pp. 21 – 39

CHAPTER 2

On Lossless Compression of 1-bit Audio Signals

Franziska Bittner, Dee Denteneer, Simon Kronemeijer, Misja Nuyens, Simona Orzan, Jacques Rougemont, Evgeny Verbitskiy, Dmitri Znamenski.

Abstract. In this paper we consider the problem of lossless compression of 1-bit audio signals. We study the properties of the existing solution proposed in [**5, 6**]. We also discuss possible improvements. Other methods have been cosidered, and the results are reported.

Keywords:Audio compression, linear prediction, Markov predictors, boosting, machine learning.

1. Introduction

Lossless compression of audio signals is an active area of research, which already has a wide range of practical applications such as Compact Disks (CD's) and Digital Versatile Disks (DVD's). The problem posed to the Study Group by the Philips Research Laboratories comprised two different goals. Firstly, Philips is interested in the highest possible compression ratio which can be achieved without, more or less, any restriction to the complexity of algorithms, computing power required, etc. Thus the first part of the problem has a predominantly theoretical flavor. It should indicate the limits of compression techniques for audio data in the form of binary sequences. The second part of the problem is more practical. It concerns the evaluation of the current compression technique described in [**5, 6**]. An interesting question here is whether the proposed algorithm is optimal in some sense, and whether any improvements are possible.

There is one important practical aspect to keep in mind. If a new superior compression algorithm is proposed within the class of models, currently accepted and implemented as hardware (*encoders*/*decoders*), and one would like to implement this algorithm, then only a relatively small number of *encoders* should be upgraded. The 'old' *decoders* (such as CD players) would still be capable of reproducing the audio signal.

Philips supplied us with 4 generic audio sequences and with the corresponding compression ratios achieved by their coding techniques. We will refer to these samples as samples A, B, C and D. The first sample A is considered to be 'easy', the last sequence D is considered to be 'difficult'. The

remaining sequences B and C are of average complexity. The efficiency of the compression algorithm is measured by its *gain* R,

$$R = \frac{\text{length of the original sequence in bits}}{\text{length of the compressed sequence in bits}}.$$

For reference, the Philips algorithm achieves compression gains of around 3 for sample A, 2.4–2.6 for samples B and C respectively, and 2.2 for sample D.

This paper is organised in the following way. In section 2, we describe a general approach to data compression based on statistical modelling. We also describe in more detail the algorithm proposed by Philips, and formulate the criterion for the optimal predictor within the class of linear predictors. In section 3 we discuss the efficiency of the current linear predictor. We discuss ways of improving Markov predictors in section 4. In section 5, we propose ways of improving the current linear predictor. Especially schemes based on weighted least squares seem to be very promising.

2. Statistical modelling and prediction

A large number of compression methods is known and used in practice. These include among others well-known algorithms such as Lempel–Ziv, Huffman, or arithmetic coding. We will refer to this type of compression techniques as entropy coding. Given a binary sequence of length N, an efficient entropy coder will produce an output of approximately Nh bits, where h is the *entropy* given by

$$h = -p \log_2 p - (1-p) \log_2 (1-p),$$

where p is the probability of observing a 0 in the source sequence. From this we can see that binary sequences in which one symbol occurs more often than the other – 0 say, and hence $p \gg 1-p$ – will be compressed efficiently.

It is known, however, that applying entropy coding to audio signals is not very efficient due to the presence of long-time correlations in the signal. These should be exploited in order to gain efficiency. Therefore, a preprocessing step is required, which eliminates the statistical dependencies, and leads to an almost uncorrelated source with one dominating symbol. Then standard entropy coding techniques can successfully be applied.

Suppose $\{x_i\}$, $i = 1, \ldots, N$, is a binary sequence we want to compress. A typical approach to the development of such a preprocessing stage would be the design of a scheme that tries to predict the next symbol of the sequence based on k previous symbols

$$\hat{x}_n = F(x_{n-1}, \ldots, x_{n-k}). \tag{1}$$

Next we define a new sequence $\{y_i\}$ as follows. If the prediction is successful, i.e. $x_n = \hat{x}_n$, then we let $y_n = 0$, otherwise $y_n = 1$. It is also clear that

given the parameters of the predictor F, the first k bits $(x_1, \dots, x_k)$, and the values $\{y_i\}$, $i = k+1, \dots, N$, we can reconstruct the original sequence, $\{x_i\}$, $i = k+1, \dots, N$. In practice, we would like to have N much larger than k.

If we would be able to design a predictor F which makes only very few mistakes, then 0 will be a dominating symbol in the sequence $\{y_i\}$, and we should expect a good compression gain for this sequence by entropy coding. At the same time, the decoder must be supplied with the parameters of the predictor F. We will refer to the storage space in bits, required for these parameters, as the *overhead*. Clearly, the overhead should not be too large. The gain of such a coding scheme is

$$R = \frac{\text{length of the original sequence}}{\text{overhead} + k + (N-k)h_y},$$

where h_y is the entropy of the sequence $\{y_i\}$. Hence, in this setup, the problem of efficient audio compression is reduced to the design of a prediction scheme with a minimal value of overhead $+ Nh_y$. Here we used our assumption that k is much smaller than N.

When designing the prediction scheme for a sequence $\{x_i\}$, $i = 1, \dots, N$, we are allowed to use the whole sequence. For example, even an uncompressed piece of music of just a few minutes long takes up several GigaBits of storage space. Hence, N can be quite large. On the other hand, we would like to be able to start decoding (playing music) from a more or less arbitrary position. So in practice predictors are designed for much shorter sequences (frames) of length $N \approx 40000$. In this way possible problems arising from the non-stationarity of the audio signal are avoided; we can expect a single predictor to perform reasonably well on the whole frame.

2.1. Markov predictors. Let $k \in \mathbb{N}$ be the length of words on which the prediction will be based. The state space of our Markov chain is the set $\mathcal{S}^{(k)} = \{-1, 1\}^k \approx \{1, \dots, 2^k\}$ of words of length k on a two-symbol alphabet[1].

The Markov chain will make transitions between a word $\pi = x_1 \cdots x_k$ and the words $\pi^{\pm} = x_2 \cdots x_k k_{k+1}$ for $x_{k+1} = \pm 1$. Figure 1 shows the graph of the Markov chain for $k = 3$.

Each edge of the graph will be assigned the empirical transition probability measured from the data set. Namely for each word $\pi = x_1 \cdots x_k$, we compute

$$\begin{aligned} U^{(k)}(\pi) &= \text{number of times the word } x_1 \cdots x_k, +1 \text{ is found in the data ,} \\ D^{(k)}(\pi) &= \text{number of times the word } x_1 \cdots x_k, -1 \text{ is found in the data .} \end{aligned}$$

[1]In this paper we are dealing with binary data. Everywhere below we will be using equivalent representations of the data in alphabets {0,1} and {-1,1} freely, without announcing it explicitly. In every case, however, it will be clear which representation we are using.

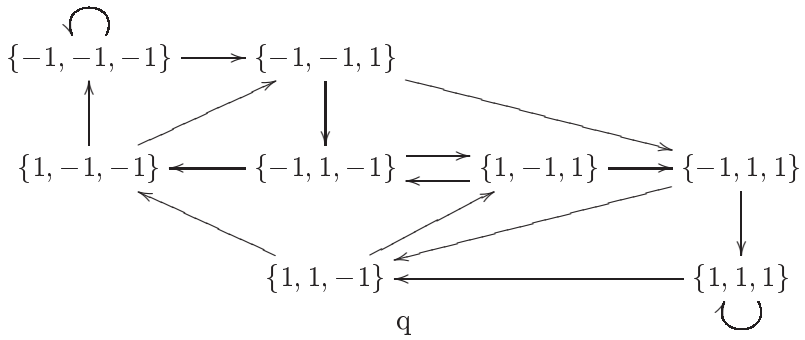


FIGURE 1. Graph of the Markov chain on $\mathcal{S}^{(3)}$

The empirical transition probabilities are given by

$$\begin{aligned} \mathbf{P}^{(k)}\big(\pi \to \pi^+\big) &= \frac{U^{(k)}(\pi)}{U^{(k)}(\pi) + D^{(k)}(\pi)} \,, \\ \mathbf{P}^{(k)}\big(\pi \to \pi^-\big) &= 1 - \mathbf{P}^{(k)}\big(\pi \to \pi^+\big) \,. \end{aligned} \tag{2}$$

These probabilities are not needed in practice, we only provide them for a complete definition of the Markov chain.

The Markov predictor of order k, $\widehat{x}_n = p^{(k)}(\pi)$ on the word $\pi = x_{n-k} \cdots x_{n-1}$ is defined by

$$p^{(k)}(\pi) = \begin{cases} +1 \,, & \text{if } U^{(k)}(\pi) \geq D^{(k)}(\pi) \,, \\ -1 \,, & \text{if } U^{(k)}(\pi) < D^{(k)}(\pi) \,. \end{cases} \tag{3}$$

Hence, in this way we predict the most probable symbol to follow π.

The overhead of the Markov predictor is 2^k bits. Hence, practical application of Markov predictors is feasible only for relatively small k's.

2.2. Linear predictors. Linear prediction is amongst the most successful tools in signal processing. In data compression, schemes based on linear prediction show good compression gains for various data types, e.g. speech. Linear predictors, in general, are trying to predict the next observation by a linear combination of several previous values:

$$\hat{x}_n = \beta_0 + \beta_1 x_{n-1} + \beta_2 x_{n-2} + \ldots + \beta_k x_{n-k},$$

Since we are dealing with binary data, $x_n \in \{-1, 1\}$, a natural way to incorporate this into our predictor is to consider the following predictors

$$\hat{x}_n = \operatorname{sign}\big(\beta_0 + \beta_1 x_{n-1} + \beta_2 x_{n-2} + \ldots + \beta_k x_{n-k}\big),$$

where $\operatorname{sign}(x) = 1$ for $x \geq 0$, and -1 otherwise.

The design of an optimal linear predictor hence consists of the selection of a vector of parameters $\beta = (\beta_0, \ldots, \beta_k) \in \mathbb{R}^{k+1}$ in a such way that the number of instances where $x_n \neq \hat{x}_n$ is minimal. The overhead of a linear

predictor is $(k+1)M$, where M is the number of bits used to represent a real number. Hence, the overhead is growing only linearly with k. This is a great advantage of linear predictors over Markov predictors, and allows them to go to much higher values of k. It is also clear that for the same order k, Markov predictors are at least as good as linear predictors.

2.2.1. *Philips predictor.* In [**5, 6**], a linear predictor of the following form has been used

$$\hat{x}_n = \operatorname{sign}\bigl(\beta_1 x_{n-1} + \beta_2 x_{n-2} + \ldots + \beta_k x_{n-k}\bigr), \tag{4}$$

where the coefficients $\beta_1, \ldots, \beta_k$ have been selected to minimize the following expression

$$\sum_{n=k+1}^{N} \bigl|x_n - (\beta_1 x_{n-1} + \beta_2 x_{n-2} + \ldots + \beta_k x_{n-k})\bigr|^2 \to \min. \tag{5}$$

The problem (5) is a standard least squares problem, which admits an efficient practical solution. This simple approach produces a predictor of a remarkable quality. An optimal value of the order k for such a linear predictor has also been investigated in [**5, 6**]. In fact, optimal k may vary and depends on a particular frame, but typically $k = 128$ gives good results. Increasing k does lead to a better predictor, but the corresponding growth of the overhead is not compensated by the gain in the quality of the prediction.

2.2.2. *Optimal linear predictor.* In fact, solving problem (5) gives only an approximation of the *optimal* linear predictor. An optimal predictor minimizes the number of errors, i.e. instances when $x_n \neq \hat{x}_n$. We can reformulate the criterion for the optimal predictor (4) in the following way: coefficients $(\beta_1, \ldots, \beta_n)$ of the optimal linear predictor are such that in the system of inequalities

$$\begin{array}{cc} \beta_1 x_{k+1} x_k + \beta_2 x_{k+1} x_{k-1} + \ \ldots \ + \beta_k x_{k+1} x_1 & > 0 \\ \vdots & \vdots \\ \beta_1 x_N x_{N-1} + \beta_2 x_N x_{N-2} \ + \ \ldots \ + \beta_k x_N x_{N-k} & > 0 \end{array}$$

the number of valid inequalities is *maximal*.

Let us denote by A the matrix with elements $x_n x_{n-j}$, $n = k+1, \ldots, N$, $j = 1, \ldots, k$. Hence the problem of finding the optimal linear predictor is equivalent to finding a vector β such that a vector $A\beta$ has a maximal number of positive coordinates.

In general, for a given matrix A it is quite easy to check whether there exists a vector β such that *all* the coordinates of $A\beta$ are positive. If such a vector exists, then we say that A defines a *feasible* system of inequalities. This is a best possible case, because the corresponding linear predictor will not make a single error. To check whether A is indeed feasible, we can use

standard methods of Linear Programming, which give an answer in polynomial time. However, one should not expect that the matrix A obtained from the data will lead to a feasible system of inequalities. The problem of finding the optimal linear predictor then is equivalent to the problem of finding the maximal feasible subsystem, i.e. a matrix A', made out of rows of A, which has maximal dimension and still gives a feasible system of inequalities. This problem is known to be NP hard, but also there is no polynomial approximation, see [**1, 2**]. However, there are several heuristic methods, which are known to perform relatively well. Unfortunately, we were not able to pursue this idea further. In the next section however, we will discuss briefly how close the linear predictor, given by (5), is to the optimal linear predictor.

3. Comparing predictors

In the previous section we have seen that the Philips predictor is, in principle, different from the optimal linear predictor. Nevertheless, the Philips predictor performs remarkably well: on average, 1 error for every 10 predictions made. This suggests that maybe the Philips predictor is not that far away from the optimal one. However, as was mentioned above, design of an optimal predictor is a known NP hard problem. On the other hand, we can try to compare Philips predictor with a Markov predictor.

For every pattern, the linear predictor makes at least as many mistakes as the Markov predictor. If it makes more, these extra errors are a measure for the quality of our predictor. We can calculate these numbers for our data. We should not make k too large; then almost every pattern would only occur once at most, and the Markov predictor would make no errors. This is not what we want; for each pattern, we should have a reasonable number of occurrences, or none.

It is useful to develop some terminology for these errors: we say a k-bit linear predictor makes a 'Type I' (or unavoidable) error if it makes an error, but agrees with the Markov prediction. In that case, the predictor can not be improved by repairing this error. For example, if our data contains a pattern of k bits, which appears twice, once followed by 0 and 1 at the second instance, then the Markov (and hence, linear) predictor will make an error. If the predictor makes an error, while the Markov predictor is correct, we call this a 'Type II' (or avoidable) error. This error may be due to the linearity of the predictor, or it may be that we have not found the best linear predictor. We will not try to distinguish between these cases. If we find a type II error, we can, in principle, improve our predictor. However, this could mean we have to leave the class of linear models, which might not be a desirable solution from a practical point of view.

Predictor length	No. of errors	Unavoidable (type 1)	Avoidable (type 2)
7	6301	6301	0
30	5660	3070	2590
90	4328	0	4328
128	3764	0	3764

FIGURE 2. The Philips predictor applied to the first frame of sample A (frame size is 37632 bits)

Table 2 suggests that the Philips predictor is indeed optimal for small k. As expected, for large k the Markov predictor will not make any mistake. This, however, has no practical value. On the other hand, comparison between linear and Markov predictors is not really fair. It would be more interesting to define Type I and Type II errors for the class of linear predictors. For example, if our data contains $1, \ldots, 1$ and $-1, \ldots, -1$ (both k times), followed by 1, then any linear predictor is bound to make a mistake. The precise identification of avoidable/unavoidable mistakes seems to be out of reach at the moment.

4. Improving the Markov predictor

We have seen earlier that Markov predictors of high order have superior quality. At the same time, huge overhead (2^k bits) makes them unpractical. In this section we consider two attempts to produce predictors of comparable quality, but with smaller overheads.

4.1. Adaptive Markov Predictor. For each word π, the number of correct predictions will be $\max\big(U^{(k)}(\pi), D^{(k)}(\pi)\big)$, see (3). Consequently the total number of incorrect predictions is

$$\sum_{\pi\in\mathcal{S}^{(k)}} e(\pi) \;=\; \sum_{\pi\in\mathcal{S}^{(k)}} \min\big(U^{(k)}(\pi), D^{(k)}(\pi)\big)\ .$$

As shown in Figures 3 and 4, most of those errors are typically due to just a few words. An adaptive Markov scheme can be designed in the following way:

: 1) Construct the Markov predictor of order k (see (3)).

: 2) Sort the 2^k elements of $\mathcal{S}^{(k)}$ by the number of errors they account for, that is number the words $\pi(1), \pi(2), \ldots, \pi(2^k)$ in such a way that $e(\pi(1)) \geq e(\pi(2)) \geq \cdots \geq e(\pi(2^k))$.

: 3) Choose $k' > k$, $I \in \{1, \ldots, 2^k\}$ and compute the Markov predictor $p^{(k')}(\rho)$ for each word $\rho \in \mathcal{S}^{(k')}$ of the form $\rho = \pi_1\pi_2$, where π_2 is one of $\pi(1), \ldots, \pi(I)$.

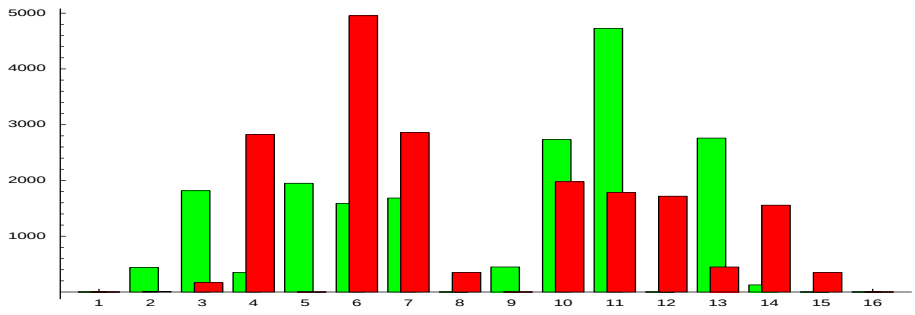


FIGURE 3. Word counts $U^{(4)}$ and $D^{(4)}$ computed for sample A

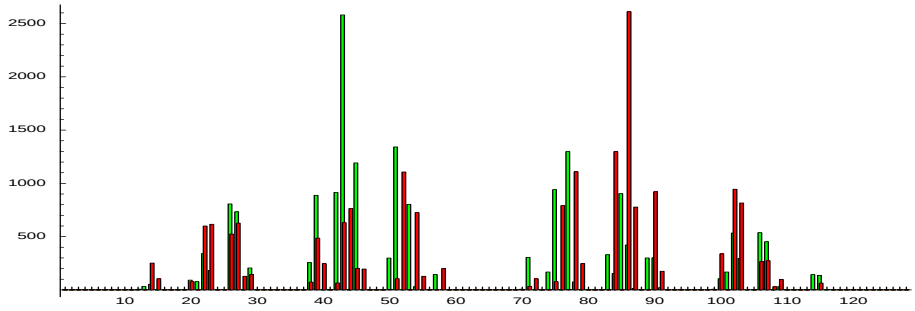


FIGURE 4. Word counts $U^{(7)}$ and $D^{(7)}$ computed for sample A

: 4) Define a predictor $\widehat{x}_n = p_{\mathrm{ad}}^{(k,k')}(\pi_1\pi_2)$, where $\pi_1 = x_{n-k'}\cdots x_{n-k-1}$ and $\pi_2 = x_{n-k}\cdots x_{n-1}$, as follows:

$$p_{\mathrm{ad}}^{(k,k')}(\pi_1\pi_2) = \begin{cases} p^{(k)}(\pi_2)\ , & \text{if } \pi_2 \in \{\pi(I+1),\ldots,\pi(2^k)\}\ , \\ p^{(k')}(\pi_1\pi_2)\ , & \text{if } \pi_2 \in \{\pi(1),\ldots,\pi(I)\}\ . \end{cases} \tag{6}$$

Equation (6) defines our adaptive predictor. The overhead for this is of the order of $2^k + 2^{k'-k}I$ bits.

Such an adaptive predictor has been constructed based on the first 37632 bits of sample A. Firstly, a Markov predictor of order $k = 4$ is constructed. The values of $U^{(4)}$ and $D^{(4)}$ for each word (numbered $1,\ldots,16$) are displayed in Figure 3.

We see that $I = 4$ words (numbers $6, 7, 10$ and 11) account for 86% of the errors (7034 out of 8137). A Markov predictor of order $k' = 8$ on those words can correct 1911 of those errors (a 23% reduction). Hence the adaptive Markov predictor $p_{\mathrm{ad}}^{(4,8)}$ makes 6226 mistakes. The overhead would be around 100 bits.

Starting from a predictor of order $k = 7$ (see Figure 4), and taking $I = 10$ (accounting for $4699/6217 \approx 76\%$ of all errors), a predictor of order

$k' = 14$ makes 3924 errors on those words (an overall reduction of 12%). The adaptive Markov predictor $p_{\text{ad}}^{(7,14)}$ thus makes 5442 errors, with an overhead of about 1400 bits.

4.2. Compressed Markov predictor. The main weakness of Markov predictors is the large size of the overhead. It might be feasible to try to 'compress' this overhead, by representing the Markov predictor in an equivalent, but more economic form. Our idea is to start with a boolean function which describes the Markov predictor and then to minimize its size using Quine's algorithm ([**8**]) for minimization of boolean functions.

Consider the Markov predictor given by (3). Let $\Pi_1^{(k)}$ be the set of words (patterns) of size k, such that Markov predictor predicts 1 on those patterns:

$$\Pi_1^{(k)} = \{\pi = (\pi_1, \dots, \pi_k) \in \{0,1\}^k : p^{(k)}(\pi) = 1\}.$$

Then the following boolean function gives an equivalent representation of our Markov predictor

$$f(x_1, \dots, x_k) = \bigvee_{\pi \in \Pi_1} \bigwedge_{1 \le i \le k} (x_i = \pi_i).$$

Now we can try to minimize the size of f using the Quine minimization algorithm, which minimizes the number of boolean operators in a logical expression.

EXAMPLE 4.1. Suppose the source sequence is 011001 and $k = 2$. The Markov predictor is given by

$$f(x_0, x_1) = ((x_0 = 0) \wedge (x_1 = 1)) \vee ((x_0 = 0) \wedge (x_1 = 0))$$

This Markov predictor does not make any mistakes for our data. After minimization, we conclude that $f(x_0, x_1) = (x_0 = 0)$.

Some tests performed on the real date are summarized in the following table. Frame size was set to 37632.

Order k	No. errors (all are type 1)	Overhead (bits occupied by f)
2	8038	2
7	7917	98
10	7464	1160

Unfortunately, this method does not seem to lead to any substantial improvement of compression ratios.

5. Improving the linear predictor

A coding approach based upon the linear predictor was shown to be very successful, as compared to approaches based on other prediction schemes

such as the Markov predictor. Therefore, it makes sense to take the Philips-approach based on the linear predictor as a starting point and to try variations on this scheme. This will be the subject of this section. We consider the following variations. In subsection 5.2, we try linear predictors with coefficients chosen from a finite set, either $\{0,1\}$ or $\{-1,0,1\}$. In subsection 5.4, we change the optimization criterion to a weighted optimization criterion. In subsection 5.5, we investigate the effect of changing the prediction order k of the linear predictor. Finally, in subsection 5.6, we consider the effect of a lagged estimation scheme in which $\hat{\beta}$ is estimated from the previous bit string rather than from the current bit string. Firstly, let us discuss briefly how the binary data is obtained from an analog signal. An understanding of this transformation might lead to improvements of prediction schemes, see subsection 5.3 below.

5.1. Sigma-Delta Modulation. An audio signal – a relatively smooth function of time – is digitized using a *sigma-delta modulator* (SDM) (see Figures 5 and 6). So the additional information, which, in principle, can help improving the quality of the prediction is the following:

(1) The original signal $X(t)$ is an audio signal, and is a smooth function of time.
(2) The SDM is, theoretically, a deterministic function of $X(t)$, and it produces the same binary sequences as an output, given the same audio signal as an input. But in practice, some noise is always present in the working circuit of an SDM. This noise can occasionally invert an output bit of the SDM.

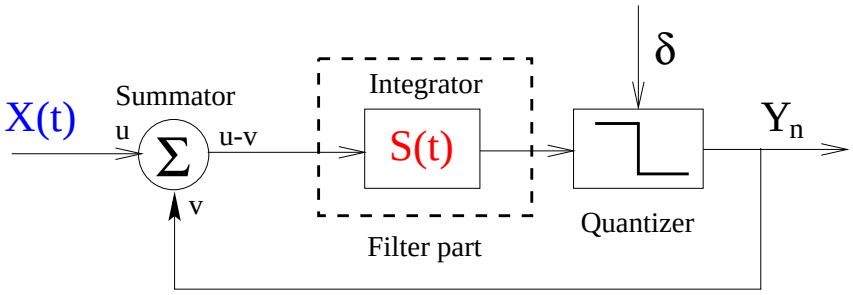


FIGURE 5. A primitive 1-st order SDM with one integrator. The input $S(t)$ to the quantizer is the integral of the difference between the original signal $X(t)$ and Y_n, the output of the quantizer.

It follows from (1), that if some frequency f is essential in the spectrum of $X(t)$, then $X(t)$ and $X(t+1/f)$ are highly correlated. Since $1/f$ contains $1/(\delta f)$ quanta of time, there are dependencies in the ouput binary stream (x_n) at distance $k = 1/(\delta f)$. For example, the frequency $f = 1000$ Hz

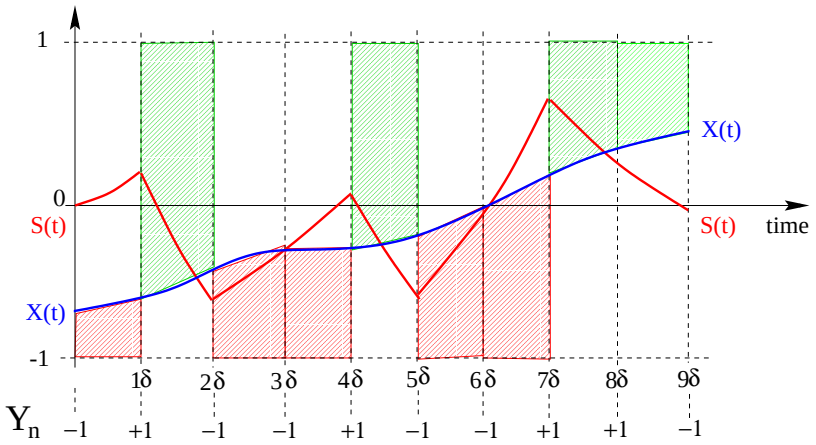


FIGURE 6. A primitive 1-st order SDM. The integrated difference $S(t)$ as a function of the original signal $X(t)$.

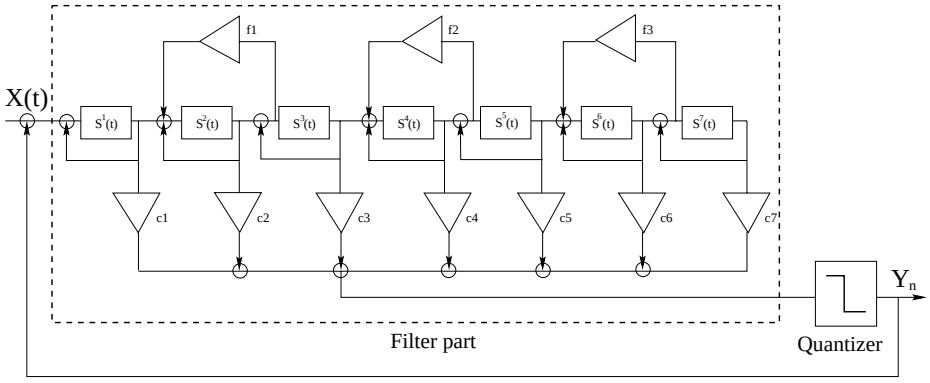


FIGURE 7. A practical implementation of a 7-th order SDM in [**7**]

corresponds to dependencies at distance $k = 44.1 * 64 = 2822.4$ quanta of time, if one quantum of time $\delta = (44100 * 64)^{-1}$ seconds. This suggests that a predictor of length 2000–3000 should be used. In practice however, predictors of much smaller order k are used. Again, long predictors are not efficient due to a large overhead. It is nevertheless possible to use longer predictors, provided we use coefficients that can be stored using only a few bits: for example, binary ($\beta_i \in \{-1, 1\}$) or ternary ($\beta_i \in \{-1, 0, 1\}$).

5.2. Linear predictors of low pression. We have seen that finding the optimal linear predictor is equivalent to finding a vector $\beta = (\beta_1, \ldots, \beta_k) \in \mathbb{R}^k$ such that the scalar product $\beta \cdot Y_n$ is strictly bigger than 0 as often as possible, where $Y_n = (x_n x_{n-1}, \ldots, x_n x_{n-k}) \in \{-1, 1\}^k$. In other words, we

need to find a hyperplane containing 0 in $\mathbb{R}^k$ such that as many Y_n's as possible lie on the same side.

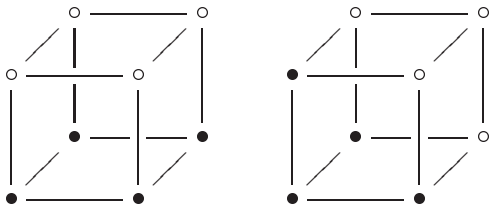
This problem can theoretically be solved. Moreover, the fact that all the Y_n lie on the cube $\{-1,1\}^k$ should help. It looks as if the only β's one needs to try are the ones which have entries in $\{-1,0,1\}$, where one should exclude the β's such that the number of nonzero entries is even, to avoid that $\beta \cdot Y_n = 0$ for some Y_n.

A 'linear' predictor for us is a special function

$$\{-1,1\}^k \longrightarrow \{-1,1\}$$

which takes the value 1 on one side of a hyperplane and the value -1 on the other. In particular it maps half of the elements to 1 and half of them to -1.

For $k = 3$, up to symmetry we get the following possibilities:

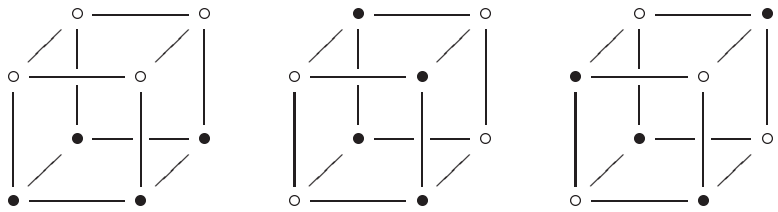


Here we denote points which are mapped to 1 by $\circ$ and points which are mapped to -1 by $\bullet$. There are 6 'linear' predictors of the first type (as the cube has 6 faces) and 8 'linear' predictors of the second type (corresponding to the 8 vertices of the cube).

Functions $\{-1,1\}^k \longrightarrow \{-1,1\}$ are the same as functions $\{0,1\}^k \longrightarrow \{0,1\}$. Regarding $\{0,1\}$ as the field $\mathbb{F}$ with two elements we can consider special functions of the form

$$\mathbb{F}^k \longrightarrow \mathbb{F}, \qquad (y_1, \ldots, y_k) \mapsto \sum \beta_i y_i + c,$$

where $\beta = (\beta_1, \ldots, \beta_k) \in \mathbb{F}^k$ and $c \in \mathbb{F}$. We assume that β is not the zero vector, such that half of the elements is mapped to 0 and half of the elements is mapped to 1. As multiplication and addition in $\mathbb{F}$ are just AND and XOR (where 1 corresponds to TRUE and 0 corresponds to FALSE) these functions are easy to handle on a computer. The class of functions one gets is different from the class of 'linear' predictors. Again we draw the case of $k = 3$:



Now we denote points which are mapped to 0 by ∘ and points which are mapped to 1 by •. We get 6 functions of the first and second kind and 2 of the third kind. Here the first kind corresponds to a β containing two zero entries, the second kind to a β containing one zero entry, and the third one to $b = (1, 1, 1)$.

The predictors on $\mathbb{F} = \mathbb{Z}_2$. Let for the moment $x_n \in \{0, 1\}$ and consider predictors

$$\hat{x}_n = \beta_0 + \sum_{i=1}^{k} \beta_i x_{n-i} \ (\mod 2), \quad \beta_i \in \mathbb{F}.$$

We compared, for every $k \leq 12$, all 2^{k+1} predictors and selected the optimal one. The minimal percentage of errors in the frames varies from 9% to 22%, and is equal to $\sim 17.5\%$ on average, independent of the sample.

The ternary predictors. Here we return to $x_n \in \{-1, 1\}$ and to predictors of the form

$$\hat{x}_n = \operatorname{sign}\Big(\beta_0 + \sum_{i=1}^{k} \beta_i x_{n-i}\Big),$$

where $\beta_i \in \{-1, 0, 1\}$. We have used the fast gradient method to minimize the number of errors. The percentage of errors p equals, on average, 18.5% independent of the sample. The optimal k was of order 10.
We remark here, that it is strange that ternary predictors give worse results than binary. Probably it is due to a poor optimisation that we do get worse results here.

As a conclusion we would suppose that the idea to use the low resolution predictors is not promising and could not provide the desirable $p \sim 5\%$.

5.3. SDM predictors. An SDM is an almost deterministic mapping of a signal $X(t)$ into the binary sequence $\{x_i\}$. This means that, given a signal $X(t)$, one could write an adjustable model of an SDM to produce the output sequence $\{x_i\}$ almost without errors. We could write it in a form

$$\hat{x}_n = M\Big(X(t), t \leq \delta n; x_i, i < n\Big), \tag{7}$$

where M is some fixed map, depending on the realization of the SDM, and $\hat{x}_n$ predicts x_n almost without errors.

In our setup, the signal $X(t), t \leq \delta n$, is unknown. However, we can reconstruct and extrapolate it from $x_j, j < n$ applying the low pass filter with some coefficients. One can obtain those coefficients by analyzing the DAC of the decoder. Therefore, we have

$$\hat{X}(t) = f(t;\ x_i, i < n), \qquad t \leq \delta n.$$

Substituting the above estimate for $X(t)$ in (7), we should obtain an accurate estimate for x_n.

We have not completed this approach because it demands the exact knowledge of the SDM diagram and DAC filter characteristics. Nevertheless, to see how promising this approach can be, we have tried a simplified version of the algorithm. The following predictor

$$\begin{aligned} \hat{X}(t) &= (x_{n-1} + \cdots + x_{n-300})/300, \\ S(t) &= S(t-\delta) + x_{n-1} * \text{const}, \\ \hat{x}_n &= \text{sign}\Big(\hat{X}(t) - S(t)\Big). \end{aligned}$$

gives about $\sim 18\%$ of errors, which is quite promising.

5.4. Weighting. A promising approach to improve predictors is 'boosting', see e.g. [**9**] or [**4**]. In boosting, a number of predictors is constructed that are combined to yield the final predictor. The individual predictors that make up the final one are constructed by training a given base predictor from the same training data, using different weights. The weights are such that cases that were predicted wrong frequently with the predictors already constructed, are given more weight during the construction of the next one.

We have not fully explored the possibilities inherent in boosting. However, we did an experiment with the reweighting of badly predicted cases that is at the heart of boosting. Thus, we arrived at the following scheme to estimate the coefficient vector of the linear predictor.

$$\begin{aligned} \hat{\beta}_0 &:= \text{argmin}_\beta \sum_{n=k+1}^{N} \Big(x_n - \sum_{j=1}^{k} \beta_j x_{n-j}\Big)^2 \\ w_n &:= \begin{cases} 1 & \text{if } \Big|\sum_{j=1}^{k} \beta_j x_{n-j}\Big| \leq 1/2 \\ 0 & \text{otherwise, for } n = k+1, \ldots, N \end{cases} \\ \hat{\beta}_1 &:= \text{argmin}_\beta \sum_{n=k+1}^{N} w_n \Big(x_n - \sum_{j=1}^{k} \beta_j x_{n-j}\Big)^2. \end{aligned}$$

Thus the algorithm starts by constructing the basic linear predictor using ordinary least squares. It then throws away all 'sure' predictions by giving weight 0 to all cases for which the absolute predicted value exceeds 1/2. It

then constructs a coefficient vector from the remaining cases in the sample. The coefficient vector $\hat{\beta}_1$ thus found will be used for the linear prediction.

This simple scheme performs remarkably well; some numerical results are displayed in Figure 8. It can be observed that weighting improves the basic scheme by something between 0.5% and 2%. It is remarkable that the improvement was uniform: on all frames tried in all sequences did weighting improve.

Of course, many variants of this basic scheme are possible. Obvious possibilities are

- Changing the threshold from 1/2 to other values
- Iterating the scheme

However, the basic scheme given above proved to be very difficult to improve upon, and it will be used in the remainder.

5.5. Changing the prediction order. In this subsection, we investigate the effect of the prediction order on the prediction accuracy. Now, increasing the prediction order will 'obviously' improve the prediction accuracy. However, this will not necessarily lead to a more efficient coding scheme: the predictor must also be transmitted and longer predictors require more bits.

We have not investigated the efficient coding of the linear predictor separately. However, to compensate for this effect, we have simultaneously changed the length of the bit sequence on which the linear predictor is based. We expect that, roughly, the coding of a linear predictor is proportional to the length of the predictor. Hence it will be as efficient to use a linear predictor of length k for a bit sequence of length N as it is to use a linear predictor of length ck on a bit sequence of length cN for real valued c.

Qualitatively, it was found that decreasing the order of the predictor did always deteriorate the prediction performance. Increasing the order of the predictor to 256 did yield improvement; but there was no further gain in increasing the order to 512. The gain was only marginal with the basic linear predictor, but more substantial with the linear predictor obtained by reweighting as described in the previous section. It was found that the increased complexity of the predictor could be compensated for by increasing the frame size.

Figure 8 gives qualitative results and displays the prediction error of the linear predictor with prediction order $k = 256$ estimated from frames of size $N = 100000$. These can be compared with the errors with the other approaches described above. It is easily observed that the proposal from this section outperfroms the other approaches. Again, the improvement was uniform in that the current approach was better on all frames tried in all sequences.

5.6. Reducing the overhead. A final possibility to improve on the basic linear predictor scheme is to reduce the overhead by computing the predictor from the bits that have already been transmitted, at the decoder, rather than transmitting the predictor from encoder to decoder. Of course, the former scheme has several disadvantages as compared to the latter:

- It requires computational power at the decoder that is not needed with the current set-up.
- It makes the music less accessible: we cannot decode a given bit sequence without decoding its full past.

There are two basic strategies to implement such a scheme: adapting a current solution or recomputing a solution. In the former strategy, the coefficient vector of the linear predictor is updated continuously, i.e. after every new bit. The second possibility is to recompute the coefficient vector from a previous block of bits: the coefficient vector to predict the bits in the current frame is computed from the bits in the previous frame.

We have experimented only with the latter scheme, which we will call the lagged scheme. We give qualitative results only. Generally, the predictions from the lagged scheme are as good as the predictions from the original scheme. This holds true both for the original unweighted optimization and the proposed weighted optimization. This is the case for sequences A, B, and C. For sequence D the results are mixed. For the major part of the sequence the results are comparable. However, in the middle of sequence D the lagged predictor works very badly.

For this reason, the lagged predictor cannot be used without precautions. However, some approximations to the lagged scheme can be practical. We suggest the following options:

- The keep bit, which informs the coder to keep the coefficient vector from the previous frame. As the lagged predictor is almost as good as the predictor based on the current frame, this amounts to a saving of about 50% in the transmission of the coefficient vector.
- the link bit, which tells the decoder where it can find the required coefficient vector
- the recompute info, e.g. the lag, which tells the decoder which parameters to utilize in the recomputation

However, the savings that can be expected from this lagged scheme are limited as it can save at most the transmission of the coefficient vector. Because of these limited savings and the inherent problems, we have not pursued this lagged scheme much further.

Another possibility of saving on the transmission (storage) of the coefficients, is to use some methods of the theory of machine learning. For example, one of its oldest algorithms, the so-called *Perceptron Algorithm*. Let $\beta = (\beta_1, \dots, \beta_k)$ be a real vector, and $X_{n-k,n-1} = (x_{n-k}, \dots, x_{n-1})$ are

the last k observed bits. We predict $\hat{x}_n = 1$ if $(\beta, X_{n-k,n-1})$ is positive, and -1, otherwise. However, we are going to update β after each mistake:
a) if we predict $\hat{x}_n = -1$, while $x_n = 1$, let $\beta' = \beta + X_{n-k,n-1}$;
b) if we predict $\hat{x}_n = 1$, while $x_n = -1$, let $\beta' = \beta - X_{n-k,n-1}$.
To start the algorithm we may choose $\beta = (1, 1, \dots, 1)$. Motivation for the updating rules as above is the following:

$$(\beta', X_{n-k,n-1}) = (\beta, X_{n-k,n-1}) \pm (X_{n-k,n-1}, X_{n-k,n-1}),$$

so the value of $(\beta', X_{n-k,n-1})$ is closer to x_n, then $(\beta, X_{n-k,n-1})$.

We have applied the Perceptron Algorithm to our data. The quality of the prediction is substantially worse. On average, the perceptron algorithm makes twice the number of errors of the least squares predictor. This poor quality of the predictor is not compensated by the space we saved by not transmitting the coefficients. Therefore, the overall compression of the scheme based on Perceptron algorithm is lower. It is interesting to mention that after training the perceptron algorithm on a long sequence, the corresponding vector of coefficients β is very close to the one obtained from the Philips prediction scheme (5).

5.7. Conclusion. The linear prediction scheme suggested in [**5, 6**] leads to an efficient compression algorithm. It seems that in many cases the Philips linear predictor is quite close to the optimal linear predictor. Nevertheless, further improvements are still possible. Let us summarize our results on possible ways of improving the quality of linear predictors.

Switching to the longer predictors of low precision would probably not improve the overall performance of a linear predictor.

Weighting improves. Note that this improvement can be achieved with the current decoders as it requires change only for the encoder.

Some further benefits can be obtained by fine-tuning the frame size and the prediction order: it is particularly advised to increase the prediction order to 256. This can be achieved without loss in efficiency if the frame size is increased from 40000 to, say, 100000. Further optimizations are possible here.

Note that the improvements for frame A are then substantial: they amount to a reduction of approximately 75% of the storage of the bit string; the reduction obtained with the original approach is roughly 66%. The improvements on the other frames are less impressive. Whether these changes are worth the trouble will obviously depend on which of the frames is more typical.

Moreover, the cost of transmission of the coefficient vector can be reduced by at least 50% if a keep bit is defined.

Adaptive schemes, like the perceptron algorithm, seem to be unfeasible. In our opinion, it simply takes a very long time (and hence, a lot of mistakes

will be made) for such a scheme to achieve a quality comparable to that of the least squares predictor.

Finally, the optimal linear predictors cannot be computed (or estimated) efficiently at the present time.

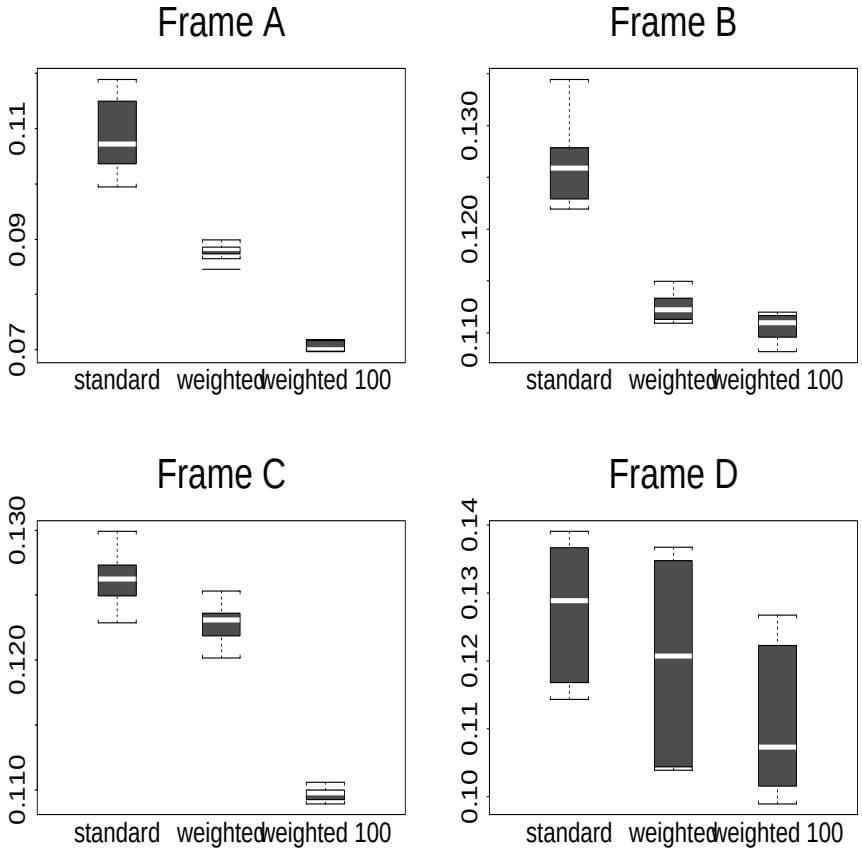


FIGURE 8. Prediction errors for several frames in given sequences for several approaches based on linear prediction. Standard: coefficient vector of length 128 estimated from sample of size 40000 using unweighted least squares. Weighted: coefficient vector of length 128 estimated from sample of size 40000 using weighted least squares. Weighted 100: coefficient vector of length 256 estimated from sample of size 100000 using weighted least squares.

Bibliography

[1] E. Amaldi, M.E. Pfetsch, L.E. Trotter, Some structural and algorithmic properties of the maximum feasible subsystem problem. *Integer programming and combinatorial optimization (Graz, 1999)*, 45–59, Lecture Notes in Comput. Sci., 1610, Springer, Berlin, 1999.

[2] E. Amaldi, V. Kann, On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoret. Comput. Sci.* 209 (1998), no. 1-2, 237–260.

[3] H. Arora, A. McLean, Stability Analysis of 1st and 2nd Order Sigma Delta Analog to Digital Converter, *preprint* `www.duke.edu/~ha/HimanshuArthur.pdf`

[4] L. Breiman, Arcing classifiers. The Annals of Statistics 26, 3, pp. 801-849, 1998.

[5] F. Bruekers, W. Oomen, R. van der Vleuten, and L. van de Kerkhof, Lossless coding of 1-bit audio signals. *AES 8th Regional Convention, Tokyo, Japan,* 1997.

[6] F. Bruekers, W. Oomen , R. van der Vleuten, and L. van de Kerkhof, Improved lossless coding of 1-bit audio signals. *AES 103rd Convention, New York*, 1997.

[7] D. Reefman, P. Nuijten, Why Direct Stream Digital is the best choice as a digital audio format, *Audio ES, Convention Paper, preprint*, 2001.

[8] W. Quine, The problem of simplifying truth functions, American Mathematical Monthly, 1952, 59, pp.521-531.

[9] R. Schapire, Y. Freund, P. Bartlett, and W. Lee, Boosting the margin: a new explanation for the effectiveness of voting. The Annals of Statistics 26, 5, pp. 1651-1686, 1998.