

Digital Audio Coding

Lab Session 2 – SHRINK

Sébastien Boisgérault,
Mines-ParisTech.

Nov. 9, 2012



This work is licensed under a [Creative Commons Attribution 3.0 Unported License \(CC BY 3.0\)](https://creativecommons.org/licenses/by-sa/3.0/). You are free to **share** – to copy, distribute and transmit the work – and to **remix** – to adapt the work – under the condition that the work is properly attributed to its author.

About SHRINK

SHRINK is an simple audio file format that relies on lossless audio compression algorithms and hence is similar to [FLAC](#), [ALAC](#) or [SHORTEN](#).

The command-line program **shrink** implements a SHRINK *codec* (coder/decoder pair): the command

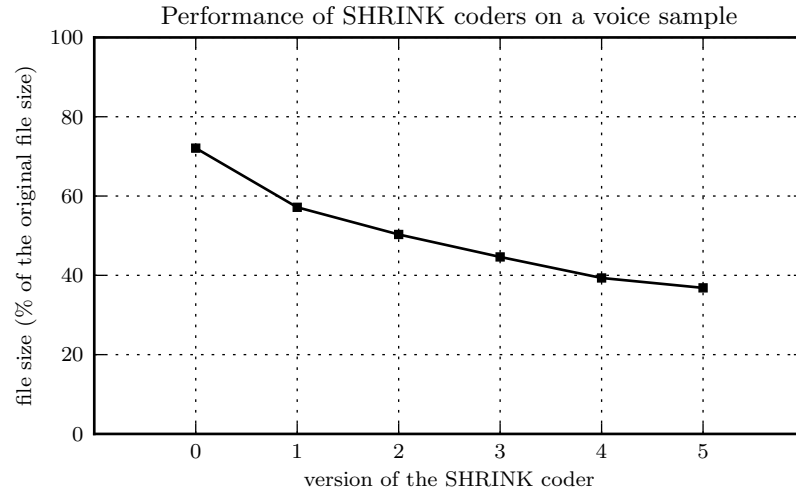
```
$ shrink sound.wav
```

turns the uncompressed WAVE file "**sound.wav**" into a "**sound.shk**" file (only 44.1 kHz, 16-bit linear PCM content is supported) and conversely the command

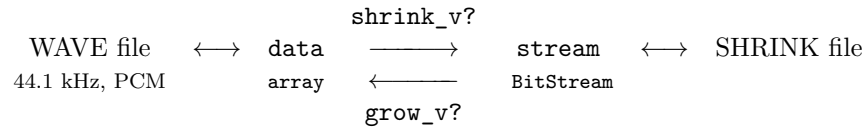
```
$ shrink sound.shk
```

uncompresses the SHRINK file "**sound.shk**" into "**sound.wav**". The command **shrink --help** provides more options.

The **shrink** program is implemented on top of a Python module also named **shrink**. The SHRINK format actually gathers different codecs versions – numbered 0 to 5 – of increasing complexity and performance.



Each version of the codec is defined by two functions: a coder `shrink_v?` and a decoder `grow_v?` where `?` stands for the version number; these two functions transform `data` – a numpy array of type `int16` – into `stream` – a bitstream – and reciprocally, according to the following diagram:



The aim of the lab session is to study the algorithms behind the design of SHRINK and to implement the main features of this coder.

SHRINK Format Structure

The high-level structure of the SHRINK file format is given by:

Size	Unit	Content
89	bit	header
?	bit	channel 1
?	bit	channel 2
< 8	bit	zero-padding

where the content of the header is:

Size	Unit	Name	Type	Range/Value
6	byte	magic	ASCII, big endian	"SHRINK"
1	byte	version	unsigned integer	[0, ..., 5]
4	byte	length	unsigned integer, big endian	number of samples (per channel)
1	bit	stereo	boolean	True or False

The format of the channel fields is version-dependent.

1. What is the purpose of the zero-padding in this context ?
2. Generate a SHRINK bitstream with no channel data (for example with version 0 and a single channel). Make sure that the function `grow_v0` from the Python module `shrink` can successfully decode this stream.

Amplitude Rice Coder

For each channel, the version 0 of the SHRINK compression algorithm:

- determines an appropriate value of the Golomb parameter,
- encodes it in the channel field as a 16-bit big-endian unsigned integer,
- encodes the values of the channel array with this instance of Rice coder.

Size	Unit	Content	Type
8	bit	Golomb parameter	unsigned integer
?	bit	sample values	Rice encoded data

Table 1: SHRINK version 0 channel format.

1. What is maximal value of the Golomb parameter that it makes sense to consider for 16-bit signed integer data ? Are 8 bits enough to store the Golomb parameter ?
2. What does a Golomb parameter of 0 correspond to ? Compute a worst-case estimate of the size of the channel data of **A4**, coded with a Golomb parameters of 0.

3. Use the heuristic provided in the `rice` coder to compute the “best” value of the Golomb parameter p . Check the size of the coded data for Golomb parameters between $p - 3$ and $p + 3$.
4. Implement the coder function `shrink_v0`. Test it on the data from "A4.wav". Did we achieve any compression ? Why ? Generate an audio data signal more likely to be effectively compressed by `shrink_v0` and test this hypothesis.

Simple Prediction Coders

Convention: finite sequences of numeric values $(x_0, x_1, \dots, x_{N-1})$ – represented as one-dimensional arrays of length N – are assimilated to infinite sequences indexed on \mathbb{Z} , denoted $(x_n)_n$, with a value of 0 outside of the initial index range.

The difference operator Δ on sequences is defined by

$$\Delta(x_n)_n = (x_n)_n - (x_{n-1})_n$$

1. Implement the restriction of Δ to arrays as a function `D`. Make sure that the length of the output array is the same as the input array. Is the operator Δ invertible ? What about the function `D` ?

Implement version 1 of the SHRINK coder `shrink_v1` that encodes the channel data as:

Size	Unit	Content	Type
8	bit	Golomb parameter	unsigned integer
?	bit	sample difference	Rice encoded data

Table 2: SHRINK version 1 channel format.

Study the reduction of average “size” of the difference values with respect to the original sample values and the corresponding reduction of the Golomb parameter. Compute the compression rate achieved on the file "A4.wav".

2. A *predictive coder* encodes the *prediction residual* $e_n = x_n - \hat{x}_n$ where \hat{x}_n is a prediction of x_n based on the past values of this sequence. Show that version 0 and 1 of SHRINK are simple predictive coders. What is their prediction of x_n ?
3. Implement version 2 of the SHRINK coder `shrink_v2` that replaces the previous predictor schemes with a linear extrapolation of x_n based on x_{n-1} and x_{n-2} .

Size	Unit	Content	Type
8	bit	Golomb parameter	unsigned integer
?	bit	linear extrap. residual	Rice encoded data

Table 3: SHRINK version 2 channel format.

Study the reduction of average “size” of the new residual and the corresponding reduction of the Golomb parameter. Compute the new compression rate achieved on the file "A4.wav".

Adaptative Polynomial Prediction Coder

For any $n \in \mathbb{Z}$ and any numbers $x_{n-m-1}, x_{n-m}, \dots, x_{n-1}$, there is a single polynomial P_n whose order is at most m such that:

$$P_n(n-m-1) = x_{n-m-1}, P_n(n-m) = x_{n-m}, \dots, P_n(n-1) = x_{n-1}.$$

The *polynomial prediction of order m* maps a sequence $(x_n)_n$ to the sequence $(\hat{x}_n)_n$ defined by $\hat{x}_n = P_n(n)$.

1. Can you describe versions 1 and 2 of the SHRINK coders in terms of polynomial prediction ? Show for a given prediction order m , there is a sequence of numbers (a_0, a_1, \dots, a_m) such that for any $n \in \mathbb{Z}$:

$$\hat{x}_n = a_0 + a_1 x_{n-1} + \dots + a_m x_{n-m-1}$$

2. Prove that the prediction residual $(e_n)_n = (x_n)_n - (\hat{x}_n)_n$ associated to the prediction of order m can be expressed simply with the difference operator. Prove that the prediction residual has only integer values when the initial sequence has only integer values, and that the values to prediction residual mapping may be inverted.
3. Implement a function that given a signal computes its average absolute value, then compute the average absolute value of the polynomial prediction of order 0, 1, etc. until this average value stops to decrease, in order to determine the optimal prediction order and the corresponding residual. We take as a convention that no prediction (direct encoding of the signal values) corresponds to a prediction order of -1 .

Implement version 3 of the coder `shrink_v3`, with the following channel format:

Size	Unit	Content	Type
?	bit	prediction order + 1	Rice encoded data.
			Golomb param. 3, unsigned.
8	bit	Golomb parameter p	unsigned integer
?	bit	prediction residual	Rice encoded data.
			Golomb param. p , signed.

Table 4: SHRINK version 3 channel format.

Compute the new compression rate achieved on the file "A4.wav".

Framed Prediction

The properties of an audio signal may change completely in different sections of its time frame but he still probably has some local consistency. We may therefore search for the best predictor locally instead of globally and expect higher compression ratios.

1. Implement the `shrink_v4` coder that splits a signal into frames of 20 ms, and applies the channel data coder from the version 3 to each frame separately.
2. This method has the drawback to include a “cold restart” of the algorithm for each new frame: every frame analysis is done as if the previous values of the signal were all zeros and the first predicted values are therefore probably very inaccurate. Modify the previous coder into `shrink_v5` to solve that problem.
3. Study the performance of version 4 and 5 of the coders with respect to the version 3 on smooth synthetic signals, such as A4, and on more realistic signals, such as music or voice signals.