Marker Codes for Channels with Insertion, Deletion and Substitution Errors

Daniel Marco

PhD Qualifying Examination Part II
May 2001

Contents

Abstract

In channels where insertion and deletion errors might occur, synchronization becomes an important issue. This report suggests a way to tackle such errors (in addition to the usual substitution errors that can occur). We introduce a block code that is capable of correcting multiple insertions, deletions and substitution errors. The block code consists of an inner code whose purpose is to maintain synchronization (this is obtained by means of markers) and an outer code, which provides the error correcting capability. Theoretical analysis of the performance of such a block code is made and simulation results are presented as well.

I. Introduction

A. Problem statement

Many different kinds of codes have been devised to combat the effect of noise. Most of these codes are capable of correcting a certain number of substitution errors, in which the transmitted symbol is replaced by another symbol, i.e. 1->0 or 0->1 in the binary case (which we will use). When a fixed block length code is used over a channel that produces these types of errors, there is no synchronization problem. That is, the decoder knows where each received bit belongs in the original message transmitted. This however, is not the case when the channel used for transmission produces insertions or deletions. An *insertion* is the event in which a bit, which was not transmitted, is detected at the output. A *deletion* is the event in which a bit, which was transmitted, is not detected at the output. For example, suppose that the sequence 0001100 were transmitted. If a 1 were inserted between the second and third bits of the original sequence, we would receive 00101100. If the fourth bit were deleted, the received sequence would be 000100. As we can see, in both cases the received sequence is of different length than that of the transmitted sequence. It is this fact that causes a synchronization problem. One may observe that an equal number of insertions and deletions results in a received sequence that is of the same length as that of the transmitted one. Such an event is equivalent to having only substitution errors. However, one cannot expect a channel that produces insertion and deletion errors (together referred to as *synchronization errors*) to always produce the same number of these types of errors for each codeword. Therefore, when dealing with a channel that has synchronization errors, one must be ready to deal with the varying length responses to transmitted codewords.

B. Previous work

Codes for insertion and deletion errors were first considered in 1965 by Levenshtein [1]. He obtained bounds on the number of codewords in a code capable of correcting any constant number (not fraction) of synchronization errors in a codeword, and suggested the use of strings of bits, called buffers, to serve as delimiters between codewords in a transmission of multiple codewords. He also introduced a new distance measure, which was called *Levenshtein distance* and denote by LD: for any two sequences x, y (of equal or different length), LD(x,y) is the smallest number of substitutions, deletions and insertions necessary to transform x into y. It is easy to verify that this is a metric. As in the case for the Hamming distance, if a code is capable of correcting up to e errors (substitution and/or synchronization) in any one word a, it is necessary for a to have Levenshtein distance at least 2e+1 from any other word. This is a minimum distance type of requirement that uses the Levenshtein distance as the distance metric. The probabilities of insertion, deletion and substitution errors are not considered. Tanaka and Kasai [2] generalized the Levenshtein distance and introduced a *Weighted Levenshtein Distance* (WLD), which gives different weights to the possible types of errors, thus taking into account the possible different values of probabilities of insertion, deletion and substitution. It can be shown that WLD is a metric when insertions have the same weight as deletions. Furthermore, WLD reduces to LD when insertions, deletions and substitutions all have weight 1, and reduces to Hamming distance when substitutions have weight 1 and insertions and deletions have infinite weight.

Subsequent work after [1] dealt mostly with various types of codes capable of correcting a limited number of synchronization errors with or without added substitution errors. Calabi and Hartnett [3] described codes parameterized by a positive integer $t \geq 3$ that are capable of correcting, in every t consecutive codewords, either one-substitution error in each one of, at most, t-2 codewords, or one synchronization error (but not both). The parameter - t, may be changed by the decoder to suit the channel conditions. It is chosen to be small or large depending on whether the channel error probabilities are high or low respectively. In order to decode any one codeword, their decoder needs to receive at most the next t-1 codewords. These codes were slightly generalized in [2].

Tenegolts [4] presented codes that can correct one deletion and a possible substitution error just before the deleted bit or 2 substitutions in adjacent bits. No synchronization is discussed, however, and the assumption made is that the boundaries of the codeword are known. Tenegolts [5] showed an interesting non-binary code capable of correcting one deletion or insertion in a codeword. The idea was to associate each non-binary codeword with an appropriate binary codeword according to some rule and having both satisfy a certain system of congruence. He then demonstrated that such a code is capable of correcting one deletion or insertion error. In fact, the idea of system of congruence was first introduced in [1] and was methodically employed in subsequent work [3], [4], [5], [8], [9].

Some work has been done in detecting synchronization misalignment. For example, comma-free codes, which were introduced to the coding community by Golomb, Gordon and Welch [6] are codes that have the property that if $x=x_1x_2..x_n$ and $y=y_1y_2..y_n$ are codewords, then none of the overlaps $x_i...x_n\ y_1...y_{i-1}$ $1 < i \le n$, is a valid codeword. Thus if an incorrect block alignment is chosen, the data do not form valid codewords. These codes allow the receiver to resynchronize after a synchronization error with a delay of at most two blocks of data. Prefix Synchronized codes (PS-codes) discussed in [7] belong to a sub class of comma-free codes, which have the property that if an insertion/deletion occurs in a codeword, and no errors occur in the next codeword, then it is always possible to resynchronize at the beginning of that next codeword. In all these cases, however, no correction is provided. Morita, Wijngaarden and Han Vinck [8] presented a class of PS-codes capable of correcting a single insertion or deletion in a codeword. Substitutions are not mentioned and the assumption made is that the decoder knows when the data stream of codewords started (that is – the decoding procedure does not start in the middle of the stream).

Kabatjanskii, Han Vinck and Wijngaarden [9] investigated special types of errors induced by the bit stuffing (BS) procedure. The BS procedure is used to prevent certain patterns from occurring in the bit stream. For example, in order to prevent the pattern 01^j0 from occurring, the BS procedure inserts a 0 after j-1 consecutive 1's. When BS is used in variable length codes, certain single substitution errors cause an additional insertion or deletion errors in the received data. Kabatjanskii, Han Vinck and Wijngaarden described a class of variable length codes that correct single insertion/deletion errors and single substitution errors as a result of BS. We should stress out that since variable length codes are very often used in data compression schemes the work presented in [9] might be useful in some source coding techniques.

C. General description of scheme presented

In this report we present fixed length block codes capable of correcting insertion, deletion and substitution errors. As previously mentioned, in channels where there are only substitution errors, synchronization poses little problem for fixed length block codes. In contrast, when insertion and deletion errors occur as well, synchronization becomes a serious issue. Block boundaries are no longer easy to detect, and thus it is no longer possible to simply hand the decoder one block after the other. The way we have chosen to tackle the synchronization problem is by introducing another layer of coding. Thus our system essentially encodes the data twice. The first encoding scheme is referred to as the "outer code" and the second as the "inner code". The purpose of the inner code is to cope with the synchronization errors, whereas the purpose of the outer code is to provide the error correcting capability. We use a Reed-Solomon code over GF(8) as the outer code and we use natural marker codes [9] as the inner code. The inner code is later improved in order to optimize performance.

The work presented here focuses mostly on the inner code and how to optimize its performance given that it works in conjunction with the specified outer code. We first introduce the "basic inner code", which is a code whose design does not depend nearly at all on the outer code. This code did not work. However, the concepts behind it were not without merit. By making slight modifications to the basic inner code we obtained a code, whose performance was more reasonable, which we call the "improved inner code". In order to optimize performance and maximize the effect of the combined redundancy of both the inner code and the outer code, the improved inner code and outer code's parameters have to be jointly designed.

The use of two layers of coding was also employed by Schulman and Zuckerman [10]. However, no performance curves or actual results were given. Recently Davey and MacKay [11] presented a block code aimed at correcting multiple insertion, deletion and substitution errors. The code they presented consists of three layers of coding: non-linear inner codes, which are called "watermark" codes, a Sparsifier code and low-density parity check codes over non-binary field. As in our case, their inner code is used for synchronization purposes. They modeled the received block as having been produced by a hidden Markov model (HMM) and thus used the forward-backward algorithm (a detailed description of which can be found in [12]) in order to decode the inner code.

When developing our block codes, one of our main goals was to keep the decoding procedure as simple as possible. As opposed to using a sophisticated HMM model which requires the backward-forward algorithm, we sought to find codes whose decoding procedure would be much more straightforward. In addition, the goal was to develop a system, which could be used in a "real life" situation where block boundaries have to be determined by the system itself, rather than having each block handed to the decoder by some omnipotent observer. Since Davey and MacKay's work is the only one we found that copes with the same "real life" situation and provides actual results, we make all of our comparisons to their work.

In order to save the reader the suspense we state the end result here. The first goal of simplicity has indeed been achieved. The marker codes are simple to decode and the decoding complexity is of the order of NlogN, where N is the number of symbols in each data block (in our case N=255). As for their error correcting capabilities – the basic inner code does not work. However, with a relatively simple change, a useful code (namely the improved inner code) can be designed, and so the basic inner code sets the pattern for a potentially useful technique. Although the improved inner code is far better than the basic inner code, it still falls short of the performance achieved by the code presented by Davey and MacKay. Moreover, the analysis presented in this report indicated that it could never be made as good as Davey and MacKay's code. For this reason I abandoned it and switched to a new research topic last November.

D. Outline of report

This report is organized as follows: Section II provides an overall description of the system. It discusses the outer code and its error correcting capabilities and it provides a description of the structure of the basic inner code and its encoding and decoding procedures. In Section III a description of the channel model is given. In Section IV simulation results for the basic inner code are provided, and theoretical analysis of its performance is made. The section also discusses approximations that are made throughout this report. Since the basic inner code was a naïve first attempt, which performed poorly in comparison to the performance obtained by Davey and MacKay, an improved version was developed and is presented in Section V. As in the case for the basic inner code, a decoding algorithm is provided and performance analysis is made. No simulations were made for the improved inner code but rather its performance is predicted via theoretical analysis. In Section VI we provide a short summary of this report and state our conclusions regarding the overall performance of marker codes for channels with insertion/deletion and substitution errors in comparison to the performance achieved by Davey and MacKay's watermark code. Section VII discusses future possible avenues of research that might produce further improvement.

II. System Description

A. Overview

A general block diagram of the system is shown in Figure 1. As described in the figure, the encoding procedure is composed of two sequential stages. Let a data symbol be a string of bits of some fixed length. In the first stage, K data symbols are mapped into N data symbols by the outer encoder, which uses an RS code for this purpose. Clearly N>K, since the outer encoder introduces some redundancy in order to obtain error-correcting capability. In the second stage the inner encoder places markers among the N data symbols. Markers are strings of bits, which contain synchronization information about the data symbols. Once the original K data symbols pass the two encoding stages, we refer to the result as a "block". These two stages are repeated for every K data symbols in the data symbol stream. Therefore, the encoding procedure for the whole data symbol stream is a sequential one: a group of K data symbols are encoded (using the two encoding stages) each at a time, forming a sequence of blocks.

At the receiver end, the decoding procedure is also composed of two stages, which are the reverse operations performed at the encoder end and are applied in reverse order. The received encoded blocks are received as a continuous string of bits; therefore decoding block-by-block requires identifying block boundaries. This is obtained using the inner decoder whose purpose is to estimate those boundaries. Once the block boundaries have been determined the inner decoder produces a reproduction of the inner encoded N symbols. These N symbols are then passed on to the outer decoder, which produces a reproduction of the original K data symbols that were originally encoded. An essential requirement for this procedure to work is for the inner code to have the ability to maintain synchronization, i.e. to estimate block boundaries well, and for the outer code to have sufficient error correcting capability, i.e. to be able to cope with corrupted reproductions of the transmitted N data symbols.

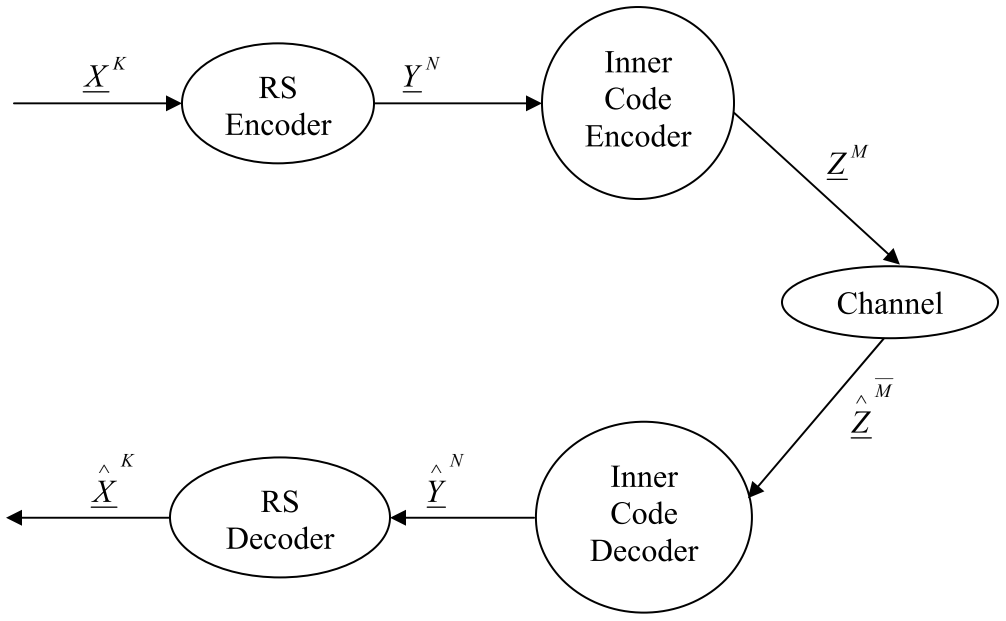

Fig. 1: Block diagram for the whole procedure. $\underline{X}^N$ denotes K data symbols. These are passed through the outer encoder (i.e. the RS encoder) to produce a block of N symbols: $\underline{Y}^N$. The inner code encoder then encodes the N data symbols and the result $\underline{Z}^M$ (referred to as a "block") is sent through the channel. M denotes the number of bits in the block. The received block $\hat{\underline{Z}}^{\overline{M}}$ may not be of the same size as the transmitted one! Thus $\overline{M}$ may not equal M. The encoding operations are reversed on the receiving end. $\hat{\underline{Y}}^N$ and $\hat{\underline{X}}^K$ denote the reconstructed versions of $\underline{Y}^N$ and $\underline{X}^K$ respectively.

B. The outer code

The outer code that is being used for each block is a Reed Solomon (RS) code over GF(8). Thus eight bits represent each data symbol. Let A denote a sequence of bits. Then we denote by $|A|$ the length of the sequence. Thus, if we let D denote a data symbol, then $|D|$ is its length (and in our case $|D|$ = 8). The relationship between the length of the data symbols and the parameter N is given by: N=$2^{|D|}$ -1.

In this code the redundancy is obtained by coding K data symbols into N data symbols. As in the case for binary codes, the code can correct up to $\lfloor (d_{\min}-1)/2 \rfloor$ erroneous symbols, where d_{min} is the minimum number of symbols in which any two blocks of N symbols differ. It can be shown that for any code $d_{\min} \leq N-K+1$. RS codes attain the maximum value for d_{min}, and thus they are capable of correcting up to $\lfloor (N-K)/2 \rfloor$ erroneous data symbols.

When decoding a block of N symbols, some symbols may be corrupted and some may be unknown. An unknown symbol is referred to as an *erasure.* A data symbol that is considered an erasure no longer has either a positive contribution (i.e. correct data symbol), or a negative contribution (i.e. an erroneous data symbol) towards decoding the block of N data symbols. An RS code that codes K data symbols into N data symbols can decode correctly a block of N symbols that has up to (N-K) erasures. Clearly, we might have cases where some symbols are incorrect and some are erasures. With the above in mind we obtain the following error/erasure correcting capability of the outer code:

$$2 * (\#errors) + (\#erasures) \leq N - K \tag{1}$$

The rate of the outer code is

$$\mathrm{R_{out}} = \frac{K}{N} \tag{2}$$

C. The basic inner code

1. Description

The basic inner code is a code whose redundancy is in the form of markers, which are added to the stream of data symbols. This code is in fact a natural marker code [9], that is - a marker is placed before every data symbol. Each marker consists of two parts: The flag, denoted by 'F', and the marker number, denoted by 'M#'. The purpose of the flag is to identify the marker as a marker rather then a part of the data symbols stream. The marker number is used to index the symbol that follows the marker. Since there are N symbols in a block of data symbols, we require $\lceil \log_2 N \rceil$ bits to index each symbol, thus $|M\#| = \lceil \log_2 N \rceil$.

The flag that we use is a string of bits set to "1". Naturally, we must make sure the flag does not appear accidentally as part of the data symbols or as part of the marker number, therefore we require $|F| = \max(|M\#|, |D|) + 1$. We further require that the flag not appear in the wrong position, thus it is not enough for the data symbols to simply not contain the flag. For example, a symbol that ends with a '1' would cause the flag to appear earlier. Similarly, a marker number that starts with a '1' would cause the flag to appear in a later position (should the data block be scanned from end to start). Also, a marker number that ends with a '1' would produce a whole new flag when concatenated with a data symbol that begins with a sufficient run of 1's. In order to solve these problems we use zeros as delimiters between data symbols and flags, flags and marker numbers and marker numbers and data symbols.

A visualization of the basic inner code is given in the following figure:

	...	F	0	M#	0	D	0	F	0	M#	0	D	...	

Fig. 2: The layout of the basic Inner Code. D denotes a data symbol (passed from the outer code), F denotes the flag, M# denotes the marker number. Zeros serve as separators. The total number of sets of the form: "F 0 M# 0 D 0" per block is N.

The rate of the basic inner code is then given by:

$$R_{in} = \frac{|D|}{|F| + |M\#| + |D| + 3} \qquad (3)$$

2. Decoding algorithm

The inner decoder is viewed as a device, whose input is a bit stream from the channel and whose output is a sequence of blocks of data symbols. Each output block consists of N data symbols, which are supposed to match the ones generated by the outer encoder. Since the outer code's error correcting capabilities are well known, there is no need to actually decode these blocks further for the purpose of analyzing the performance. The inner decoding process is a sequential one – the inner decoder produces one decoded block at a time. Thus, special care has to be given to identifying block boundaries. Since the channel has insertion/deletion errors, the channel output in response to encoded blocks may vary in length. An analysis of this length is provided in Section III.

The inner decoder has two modes: In the first mode it assumes that it knows approximately the starting position of the next block in the received bit stream, and E bits before that position is the position from which the next block is actually read (where E is some number to be discussed shortly). The actual length of the block is not known exactly, due to possible insertion and deletion errors. Therefore in order to consider all the bits that are associated with the block, the number of bits that is read is bigger than the expected length of a received block. A number of E extra bits are read on either side of the block to make sure all the bits that belong to the block are read. Thus if BL where to denote the expected length of a received block, then when reading the next block from the bit stream, BL + 2E bits are read from the position where next block should be read. Evidently, bits on the edges of blocks are read twice: once for the block before and once for the block after - but this does not pose any problem. An illustration is given in the following figure:

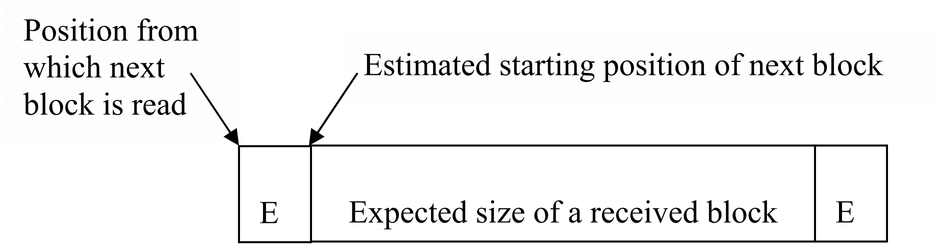
Position from which next block is read
Estimated starting position of next block
E
Expected size of a received block
E

Fig. 3: From the estimated starting position of the next block, E bits are subtracted and the new position is the position from which the next block is read. The total number of bits read are thus the expected number of bits in a received block + 2E.

In the second mode, the inner decoder does not know at all where the next block starts, and thus a more direct technique (referred to as "reset") is used to identify its starting position. Keeping the above in mind, the actual decoding algorithm is as follows:

1. Mode 1 - read the next block from the bit stream
2. Find all the flags and the corresponding marker numbers
3. Find the Largest Increasing Subsequence of marker numbers (LIS)
4. If # of elements in LIS $\geq$ K then

   For each marker in the subsequence copy the corresponding data symbol as is. For the markers that are not part of the subsequence (at most N-K such markers), place erasures as the data symbols. Set the counter of consecutive initially corrupted blocks to 0, mark the location in the bit stream where the next block should be read from and go to step 1

   Otherwise

   Define the whole block as *initially corrupted* (as it is already beyond the decoding capability of the outer code). Increment the counter of consecutive initially corrupted blocks, and if a certain threshold has not yet been reached go to step 1, otherwise go to step 5
5. Mode 2 - mark the location in the bit stream where the next block should be read from using "reset" and go to step 1

The "reset" in step 5, is a procedure that is applied when a few blocks in a row are initially corrupted. When this happens, the decoder assumes that it is out of sync, and hence some more sophisticated resynchronization is applied. The bit stream is searched for two flags whose corresponding marker numbers are consecutive and in the range of N-3 to 2 (i.e. N-3, N-2, N-1, 0, 1, 2). Once these are found, the expected starting position of the block is extrapolated, by advancing forward or backward in the bit stream from the found markers to the point where the first marker of the block is supposed to be. The direction of the advancement, i.e. forward or backward, is determined by whether the marker numbers found are in the range N-3 to N-1 or 0 to 2 respectively. The advancing amount is calculated using the known length of the markers and data symbols. Once the expected starting position has been extrapolated, the decoder goes E bits backward in the bit stream, and the new position is the position from which the next block is to be read, so step 1 in the decoding algorithm would follow.

The reason for choosing the range of consecutive markers to be quite small, i.e. N-3 to 2 lies in the fact that the further away from the starting position of the block the extrapolation is done, the less accurate it becomes.

.

Notes:

- In step (2) finding the "corresponding" marker numbers means that the $|M\#|$ bits after the zero bit, which follows the flag, are considered the marker number. We are assured that a zero follows the flag, since the flag is considered as the last $|F|$ bits in a run of 1's. The same goes in step (4) when data symbols of corresponding markers are copied: one bit after the $|M\#|$ bits of the marker is skipped (so this skipped bit may be in error and this it may have no effect!) and the next $|D|$ bits that follow are regarded as the data symbol.
- In both (2) and (4) errors might occur in either the marker number or in the data symbol. An error in the marker number is not very likely if the marker number ends up being part of the LIS. An error in the data symbol (be it substitution or synchronization) is fairly possible, regardless, and we rely on the error correcting capability of the outer code in this case.

3. Complexity

The operations performed in the decoding algorithm are: reading a block from the bit stream, finding all the flags, finding the marker numbers and finding the LIS. The first two operations are linear in the number of bits. That is, given that B bits are read then the first and second operations have complexity O(B), the third is linear in the number of data symbols, i.e. O(N) where N denotes the number of data symbols in a block. The fourth has complexity O(NlogN). The following is explains how these complexities are obtained.

The fact that reading the bits has complexity O(B) is trivial. Finding the flags requires parsing through the read bits and keeping track of the number of consecutive 1's. Once a sufficient number of 1's is found followed by a zero, a flag is found. This operation has complexity O(B) as well, since the bits are parsed once, and the operation done for each read bit has constant complexity. Once a flag is detected, the appropriate bits that follow are considered the marker number. Thus, obtaining a marker number (after the corresponding flag was found) is done in O(1). The total complexity of obtaining all the marker numbers (assuming flags were already found) is thus O(N), where N is the number of data symbols in a block (note, that less than N markers might be found – however, this does not change the complexity analysis).

The most complicated operation is finding the LIS. This problem is a specific case of the more general problem of finding the Longest Common Subsequence (LCS) of two sequences of symbols. The general problem reduces to our problem by choosing the sequences as follows: the first sequence is the sequence of numbers from 1 to N, which are in ascending order and the second sequence is the sequence of marker numbers found. The LCS of these two sequences is the largest increasing subsequence marker numbers. A brute-force approach to solving the LCS problem is to enumerate all the subsequences of marker numbers and check each subsequence to see if it is also a subsequence of 1 to N. This approach is impractical since it requires exponential time. In [14] it is shown that the LCS problem can be solved using dynamic programming and the complexity of finding the largest increasing subsequence of N numbers can be reduced to O(NlogN). We should point out that although the number of marker numbers found differs from one block to another (due to errors), it is roughly N.

The overall complexity of the decoding algorithm is O(B) + O(NlogN). Since B is a linear function of N, the overall complexity then becomes O(NlogN).

III. Channel Model

The channel model we have chosen is the same as the one used by Davey and MacKay. The channel is a binary channel described by three parameters: P_i – probability of insertion, P_d – probability of deletion and P_s – probability of substitution. The string of bits to be transmitted can be viewed as if they are placed in a queue. At each time, the bit at the beginning of the queue is considered to be the *current* bit. The notion of the current bit is important when describing the response of the channel to the string of bits to be transmitted. The transmission of the current bit is as follows: let the k^{th} bit be the current bit. With probability P_i a random bit is transmitted (this is considered an insertion), and the k^{th} bit remains the current bit so that another attempt is made at transmitting the k^{th} bit. With probability P_d the current bit is deleted (i.e. ignored and the channel produces no response to it), and the $(k+1)^{th}$ bit becomes the current bit. With probability $P_t = (1-P_i-P_d)$ the current bit is transmitted and has a probability P_s of being inverted, and in either case the $(k+1)^{th}$ bit becomes the current bit. This scheme is shown in the following figure:

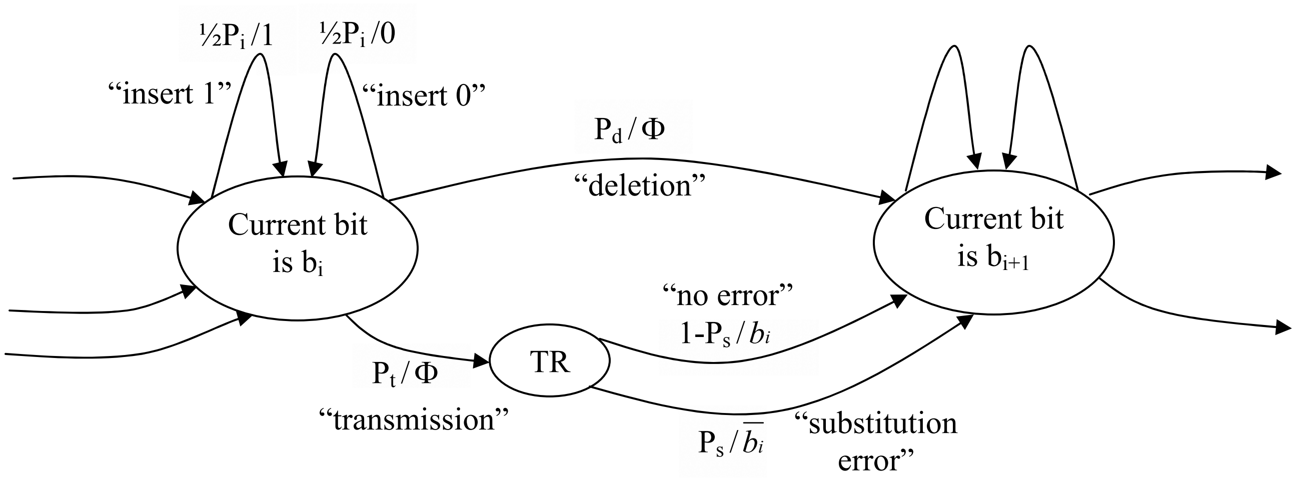

Fig. 4: State machine describing the Insertion/Deletion/Substitution channel with probabilities P_i, P_d and P_s respectively. Each transmission labeled with P/b , where P is the probability of that transmission and b is the bit produced. "Φ" indicates that nothing is produced.

Since we shall be using the channel model extensively it is important to understand how it behaves. However, for our purpose we only need to know $\Pr(z_1 z_2 \ldots z_n | z_1 z_2 \ldots z_n)$, i.e. the probability of receiving the sequence z_1 $z_2 \ldots z_n$ without an error. One way to have no error in the received sequence is if no error of any type (insertion, deletion or substitution) occurred in the transmitted sequence. The probability of such an event occurring is $\left[(1-P_i-P_d)(1-P_s)\right]^n$, that is, for every bit there was no insertion or deletion, nor was there a substitution. Clearly, there are other ways to obtain the transmitted sequence. Let an *error pattern* represent the errors that occur in a transmitted sequence. We then define a *valid error pattern* to be an error pattern whose errors "cancel each other" and thus results in an uncorrupted received sequence. An example of a valid error pattern is the following: An insertion in the first bit of a bit that has the same value as the first bit, followed by a deletion of the first bit and no other errors of any type in the rest of the sequence. Such a string of errors causes the received sequence to be identical to the transmitted one – hence it is a valid error pattern. The probability of this error pattern is: $\left(2^{-1}P_iP_d\right)\left[(1-P_i-P_d)(1-P_s)\right]^{n-1}$. The (n-1)th power represents having no errors in the last n-1 bits. The factor ½ is due to the fact that the inserted bit has equal probability of being either 1 or 0, and we require it to be the same as z_1 in order to obtain the original transmitted sequence. Similarly one can proceed in this manner and enumerate all the possible valid error patterns (there is a finite number of them). This however, turns out to be a lengthy operation that can be simplified by not counting all the valid error patterns, which have a negligible probability. In part E of Section V, we discuss such approximations and show their validity.

Let S denote a sequence of bits that is transmitted, and let R denote the corresponding received sequence. Then the mean and standard deviation of $|R|$ are given by:

- $E(|R|) = |\mathrm{S}| \cdot \dfrac{1-P_d}{1-P_i}$ (4)
- $\sigma(|R|) = \dfrac{1}{1-P_i}\sqrt{|\mathrm{S}| \cdot \dfrac{(P_i+P_d)}{(1-P_i)}}$ (5)

In order to find the expected length and standard deviation of the length of a received block, we simply replace $|S|$ by $|\text{Encoded Block}|$. These quantities are used in the decoding procedure of the inner code when block sizes and boundaries are estimated. The number of extra bits E read on either side of a block (as discussed in Section II part B.2) is taken to be a few standard deviations - $\sigma(|\mathrm{B}|)$, where B is the response of the channel to an encoded block. Thus the total number of bits read is determined by $E(|B|)$ and $\sigma(|\mathrm{B}|)$.

IV. Results for the basic inner code

A. Results

The outer code parameters we used were: $N = 255$, $K = 233$. As mentioned in Section II part B, N is a function of $|D|$. The results presented here are for $|D| = 8$, but upon examination of other values of $|D|$, we observed that the performance did not change significantly. As for K, the choice was somewhat arbitrary with the idea of not making K too small (so as to avoid too low a rate) or too big (so as to still have a reasonable error correcting capability). $|D|$ and N induce $|F| = 9$ and $|M\#| = 8$ (as explained in Section II part C.1). Given these parameters the outer rate, R_{out}, is 0.914, the inner rate, R_{in}, is 0.286 and the resulting overall rate of the code then becomes: $R = R_{out} \cdot R_{in} = 0.26$.

In Table 1 a summary of simulation results, as well as the corresponding theoretical predictions (discussed shortly) are presented. We should point out that all comparisons are made to the Davey and MacKay's code of rate 0.71. The 6th column of the table provides the probability of initially corrupted block. As we recall, a block is initially corrupted if the number of marker numbers in the largest increasing subsequence of marker numbers is less than K. In the 7th column, the probability of block error includes the cases in which the block is initially corrupted.

Pi	Pd	Ps	Frequency of Initially corrupted block	Frequency of block errors	Probability of initially corrupted block	Probability of block error	Davey& MacKay's frequency of block error
0.0020	0.0020	0.003	0.8254	1.0000	0.883285	0.999997	~0.01
0.0015	0.0015	0.003	0.6232	0.9996	0.675725	0.999888	~0.001
0.0035	0.0035	0.002	0.9755	1.0000	0.992138	1.000000	~0.1
0.0018	0.0018	0.002	0.4263	0.9987	0.515519	0.999490	~0.001
0.0030	0.0030	0.001	0.7123	0.9999	0.845610	0.999995	~0.01
0.00175	0.00175	0.001	0.0903	0.9723	0.148724	0.985224	~0.001
0.0033	0.0033	0	0.5195	0.9999	0.737231	0.999973	~0.01
0.0015	0.0015	0	0.0004	0.4446	0.002298	0.645192	~0.0001

Table 1: Simulation results of the basic inner code. 10,000 blocks were used for each case. The rate of the inner code is 0.286 and the rate of the outer code is 0.914. The overall rate is thus 0.26. Note: Davey and MacKay's code to which the comparison is made, has rate 0.71.

As we can see from the above table, the basic inner code is no match for the results obtained by Davey and MacKay and, in fact, it simply does not work. The probability of block error is nearly 1 for all the cases examined, except for the last one. Furthermore, the rate is 0.26 whereas Davey and MacKay's code rate is 0.71. We should point out that the probabilities of block error for Davey and MacKay's code are not precise since in their paper no exact results are provided, so these probabilities had to be estimated from their plots.

The basic inner code is improved in Section V, where an improved inner code is discussed. Although the improved inner code is also outperformed by Davey and MacKay's code, it nonetheless has a reasonable performance as opposed to the basic inner code.

B. Theoretical Analysis

There are two quantities that we are interested in, and would like to calculate beforehand: the probability of a block being initially corrupted and the probability of a block being erroneous. Before we proceed we would like to stress that the analysis made involves approximations throughout. These approximations are discussed in the next part of this section. Also, we will be calculating the probability of block being decoded correctly, rather than the probability of a block being erroneous, but the latter is obtained by subtracting the former from 1.

- Probability of block being initially corrupted

$$\Pr\,(\text{block initially corrupted}) \cong \sum_{i=N-K+1}^{N} \binom{N}{i} \cdot P^{i} \cdot (1-P)^{N-i} \qquad (6)$$

Where P is the probability of a marker being decoded incorrectly. We approximate P as follows:

$$P \cong 1 - \left[(1-P_i-P_d)\cdot(1-P_s)\right]^{(|M\#|+1)} \cdot \left[\left(1-\frac{1}{2}P_i-P_d\right)\cdot(1-P_s)\right]^{|F|} \cdot \left(1-\frac{1}{2}P_i\right) \qquad (7)$$

The term $\left[(1-P_i-P_d)\cdot(1-P_s)\right]^{(|M\#|+1)}$ stands for the probability of having no errors whatsoever in the marker number and the preceding zero. The term $\left[\left(1-\frac{1}{2}P_i-P_d\right)\cdot(1-P_s)\right]^{|F|}$ stands for the probability of the flag having no errors except for insertions of 1's (which is the reason for having a factor ½ before P_i). We allow insertions of 1's in the flag because they do not cause errors in the decoding procedure of the inner code (although the flag is in a sense erroneous). The term $\left(1-2^{-1}P_i\right)$ stands for not having an insertion of 1 in the zero that precedes the flag. Such an error might lead to a creation of a new flag, where there was none, if there is a sufficient run of 1's before. We allow an insertion of zero, a deletion or an inversion of the bit, since these errors do not cause errors in the decoding procedure of the inner code.

- Probability of block being decoded correctly

$$\Pr\,(\text{block correct}) \cong \sum_{S=0}^{N-K} \sum_{R=0}^{\left\lfloor \frac{N-K-S}{2} \right\rfloor} \binom{N}{S} \cdot P^{S} \cdot (1-P)^{N-S} \cdot \binom{N-S}{R} \cdot q^{R} \cdot (1-q)^{N-S-R} \qquad (8)$$

Where P is the probability of a marker being decoded incorrectly, q is the probability of a data symbol being incorrect, S is the number of erasures and R is the number of erroneous data symbols. The above is derived by keeping in mind (1). P is given by (7) and q is approximated as follows:

$$q \cong 1 - \left[(1-P_i-P_d)(1-P_s)\right]^{|D|+1} \qquad (9)$$

The reason for having $|D|+1$ as the power is that the preceding zero before the data symbol is considered part of the data symbol for this analysis.

Table 1 shows that the above analysis is quite close to the actual results obtained, both for the probability of block error and for the probability of a block being initially corrupted.

C. Approximations

A recurring theme in this report is that approximations are made. We make two types of approximations. The first type is the one seen in equation (6). I this case the binomial is an approximation as it does not take into account cases where a whole string of markers might be inserted somewhere within the block or other cases of this nature. These however, have negligible probability compared to the sum shown in (6). A more detailed discussion regarding the probability of events of this nature is discussed in the following second type of approximations.

The second type of approximations is the ones made when evaluating probabilities such as q in (9). The reason the expression in (9) is not an exact one, lies in the fact that any insertion/deletion or substitution is assumed to cause an error in the data symbol. This assumption is not true, although, we will show that it holds with very high probability – the result of which is that the approximation is a valid one. Examples where insertion/deletion errors do not corrupt the data symbol can be created as follows: Have no errors whatsoever for all the bits in the data symbol that precede bit b_i, then when trying to transmit bit b_i, first insert a bit that has the same value as bit b_i, then delete bit b_i and have no errors whatsoever in the rest of the bits of the data symbol. This chain of events leads to a data symbol, which is uncorrupted although errors have occurred. Hence a more precise formula for (9) would be:

$$q \cong 1 - \sum_{k=0}^{|D|+1} \binom{|D|+1}{k} \frac{1}{2^k} P_i^k P_d^k \left[(1-P_i-P_d)(1-P_s)\right]^{|D|+1-k} \qquad (10)$$

where k stands for the number of deletions and hence runs from 0 to $|D|+1$. The terms P_d^k and P_i^k are the probability for k deletions and insertions respectively. The term 2^{-k} is the probability that the insertions produced the same bits as the deleted ones. The term $\left[(1-P_i-P_d)(1-P_s)\right]^{|D|+1-k}$ is the probability of having no errors occur for the remaining $|D|+1-k$ bits. Our claim is, that this sum can be approximated by its first term. The reason for this lies in the fact that as k gets larger the corresponding term in the summation becomes very small due to $2^{-k} P_i^k P_d^k$. For example, for $P_i = P_d = 0.0035$ and $P_s = 0.002$, which are the highest values of P_i, P_d and P_s considered in this report, the second term (k=1) is slightly more than 5 orders of magnitude smaller than the first term. The third term is slightly more than 10 orders of magnitude smaller than the first term and so forth.

The reason why the subsequent terms become so small is due to the fact that the terms $(1-P_i-P_d)$ and $(1-P_s)$ are close to 1 when P_i, P_d, and P_s are small and thus their product does not increase much when they are raised to the (n-k)th power rather than to the nth power. However, the additional multiplying factor $2^{-k} P_d^{\,k} P_i^{\,k}$ is very small and thus reduces the whole term significantly. The multiplying factor $\binom{|D|+1}{k}$ is far too small to make a difference. As one can see, the smaller the probabilities (P_i, P_d, P_s) the better the approximation. The actual numbers provided above are for the highest probabilities we consider and hence stand for the worst case. Therefore (9) is clearly a valid approximation of (10).

As stated, (10) too, is an approximation. It does not take into account the possible substitution errors and possible deletion and insertion errors that together result in an uncorrupted data symbol. As discussed in Section III an error pattern represent the errors that occur in a data symbol, and a valid error pattern is an error pattern whose errors cancel each other out resulting in an uncorrupted symbol. From the above we conclude that the greater the number of errors in a valid error pattern, the smaller the probability for the error pattern to occur. The reduction, as we have seen, is very rapid.

We conclude from this discussion that the probability of any valid error pattern is negligible compared to the probability of the trivial valid error pattern of no errors. It can thus be ignored so as to simplify the expressions. In the remainder of this report, we shall make this approximation time and again. In all such cases, the same analysis as above applies.

V. The Improved Inner Code

A. Description

The improved inner code is very similar to the basic code in the fact that it uses the same marker structure. The difference between the two lies in the density of the markers. In the basic inner code a marker is placed before every data symbol. This significantly reduces the rate of the inner code. Furthermore, the many redundant bits are themselves susceptible to channel errors, which might, in a sense, cause more damage than good. That is the reason for contemplating the concept of an improved marker code, the idea behind which is to place markers every certain number of data symbols, denoted as L (see Figure 5).

Naturally there is a tradeoff. If a marker is damaged the whole set of L data symbols associated with that marker is regarded as L erasures. Hence the cost of a marker being in error increases significantly. On the other hand having fewer markers saves a lot of redundancy, which in turn may be transferred to the outer code, enabling a higher error correcting capability. It is thus clear, that the goal is to optimize the parameter L of the improved inner code.

Note: The basic inner code is in fact an improved inner code with L=1.

	...	D	0	D	0	D	0	F	0	M#	0	D	...	

	...	D	D	D	F	0	M#	D	D	D	F	0	...	

Fig. 5: Top: Improved Inner Code of type 'A'. Bottom: Improved Inner Code of type 'B'. In both cases L=3.

There are two types of improved Inner Codes. Type 'A', in which a 0 is put after every data symbol in order to ensure that the flag does not appear accidentally as part of the data symbols and type 'B', in which redundancy is saved by not separating the data symbols with 0's. It is important to note, that this may cause the decoder to find flags were there are none. One can cope with this problem by keeping track of flag locations within the analyzed block, and since the number of bits between flags is approximately known, flags that do not appear in reasonable locations can be ignored.

Again, as before, there is a tradeoff between complexity and performance. The improved inner code of type 'A' has a lower rate, but its decoder is simpler, whereas the improved inner code of type 'B' has a higher rate, but requires a more sophisticated decoder (one that keeps track of flag locations).

These codes were not simulated, but rather their performance was predicted via theoretical analysis. Let us examine the rates of the inner codes of both types:

- For type 'A': $R_{Ain} = \dfrac{L \cdot |D|}{L \cdot (|D|+1) + |F| + |M\#| + 2}$ (11)
- For type 'B': $R_{Bin} = \dfrac{L \cdot |D|}{L \cdot |D| + |F| + |M\#| + 1}$ (12)

As can be seen, $R_{Bin} > R_{Ain}$, and as L gets larger the difference in rate becomes more significant. Furthermore, when L=1, R_{Ain} reduces to R_{in} of the basic inner code as given by (3).

B. Decoding algorithm

The improved inner code was not simulated, therefore we need only keep in mind a general idea of a possible decoding algorithm, so that theoretical analysis can be done. The decoding algorithm would be similar to the decoding algorithm of the basic inner code. One important place, where the two algorithms differ, is the counting of bits after the marker. While the decoding algorithm of basic inner code does not perform any such counting, the improved decoder does count. The reason for this is the "high price" of L data symbols being in error. Thus, bits are counted between every two flags, and if the number found is not the expected one, there is an erasure, and all the L data symbols associated with the marker, are considered erasures. When a type 'B' code is used, extra effort has to be put in discerning between accidental flags and real ones. This can be done, by keeping track of flag locations within the block. Since the number of bits between every two flags is approximately known, it can be used to determine which flags are "real" and which are not.

C. Complexity

The decoding algorithm of the improved inner code has to perform the same operations as those performed by the decoding algorithm of the basic inner code, and has to perform some additional operations as well. Before considering the additional operations, we examine the complexity of performing the operations that are common to the decoding algorithms of both the basic inner code and the improved inner codes. As mentioned in Section II part C.3, the decoding complexity of the basic inner code is O(NlogN), where N is the number of data symbols in a block. This was due too the complexity of finding the largest increasing subsequence of marker numbers and N was the number of marker numbers that had to be considered. In the case of the improved inner codes, the expected number of marker number is no longer N, but rather $\left\lceil \frac{N}{L} \right\rceil$. Therefore the complexity of finding the largest increasing subsequence of marker numbers becomes $\mathrm{O}\left(\frac{N}{L}\log\left(\frac{N}{L}\right)\right)$.

One additional operation that has to be performed by the decoder of the improved inner code is the counting of bits after the marker number, the complexity of which is linear in the number of bits in the block. The decoder for the improved inner code of type B, has to also keep track of flag locations within the block and determine whether a flag that is found is indeed a "real" flag or whether it was accidentally formed. Although we have not fully analyzed the complexity of such an operation it does not seem like it would require a larger amount of computation than finding the largest increasing subsequence of marker numbers.

The overall complexity of the decoding algorithm for the improved inner codes is therefore $\mathrm{O}\left(\frac{N}{L}\log\left(\frac{N}{L}\right)\right)$, which is in fact lower than the complexity of the decoding algorithm of the basic inner code! We should stress out that this does not mean that the decoders for the improved inner codes are simpler in terms of implementation (they are not), but rather the total amount of calculations (i.e. complexity) that they perform is smaller. This is because the dominating factor in the amount of calculations performed is finding the largest increasing subsequence of marker numbers, of which the improved inner codes have less.

D. *Theoretical analysis*

The quantity that we seek is the probability of a block being decoded correctly. As before, we make some approximations, which do not take into account all possible cases, but rather the most likely ones.

$$\Pr(\text{block correct}) \cong \sum_{S=0}^{\left\lfloor \frac{N-K}{L} \right\rfloor} \sum_{R=0}^{\left\lfloor \frac{N-K-S\cdot L}{2} \right\rfloor} \binom{N/L}{S} \cdot P^S \cdot (1-P)^{\frac{N}{L}-S} \cdot \binom{N-S\cdot L}{R} \cdot q^R \cdot (1-q)^{N-S\cdot L-R} \tag{13}$$

Where P is the probability of having an erasure (which is in fact L erasures). The event "erasure" happens when either the marker is decoded incorrectly or the L data symbols do not have the right number of bits. q is the probability of a data symbol being incorrect given that L data symbols have the right size. The above expression reduces to (8), when L=1, P is the probability of the marker being decoded incorrectly or that the following data symbol does not have the right size and q is the probability of a data symbol being incorrect, given it has the right size. It is important to notice that (12) holds for both type 'A' and type 'B'. The differences between the two come into play when approximating P and q. Note that if N/L is not an integer then there are some edge effects, which is one of the reasons for the having the "$\cong$" sign. These effects, however, make little difference due to the fact that L<<N.

In order to calculate P and q we need to evaluate first the probability that the length of a sequence of bits remains the same after the sequence is transmitted. For a sequence whose length is m, we denote this probability by S(m), and obtain the following expression for it:

$$S(m) = \sum_{k=0}^{m} \binom{m-1+k}{k} \cdot P_i^k \cdot \binom{m}{k} \cdot P_d^k \cdot (1-P_i-P_d)^{m-k} \tag{14}$$

In order for m bits to have the right size, the number of insertions has to equal the number of deletions. Since there can be m deletions at the most, the sum runs from 0 to m. For a given k, there are k insertions and k deletions. Hence the terms P_i^k and P_d^k. The number of ways to have k insertions (where more than one insertion per bit may occur) is given by $\binom{m-1+k}{k}$. This expression is derived using the analogous occupancy problem of finding the number of ways to place k identical balls in m cells [13]. The number of ways to choose which bits are deleted is given by the term $\binom{m}{k}$. Since k bits are deleted, it means that (m- k) bits are not deleted. The only way to proceed to the next bit from a non-deleted bit (after perhaps having some insertions) is by having neither an insertion nor a deletion. Thus we have the term $(1-P_i-P_d)^{m-k}$.

S(m) can be approximated, however when doing so, one must pay special attention to the parameter 'm'. Upon investigation of the function S(m) we found that its terms are not monotonically decreasing. The terms first increase monotonically until they reach their peak, and then they decreases monotonically. Therefore, one cannot approximate S(m) by its first term. The rates with which the terms reach their peak and with which they decrease depend on m and on the probabilities P_i and P_d in the following way: the smaller the m and the smaller the probabilities the faster the increase and the decrease. In our case we found that for the largest probabilities that we considered and for m=9, S(m) can be approximated by its first term up to an accuracy of 10^{-3}. This approximation gets better as the probabilities P_i and P_d become smaller. The reason for stating the result for m=9 is that when considering S(m) for a single data symbol, m is either 9 or 8 depending on whether the system considered is A or B respectively. In this case we can then write:

$$S(m) \cong (1 - P_i - P_d)^m \tag{15}$$

For the case when m is a few thousands and for the largest probabilities P_i and P_d that we considered, approximating S(m) by the first 15 terms yields an accuracy of 10^{-8}. As before, making the probabilities P_i and P_d smaller or making m smaller, results in a better approximation. This case happens when considering L data symbols together. The largest m is then 2996 and is achieved for system A with L=255.

In order to approximate P we proceed with a couple more definitions:

- X is the event that a marker is decoded incorrectly
- Y is the event that L data symbols do not have the right number of bits

We will use X_A, X_B and Y_A, Y_B in order to refer to X and Y of the appropriate type. With these definitions in mind we proceed as follows:

$$P \cong \Pr(X \cup Y) = \Pr(X) + \Pr(Y) - \Pr(X \cap Y) = \Pr(X) + \Pr(Y) - \Pr(X) \cdot \Pr(Y) \tag{16}$$

where X and Y are independent events due to the fact that channel errors are independent and since X and Y deal with errors that occur at different locations. $\Pr(X_A)$ is given by (7), and $\Pr(X_B)$ is given by (7) with the term $(1 - 2^{-1} P_i)$ removed, since there is no zero before the flag in type B code. Care has to be given, however, when calculating $\Pr(X_A)$ and $\Pr(X_B)$ using (7) or a slightly modified version of (7) respectively. The number of bits in the marker numbers depends on the parameter L. So one has to plug in (7) or in the modified version of (7), $|M\#| = \left\lceil \log_2 \frac{N}{L} \right\rceil$ instead of $|M\#| = \lceil \log_2 N \rceil$.

The probabilities of the events described by Y_A and Y_B are given as follows:

$$\Pr(Y_A) = 1 - S\left(L\cdot\left(|D|+1\right)+1\right) \tag{17}$$
$$\Pr(Y_B) = 1 - S\left(L\cdot|D|\right) \tag{18}$$

Note that for type A code, the zero after the marker number is not considered part of the marker, and thus its affect is counted in the expression for $Y_{A,}$ rather than in the expression for X_A.

In order to calculate q (q_A, q_B – depending on the type), we make a few more definitions:

- W is the event that a data symbol is correct
- Z(L) is the event that L data symbols have the right size

We use W_A, W_B and $Z_A(L)$, $Z_B(L)$ to refer to W and Z of the appropriate type. Using these definitions we obtain:

$$q = 1-\Pr\left(W\middle|Z(L)\right) = 1-\frac{\Pr\left(Z(L)\middle|W\right)\cdot\Pr(W)}{\Pr\left(Z(L)\right)} = 1-\frac{\Pr\left(Z(L-1)\right)}{\Pr\left(Z(L)\right)}\cdot\Pr(W) \tag{19}$$

Where the last equality is due to the fact that if the symbol is correct it has the right size, therefore, the probability of the L symbols, including the correct symbol, having the right size is the probability of L-1 symbols having the right size, where the correct symbol is not one of them. Using the definition in (14) we thus obtain the following expressions for q_A and q_B:

$$q_A = 1-\frac{S\left((L-1)\cdot\left(|D|+1\right)+1\right)}{S\left(L\cdot\left(|D|+1\right)\right)}\cdot\Pr(W),\quad q_B = 1-\frac{S\left((L-1)\cdot|D|\right)}{S\left(L\cdot|D|\right)}\cdot\Pr(W) \tag{20}$$

Interestingly, (19) can be approximated in two different ways for the two possible extreme values of L. For large values of L, $\Pr\left(Z(L)\right)\cong\Pr\left(Z(L-1)\right)$ and we can approximate $q\cong 1-\Pr(W)$, which is simply the probability of a data symbols being incorrect. This expression is given by (9) for q_B, and if $|D|$ is replaced by $|D|+1$ in (9) then the expression for q_A is obtained. This result makes sense, since when L is large the conditioning $Z(L)$ gives little information about a given symbol and can thus be omitted in order to obtain the approximation.

For small values of L we can use the approximation in (15) to evaluate (20) and we obtain:

$$q_B \cong 1-\frac{\left(1-P_i-P_d\right)^{(L-1)\cdot|D|}}{\left(1-P_i-P_d\right)^{L\cdot|D|}}\cdot\left[\left(1-P_i-P_d\right)\cdot\left(1-P_s\right)\right]^{|D|} = 1-\left(1-P_s\right)^{|D|} \tag{21}$$

q_A is obtained by replacing $|D|$ by $|D|+1$. This result has intuitive meaning as well. When L is small, the conditioning on Z(L) can be regarded as knowing that the data symbol has the right size. Which means that the probability of a data symbol having the right size and not being correct is roughly the probability of having only substitution errors. We say it is ”roughly”, since in addition to using the approximation in (15) we also approximate Pr(W) using (9). The meaning of (21) can thus be captured in the following way: a data symbol that is incorrect and has the right size is most likely incorrect due to substitution error/errors rather then due to insertion and deletion errors that cancel themselves out. This is in a sense a restatement of the fact that once insertions/deletions occur it is “highly” unlikely that their affect would cancel out (even if the canceling out were in terms of symbol length only, since the difference in this case from the example discussed in Section III is just a factor of ½). This is illustrated in the following figure:

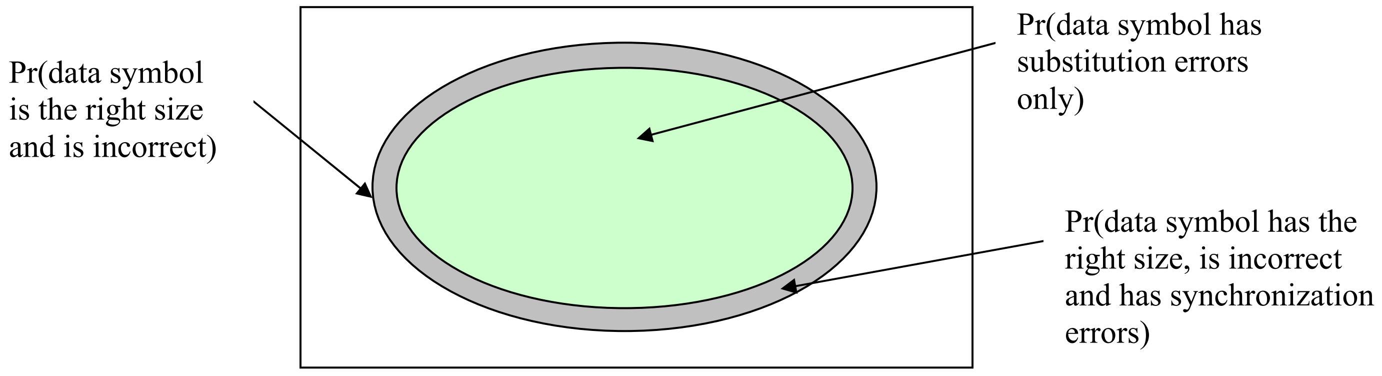
Pr(data symbol is the right size and is incorrect)
Pr(data symbol has substitution errors only)
Pr(data symbol has the right size, is incorrect and has synchronization errors)

Fig. 6: Illustrating the concentration of probability for a data symbol that is of the right size and is incorrect.

E. Predicted performance

This section examines the performance of the improved inner codes in two different approaches. In the first approach, the channel error probabilities that are used are the same as those used in Section IV (and by Davey and MacKay), and the probability of block error for different rates is examined. Thus we seek the rates for which codes of type A and type B achieve the same probability of block error as Davey and MacKay’s code of rate 0.71. In the second approach, the rate is fixed to be 0.71 (as in Davey and MacKay’s code), and the channel error probabilities are reduced so as to obtain the same block error probabilities as the ones obtained by Davey and MacKay’s code (for the non reduced channel error probabilities).

We would like to make two notes: 1. In all cases when we refer to rate we shall be referring to the overall rate of the outer an inner code combined and 2. The comparison made is always to Davey and MacKay’s code of rate 0.71.

The improved inner code has a few parameters, which affect its performance. These parameters are: N, K and L. Fixing the overall rate to a desired value and fixing N, enables us to optimize the performance by optimizing over L. K is then determined as a function of the overall rate, N and L. Since N is not very large (255 in the cases we examined), the easiest way to optimize over L is simply by trying all the possible L's. So the procedure is to calculate Pr(block correct) as in (13) for all L's and to choose the one that maximizes this probability.

We have done so for the channel parameters simulated in Section IV, for various overall rates. These results are summarized in the Table 2. As stated in Section IV, the probabilities of block error for Davey and MacKay's code are not precise since in their paper no exact results are provided, so these probabilities were estimated from their plots.

Pi	Pd	Ps	Rate	Optimal L for type A	Optimal L for type B	Probability of block error for type A	Probability of block error for type B	Davey& MacKay's frequency of block error
0.0020	0.0020	0.003	0.26	4	3	0.000000	0.000000	
			0.35	4	4	0.001863	0.000010	
			0.40	5	4	0.078864	0.001951	
			0.71	43	32	0.996061	0.974716	~0.01
0.0015	0.0015	0.003	0.26	3	3	0.000000	0.000000	
			0.35	4	4	0.000099	0.000000	
			0.40	5	4	0.013884	0.000123	
			0.71	43	19	0.990116	0.931956	~0.001
0.0035	0.0035	0.002	0.26	3	2	0.000060	0.000000	
			0.35	4	4	0.090203	0.003113	
			0.40	6	4	0.543751	0.079493	
			0.71	43	64	0.999409	0.991991	~0.1
0.0018	0.0018	0.002	0.26	3	2	0.000000	0.000000	
			0.35	4	4	0.000097	0.000000	
			0.40	4	4	0.014570	0.000110	
			0.71	43	19	0.994053	0.957975	~0.001
0.0030	0.0030	0.001	0.26	3	2	0.000000	0.000000	
			0.35	4	4	0.008819	0.000097	
			0.40	4	4	0.192153	0.008209	
			0.71	43	64	0.998825	0.988909	~0.01
0.00175	0.00175	0.001	0.26	2	4	0.000000	0.000000	
			0.35	4	4	0.000006	0.000000	
			0.40	4	4	0.002527	0.000008	
			0.71	43	11	0.993053	0.942017	~0.001
0.0033	0.0033	0	0.26	2	2	0.000000	0.000000	
			0.35	4	4	0.008402	0.000115	
			0.40	4	4	0.196976	0.008886	
			0.71	43	64	0.999081	0.990388	~0.01
0.0015	0.0015	0	0.26	2	4	0.000000	0.000000	
			0.35	4	3	0.000000	0.000000	
			0.40	4	4	0.000078	0.000000	
			0.71	43	11	0.987977	0.858856	~0.0001

Table 2: Probability of block error for improved inner codes type A and type B, and the optimal L that achieves the smallest probability. Four rates are examined: rate 0.26 - as for the basic inner code, rates 0.35 and 0.4 – which achieve the same probability of block error for codes type A and B respectively as Davey and MacKay's code and rate 0.71 as in Davey and MacKay's code.

As can be seen in Table 2, both codes of type A and B perform far better than the basic inner code. Both codes have good performance for the low rate of 0.26. At rates 0.35 and 0.4 codes of type A and B, respectively, attain the same block error probabilities as the ones attained by Davey and MacKay's code. At rate 0.71 both codes of type A and B perform poorly and have very high probability of block error. We further see from the table that type B code performs quite better than type A code and from now on we shall focus on type B codes.

In the following figure the performance of type B code is examined for rates that attain probability of block error close to that attained by Davey and MacKay's code.

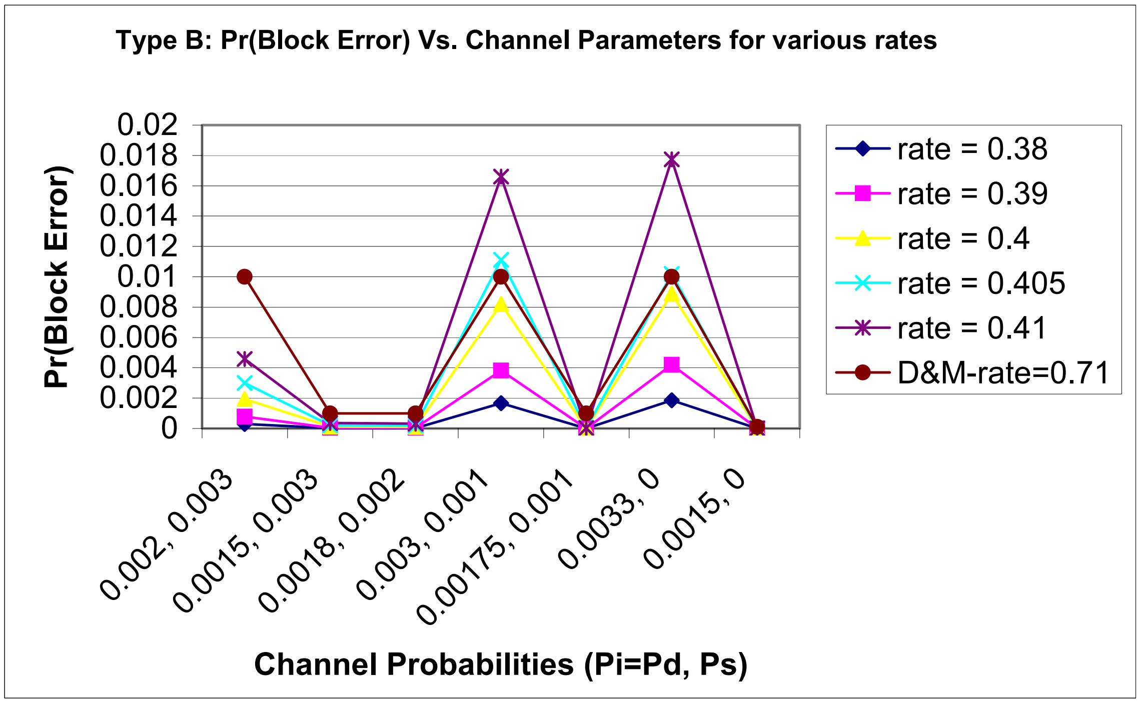

Fig. 7: The probability of block error for type B code for various rates in the range of 0.4 that achieve performance close (some slightly better and some slightly worse) to that attained by Davey and MacKay's code of rate 0.71.

As seen in the figure, type B code at rate 0.4, achieves about the same performance as Davey and MacKay's code. We should note that the X-axis in the above figure is used to simply state different channel error probabilities, and the connecting lines between the various points are shown so as give an impression of performance various rate. The X-axis does not have an order relation.

The point corresponding to channel error probabilities P_i=P_d=0.0035, P_s=0.002 is not plotted since the probability of block error at this point is much higher than the rest of the probabilities of block error and would not enable us to examine what happens at low probabilities of block error, thus rendering the graph useless.

As an alternate method of comparison of our code and Davey and MacKay's, we consider how much smaller must be the channel error probabilities (P_i, P_d, P_s) in order that for our type B code to attain the same block error probability as Davey and MacKay's code, where both codes have rate 0.71. With this in mind each row of Table 3 lists a particular set of channel error probabilities, a probability reduction factor (in bold), the block error probability of type B code and the resulting Davey and MacKay frequency of block errors. We found that among the several reduction factors tested, 17 was the best on the average and the block error probabilities of type B code corresponding to 17 are marked in bold.

Pi	Pd	Ps	Probability of block error for type B	Davey& MacKay's frequency of block error (for the original channel probabilities!)
0.0020 / **14**	0.0020 / **14**	0.003 / **14**	0.005303	~0.01
0.0020 / **17**	0.0020 / **17**	0.003 / **17**	**0.002689**	
0.0020 / **20**	0.0020 / **20**	0.003 / **20**	0.001503	
0.0015 / **14**	0.0015 / **14**	0.003 / **14**	0.00217	~0.001
0.0015 / **17**	0.0015 / **17**	0.003 / **17**	**0.001077**	
0.0015 / **20**	0.0015 / **20**	0.003 / **20**	0.000593	
0.0035 / **14**	0.0035 / **14**	0.002 / **14**	0.027027	~0.1
0.0035 / **17**	0.0035 / **17**	0.002 / **17**	**0.014549**	
0.0035 / **20**	0.0035 / **20**	0.002 / **20**	0.008487	
0.0018 / **14**	0.0018 / **14**	0.002 / **14**	0.003348	~0.001
0.0018 / **17**	0.0018 / **17**	0.002 / **17**	**0.001678**	
0.0018 / **20**	0.0018 / **20**	0.002 / **20**	0.000931	
0.0030 / **14**	0.0030 / **14**	0.001 / **14**	0.015658	~0.01
0.0030 / **17**	0.0030 / **17**	0.001 / **17**	**0.008233**	
0.0030 / **20**	0.0030 / **20**	0.001 / **20**	0.004723	
0.00175 / **14**	0.00175 / **14**	0.001 / **14**	0.002649	~0.001
0.00175 / **17**	0.00175 / **17**	0.001 / **17**	**0.001321**	
0.00175 / **20**	0.00175 / **20**	0.001 / **20**	0.000730	
0.0033 / **14**	0.0033 / **14**	0	0.019639	~0.01
0.0033 / **17**	0.0033 / **17**	0	**0.010426**	
0.0033 / **20**	0.0033 / **20**	0	0.006022	
0.0015 / **14**	0.0015 / **14**	0	0.001312	~0.0001
0.0015 / **17**	0.0015 / **17**	0	**0.000645**	
0.0015 / **20**	0.0015 / **20**	0	0.000353	

Table 3: Probability of block error for type B code. The rate is 0.71 for all entries. The channel probabilities that are examined are the ones examined in Section IV reduced by factors of 14, 17 and 20. The block error probabilities that achieve the required performance are set in bold.

As stated, a reduction factor of 17 enables type B code to achieve the same block error probabilities as those achieved by Davey and MacKay's code (both operating at rate 0.71). The values of Table 3 are plotted in Figure 8.

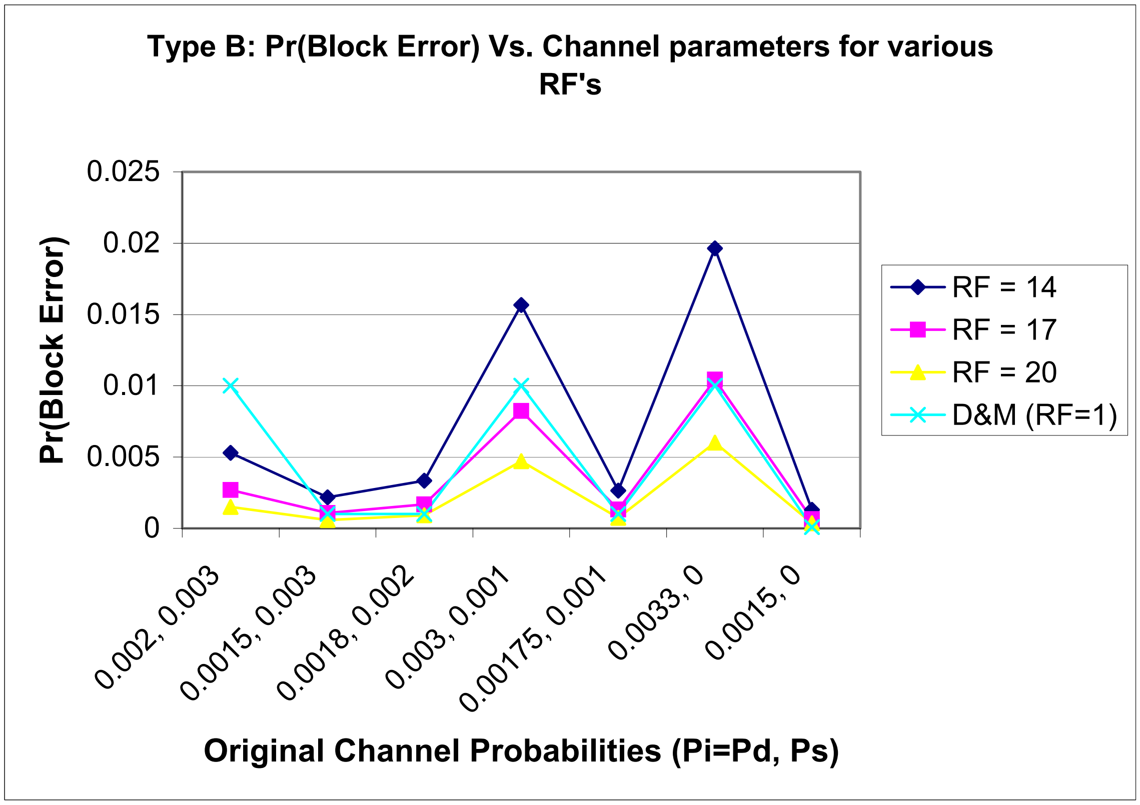

Fig. 8: The probability of block error for type B code with rate 0.71 for various RF's and Davey and MacKay's code applied with RF=1 (i.e. original channel error probabilities).

We should note that the X-axis in the above figure has the same meaning as the X-axis in Figure 7. Also, the point corresponding to channel error probabilities P_i=P_d=0.0035, P_s=0.002 is not plotted for the same reason as in Figure 7.

VI. Summary

Our purpose in developing the block codes described in this report was to obtain a code, which would have low decoding complexity and a performance comparable to that achieved by Davey and MacKay's code.

As far as complexity is concerned, the goal that we set was indeed met. The decoding complexity of the basic inner code is O(NlogN), where N is the number of data symbol in a block to be decoded and the decoding complexity of the improved inner codes is $O\left(\frac{N}{L}\log\left(\frac{N}{L}\right)\right)$, where L is the number of data symbols between markers. Furthermore the decoders themselves, in both cases, are fairly simple and straightforward.

The performance of our codes, however, was not competitive relative to Davey and MacKay's. The basic inner code, which was the first inner code we examined, did not work at all, and thus we had to make some modifications to it. Although these modifications have improved significantly the performance of our inner code, the improvement was still uncompetitive. We examined two types of improved inner codes, which we named 'type A' and 'type B'. Type B codes performed better than type A codes but at the cost of a somewhat more complicated decoder. Accepting the price of a more complicated decoder, we focused our attention on type B codes, in order to analyze the most optimistic case and see how well it performs.

The analysis of both the basic inner code and the improved inner codes consisted of approximations, which were shown to be valid. In the case of the basic inner code, the actual simulations that were conducted further supported those approximations.

In their paper, Davey and MacKay did not provide exact numbers that describe the performance of their code, thus these had to be approximated from the plots that were provided. Therefore, the comparison is not exact, however, the accuracy used was sufficient to capture the performance of our codes in comparison to theirs.

As mentioned the improved inner codes performed much better than the basic inner code. However, they too, fell short of the performance attained by Davey and MacKay's code. We thus quantified in two different ways how worse our codes performed. In the first way, we considered the same channel error probabilities for the improved inner codes and for Davey and MacKay's code, but reduced the rate of the improved inner codes up to the point where the same probability of block error as for Davey and MacKay's code was achieved. We found that in order to obtain the same probability of block error, the inner code of type A required half the rate and the inner code of type B required about 0.55 of the rate. In the second way, the rate was the same for both the improved inner code of type B (which was the only one examined) and Davey and MacKay's code, but the channel error probabilities were reduced to the point where the same probability of block error was achieved. We found that a decrease of the channel error probabilities by a factor of about 17 was required in order to obtain the same probability of block error for the given rate.

These results have led us to believe that the inner codes that we examined are inherently inferior to those suggested by Davey and MacKay. We postulate that one of the main reasons for their inferiority is the fact that all the error correction is done by the outer code. By doing so, when a single synchronization error occurs in a group of L symbols, they are all considered as erasures. This is so because the inner code performs no error correction and thus a single synchronization error in a group L symbols will cause all the subsequent symbols in the group to be corrupted, the cost of which is very high. This problem is inherent in our scheme and thus we abandoned this research several months ago and moved on to other areas.

VII. Future Work

One possible way to improve the performance of the inner code is to consider the use of an inner code that has shorter markers. This would reduce redundancy, which could then be transferred to the outer code and result in a higher error correcting capability and consequently a lower probability of block error. An example of such a code is a cascaded code [9], which is a code that, like natural marker codes, uses markers for synchronization. The redundancy of its markers, however, is smaller than that of a natural marker code. This reduction in redundancy is obtained by having a shorter flag at the expense of representing the marker indices by sequences of k bits that do not contain the flag. Clearly, the new sequences of bits representing the marker numbers would have longer length, but the overall length of the markers is reduced and redundancy is saved.

Another way of improving our coding scheme is to tackle the problem of L symbols being considered as L erasures due to a single synchronization error that occurs in one of them. As stated in the Summary, we conjecture that this is one of the main reasons for the inferior performance achieved by our coding scheme. This problem might be solved by introducing a third level of coding. The idea is to treat each group of L symbols, as a separate codeword. One can then use one of the codes described in the introduction, which correct a single insertion or deletion error in a codeword given the boundaries of the codeword are known. The inner code provides such boundary information, and so this requirement can be met. Clearly such a system would require optimizing the parameters of three types of codes – namely, the outer code, the improved inner code and the added third level code. Although, adding third level coding adds additional redundancy, it would come at the expense of the outer code's redundancy, which could be reduced, since according to our conjecture the outer code would not be required to have as high an error correcting capability as before.

References

[1] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” *Soviet Physics Doklady,* vol. 10, no. 8, pp. 707-710, Feb 1966. English translation of the Russian paper in *Dokl. Akad. Nauk* SSSR, vol. 163, no. 4, pp. 845-848, Aug.1965.

[2] E. Tanaka and T. Kasai, “Synchronization and substitution error-correcting codes for the Levenshtein metric,” *IEEE Transactions on Information Theory*, vol. 22, no. 2, pp. 156-162, Mar. 1976.

[3] L. Calabi and W. E. Hartnett, “A family of codes for the correction of substitution and synchronization errors,” *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 102-106, Jan.1969.

[4] G. M. Tenegolts, “class of codes correcting bit loss and errors in the preceding bit,” *Automation and Remote Control*, vol. 37, no. 5, pp. 797-802, May 1976. English translation of the Russian paper in *Automatika i Telemekhanika*.

[5] G. Tenegolts, “Nonbinary codes correcting single deletion or insertion,” *IEEE Transactions on Information Theory*, vol. 30, no. 5, pp. 766-769, Sep.1984.

[6] S.W. Golomb, B.Gordon and L. R. Welch, “Comma-free codes,” *Canadian Journal of Mathematics*, vol. 10, no. 2, pp. 202-209, 1958.

[7] E. N. Gilbert, “Synchronization of binary messages,” *IRE*, *Transactions on Information Theory*, vol. IT-6, pp. 470-477, 1960.

[8] H. Morita, A. J. van Wijngaarden and A. J. Han Vinck, “Prefix-synchronized codes capable of correcting single insertion/deletion errors,” *Proc. IEEE International Symposium on Information Theory*, Ulm, Germany, p. 409, June 1997.

[9] N. Kashyap and D. L. Neuhoff, “Data Synchronization with Timing,” to appear in *IEEE Transactions on Information Theory*.

[10] L J. Schulman and D. Zuckerman, “Asymptotically good codes correcting insertions, deletions, and transpositions,” *IEEE Transactions on Information Theory*, vol. 45, no. 7, pp. 2552-2557, 1999.

[11] M. C. Davey and D. J. C. MacKay, “Reliable communication over channels with insertions, deletions and substitutions,” *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 687-698, Feb. 2001.

[12] L. R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” *Proc. IEEE*, vol. 77, no. 2, pp. 257-286, Feb. 1989.

[13] W. Feller, *An Introduction to Probability Theory and its Applications*, John Wiley& Sons, NY, part 1, vol. 1, pp. 38-39.

[14] T. H. Cormen, C.F. Leiserson and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, MA, pp. 314-320, 1990.