# Assignment 2 - Audio and Video Coding

Pedro Alagoa [72534] Gonçalo Grilo Alexandre [72608]

## I. Summary

This report describes the implementation and design of the audio encoder and decoder developed for the class. The purpose of the tool is to extract statistical properties of any audio file, such as the entropy, as well as encoding a new version of the file using less bits while maintaining the same data (lossless encoding).

## II. Running instructions

We have developed 4 separate tools **AudioEntropy**, **FakeAudioCodec**, **TheRealAudioCodec** and **Lossy**.

**AudioEntropy** is used to calculate the entropy of a audio file and generate histograms. It can be used as follows:

- Read input from the file *sample01.wav* and generate a histogram.

```
$./AudioEntropy ../samples/sample01.
    wav
$gnuplot plot.gp
$gwenview histogram.png
```

**FakeAudioCodec** is used to simulate/test the encoder and check the final size of the encoded file. It saves a file with the calculated values. It also does the reverse process of getting the original file from the *fake encoded*. It is used as follows:

- Read input file from sample01.wav, *fake encode* it and *fake decode* it with m=512.

```
$./FakeAudioCodec ../samples/
    sample01.wav 512
```

**TheRealAudioCodec** is the real implementation of our lossless codec. It can encode 16 bits *wav* files in different modes:

- Read input from the file *sample01.wav* and encode it using a order 2 predictor and with a blocksize of 4096 samples.

```
$./TheRealAudioCodec -e -i ../
    samples/sample01.wav -k 2 -b
    4096
```

- Read input from the file *sample01.wav* and encode it without blocks.

```
$./TheRealAudioCodec -e -i ../
    samples/sample01.wav -b 0
```

- Read input from the file *out.cod* and decode it without blocks.

```
$./TheRealAudioCodec -d -i out.cod -
    b 0
```

- Read input from the file *out.cod* and decode it with blocks.

```
$./TheRealAudioCodec -d -i out.cod
```

**Lossy** is a implementation of the lossy codec. It can encode 16 bits *wav* in the same way **TheRealAudioCodec** does, and its usage is identical, but it loses information in the encoding-decoding process.
There are other options that can be selected when using the tool.

Listing 1: Usage of the tool

```
Allowed options:
  -h [ --help ]                   produce
      help message
  -i [ --input ] arg              input file
  -k [ --order ] arg (=1)         set order
      of the model in block
      encoding
  -c [ --inter-channel ] arg (=1) inter-
      channel decorrelation, always
      on (for now)
  -o [ --output ] arg (=out)      output
      file
  -b [ --blocks ] arg (=4096)     size of
      the blocks, if less than 64
      it will not use blocks. When decoding
      use -b 0 to decode without blocks (we
      forgot this in the encoded header)
  -e [ --encode ] [=arg(=1)]      encode
  -d [ --decode ] [=arg(=1)]      decode
```

## III. External libraries

This tool uses the **Program options** library of the Boost C++ Libraries [1].

## IV. Practical work

### A. FakeAudioCodec

#### A.1 Description

As referred above,the **FakeAudioCodec** is used to simulate/test the encoder and check the final size of the encoded file. It saves a file with the calculated values. It also does the reverse process of getting the original file from the encoded file.

The method of predicting and encoding is simpler than the **TheRealAudioCodec**, because there were several improvements made to the used algorithms used in **TheRealAudioCodec**. Thus, the **FakeAudioCodec** uses an early version of the algorithms we use. The description of the final version of the algorithms and its evolution to their final state is described in IV-B.

Like stated above, the main purpose of the FakeAudioCodec is to estimate the final size of the encoded file. This is done by, adding the number of bits the quotient(**q**) and the remainder((m)) of each sample will have, in the Golomb coding phase. The size of meta-data such as headers is ignored in this calculation.

### B. *TheRealAudioCodec*

#### B.1 *Description*

For the **TheRealAudioCodec** we optimized **FakeAudioCodec** and write/read the results of encoding to a binary file. To be encoded a file goes through some stages:

- AudioReader
- Predictor
- Golomb
- Bitstream

**AudioReader** will take a *wav* file (**AudioReader** doesn't completely support all formats of *wav*) and read it's data and meta-data to memory.

This data is sent to **Predictor** where we will try to lower the data entropy. The first filter we apply is a inter-channel decorrelator where the first channel and the difference of the first and later channels are coded. From here the transformed data will pass a linear predictor.

If the block option is chosen the data is split in blocks of the define size and the filter is applied to each block independently with an order defined by the user.

In case the block option is disabled the filter will be applied to each channel and the order is the one that generates **residuals** with less entropy (using an optimized version of **AudioEntropy** to calculate entropy's).
Below we can see a comparison of the histograms generated for the original data and the residuals calculated from that data, for *sample01.wav*.

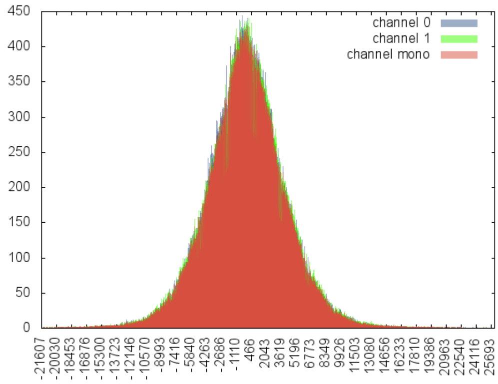


Fig. 1: Original - **Entropy** = 14.0032

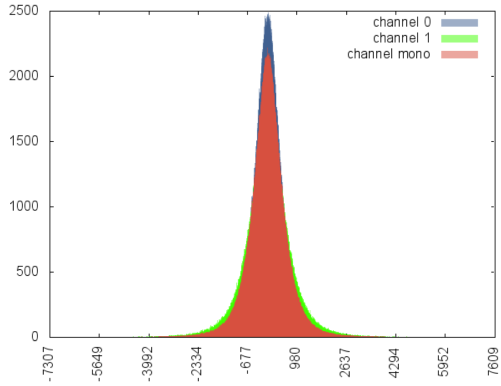


Fig. 2: Residuals - **Entropy** = 11.3429

As we can see above, the entropy of the second histogram is lower than the other, which can be inferred by the "smoother" histogram.

The next stage is the **Golomb** encoder. If the data is divided by blocks we will try to find the best m for each block. If we do not have blocks we try to find the best m for the whole data. This is where the blocking will gain advantage from the blockless solution.

The algorithm to chose the best m will simply calculate the size of the encoded data for all possible values of m and chose the lower one. We only allow values for m that are powers of 2 and only calculate values until 32768.

The last step is the **Bitstream** where the encoded data will be written to a file. In the first bits of this file are placed the struct SF_INFO that contains the number of frames, the sample rate, number of channels and other important info to rebuild the wav or play the file.

If the encoded data is divided by blocks the size of the blocks is written after SF_INFO. The blocks are written to the file, and before each block there is a small header with the used predictor order, the used *m* and the size of the block.
The order in this header is not currently used (because we use the same order for every block) but

in the future we intend to choose the best order for each block and save that value in that position.
The size field could also be optimized away as it is only used to stop reading at the end of the channels when the last block has less samples. But keeping the size field allow us to make blocks with different sizes in the future.

To ensure that the original file and the decoded file had, in fact, the same content, the **md5** of both files was calculted and compared. This was done with a *bash script* called *test_samples.sh* that was sent along with the code.

Below there is a sample output of the script:

```
05229df2eaac96458a23e5a80c7db9f2  d_s1_b_o1.
    wav
05229df2eaac96458a23e5a80c7db9f2  ../samples
    /sample01.wav
502b0b6460d5663ca309ee383be87fd3  d_s2_b_o1.
    wav
502b0b6460d5663ca309ee383be87fd3  ../samples
    /sample02.wav
51c3c77f8979254d33d36542f7d51dbb  d_s3_b_o1.
    wav
51c3c77f8979254d33d36542f7d51dbb  ../samples
    /sample03.wav
970cb81c1c9746a97013a9245c3a67e6  d_s4_b_o1.
    wav
970cb81c1c9746a97013a9245c3a67e6  ../samples
    /sample04.wav
14d3068f0098a2055f21ca29adcf248a  d_s5_b_o1.
    wav
14d3068f0098a2055f21ca29adcf248a  ../samples
    /sample05.wav
e0008652ef304a281e13bbc0cae970f9  d_s6_b_o1.
    wav
e0008652ef304a281e13bbc0cae970f9  ../samples
    /sample06.wav
759190478cedbf2b3418fcffcf76b1d6  d_s7_b_o1.
    wav
759190478cedbf2b3418fcffcf76b1d6  ../samples
    /sample07.wav
```

### C. *Lossy*

#### C.1 *Description*

Our lossy version behaves exactly like our lossless version except in two phases:

- **Encoding phase** - Before passing the data to the **Predictor**, we divide the values of the samples by a constant (e.g. 64).
- **Decoding phase** - After the **Predictor** is done with its reverse process, we multiply the values of each sample by the same constant used in the encoding phase.

This way, the compression ratio is much higher than the **TheRealAudioCodec**, although there is loss of some information. The constant we use (64) seemed to be the optimal trade-off between the encoded size and the perception of quality of the resultant decoded file.

## V. Benchmarks and Results

To make the tests, we compared the results of 7 different wav files.

- **sample01.wav** - Size: 5,0M Length: 00:29
- **sample02.wav** - Size: 2,5M Length: 00:14
- **sample03.wav** - Size: 3,4M Length: 00:20
- **sample04.wav** - Size: 2,3M Length: 00:13
- **sample05.wav** - Size: 3,5M Length: 00:20
- **sample06.wav** - Size: 4,1M Length: 00:24
- **sample07.wav** - Size: 3,6M Length: 00:21

A table with all of the results and charts can be found in the Appendix.

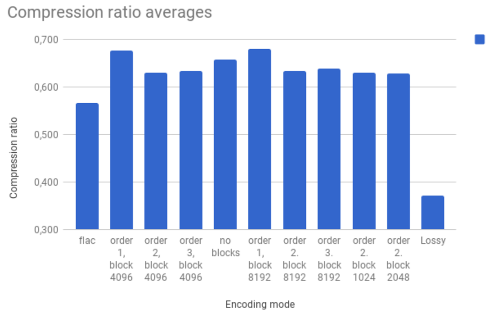


Fig. 3: Chart of the averages compression ratios of each mode

By analyzing the results, we came to the following conclusions:

- The best encoding mode is different for each file.
- Our algorithm can not beat FLAC in compression ratio nor in encoding speed. But our compression ratio is very similar.
- The encoding mode that has a better average compression ratio is the one with **order 2** and **block size of 2048 samples**.
- Generally, the best order to use is order 2.
- In some cases it is better not to use blocks at all, especially if the block size is large.
- We think that our algorithm, despite being slower than FLAC (which has been optimized throughout the years), has a good encoding and decoding speed.
- The lossy encoder can massively reduce the size of a file such that the loss of audio quality is not easily perceptible.

## VI. Conclusions

We are not totally satisfied with our codec. We think that with a few improvements it could definitely equal or even surpass FLAC. This could be achieved by several methods:

- Using a more effective predictive model.
- Having a specific coding for silence in the file, so that the number of bits describing silence in the encoded file is smaller.

- Estimate the final size of each block with the optimal entropy before passing the block through the Predictor and the Golomb.

Despite this, we are convinced that our encoding and decoding process is fast, though it could benefit with the usage of threads to compute each block in parallel.

Regarding the Lossy codec were impressed with how easily such a simple method of losing information (dividing the original values by a constant) can have such a great impact in reducing the file size without losing much audio quality in the process.

## REFERENCES

[1] David Abrahams Beman Dawes. Boost C++ Libraries. http://www.boost.org/. [Accessed: 14 October 2017].

# Appendix A

# Results

| | File | Size (bytes) | Ratio | Encoding time | Average ratio |
|---|---|---|---|---|---|
| | sample01.wav | 5176796 | 1,000 | 0 | |
| | sample02.wav | 2589596 | 1,000 | 0 | |
| | sample03.wav | 3530396 | 1,000 | 0 | |
| **Original** | sample04.wav | 2354396 | 1,000 | 0 | **1,000** |
| | sample05.wav | 3647996 | 1,000 | 0 | |
| | sample06.wav | 4235996 | 1,000 | 0 | |
| | sample07.wav | 3765596 | 1,000 | 0 | |
| | sample01.wav | 3441745 | 0,665 | 0m0,097s | |
| | sample02.wav | 1626399 | 0,628 | 0m0,212s | |
| | sample03.wav | 1975962 | 0,560 | 0m0,106s | |
| **FLAC** | sample04.wav | 1349914 | 0,573 | 0m0,220s | **0,566** |
| | sample05.wav | 1701825 | 0,467 | 0m0,122s | |
| | sample06.wav | 1434308 | 0,339 | 0m0,268s | |
| | sample07.wav | 2750679 | 0,730 | 0m0,260s | |
| | sample01.wav | 3669206 | 0,703 | 0m0,522s | |
| | sample02.wav | 1820638 | 0,703 | 0m0,286s | |
| | sample03.wav | 2197112 | 0,622 | 0m0,319s | |
| **No blocks** | sample04.wav | 1497826 | 0,636 | 0m0,239s | **0,657** |
| | sample05.wav | 1921249 | 0,527 | 0m0,357s | |
| | sample06.wav | 1759689 | 0,415 | 0m0,322s | |
| | sample07.wav | 2969781 | 0,789 | 0m0,475s | |
| | sample01.wav | 3734129 | 0,721 | 0m0,431s | |
| | sample02.wav | 1864614 | 0,720 | 0m0,246s | |
| | sample03.wav | 2315759 | 0,656 | 0m0,279s | |
| **Order 1, block 4096** | sample04.wav | 1682595 | 0,715 | 0m0,210s | **0,677** |
| | sample05.wav | 2148205 | 0,589 | 0m0,250s | |
| | sample06.wav | 2242284 | 0,529 | 0m0,264s | |
| | sample07.wav | 3040026 | 0,807 | 0m0,350s | |
| | sample01.wav | 3640081 | 0,703 | 0m0,537s | |
| | sample02.wav | 1837338 | 0,710 | 0m0,221s | |
| | sample03.wav | 2215912 | 0,628 | 0m0,252s | |
| **Order 2, block 4096** | sample04.wav | 1494701 | 0,635 | 0m0,200s | **0,629** |
| | sample05.wav | 1918096 | 0,526 | 0m0,239s | |
| | sample06.wav | 1747178 | 0,412 | 0m0,233s | |
| | sample07.wav | 2979162 | 0,791 | 0m0,345s | |
| | sample01.wav | 3748732 | 0,724 | 0m0,434s | |
| | sample02.wav | 1906308 | 0,736 | 0m0,225s | |
| | sample03.wav | 2245208 | 0,636 | 0m0,273s | |
| **Order 3, block 4096** | sample04.wav | 1440889 | 0,612 | 0m0,185s | **0,633** |
| | sample05.wav | 1910435 | 0,524 | 0m0,244s | |
| | sample06.wav | 1690475 | 0,399 | 0m0,232s | |
| | sample07.wav | 3025498 | 0,803 | 0m0,383s | |

Table A.1: Results - Part 1

| | File | Size (bytes) | Ratio | Encoding time | Average ratio |
|---|---|---|---|---|---|
| | sample01.wav | 3737472 | 0,722 | 0m0,408s | |
| | sample02.wav | 1884346 | 0,728 | 0m0,212s | |
| | sample03.wav | 2331365 | 0,660 | 0m0,261s | |
| **Order 1, block 8192** | sample04.wav | 1682389 | 0,715 | 0m0,193s | **0,679** |
| | sample05.wav | 2153798 | 0,590 | 0m0,252s | |
| | sample06.wav | 2244187 | 0,530 | 0m0,275s | |
| | sample07.wav | 3052578 | 0,811 | 0m0,350s | |
| | sample01.wav | 3646127 | 0,704 | 0m0,422s | |
| | sample02.wav | 1864176 | 0,720 | 0m0,209s | |
| | sample03.wav | 2242289 | 0,635 | 0m0,274s | |
| **Order 2, block 8192** | sample04.wav | 1493712 | 0,634 | 0m0,191s | **0,633** |
| | sample05.wav | 1922125 | 0,527 | 0m0,249s | |
| | sample06.wav | 1742725 | 0,411 | 0m0,237s | |
| | sample07.wav | 3003011 | 0,797 | 0m0,359s | |
| | sample01.wav | 3757810 | 0,726 | 0m0,442s | |
| | sample02.wav | 1936015 | 0,748 | 0m0,227s | |
| | sample03.wav | 2282296 | 0,646 | 0m0,272s | |
| **Order 3, block 8192** | sample04.wav | 1438623 | 0,611 | 0m0,175s | **0,638** |
| | sample05.wav | 1913075 | 0,524 | 0m0,246s | |
| | sample06.wav | 1678938 | 0,396 | 0m0,235s | |
| | sample07.wav | 3057971 | 0,812 | 0m0,359s | |
| | sample01.wav | 3645167 | 0,704 | 0m0,437s | |
| | sample02.wav | 1816255 | 0,701 | 0m0,221s | |
| | sample03.wav | 2194245 | 0,622 | 0m0,265s | |
| **Order 2, block 1024** | sample04.wav | 1504082 | 0,639 | 0m0,198s | **0,629** |
| | sample05.wav | 1929965 | 0,529 | 0m0,250s | |
| | sample06.wav | 1785265 | 0,421 | 0m0,238s | |
| | sample07.wav | 2970829 | 0,789 | 0m0,347s | |
| | sample01.wav | 3638730 | 0,703 | 0m0,447s | |
| | sample02.wav | 1816255 | 0,703 | 0m0,211s | |
| | sample03.wav | 2194245 | 0,622 | 0m0,245s | |
| **Order 2, block 2048** | sample04.wav | 1504082 | 0,636 | 0m0,178s | **0,628** |
| | sample05.wav | 1929965 | 0,527 | 0m0,256s | |
| | sample06.wav | 1785265 | 0,415 | 0m0,238s | |
| | sample07.wav | 2970829 | 0,789 | 0m0,343s | |
| | sample01.wav | 2114953 | 0,409 | 0m0,440s | |
| | sample02.wav | 1047539 | 0,405 | 0m0,202s | |
| | sample03.wav | 1223049 | 0,346 | 0m0,246s | |
| **Lossy** | sample04.wav | 960913 | 0,408 | 0m0,175s | **0,372** |
| | sample05.wav | 1119881 | 0,307 | 0m0,254s | |
| | sample06.wav | 988260 | 0,233 | 0m0,243s | |
| | sample07.wav | 1861620 | 0,494 | 0m0,322 | |

Table A.2: Results - Part 2

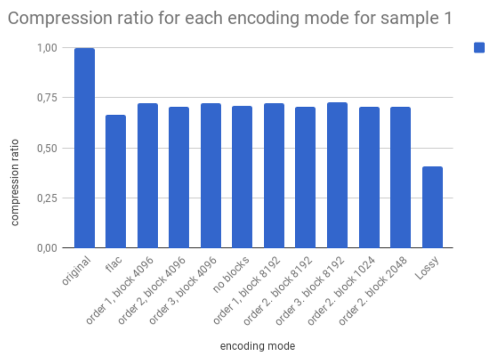


Figure A.1: Sample 1 chart

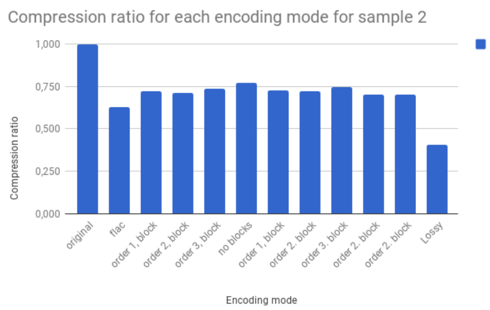


Figure A.2: Sample 2 chart

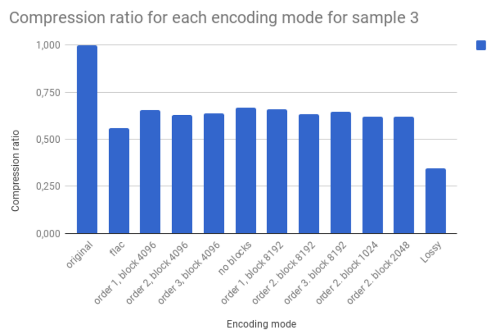


Figure A.3: Sample 3 chart

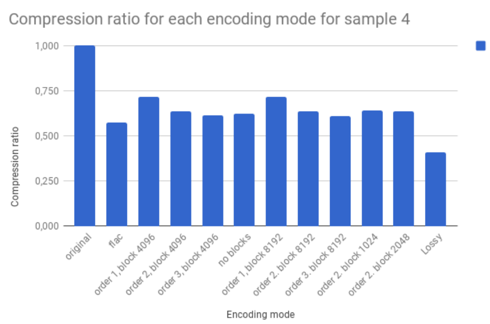


Figure A.4: Sample 4 chart

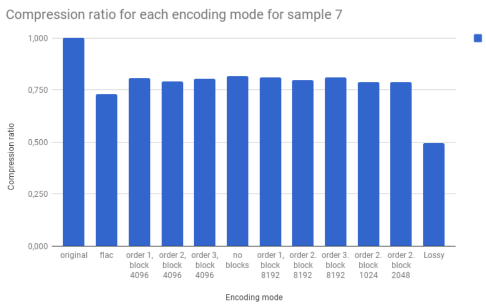


Figure A.5: Sample 7 chart