Codes for Channels with Insertions, Deletions and Substitutions
Edward A. Ratzer and David J.C. MacKay
Cavendish Laboratory, Cambridge, CB3 0HE, United Kingdom. E-mail: ear23,mackay@mrao.cam.ac.uk.

Abstract: We compare the performance of two codes for channels with insertion and deletion errors: marker codes [1] and watermark codes [2], [3]. Both codes are decoded using the sum-product algorithm. Marker codes can be viewed as irregular watermark codes. Our experiments give evidence that irregular constructions can improve watermark codes’ performance.

Keywords: Levenshtein distance, synchronization error, sum-product algorithm, Gallager code.

1. Introduction

Insertion-deletion channels are channels with synchronization problems. Bits are lost and gained between the source and the receiver at unknown positions in the bit stream. Examples of channels that might be modelled as insertion-deletion channels include:

Serial line: The clock speed of the transmitter may not be accurately known (for instance due to temperature variations in the clock) and hence the time of arrival of each transmitted bit is not known.

Hard disc: Variations in the rotation speed of the platter cause the bit rate to be inaccurately known.

DAT tape: As the tape goes through the reader or writer it can stretch, leading to insertion errors.

In some channels, synchronization problems are worst when several consecutive 1s or 0s are transmitted, so synchronization errors can be suppressed by the use of run-length limiting codes. In this paper, however, we study channels for which this fix is not adequate.

We assume that insertions and deletions occur at random, whatever the transmitted sequence (figure 1). This is the probabilistic channel model implicit in the Levenshtein metric [4]. We look for large blocklength encoding and decoding methods that can achieve reliable communication over such channels with insertion and deletion rates of the order of 1% or 10%.

Figure 1. Flow chart describing the probabilistic insertion-deletion channel with insertion probability $P_{ins}$, deletion probability $P_{del}$, transmission probability $P_k = 1 - P_{ins} - P_{del}$, and substitution probability $P_{sub}$.

Techniques for dealing with such channels are reviewed in [3]. There are few practical methods capable of correcting substantial numbers of insertions and deletions in a single block. The two methods that we compare here are marker codes [1] and watermark codes [3]. Both these codes are concatenated codes: an inner code is responsible for identifying the synchronization errors, and an outer code corrects the errors associated with uncertain synchronization and substitution errors.

In [3], the chosen outer codes were low-density parity-check codes over $GF(2)$ [5], [6]. In the present paper, we focus on the inner code and ignore the choice of the outer code, assuming an idealised, capacity-achieving outer code. We now describe the two inner codes.

1.1. Marker codes

The bit stream to be transmitted has a regular header inserted in it. For example if the header is 001 and the spacing is 4, then the inner code encodes a bit stream as follows:

01101100101010 → 0110001110000101000110

Seller's decoder looks for the inserted headers and uses shifts in their positions to deduce bit loss or
gain. The affected data bits could be flagged for the outer decoder’s attention. The above inner code works as long as the number of insertions or deletions between two adjacent headers does not exceed one, and the outer code’s error-correcting capabilities are not exceeded. Sellers did not suggest the use of the sum-product algorithm to decode marker codes; we use it here because it is the optimal algorithm, given a probabilistic channel model, for inferring the synchronization.

1.2. Watermark Codes

The idea of watermark codes is as follows. [They should not be confused with the digital watermarks found in cryptology.]

Imagine that a message is written on a deformable piece of paper that contains a complex watermark. The paper is warped during transmission (this is an analogy for insertions and deletions). Assume that the receiver already knows what the original watermark looked like, and so knows that the received pattern should look like a warped version of the watermark, partially obscured by the message that has been written over it. As long as the message is not too dense (you can’t see the watermark if the page is completely covered with ink), and the warping is not too severe, the receiver will be able to figure out the warping by finding the correct alignment of the original watermark with the received image. Once the alignment has been inferred, the original data can be recovered, except for the fragments of data where the paper was stretched or compressed.

In a watermark code, we turn the user data $d$ (already encoded with the outer code) into a sparse binary stream $s$ (figure 2). This sparsified data is added, bitwise, modulo two, to a pre-agreed pseudo-random binary sequence called the watermark, $w$. The received data $r$ is a distorted and warped version of the original watermark (distorted by the addition of the data, and by substitution errors; warped by the insertions and deletions). The decoder uses the sum-product algorithm to infer the alignment of the known watermark to this received string. The differences between the two aligned strings give a not-completely-reliable indication of what the sparsified data $d$ was. This soft output of the inner decoder is fed to the outer decoder which infers the original data $m$.

In [3], the outer code was a Gallager code over $GF(q)$ with a typical value of $q = 2^k$ being 16. The sparsifier studied was a mapping from $k$ bits to the sparsest $2^k$ sequences of length $n$ bits, with, for example, $k = 4$ and $n = 5$ (figure 3).

For simplicity, the inner decoder treats the sparse data stream as being produced by a memoryless source of independent identically distributed bits with density $P(s=1) ≡ f$.

In [3], Davey and MacKay presented codes of rate 0.7 and transmitted length 5000 bits that can correct 30 insertion/deletion errors per block, and codes of rate 3/14 and length 4600 bits that can correct 450 insertion/deletion errors per block (figure 4). Watermark codes can be decoded without the block bound-

### Figure 3
A sparsifier with $k = 4$ and $n = 5$. If the input data are random, the encoded data have density $f = 25/80 = 0.3125$.

### Figure 4
Block error rate of concatenated watermark codes (from [3]).
2. The marker codes studied

In this paper, we explore the space of marker codes, attempting to find their optimum parameters, and comparing their performance with that of watermark codes.

One can create a vast number of marker codes. We explore variations in
- the spacing between headers,
- the size of the headers, and
- the type of headers.

The two types of header we have tested are fixed-sequence headers like Sellers's, and pseudo-random headers.

We have also explored the idea of non-uniform marker codes, in which the size and type of header varies periodically - for example, several small headers evenly spaced between large headers; the large headers maintain the synchronisation on a large scale and the small headers limit the number of data bits affected by a single insertion or deletion between two large headers.

One important issue for large-scale applications is whether the header structure is periodic. If, as in Sellers's marker code, the header is periodic, then, for sufficiently large transmissions, the alignment algorithm is likely to drift, with the decoding getting out of synchronization by one or more periods. We explore this effect in the present paper. This problem is the motivation for the use of pseudo-random headers.

3. Sum-product algorithm

The sum-product algorithm can be used to infer the alignment of the received data with the transmitted data, and to find the conditional probability of the transmitted bits given the received bits.

The sequence of insertions and deletions can be represented by a path across a directed trellis as shown in figure 5(a). The horizontal dimension represents receiver time and the vertical dimension transmitted time.

Figure 5(b) shows two paths across the trellis. The solid line represents an alignment of the transmitted bits with the received bits that includes some insertions and deletions. If there were no insertions or deletions the alignment would follow the dashed line. The three types of moves possible on the grid are (figure 5(c)):

Normal transmission (possibly including a bit flip)
A diagonal move in which the top left hand end of the move lies on the intersection between the row and column corresponding to the transmitted and received bits respectively.

Insertion. This corresponds to a horizontal move in which an extra random bit appears in the received stream at the left hand end of the edge.

Deletion. This is a vertical move such that the transmitted bit corresponding to the top of the move does not have a position in the received stream.

For a particular path and set of received digits the joint probability of that path and the received
Figure 5. Representation of hypotheses about synchronization by paths across a two-dimensional trellis.

Figure 6. The forward pass of the sum-product algorithm.

data is:

\[ P(\text{path}, r) = P(\text{path})P(r|\text{path}) = \prod_{\text{insertions}} P_{\text{ins}} \prod_{\text{deletions}} P_{\text{del}} \prod_{\text{normal}} (1-P_{\text{del}}-P_{\text{ins}}) \times \prod_{\text{insertions}} \frac{1}{2} \prod_{\text{normal}} \left\{ \frac{P_{\text{flip}}}{2} \right\}. \]

(2)

The probability of the path is given by a product of the probabilities for each insertion \((P_{\text{ins}})\), deletion \((P_{\text{del}})\) and normal transmission. The probability of the received datum \(r_i\) when a bit is inserted is \(\frac{1}{2}\) for either \(r_i = 0\) or \(1\). Deletions do not give any received data and hence do not contribute to the probability of \(r\). For each normal transmission the contribution to the probability of the received data is one of the three terms:

- \(P_{\text{flip}}\) (the probability of a transmitted digit being flipped) This is chosen when the transmitted digit is known and the received digit does not match it.
- \((1-P_{\text{flip}})\) – when the received digit matches the known transmitted bit.
- \(\frac{1}{2}\) This is chosen when the transmitted digit is not known (i.e., is not in a header) – assuming a 50:50 input distribution.

When decoding we do not know the path followed or the user data. To infer the user data given the received data \(r\), we marginalise across all paths using the forward-backward algorithm.

We define a forward probability at a position \((i, j)\), \(P_f(i, j)\), to be the probability that the alignment passes through \((i, j)\), and that the received data up to time \((i-1)\) is \(r_1...r_{i-1}\). There are only three ways the path can get to \((i, j)\) – it can arrive by a horizontal, vertical or diagonal move from an adjacent space (figure 5(c)). So \(P_f(i, j)\) can be found recursively:

\[ P_f(i, j) = P_{\text{ins}} \frac{1}{2} P_f(i-1, j) + P_{\text{del}} P_f(i, j-1) + \]

\[ \frac{1}{2} \left( 1-P_{\text{del}} - P_{\text{ins}} \right) \left\{ \frac{P_{\text{flip}}}{2} \right\} P_f(i-1, j-1) \]
The choice in curly brackets is made by comparing received bit $r_{i-1}$ and transmitted bit $t_{j-1}$. The boundary conditions, assuming that the synchronization error is known to be zero initially, are $p_r(0,0) = 1$, and $p_r(i,j) = 0$ for all points not on the grid.

Similarly, we define the backward probabilities, $p_b(i,j)$, to be the conditional probability of the received data from time $i$ onwards, $r_i \ldots r_N$, given that the alignment passes through $(i,j)$.

The backward message-passing algorithm for computing $p_b(i,j)$ is similar to the forward algorithm above. The boundary conditions are $p_b(i,j) = 0$ for $(i,j)$ not on the grid, and $p_b(N,i) = 1$ for all rows $i$, where $N$ is the last column of received data.

The forward and backward messages $p_f$ and $p_b$ are used to infer the posterior probability of each user bit, $P(t_j | r)$, by marginalizing over the diagonal and vertical edges in row $j$.

The above algorithm's cost is $O(N^2)$ in space and time, where $N$ is the length of the entire block of data; this cost is excessive for large blocks. However, we know for small insertion and deletion probabilities that the path is likely to run close to the leading diagonal of the grid. Thus we can restrict the computation of the forward and backward probabilities to an appropriately wide swathe down the diagonal, and assume the rest of the probabilities are zero. The width was taken to be 5 standard deviations. A more flexible approach to efficient path-finding would be to use the Uniform Cost Search algorithms [9].

4. Experiments

The whole process of adding the headers, transmitting through the channel and then carrying out resynchronisation using the forward-backward algorithm can be seen as creating an effective channel, which we model as a time-varying memoryless binary symmetric channel, with each bit's transition probability being defined by the likelihood ratio returned by the inner decoder.

The capacity of the channel at the $j$th transmitted bit is

$$C_j = 1 - H_2(p(t_j = 1 | r)).$$  \hspace{1cm} (4)

The average capacity is multiplied by the rate of the marker code to give a measure of the maximum rate of reliable communication through the entire system.

In all results, unless otherwise stated, a block size of 600 was used, and it was assumed that the correct synchronization was known at the start of the block. In all experiments, we set $p_{\text{pos}} = p_{\text{del}}$. While non-zero substitution error-rates can easily be used, the results we report here are for zero substitutions.

4.1. Uniform Marker Codes

We generated random source data and added headers of a user-definable size at user-definable intervals.

A uniform marker code is one with only one type of header, with fixed size. Two types of headers were studied. The first were headers similar to those proposed by Sellers consisting of a sequence of Os for half the header followed by Is for the other half (henceforth referred to as IBM headers, as that was how Sellers worked). The other type were headers consisting of pseudo-random bits. Examples of uniform headers are shown below (* = user data, R = random bit).

<table>
<thead>
<tr>
<th>Header spacing</th>
<th>Header size</th>
<th>IBM</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
<td>........01</td>
<td>........RR</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>........00011</td>
<td>........RRR</td>
</tr>
</tbody>
</table>

At a particular noise level we first determined the best code to use. The two different header types were tried with a range of sizes and spacings and then the combination of these that maximised the maximum rate was be found. For example the dependence on header size and spacing with IBM headers for an error rate of 4% is shown in figure 7.

The performance of a few of the best uniform marker codes with IBM headers as a function of the insertion/deletion probability is shown in figure 8(a). Obviously, which code is best varies with the noise level.

The envelope of the best uniform codes with IBM and random headers is shown in figure 8(b). Some of the best header arrangements found are shown
Figure 8. Performance of uniform marker codes. (a) Best uniform codes with IBM headers; (b) Comparison of random and IBM headers; (c) Comparison with watermark codes; (d) Some of the best headers found for uniform marker codes (* = data bit, R = random header bit).

Figure 10. (a) Performance of non-uniform marker codes; (b) comparison with watermark codes.

in figure 8(d). The best uniform marker codes are compared with the watermark codes of Davey and MacKay in figure 8(c). The results suggest that marker codes might perform better than watermark codes at low error rates.

4.2 Dependence on block size

The dependence on block size of the best uniform codes at 1% and 10% error rates is shown in figure 9. The dependence is not particularly large at low error rates (1%); at higher error rates (10%) the performance of a code with the IBM headers drops off with increasing block size while the code with random headers is unaffected.

4.3 Non-uniform marker codes

In a non-uniform marker code, a combination of small and large headers is used. We placed equal amounts of data between adjacent headers. All-random and all-IBM headers were tested, and large random headers combined with small IBM ones. The envelopes of the best codes are shown in figure 10(a). Some of the best header structures are shown in table 1. A comparison of the best non-uniform codes found with watermark codes is shown in figure 10(b).

The effect of changing block size is shown in figure 11. At high error rates, the periodic code with only IBM headers is the only code noticeably affected by block size.
Figure 9. Effect of block size on uniform marker codes at 1% and 10% insertion/deletion probability.

<table>
<thead>
<tr>
<th>Insert/Delete Probability</th>
<th>Header Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5%</td>
<td>01</td>
</tr>
<tr>
<td>2%</td>
<td>01................01</td>
</tr>
<tr>
<td>4%</td>
<td>01................01</td>
</tr>
<tr>
<td>8%</td>
<td>01................01</td>
</tr>
<tr>
<td>12%</td>
<td>RRRRRRRRRRRRRR</td>
</tr>
</tbody>
</table>

Table 1. Best non-uniform headers.

Figure 11. The effect of block size on non-uniform codes at 2% and 10% error rate.
5. Discussion

We note the following patterns in the codes found. First, all the best codes on the channels studied use small headers (3 bits or smaller). At $p_{\text{ins}} = p_{\text{del}} = 0.5\%$ uniform codes (made out of IBM headers) give the best performance. Up to $p_{\text{ins}} = p_{\text{del}} = 6\%$ non-uniform codes with IBM headers perform the best. However the periodic IBM codes are adversely affected by increasing block size so if synchronization is to be maintained over long sequences, some form of pseudorandom header should be used. Above $p_{\text{ins}} = p_{\text{del}} = 6\%$, non-uniform codes with random headers were the best marker codes. Above about $p_{\text{ins}} = p_{\text{del}} = 10\%$, the results for regular watermark codes were the best.

When making marker codes with pseudorandom headers, it seems plausible that, rather than using completely random sequences, the best choice would be to pick pseudorandomly from a subset of good-quality IBM-like headers. For example, the headers 001, 110, 100, and 011 could be used in a pseudorandom sequence.

It will be interesting to see whether irregular watermark codes can outperform marker codes under realistic conditions.

ACKNOWLEDGEMENTS

This work was supported by the Gatsby Foundation and by a partnership award from IBM Zurich Research Laboratory. We thank M.C. Davey for providing the watermark code data, and for comments on the manuscript. E.A.R. thanks Christ's College, Anton Altpamakov, Richard van der Hoff, Matthew Bentham and Adam Johansen for providing computer resources. We are grateful to Olaf Wagner for fixing a bug in Modula-3's random number generator.

REFERENCES


Published in the proceedings of the 2nd International Symposium on Turbo Codes and Related Topics, Brest, France, 4–7 September 2000.