aligned} \tag{4}$$

In our solution we use the following process. The block is encoded by first computing the residuals generated with all the predictors. We develop a function that recursively calculates the residuals of predictors up to any order. Based on the fact that these residuals are easily calculated the process of calculating the residuals that would be generated by all of the predictors is not very costly and allow us to choose the best for every block (since every block is different and has different properties). So, as previously said, since we have all the residuals of every possible predictor we can choose the best predictor to use. To explain this we need to take a step back and explain another important part of this section, the coding of the residuals.

So a block of encoded data is composed by the Frame Header (as we said previously) and by the residuals. These residuals are encoded using the Golomb Code.

After understanding this we can easily see how the best predictor is chosen. The best predictor is the one that will allow us to save more space in the encoding of the residuals, therefore, because we use the Golomb code to encode them, the best predictor is the one that generates the lowest residuals (it's very intuitive, the best predictor is the one that is more close to the real result). The way we choose it is by calculating the average of the absolute value of the residuals. We use the absolute value because although the residuals can be negative when they are encoded into Golomb they will turn positive. About this is important to talk about is the calculation of the M for the Golomb Encoding. In [2] it is proposed a method to make an estimative for the M taking into consideration the average of the absolute value of the residuals, which luckily we already calculate.

$$\begin{aligned} m &= 2^k \\ k &= \lceil \log_2 E(|\hat{r_n}|) \rceil \end{aligned} \tag{5}$$

So far we talk about how we do the Intra Channel Decorrelation, using a predictive model based on simple polynomials, and the Entropy encoding, encoding the residuals with Golomb codes. However, we didn't discuss how we handle different channels. And the Inter Channel Correlation. We consider, backed with literature that the correlation between channel is very weak in most samples so, the way we handle stereo channels is by encoding the left channel in one block and then the differences to the right channel in another block.

Before closing this section there is one last feature in our approach to take into consideration. So far we assumed that the block samples varied in time. But what if the block was constant? In that case, the predictor of order 1 would be perfect

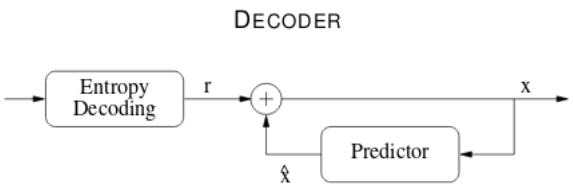

Fig. 2. Schematic of the Lossless Decoder

and we would be only writing zeroes in the residual values. We handle this by having an if clause for when the frame is constant [2] and then we only write the value that is going to be repeated block size, saving the space we would otherwise waste writing zeroes.

Closing out this section lets review the frame header and its values and how they come to play into the new information presented. First, the 2 bits that say if the frame is constant or not. the next 2 bits are there to tell the decoder wich one of the predictors was used. Finally, the M with 4 bits tells the decoder which M was used for the Golomb code.

3) Decoding a CAVLAC file: On the other side of the process, there is the decode. The decoder job is very simple, it will simply read the encoded file. Decipher the headers and act accordingly. The headers are easily read because they have a fixed size, both the initial one of the file containing

```
number of frames - 32 bits
sample rate - 32 bits
channels - 16 bits
format - 32 bits
block size - 16 bits
```

and the ones that are appended to the beginning of the frames.

In them there is ,basically, everything that is needed to reconstruct the file like the original one. In the block frame there is also the seeds for the predictor used, so the process is very simple to understand. We basically predict the value that would be using the predictor indicated in the header and then, we add the residual difference so that we can have the original value again.

IV. Lossy Codec

The main difference between this codec and the previous one is how the predictor is fed. In the previous one, we fed the predictor with the real values, in this one we must feed it we reconstructed values. The flow is the following:

- Generate the residual between the real value and the value predicted.
- Take this residual and dispose of the least significant bits.
- Take this new residual and reconstruct the value of the sample
- Feed this value to the predictor.

[2]The way we check if a frame is constant is by checking if the residuals average for the predictor 1 is 0

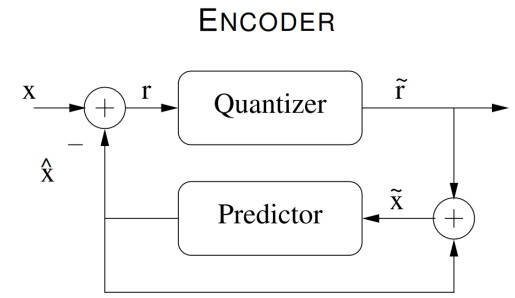

Fig. 3. Schematic of the Lossy Encoder

This method is almost a hybrid one, taking advantage of both forms of encoding, lossless and lossy. Decoding this file will result in a reconstruction of the file with the "Reconstructed values". Values that are not exactly the original ones, but have an associated error.

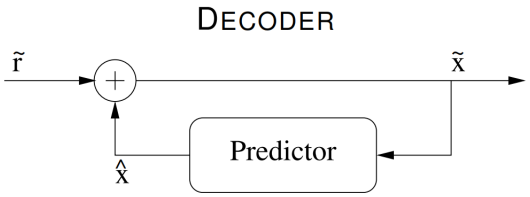

Fig. 4. Schematic of the Lossy Decoder

The rest of the process is very similar to the one previously discussed.

V. Results

TABLE II
CODING AND DECODING TIMES LOSSLESS

Sample	Coding (s)	Decoding (s)
Sample 1	0,285	1,378
Sample 2	0,144	0,697
Sample 5	0,183	0,918

TABLE III
SNR AND COMPRESSION RATION LOSSY CODING

Sample	Quantization Bits	SNR	Compression Ratio
Sample 1	2	70,07	1,64
Sample 1	4	58,51	2,07
Sample 1	8	34,46	4,28
Sample 1	10	22,42	7,06
Sample 2	2	66,51	1,84
Sample 2	4	54,87	2,38
Sample 2	8	6,14	5,07
Sample 2	10	5,19	7,08

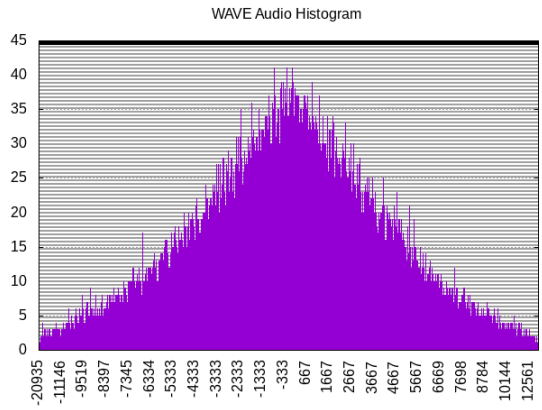

Fig. 5. Histogram of Sample 2 Mono Version

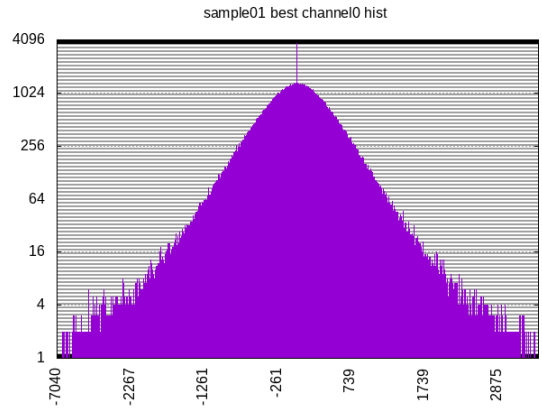

Fig. 6. Sample 1 best predictors channel 0

A. Histograms

The group decided to present two simple histograms:

- Figure 5 shows the sample count of the mono version of sample 1.
- Figure 6 show the residual count of sample 1. In this histogram, only the best predictor for each block is used. The block size used is 1152.

The two histograms have a similar key property: the distribution of the samples follows a normal curve around the point 0. The main difference between the two is the scale of the y-axis:

- The mono version sample count fluctuates between 0 and 42.
- The residual count varies between 0 and 1100 (a logarithmic scale is used to represent it).

A big residual count of small values (around 0 either negative or positive) allows better compression rates when using an entropy coder like Golomb.

VI. Conclusion

In this study, we learn more about predictive coding, lossless and lossy codecs. In our opinion, we achieved great results.

Based on Table II the encode and the decode time are good and expected to be in that order of magnitude. As we already told, the obtained compression ratio needs to be carefully chosen for not increase a header overhead or for not compromise the performance of the predictor. In table three we measured SNR and compression ratio of the lossy coding, concerning the sample01, regardless of the number of quantization bits, we observed that the SNR was always satisfying. As expected the compression ration increased if the number of quantization bits increases, and in our opinion compression ratio of 2 two or higher is great. About the sample02 we expected the same good results, however, with eight and ten quantization bits the SNR was a bit off. We think that there might be a bug in our lossy codec. The audio starts with a strange bip but it is quite audible therefore. With this study we can conclude that when lossy compression is not viable for our application we still have the option off lossless coding, however we are trading quality with size of the file.

REFERENCES

[1] Mat Hans, Ronald W. Schafer* *Lossless Compression of Digital Audio* HP Laboratories Palo Alto, HPL-1999-144, November, 1999

[2] M. J. Weinberger, G. Seroussi and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS," in IEEE Transactions on Image Processing, vol. 9, no. 8, pp. 1309-1324, Aug. 2000. doi: 10.1109/83.855427