Convolutional code performance as a function of decoding delay

Edward A. Ratzer

Cavendish Laboratory, Madingley Road, Cambridge CB3 0HE
Phone: 01223 337344
Email: ear23@cam.ac.uk

Abstract

Unterminated rate $3/4$ convolutional codes are studied experimentally using the forward-backward algorithm over a 1% binary symmetric channel. A lower bit error probability is found for systematic compared to non-systematic generator matrices. It is further found that a decoding delay of $7m$ is needed to fully exploit the error-correcting capabilities of the code.

1 Introduction

To achieve reliable communication with block codes Shannon showed that the block size needs to become asymptotically large [6]. Block codes that reach 0.1 dB of the Shannon Limit have been found [5], however these required blocks of 10^6 bits. For very slow data rates blocks of this size are impractical. A code that allows early decoding and good performance is needed.

Infinite random tree codes (figure 1) can also reach the Shannon Limit [9][4]. They seem appropriate for a low data rate application: a decoding can be attempted at any time and the data earlier on in the transmission becomes progressively better protected by data received later. Unfortunately infinite random tree codes have infinite complexity. Convolutional codes [3] approximate this structure and are simple to encode and decode. An encoder can be visualised as a linear sequential circuit, for example figure 2. The encoding can alternatively be viewed in a similar manner to a tree code, in which the number of states of the system remains bounded, forming a trellis, for example figure 3.

Convolutional codes are often used as block codes by terminating the input stream with dummy zeros. This termination has the effect of forcing the system to a known state at the end of a block and hence provides equal protection to the bits at the beginning and end of the block. However if one is decoding a continuous stream or attempting a decoding in the middle of a block, one is dealing with an unterminated code. In this paper we will study this case.

To test the performance of unterminated convolu-

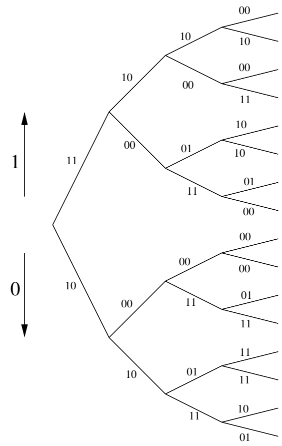

Figure 1: A diagram of the beginning of a $R = 1/2$ random tree code. The encoding of a stream can easily be read off the tree – start at the left and for each input bit move one state to the right and up for `1` or down for `0`. The output stream consists of the labels (made up of random i.i.d. bits) along the route.

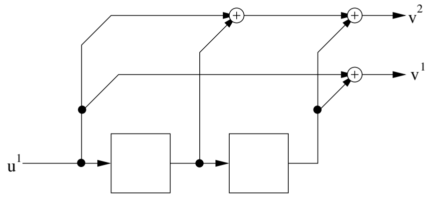

Figure 2: A diagram of a $R = 1/2$, $m = 2$ convolutional code in controller-canonical form. The boxes represent memory elements and $\oplus$ modulo 2 binary addition.

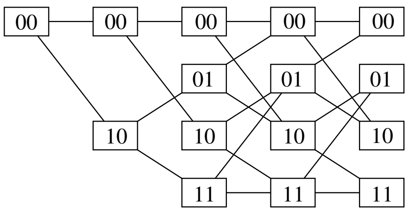

Figure 3: A trellis for the system shown in figure 2 (with four states). Only the content of the memories are shown, other labels are omitted for clarity.

tional codes many blocks were encoded, transmitted over a simulated noisy channel, and then decoded. Various algorithms for the decoding stage were considered. Three algorithms that fully exploit the error-correcting abilities of the code are widely used:

- The Viterbi algorithm [7] provides the maximum likelihood decoding of a block by visiting every branch.
- Sequential decoding [8] also provides a maximum likelihood decoding. In general it uses less resources than the Viterbi algorithm as it aims to explore only branches close to the maximum likelihood path.
- The forward-backward algorithm [1] provides the marginal posterior probabilities for each bit – this is called an *a posteriori* probability decoding. The algorithm is slightly slower than the Viterbi algorithm as it visits every branch twice.

The maximum likelihood path produces the minimum block error probability but not necessarily the minimum bit error probabilities. The forward-backward algorithm was chosen for further study as its *a posteriori* probability decoding behaviour minimises the bit error probability.

2 Theoretical Introduction

2.1 Convolutional codes

The following description is after Johannesson and Zigangirov [3].

Convolutional codes are described in terms of several parameters. The first is the rate, R. It is traditionally written in a fractional form with the number of input symbols to the sequential circuit over the number of output symbols; an $R = 2/4$ code hence has a different form than an $R = 1/2$ code. The other generally cited parameters are the memory, m, and the overall constraint length, ν. If each input has ν_i memory elements (for example in figure 2, $\nu_1 = 2$), we then define $m = \max_i \nu_i$ and $\nu = \sum_i \nu_i$.

Instead of using a sequential circuit form for describing convolutional codes, we can use a matrix representation of the system under a D-transformation (where operations are carried out modulo 2). Then $\boldsymbol{v}' = \boldsymbol{u}'\mathbf{G}$, where $\boldsymbol{u}'$ and $\boldsymbol{v}'$ are the vectors representing the D-transforms of the input time sequences $\boldsymbol{u}_t$ and output time sequences $\boldsymbol{v}_t$ respectively. For the example in figure 2:

$$\mathbf{G} = \begin{pmatrix} 1+D^2 & 1+D+D^2 \end{pmatrix} \qquad (1)$$

Convolutional codes can be catastrophic: a finite number of channel corruptions can lead to an infinite sequence of errors after decoding. To avoid this situation, the convolutional encoder needs to be chosen appropriately. Systematic convolutional codes (codes where the input stream appears in the clear in the output stream) are known to be non-catastrophic.

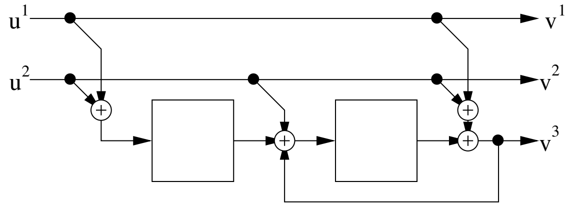

Figure 4: Observer-canonical representation of the generator matrix shown in equation 5

The codes so far described have not necessarily been systematic. It is known that if you create a systematic generator matrix from a non-systematic generator matrix by replacing the left hand side of the matrix with the identity matrix, a bad code is produced. Alternatively we can create a systematic matrix from a generator matrix, $\mathbf{G}$, by multiplying by the inverse of a square-matrix component, $\mathbf{T}$, for example:

$$\mathbf{G} = \begin{pmatrix} 1+D^2 & D+D^2 & 1+D \\ D^3 & D^3+1 & D^2 \end{pmatrix} \qquad (2)$$

$$\mathbf{T} = \begin{pmatrix} 1+D^2 & D+D^2 \\ D^3 & D^3+1 \end{pmatrix} \qquad (3)$$

$$\mathbf{T}^{-1}\mathbf{G} = \begin{pmatrix} 1 & 0 & \frac{1+D}{1+D^2+D^3+D^4} \\ 0 & 1 & \frac{D^2+D^3}{1+D^2+D^3+D^4} \end{pmatrix} \qquad (4)$$

If we use $\mathbf{T}^{-1}\mathbf{G}$ as our new generator matrix it forms the same code as $\mathbf{G}$; $\mathbf{T}^{-1}$ only defines a convolutional scrambling of the input bits. If $\mathbf{T}$ is selected so that its determinant is in the form $1 + f(D)$, then we can realise it in a sequential circuit form with the denominator corresponding to feedback in the delay line.

It is important to be able to represent a convolutional code as a sequential circuit with a minimal set of memory elements. The contents of the memories define the state; if we can minimise the number of memories we can reduce the trellis size and hence the complexity of decoding. In general this is the hard problem of finite-state-machine minimization; however for $R = N/(N+1)$ systematic codes the problem becomes trivial. We can use the distributive and associative properties of the finite field to reverse the order of the delay and addition properties to come up with another simple representation called observer-canonical form. For example for the systematic encoder

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & \frac{1+D^2}{1+D} \\ 0 & 1 & \frac{1+D+D^2}{1+D} \end{pmatrix} \qquad (5)$$

we can represent it using two memory elements rather than four as we would have in controller-canonical form. This form is shown in figure 4.

2.2 Forward-backward algorithm

We will start by defining a coordinate system on the trellis (i, j) where i is a time coordinate and j a coordinate over instantaneous state. b_t^{km} represents the path from state (t,k) to $(t+1,m)$. The set $\mathcal{L}$ contains all the valid branches on the trellis: $\mathcal{L} \equiv \{(k,m):$ a branch links state (i,k) to $(i+1,m)\}$.

If we receive the sequence of binary vectors $\boldsymbol{R} \equiv \boldsymbol{r}_1 \ldots \boldsymbol{r}_N$, we then want to infer the original sequence of vectors $\boldsymbol{v}_1 \ldots \boldsymbol{v}_N$. We calculate for each bit in the original data stream:

$$\Pr(v_i^l = 1|\boldsymbol{R}) = \sum_{(k,m)\in\mathcal{L}} \Pr(v_i^l = 1|b_i^{km}) \Pr(b_i^{km}|\boldsymbol{R}) \quad (6)$$

$$\propto \sum_{(k,m)\in\mathcal{L}} \Pr(v_i^l = 1|b_i^{km}) \Pr(\boldsymbol{R}|b_i^{km}) \quad (7)$$

where we have used Bayes' Theorem with a uniform prior across the branches. The first term in equation 7 is either 0 or 1 depending on the output corresponding to the branch. We can evaluate the second term efficiently using the trellis of the convolutional code. We define forward probabilities and backward likelihoods as

$$\alpha_{i,j} = \Pr(\boldsymbol{r}_1 \ldots \boldsymbol{r}_{i-1}, \text{path through } (i,j)) \quad (8)$$

$$\beta_{i,j} = \Pr(\boldsymbol{r}_i \ldots \boldsymbol{r}_N|\text{path through } (i,j)) \quad (9)$$

These can be evaluated recursively:

$$\alpha_{i,j} = \sum_{k:(k,j)\in\mathcal{L}} \alpha_{i-1,k} \Pr(\boldsymbol{r}_{i-1}|b_{i-1}^{kj}) \quad (10)$$

$$\beta_{i,j} = \sum_{k:(j,k)\in\mathcal{L}} \beta_{i+1,k} \Pr(\boldsymbol{r}_i|b_i^{jk}) \quad (11)$$

with the following boundary conditions for an unterminated block:

$$\alpha_{1,j} = \delta_{0,j} \quad (12)$$

$$\beta_{N+1,i} = 1 \quad (13)$$

We can then evaluate

$$\Pr(\boldsymbol{R}|b_i^{km}) = \alpha_{i,k} \Pr(\boldsymbol{r}_i|b_i^{km}) \beta_{i+1,m} \quad (14)$$

3 Experiments

A computer simulation of the communication system was tested. The performance of rate $R = 3/4$ convolutional codes over the binary symmetric channel with flip probability $p_f = 0.01$ was studied. The capacity of this channel is 0.92 and the computational cut-off rate [9] is 0.74. The process of collecting the data took the following stages:

1. A random binary data sequence of q independent identically distributed (i.i.d.) bits was created.

2. This data sequence was encoded using the chosen generator matrix.

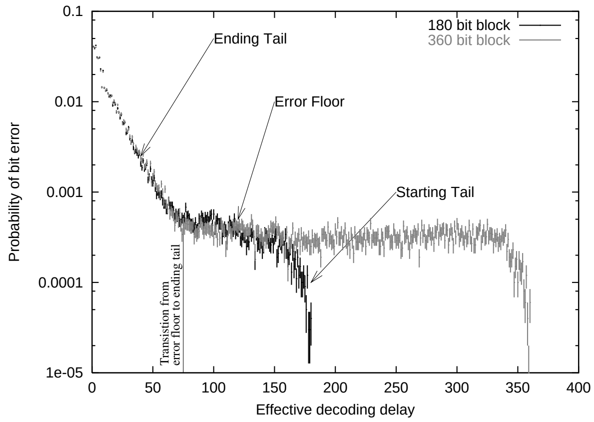

Figure 5: A graph of the bit error probability for transmissions using a $R = 3/4$ code over a 1% binary symmetric channel. The vertical lines represent 1σ error bars. The 180 bit block has been annotated with the regions described in the main text.

3. The first $q/R = t$ bits of the encoded stream were taken to simulate the effect of being in part of a longer data stream.

4. $p_f t$ bits were randomly flipped to simulate the received sequence.

5. The forward-backward algorithm was used to decode this received sequence and a hard decision returned.

6. This decision was compared to the originally transmitted sequence and the location of errors noted.

By repeating these tests many times and averaging, estimated error probabilities for each bit could be obtained. For each bit, each sample is independent; however, there will be correlations in errors between neighbouring bits as errors for convolutional codes tend to occur in bursts.

3.1 A sample set of errors

Two experiments were conducted on identical non-catastrophic $R = 3/4$ convolutional codes to determine the characteristic form of results. The transmission of 100,000 unterminated blocks of 180 data bits and 100,000 unterminated blocks of 360 data bits were simulated. The estimated probability of error for each bit in the blocks is shown in figure 5.

Each graph has three distinct regions as marked:

Ending Tail. The probability of error decreases as the decoding delay increases. The protection of the bits is increasing as the encoder's state becomes more well known in this region.

Error Floor. For data bits in the middle of the transmission there is a constant probability of error.

These bits are equally well protected and the end-effects have become unimportant – this can be confirmed by seeing that the differing length transmissions have the same error floor. It is expected that for a catastrophic code this flat region would not exist.

Starting Tail. Early in the code block the state is more well known as the encoder starts in a known state, hence the probability of error decreases.

3.2 Comparison of non-systematic and systematic generator matrices for a "good" code

An optimal $R = 3/4$, $m = 3$, $\nu = 9$ generator matrix was taken from Chang, Hwang, and Lin [2]. The transmission and decoding of 100,000 unterminated blocks of 180 data bits was simulated using this generator matrix. Then a systematic generator matrix for the same code was created and the experiment repeated. The probability of bit error for the two generator matrices is shown in figure 6. The systematic code has two desirable features:

- The probability of bit error is in general lower than the non-systematic code – a deviation from the correct path in the trellis of the code is more likely to decode to the originally transmitted bits for a systematic code.

- The bit error probability does not increase above the channel error probability. In the worst case no information on the state is known and the decoder will just report back the systematic bits as received. The bit error probability becomes "unclamped" from the channel error probability only when enough information is known about the state and the Hamming distance between trellis paths becomes great enough for error correction to occur.

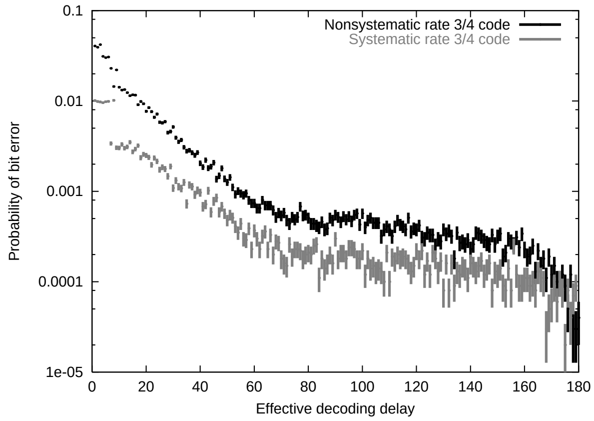

Figure 6: A graph of bit error probability for systematic and non-systematic generator matrices of the same $R = 3/4$ code over a 1% binary symmetric channel

3.3 Comparison of "good" and random systematic generator matrices

A random systematic generator matrix ($R = 3/4$, $m = 9$, $\nu = 9$) realizable in observer-canonical form was created. This generator was then compared to the systematic version of the "good" code used in section 3.2. The results are shown in figure 7. These codes show similar performance; the main difference is at very short decoding delays, where the random code performs worse. In the random construction the penultimate memory element did not have any connections to it, therefore no more protection was offered to the bits at the end of a transmission.

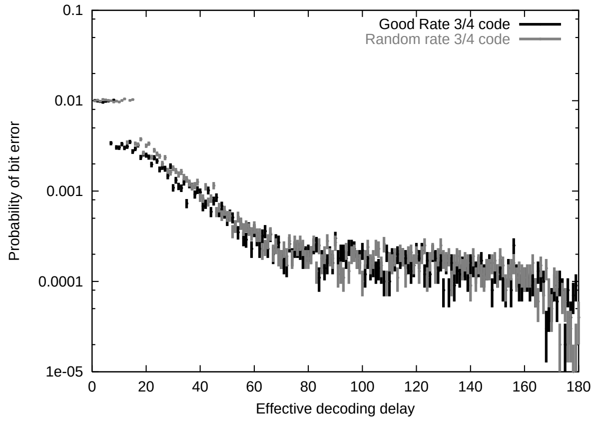

Figure 7: Comparison of a random code and a "good" code

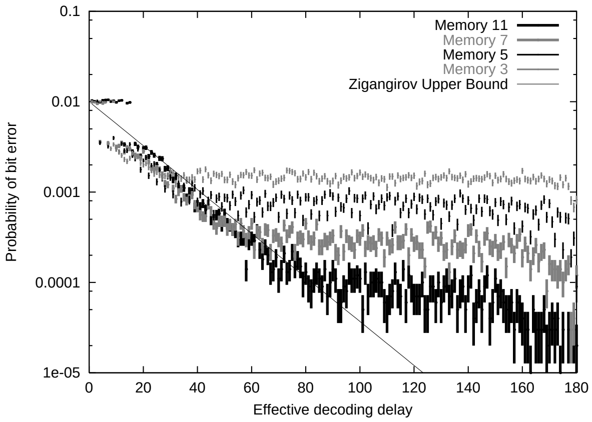

Figure 8: Comparison of different memory size

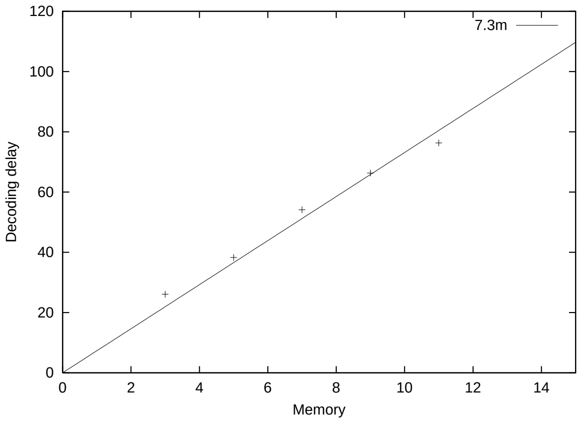

Figure 9: Transition from error floor to ending tail

3.4 Dependence on memory size

Random codes with different memory sizes were investigated and are shown in figure 8. The gradient of the ending tails is similar in each graph; the main variation between them is the level of the error floor. This variation is as expected as a more complicated code generally has a better error correcting capability. The Zigangirov upper bound on the contribution to bit error probability by finite decoding delay [9] is also shown on the graph.

The position of the transition from the error floor to the ending tail region (as shown in figure 5) effectively defines the decoding delay one should use to fully exploit the error correcting abilities of the code. The section of the graph from decoding delay 20 to 140 (these ranges were chosen so as to exclude the starting tail and remove the effect of the few "clamped" bits) was modelled as two sections. The error floor was taken to be a constant error probability and the ending tail a log scale straight line. A maximum likelihood best fit procedure was carried out with the gradient of the ending tail line kept fixed at the gradient found for the maximum memory size point (otherwise the length of the tail was too small for the procedure to work reliably). The results with a line of best fit (with a fixed zero intercept) are shown in figure 9. One can therefore deduce that the decoder should have a delay of at least $7m$ to fully exploit the error correcting ability of these codes.

4 Discussion

The Zigangirov upper bound for bit error probability against decoding delay as shown in figure 8 happens to be the same as the random-coding bound for bit error probability against block size. Using these coding bounds we can compare how decoding delay τ affects tree codes and block codes of size N. For block codes we will assume each bit has either the channel error probability or the error probability from the random coding bound depending on whether the whole block in which it lies has been received. For tree codes to outperform block codes we want:

$$\exp(-E_r\tau) \leq \begin{cases} \frac{\tau}{N}\exp(-E_rN) + (1-\frac{\tau}{N}) & \tau \leq N \\ \exp(-E_rN) & \tau > N \end{cases} \tag{15}$$

where E_r is the random coding exponent. For $\tau \ll N$ the $(1-\frac{\tau}{N})$ term dominates and the first part of the inequality will hold; the second part of the inequality holds trivially. Using Taylor expansion around $N = \tau$ the inequality was also shown to hold in this region.

It would be desirable to lower the error floor further. The obvious way of doing this, by increasing the memory size, is not immediately feasible in the set-up described in this paper; the computational complexity of the forward-backward algorithm scales exponentially with the memory size. Alternative methods could be used:

Sequential decoding. This technique would allow the memory size to be increased further as the decoder does not visit all trellis branches. However, it is only efficient below the computational cut-off rate – a lower rate code would have to be used over the example channel above. Sequential decoding is a maximum likelihood technique, so exact *a posteriori* probability decodings are not reached – this is a particular concern in the ending tail. Also when faced with bursty channels or loss of synchronization this algorithm can lose any computational advantage.

Turbo code. A turbo code design could be used. A very high rate code coupled with an $R = 3/4$ code could show a similar performance in the ending tail but be able to lower the error floor further. The form of the very high rate code would need to be carefully investigated as puncturing is known to cause problems in turbo codes and non-punctured convolutional codes become hard to handle as the rate is increased.

References

[1] Lalit R. Bahl, John Cocke, Frederick Jelinek, and Josef Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, 20:284–287, March 1974.

[2] Jinn-Ja Chang, Der-June Hwang, and Mao-Chao Lin. Some extended results on the search for good convolutional codes. *IEEE Transactions on Information Theory*, 43(5):1682–1697, September 1997.

[3] Rolf Johannesson and Kamil Sh. Zigangirov. *Fundamentals of Convolutional Coding*. IEEE, 1998.

[4] M.S. Pinsker. Bounds on the probability and of the number of correctable errors for nonblock codes. *Problems in Information Transmission*, 3(4):44–55, Winter 1967.

[5] T. Richardson, A. Shokrollahi, and R. Urbanke. Design of provably good low-density parity-check codes. Submitted to IEEE Transactions on Information Theory, available from http://cm.bell-labs.com/cm/ms/former/tjr/pub.html.

[6] C.E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, July 1948.

[7] A.J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13:751–772, 1967.

[8] Wozencraft and Reiffen. *Sequential Decoding*. MIT Technology Press and Wiley, 1961.

[9] Kamil Sh. Zigangirov. On the error probability of sequential decoding on the BSC. *IEEE Transactions on Information Theory*, 18(1):199–202, January 1972.