

Reliable Communication over Insertion-Deletion Channels

Abstract.....	1
Introduction	2
Forward-backward algorithm	5
Methods	9
Results	11
Discussion.....	17
Uniform codes	17
Non-uniform codes	17
Constructing a good code	18
Future Directions	19
Conclusion.....	21
References	22
Acknowledgements	22
Appendix: Codes Used.....	23

Abstract

The performance of various different types of marker codes was studied and the maximum rate of reliable transmission using them over channels that experience insertion and deletion errors was experimentally measured. Improved performance over watermark codes was discovered for noise levels up to $p_{ins}=p_{del}\approx 8\%$.

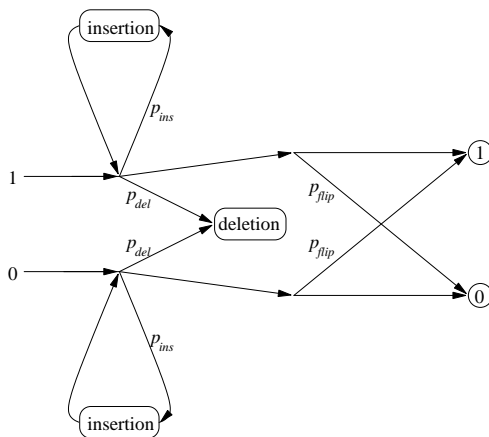
Introduction

When an insertion-deletion channel is used bits are lost and gained between the source and the receiver at unknown positions in the bit stream. These errors are also known as synchronisation errors as this is a common cause of them. Channels like this are reasonably commonplace, some examples are:

- *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 (for instance due to mechanical vibrations or shock) means the position of the head relative to the platter may be unknown. Bits can therefore be read off the disc in a slightly different place to where they were written and hence the system behaves like serial line.
- *DAT tape* As the tape goes through the reader or writer it can stretch slightly leading to the same effects as above with the tape.
- *DNA* Random mutations can lead to base pairs being added or removed from a genome.

In the first three cases above the effect is particularly acute when a whole stream of 1s or 0s are transmitted. This is because if alternate 1s and 0s are sent the boundaries between each bit become obvious (for example in the first case there will be alternating high and low voltages) and hence it is hard to get synchronisation errors between the transmitted signal and the received signal. However if a whole sequence of 1s or 0s is transmitted the clock signal is not represented in the received signal and hence the receiver's and transmitter's clocks can drift out of synchronisation. This therefore leads to the idea behind the most common way of dealing with insertion and deletion errors; one tries to ensure that these errors do not happen in the first place by restricting the maximum number of 1s or 0s that can occur in a row - this technique is called a run-length limited coding.

However problems can still occur and in this project insertion-deletion channels will be studied that can be represented as discrete memoryless channels in this form:



There are several existing techniques to deal with a channel like this:

- *Comma-free codes* [Golomb] The transmitted signal is mapped into codewords that are transmitted in sequence without punctuation between them. The codewords are constructed so that they have the property that no overlap between the codewords can be confused as a codeword. For example the codewords 1000, 1100 and 1110 form a comma-free code, as shown by looking at the overlaps between the codewords:

	1000	1100	1110
1000	10001000	10001100	10001110
1100	11001000	11001100	11001110
1110	11101000	11101100	11101110

This scheme therefore has the property that if a codeword is corrupted with an insertion or deletion it is possible to regain synchronisation again after the error, however error correction within the corrupted codeword is not generally possible [Bours].

- *Levenshtein codes* [Levenshtein] A family of coding strategies exist that is based on Levenshtein's number theoretic work. Under this scheme a transmitted codeword has the property that in the occurrence of a fixed number (or less) of insertions and deletions the codeword can not be confused with another one. In a similar manner to a Hamming distance one can define the Levenshtein distance between codewords to be the minimum number of insertions or deletions necessary to get from one codeword to another - one would then require all the codewords to have a Levenshtein distance of at least 3 between them to correct a single insertion or deletion error.
- *Watermark codes* [Davey] These allow multiple insertion and deletion errors in a single block to be corrected. They work on the idea that if a known sequence with distinctive pattern is transmitted one can infer the position of synchronisation between the receiver and transmitter. If instead you transmit this known sequence

(called the watermark) eXclusive-ORed with a stream of sparsified data then the decoder can follow the synchronisation with the watermark and also then infer what the sparsified data was.

- *Marker codes* [Sellers] The bit stream to be transmitted has a regular header (or marker) inserted in it. For example adding the header 001 with a spacing of 4 does the following to a bit stream:

01101100101010 \Rightarrow 01100011100001101000110

The decoder can then look out for the inserted headers and use any shift in their position to deduce previous bit loss or gain. These codes can be seen as a special case of watermark codes as we have alternating regions of sparse and dense data in the coded stream.

In this project the best way to construct marker codes is studied. We want to find the code that has largest maximum rate of reliable transmission (henceforth referred to as the maximum rate) on some given channel. The channel, coding and decoding (using the forward-backward algorithm as described later) were simulated on a computer and a large variety of codes and channels were studied. Hence an estimate to this maximum rate could be obtained for each combination.

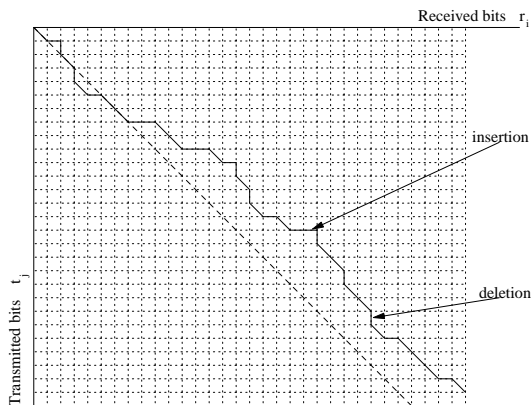
One can create a vast number of different marker codes by adjusting different parameters. First of all the spacing between headers is crucial as between each header one can not tell the position of synchronisation and hence one error can lead to half the bits between the headers being corrupted. However this problem needs to be balanced with the overhead of having too many headers. Secondly there is the choice of header type and size. Sellers in his original paper uses a header of 001 but perhaps larger headers would allow more errors to be corrected (in his paper he presents an algorithm using this header to determine a single bit loss or gain between headers). It may also be of benefit to have several small headers between large headers, the large headers can be used to maintain the synchronisation on a large scale and the small headers can be used combat the above problem of up to half the bits between headers being corrupted by a single insertion or deletion. Another factor is whether a repetitive header structure is used - if it is repetitive it is possible the decoding could get of out sync by one or more header repeat units (henceforth called chunks) - one would expect this problem to be manifested by a dependence on block size as when the block size is increased the probability of this occurring will increase.

We will therefore attempt to find the optimum parameters for these codes and compare the maximum rates achieved with watermark codes as these are a similar type of coding strategy.

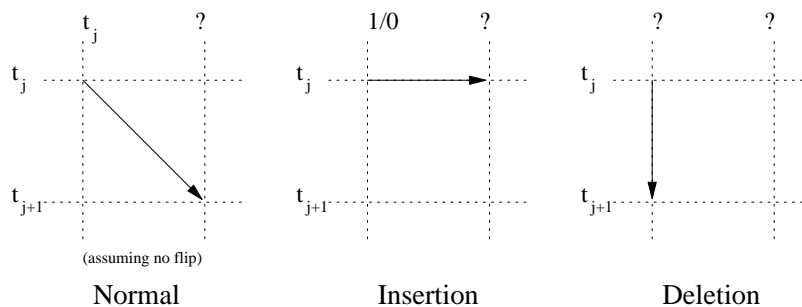
Forward-backward algorithm

The forward-backward algorithm (also known as the sum-product algorithm) is a probabilistic model which allows us to find the conditional probability distribution of the transmitted bits given the received bits.

It is helpful to develop the algorithm keeping a diagram like this in mind:



The solid line represents one particular sequence of insertions and deletions which then defines a mapping from the transmitted bits to the received bits. If there were no insertions or deletions the mapping would follow the dashed line - this represents a one-to-one mapping between the received and transmitted bits. To correspond to the three operations of the channel there are three different types of moves possible on the grid:



- *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.
- *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.

Note that to uniquely define the position of synchronisation for a complete stream the grid that the path is shown on needs to have an additional row and column over and above the transmitted and received bits. This is so we can tell for example whether the last bit received was as a result of insertion or normal transmission.

For a particular path and set of received digits it is easy to calculate the probability of that path and the received data:

$$\text{pr}(\text{path, received data}) = \prod_{\text{insertions}} p_{ins} \prod_{\text{deletions}} p_{del} \prod_{\text{normal}} (1 - p_{ins} - p_{del}) \cdot \prod_{\text{insertions}} \frac{1}{2} \prod_{\text{normal}} \left\{ \begin{array}{l} p_{flip} \\ 1 - p_{flip} \\ \frac{1}{2} \end{array} \right\}$$

The probability of the path is given by a product of the probabilities for each insertion (p_{ins}), deletion (p_{del}) and normal transmission (in a similar manner to a multinomial distribution). The probability of the received data for an inserted bit is $\frac{1}{2}$ (in our channel model it is equally likely to be a 0 or 1). Deletions do not give any received data and hence do not contribute towards the probability of the received data. For each individual normal transmission there is a choice between three possible contributions to the probability of the received data (represented by the curly brackets):

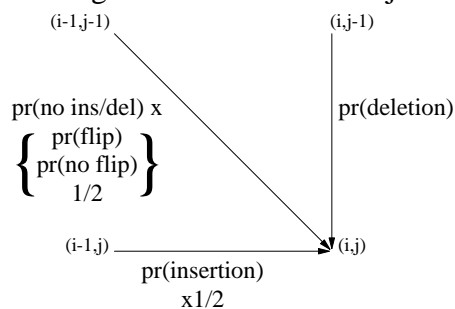
- p_{flip} (the probability of a transmitted digit being flipped) This is chosen when the transmitted digit is known and the received digit does not match it.
- $(1 - p_{flip})$ This is chosen when the received digit matches the known transmitted bit.
- $\frac{1}{2}$ This is chosen when the transmitted digit is not known (ie we are not in a known header) - this then is assuming a 50:50 input distribution.

However when decoding we do not know the path followed. We therefore want a technique that allows us to marginalise across all paths. The forward-backward algorithm allows us to do this.

We first define a forward probability p_f at a position (i, j) to be:

$$p_f(i, j) = \text{pr}(\text{path goes through } (i, j), \text{ received data up to } (i - 1))$$

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:



Therefore:

$$p_f(i, j) = p_{ins} \frac{1}{2} p_f(i-1, j) + p_{del} p_f(i, j-1) + (1 - p_{del} - p_{ins}) \cdot \left\{ \begin{array}{c} p_{flip} \\ 1 - p_{flip} \\ \frac{1}{2} \end{array} \right\} \cdot p_f(i-1, j-1)$$

The choice in curly brackets is made by comparing received bit $r_{i,j}$ and transmitted bit t_{j-1} .

This equation then allows us to iterate across the entire grid from the top left hand corner. The boundary conditions are $p_f(i,j)=0$ if not on the grid and $p_f(0,0)=1$ (we assume we start in the top-left hand corner)

In a similar manner we can also define backward probabilities $p_b(i,j)$ as:

$$p_b(i, j) = \text{pr}(\text{received data from } i | \text{path through } (i, j))$$

such that $p_f(i, j) p_b(i, j) = \text{pr}(\text{path through } (i, j), \text{received data})$

We have three possible ways of going from (i,j) , these lie in the opposite direction to the forward probabilities, thus:

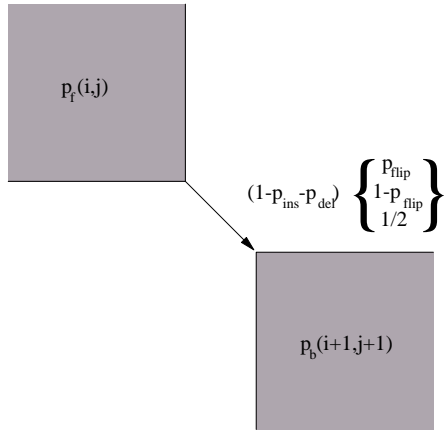
$$p_b(i, j) = p_{ins} \frac{1}{2} p_b(i+1, j) + p_{del} p_b(i, j+1) + (1 - p_{del} - p_{ins}) \cdot \left\{ \begin{array}{c} p_{flip} \\ 1 - p_{flip} \\ \frac{1}{2} \end{array} \right\} \cdot p_b(i+1, j+1)$$

We can then iterate from the bottom right hand corner using the boundary conditions $p_b(i,j)=0$ if not on the grid and $p_b(N,i)=1$ (where N is the last column and i is any row).

Then p_f and p_b may be used to infer $\text{pr}(t_j=0 | \text{received data})$ and $\text{pr}(t_j=1 | \text{received data})$.

For example:

$$\begin{aligned} \text{pr}(t_j = 1 | \text{received data}) &= \sum_i \text{pr}(\text{crossed row } j \text{ at column } i \text{ as diagonal}) \text{pr}(r_i = 1) + \\ &\quad \text{pr}(\text{crossed row } j \text{ as deletion}) \cdot \frac{1}{2} \\ &= \frac{\sum_i p_f(i, j) (1 - p_{ins} - p_{del}) \cdot \left\{ \begin{array}{c} p_{flip} \\ 1 - p_{flip} \\ \frac{1}{2} \end{array} \right\} \cdot p_b(i+1, j+1) \text{pr}(r_i = 1)}{\text{pr}(\text{data})} + \quad (*) \\ &\quad \text{pr}(\text{crossed row } j \text{ as deletion}) \cdot \frac{1}{2} \end{aligned}$$



This diagram is helpful in seeing how the first term in (*) is obtained. p_f covers all paths to the diagonal, we then use the normal probability expression for a diagonal as used previously and then p_b covers all paths after the diagonal.

The probability of the data can be calculated by marginalising p_f over all paths in the last column. All paths have to have one diagonal or horizontal move over the last column, therefore:

$$\begin{aligned} \text{pr}(\text{data}) &= \sum_j p_{ins} \frac{1}{2} p_f(N-1, j) + (1 - p_{ins} - p_{del}) \cdot \left\{ \begin{array}{c} p_{flip} \\ 1 - p_{flip} \\ \frac{1}{2} \end{array} \right\} \cdot p_f(N-1, j-1) \\ &= \sum_j p_f(N, j) - p_{del} p_f(N, j-1) \end{aligned}$$

The path has to cross the row by a diagonal or a deletion so $\text{pr}(\text{crossed row } j \text{ as deletion})$ in (*) can be calculated simply once the first term is calculated for both $t_j=0$ and $t_j=1$.

Hence (*) gives us the posterior probability distribution for each transmitted bit given the received data.

Methods

To implement the forward-backward algorithm efficiently on a computer two alterations needed to be made to the algorithm described previously.

Firstly the algorithm is $O(N^2)$ in space and time which causes problems with large blocks. If, for example, we want to deal with a block of 4000 bits this would require 32 million forward and backward probabilities to be calculated and stored - this takes a very long time and could typically use 128Mb of RAM. However we know for reasonable insertion and deletion probabilities that the path is likely to run close to the leading diagonal of the grid. Thus we can approximate the grid of probabilities instead with a swath down the middle diagonal and assume the rest of the probabilities are zero. This however means that only paths inside the swath are properly accounted for so it needs to be made sufficiently wide that the probability of being outside the swath is negligible. The width was taken to 5 standard deviations (with a block size of 4000 bits the probability of any section of the path being outside the swath is less than 0.1%), hence:

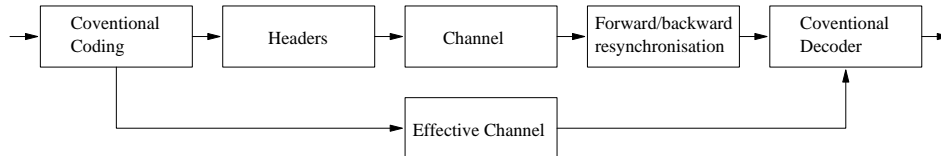
$$\text{Width} = 2 \cdot 5\sigma = 10 \sqrt{N \frac{(1 - p_{del})(p_{ins} + p_{del})}{(1 - p_{ins})^2}}$$

The second alteration is needed to deal with the very small probabilities obtained during calculation (probabilities of order 10^{-1000} are commonplace). Each column of p_f and p_b was renormalised into a manageable range during their generation and the factor stored. Then the final inference was then done in log space so that all the terms could be included. Unfortunately the entire calculation can not be easily done in log space as there are additions during each evaluation of p_f and p_b (cf. Viterbi algorithm [Durbin])

This was then coded into a Modula-3 program. The program first generated a random data stream to be transmitted and added headers of a user definable size to the data at user definable intervals. Two types of headers were studied. The first were headers similar to those proposed by Sellers consisting of a sequence of 0s for half the header followed by 1s for the other half (henceforth referred to as IBM headers). The other type were headers consisting of pseudo-random bits (using a pseudo-random number generator means that the headers used can be known to both the transmitter and receiver without additional communication). Examples of uniform repetitive header structures are (. =data, R=random bit):

Header spacing	Header size	IBM	Random
7	2 01 RR
3	5	. . . 00011	. . . RRRRR

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 another effective channel:



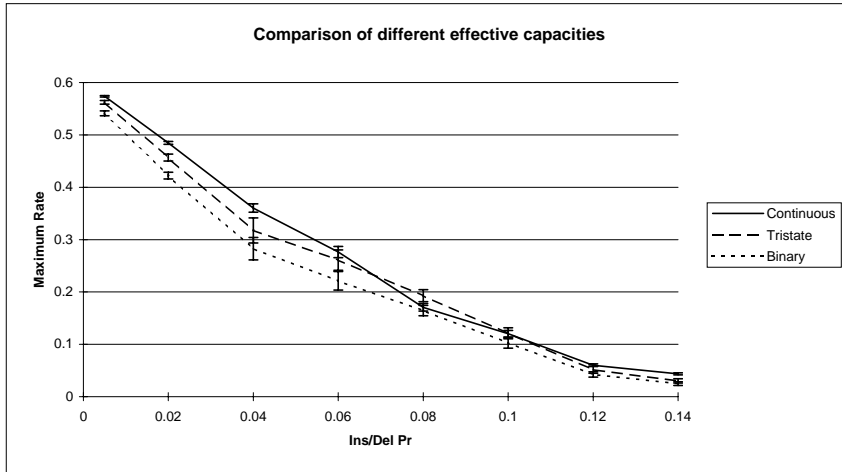
The program then modelled this effective channel in three ways. Firstly as a binary symmetric channel with a constant probability of flips occurring. Secondly as a binary symmetric channel inside an erasure channel with constant transition probabilities. Thirdly each bit was modelled as a continuous channel using each of the probabilities obtained from the forward-backward algorithm. These three different effective channels have three different capacities. These were calculated as follows:

- “*Binary capacity*” The decoder took a best guess as to whether it thought each digit was a 1 or a 0. The number of bits flipped was then counted and the capacity of the effective channel calculated using $(1 - H_2(\frac{\#bits\ flipped}{\#bits}))$ as with a binary symmetric channel.
- “*Tristate capacity*” The decoder took a best guess as to whether it thought each digit was 1, 0 or not known. The capacity of the effective channel is then given by: $(1 - \frac{\#bits\ deleted}{\#bits})(1 - H_2(\frac{\#bits\ flipped}{\#bits - \#bits\ deleted}))$.
- “*Continuous capacity*” The decoder’s conditional probability output for each bit was used in full. The capacity of each bit is $(1 - H_2(\text{pr}(t_j=1|\text{data})))$.

This then leads to a measure of the maximum rate of reliable communication through the entire system by multiplying the capacity of the effective channel produced by the rate of the inner code: $(\frac{\#data\ bits}{\#bits})$.

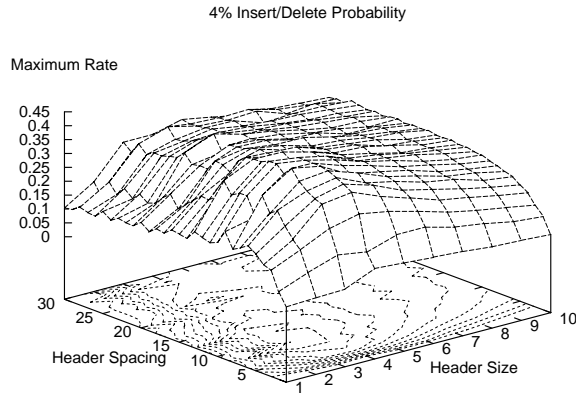
Results

A comparison of the different types of capacities listed in the Methods section was first carried out. For a particular code its performance as $p_{ins}(=p_{del}$ for all results shown) is varied can be shown, for example (using a block size of 600, as with all subsequent results unless otherwise stated):

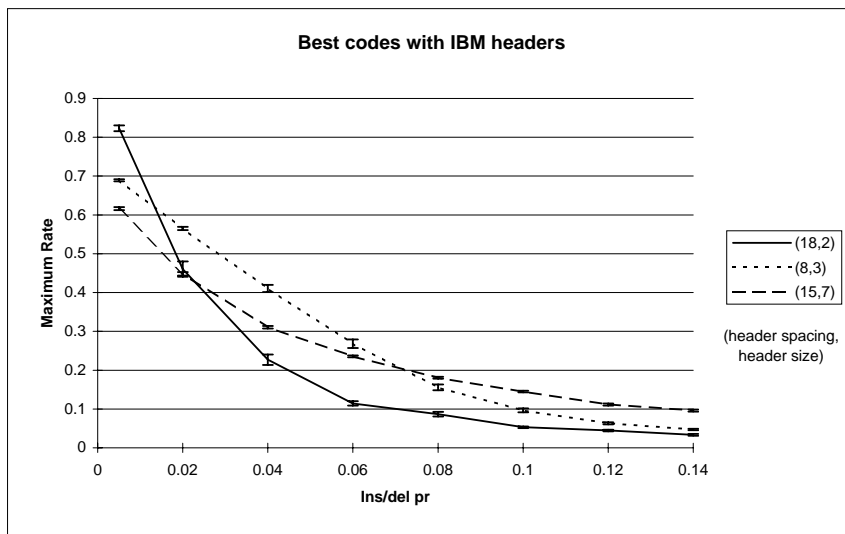


It was noticed that for most samples the continuous capacity gave the best results. However, as seen here at $p_{ins}=p_{del}=8\%$, the other methods would occasionally give a better result. For example, if the decoder output that a bit was 90% likely to be a 1 and was actually transmitted as a 1 the continuous capacity for that bit will be lower than the two other capacities. In general as more levels of detail are put in the decoding, it can be seen that there is a steady improvement in the maximum rate. The rest of the results are presented using the continuous capacity as this is the most sensible measure of the capacity of the effective channel and also its response on a particular test is generally the most consistent.

At a particular noise level the first exercise is to determine the best code to use. Initially only one header size and spacing was used throughout the block (henceforth called a uniform marker code). 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 could then be found. For example the dependence on header size and spacing with IBM headers for a particular error rate is:

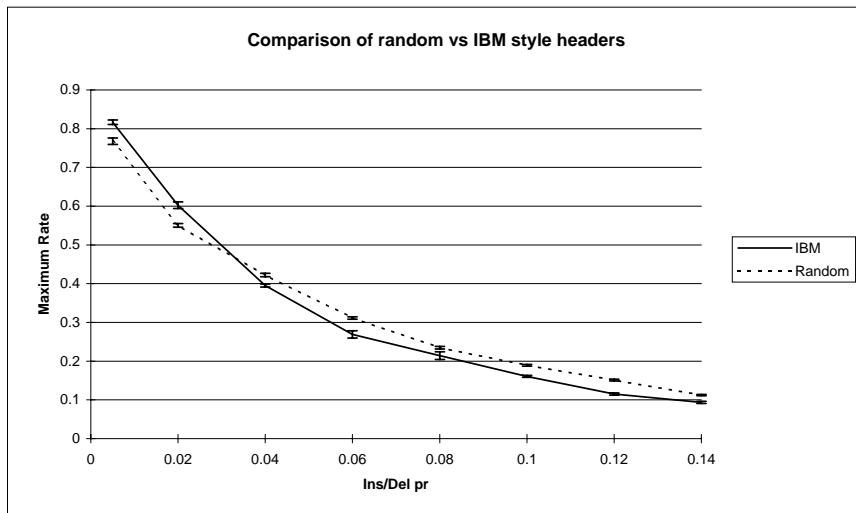


The maximum can be found which corresponds to the best code at that noise level with IBM headers. A few of these are shown below:



It can be seen that a code that performs the best at one particular error rate does not perform so well under different conditions.

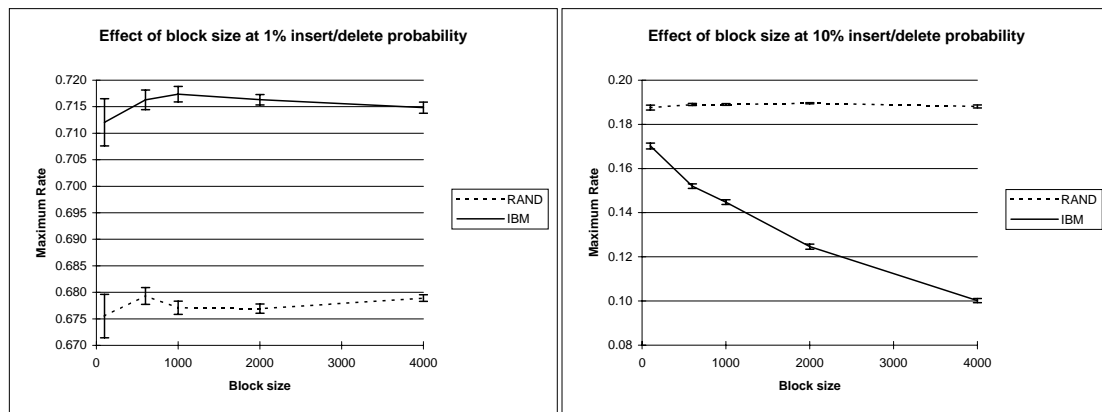
The envelope of the best uniform codes with IBM and random headers took the following form:



Some of the best header arrangements were:

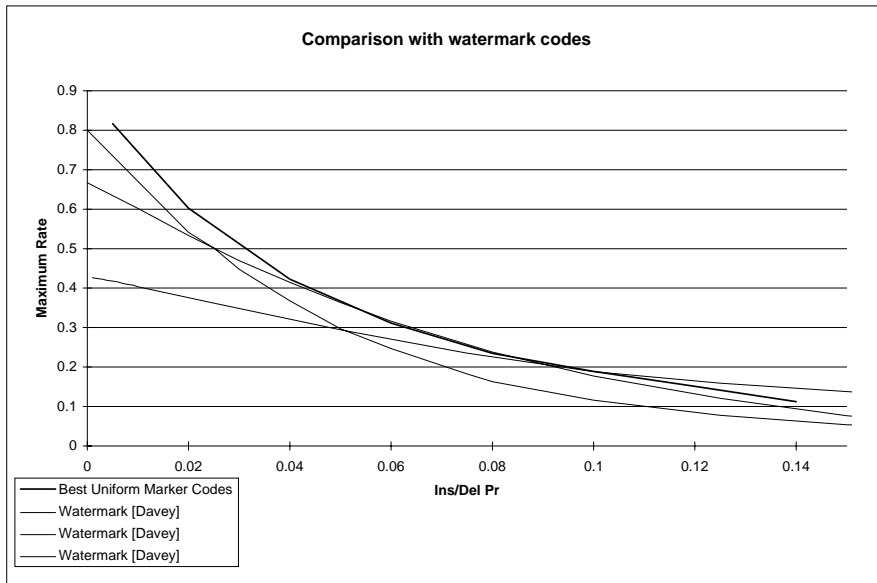
Insert Delete Probability	Repetitive Header Structure (. =data bit, R=random bit)
0.5%01RR
2%01RR
4%0011 ..R
6%001 ...RR
14%0000111 ..RR

The dependence on block size of the best uniform codes was plotted out at 1% and 10% error rates:



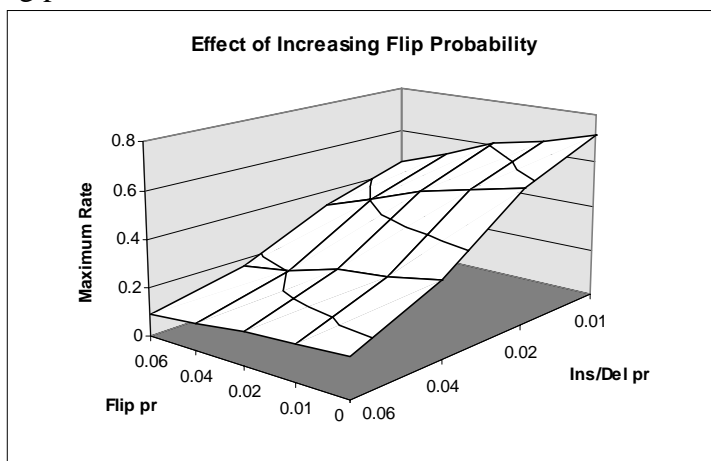
There is not a particularly large effect at low noise levels, however at higher noise levels one can see the performance of a code with the IBM style headers drops off with increasing block size while one with random headers is basically unaffected.

The best uniform marker codes were then compared against the watermark codes developed by Davey:

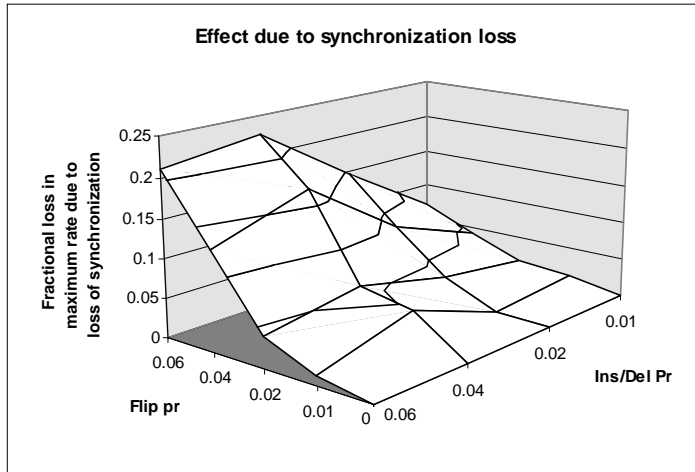


It can be seen that marker codes are performing better at low error rates and watermark codes are performing better at high error rates.

Adding a probability of flips occurring was then tried. For one particular code the following performance was obtained:

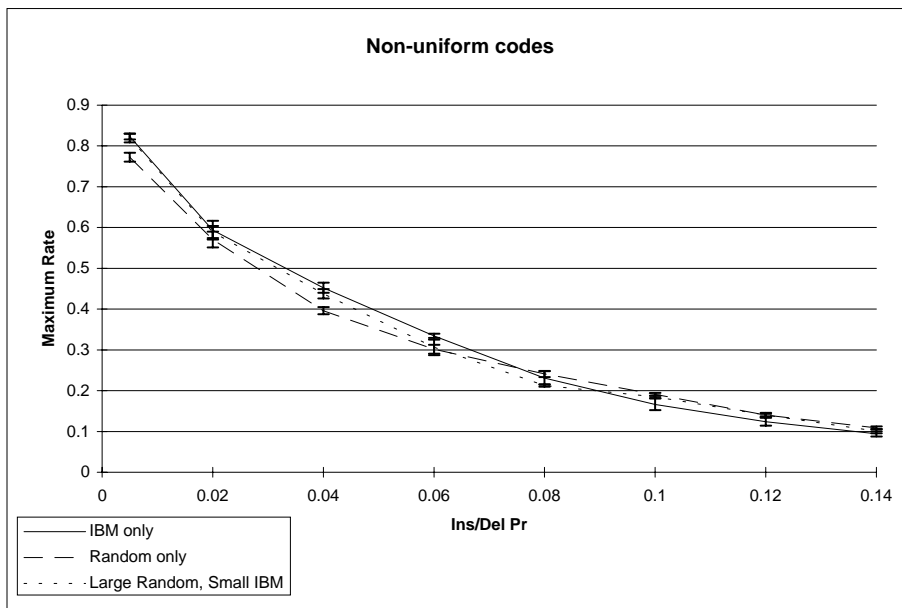


As expected it can be seen that the maximum rate falls off as the flip probability increases. The data becomes more meaningful though if we compare the maximum rate to the capacity of a binary symmetric channel in series with the effective channel with $p_{flip}=0$ (in other words the maximum rate possible if flips did not affect the synchronisation process). The change in maximum rate is:



It can thus be seen that flips are affecting the level of synchronisation of the channel and that at higher insertion/deletion probabilities the maximum rate is affected more.

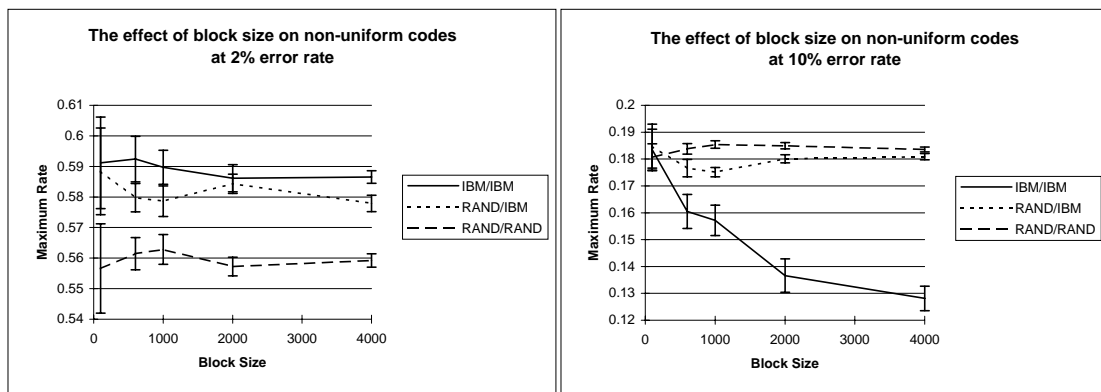
Then using a combination of small and large headers was tried. Headers combinations were studied that had the repetitive form of a fixed number of small headers followed by a large header, with a constant amount of data between them all. Using all random and all IBM headers were tested and also large random headers combined with small IBM ones. The envelopes of the best codes was found to be:



The header structure for some of these were:

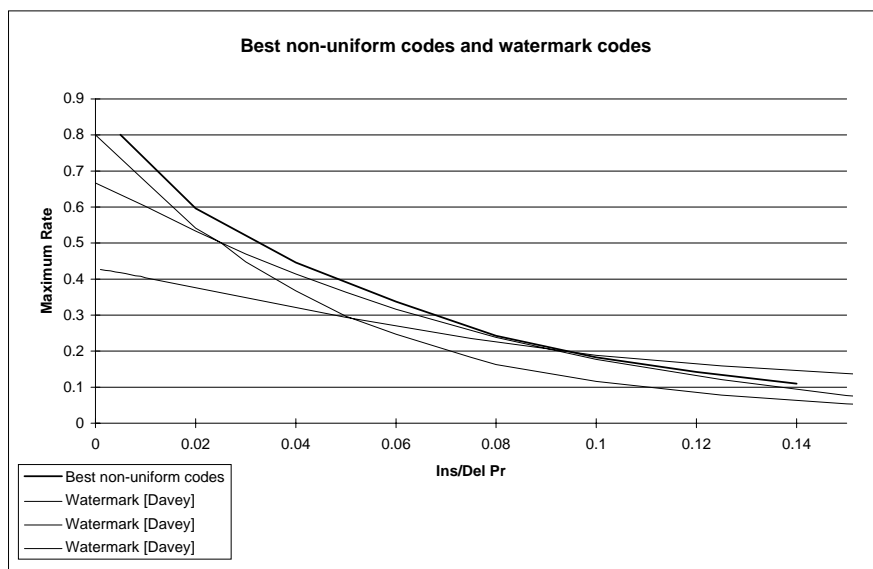
Insert/Delete Probability	Repetitive Header Structure
0.5%01
2%01.....01.....01.....001
4%01.....01.....01.....001
8%RR....RR....RR....RRR
12%	..R..R..RRR

As before the effect of changing block size was studied - the following was found:



It can be seen that the maximum rate for a code with IBM headers only at high error rates is the only one noticeably affected by block size. However the effect is less than with uniform codes.

A comparison of the best non-uniform codes found with watermark codes is shown below:



Discussion

Uniform codes

It can be seen that the IBM style headers are performing better at low noise levels and the random headers better at high noise levels. This is what one would intuitively expect. Random headers have the advantage that it is very hard to get out of sync by a chunk as there is more than one type of header - at high noise levels this is a more likely to be a problem. More evidence for this can be seen when the block size is varied as this also increases the probability of getting a chunk out of sync; it can be seen that a code with IBM style headers is affected but one with random headers is not.

However IBM style headers have the advantage that they always have a form whereby it is easy to tell if one bit (and possibly more) has been added or lost in the preceding data chunk [Sellers]. For example, with the three bit header 001, if a bit is lost in a preceding position this leaves 010 or 011 at the header position (depending on the bit following the header), both of which are indicative of an error having occurred. However if one is using random headers a particular header could be 000, if a bit is then lost beforehand we could see either 001 or 000 at the header position - for the latter we might conclude that no error had occurred. This will then lead to better low noise level performance for IBM headers.

When p_{flip} is non-zero it can be seen that in general the degree of synchronisation is affected as the maximum rate falls off faster than the binary symmetric channel in series with the effective channel created by the $p_{flip}=0$ channel. This effect is worse at high insert/delete probabilities, this is expected as under these conditions it is harder to maintain synchronisation and having the header bits yet further corrupted will make this more difficult.

Non-uniform codes

It can be seen that as the noise level increases, the spacing of the headers is reducing so as to probably prevent large sections of data being corrupted by a single error. Unlike with uniform codes though the size of each header is always remaining small. This is probably because with a non-uniform code we can place small headers regularly enough so that there is only likely to be one error between the headers - but there are still larger headers to maintain the large scale alignment if there are a lot of

errors all in similar positions. It is interesting to note that at very low noise levels a uniform code is offering the best performance.

When the block size is varied it can be seen that the maximum rate of a non-uniform code constructed from IBM headers only is affected by varying the block size, though not as significantly as for the uniform codes. This fits the same argument as for a uniform code, however with a non-uniform code the data is better protected by regular headers and hence the probability of this problem occurring should be smaller. When some random headers are introduced this problem disappears.

Large random headers combined with small IBM headers were tried as it was thought this might offer a good compromise between the two different header types. The IBM headers will be frequently spaced and should offer good protection to bits between them and the random headers should stop the problem of getting a chunk out of sync. Codes constructed in this manner give good performance across the entire range of noise levels studied but are outperformed by other codes at all points. It therefore appears that a code structure designed to be general purpose does not perform as well a header structure ideal for that noise level.

Constructing a good code

It can be seen that a code that performs well on a channel with one level of noise does not perform so well on a different channel. Therefore to make a good code one first needs to ascertain the noise characteristics of the communication channel and the block size to be used.

The simulation process used to find the best code can be quite processor intensive - it is therefore useful to note patterns in the codes found. First, all the best codes on the channels studied consist of small headers (3 bits or smaller). At $p_{ins}=p_{del}=0.5\%$ uniform codes (made out of IBM headers) are giving the best performance (and comparable performance up to $p_{ins}=p_{del}=2\%$) and therefore at low noise levels one can probably solely consider uniform codes with IBM headers to reduce the search space. Up to $p_{ins}=p_{del}=6\%$ non-uniform codes with IBM headers are performing the best, however the IBM codes are slightly affected by varying block size so if very large blocks are to be considered it may be necessary to consider both types of header. Above this non-uniform codes with random headers seem to be the best option. If computer time for the search is limited simulated annealing could possibly be used to find the maximum more rapidly.

Above about $p_{ins}=p_{del}=10\%$ watermark codes outperform the marker codes studied here and these should be used in preference. This better performance is probably due

to the fact that when the noise gets very large one needs to know synchronisation on a bit by bit basis. By using a watermark with very sparse data this is easier, as with marker codes we get no information about the synchronisation position between headers and effectively have to interpolate this between them.

Future Directions

There is a need for theoretical modelling of both the channel and marker codes. For the channel model studied no firm result has yet been derived for the capacity [Ullman]. If a result for the capacity existed this would then provide a good benchmark to measure marker codes against. Also a theoretical model of the functioning of marker codes could be developed. This work has shown that for certain regimes marker codes consisting solely of IBM style headers are performing the best. As these headers have a known form and properties it would hopefully not be difficult to develop this theoretical model. It would then be interesting to see how this model compares with the performance shown by the experimental decoding.

During the transmission of headers we are not transmitting any data which might be wasteful. It may be possible to increase the data transmission rate by, for example, having two different header types and the choice of header is then used to transmit a bit of data. When the headers are very small it seems likely that this would not work very well as it would be hard to distinguish the header in the corrupted bit stream, however when large headers produce a good code this seems more likely to work. Unfortunately it may happen that at high noise levels it would be easy to get a chunk out of sync with the decoding as each header would not be different in a manner known a priori to the receiver. The different headers could be chosen to be codewords from a Levenshtein code [Ferreira], this means that all headers would allow certain errors between them to be detected in a similar manner to the IBM headers used in this project.

The model of channel used is memoryless and hence does not exhibit bursts of errors or the run-length problems common in many every-day channels. Also the insertions in the model used are random, in many real channels losing synchronisation would give insertions that are related to the bits around them. One would expect all this to increase the capacity of the channel over the equivalent memoryless channel and it would be worthwhile investigating the size of this effect on marker codes and also to see whether a different type of marker code maximises the maximum rate of reliable communication under these conditions.

It would also be interesting to obtain graphs of block error rates for these codes so it can be seen how well they perform at noise levels near the ones presented here. To do

this an outer code would have to be implemented; a Gallager code or derivative [MacKay] would probably be a good choice as it reaches near the Shannon limit for all the effective channels used in this project.

The current decoding process is not very fast. The simulation used here is written in Modula-3 and can take up to 20 seconds to resynchronise a 4000 bit block under high noise conditions on modern computers - it could probably be hard coded in assembler without too much difficulty for modern SIMD (vectorised floating-point operations) architecture computers (e.g. one based on Intel's Pentium III processor) to significantly reduce the decoding time. However the current algorithm is $O(N^{1\frac{1}{2}})$ and thus a new one might need to be found if one wants to deal very large data blocks in reasonable time. Seller's original decoding algorithm is $O(N)$ and it may thus be worthwhile comparing the performance of this with one based on the forward-backward algorithm.

There is also the question of whether the decoding strategy presented is truly giving the maximum rate (in other words whether it is an optimal decoding strategy). By dividing the process into a synchronisation step followed by a more conventional decoder we could be losing information. A useful study may then be the development of a decoder that operated in one step rather than two.

Conclusion

The project has experimentally shown that marker codes allow reliable communication over insertion-deletion channels at a finite rate. Both uniform and non-uniform marker codes have been studied and similar maximum rates of reliable communication with them both have been found at error rates up to $p_{ins}=p_{del}\approx 2\%$. At noise levels above that non-uniform codes outperform uniform codes. When compared with watermark codes better performance has been found for error rates less than $p_{ins}=p_{del}\approx 8\%$. Hence marker codes should continue to be useful in many applications involving multiple insertions and deletions in a single block. Despite being developed before the era of extensive computation, this project has shown that Seller's work from 1962 still offers a good solution to communication over insertion-deletion channels.

References

- P.A.H. BOURS *Codes for Correcting Insertion and Deletions Errors* PhD thesis, Eindhoven Technical University, June 1994.
- M.C. DAVEY *Error-correction using Low-Density Parity-Check Codes* PhD thesis, Cambridge University, December 1999
- R. DURBIN, S. EDDY, A. KROGH, G. MITCHISON *Biological Sequence Analysis* CUP 1998
- H.C. FERREIRA, W.A. CLARKE, A.S.J. HELBERG, K.A.S. ABDEL-GHAFFAR, A.J. HAN VINCK *Insertion/Deletion Correction with Spectral Nulls* IEEE Transactions on Information Theory, 43(2):722-732, March 1997
- S.W. GOLOMB, B. GORDON, L.R. WELCH *Comma Free Codes* Canadian Journal of Mathematics, 10(2):202-209, 1958
- V.I. LEVENSHEIN *Binary codes capable of correcting deletions, insertions and reversals* Soviet Physics - Doklady, 10(8):707-710, February 1966
- D.J.C. MACKAY *Good error correcting codes based on very sparse matrices* IEEE Transactions of Information Theory, 45(2):399-431, 1999
- F.F. SELLERS, JR. *Bit Loss and Gain Correction Code* IRE Transactions on Information Theory, 8(1):35-38, January 1962
- J.D. ULLMAN *On the capabilities of codes to correct synchronisation errors* IEEE Transactions on Information Theory, 13(1):95-105, January 1967

Acknowledgements

This project was supervised by Dr MacKay who showed unfailing enthusiasm for this work and was a constant source of inspiration.

I would also like to thank Christ's College, Anton Altaparmakov, Richard van der Hoff, Matthew Bentham and Adam Johansen for providing computer time without which the amount of experimental work possible would have been severely limited.

I am also most grateful to Olaf Wagner for promptly fixing a bug in Modula-3's random number generator that was discovered when this project was started.

Appendix: Codes Used

The codes used for the graphs in the Results section where not previously stated were:

Graph	Repetitive Header Structure
Comparison of Different effective channels0011
Effect of block size at 1% insert/delete probability01RR
Effect of block size at 10% insert/delete probability00011 .R
Effect of Increasing Flip Probability01
The Effect of Block Size on a non-uniform code at 2% error rate01.....01.....01.....001RR01.....01.....01.....RRRR
The Effect of Block Size on a non-uniform code at 10% error rate001....001....00011 .R001....001....RRR