LEVENSHTEIN DISTANCE-BASED CODING FOR SYNCHRONOUS, FIXED LENGTH DECODING WINDOWS IN THE PRESENCE OF INSERTIONS/DELETIONS

W.A. Clarke and H.C. Ferreira

Department of Electrical and Electronic Engineering Science, University of Johannesburg, P O Box 524, Auckland Park, 2006, South Africa. Email: willemc@uj.ac.za

Abstract: It is firstly shown that the use of a fixed length decoding window in a channel that introduces insertions/deletions can propagate the errors to such an extent that stronger insertion/deletion correcting codes must be used. It is further shown that random insertion/deletion correcting codes based on the Levenshtein Distance are not optimal for such channels. A modification to the Levenshtein Distance is proposed and codes based on this distance are shown to be better suited to a channel with a fixed length decoding window. Rates of the codes based on the Levenshtein Decoding Distance are compared with that of codes based on the Levenshtein Distance.

Keywords: Decoding, Insertion, Deletion, Levenshtein Distance, fixed length decoding

1 INTRODUCTION

The Levenshtein distance is the distance measure often used when designing codebooks for correcting insertions/deletions. This distance measure, originally proposed by Levenshtein, and sometimes referred to as the edit distance, defines the number of symbol insertions and/or deletions necessary to transform a codeword \( x \) into another \( y \).[1]

In the body of literature on insertion/deletion correcting codes based on the Levenshtein distance, the underlying assumption is often that word framing is accomplished using means other than looking only at the codewords. In this respect, the work of Sellers [2], Ferreira et al [3] and others come to mind. They investigated the case where markers (known sequences at regular intervals) were inserted to detect the codeword boundaries. The use of these markers is very important, especially in combination with the algebraic code constructions of Levenshtein[1], Helberg[4], Tenengolts[5] and others. In the proposed decoding algorithms of these code constructions, it is assumed that the shortened/elongated errored codewords are available to the decoder. In this regard, these code constructions assume that the detection of insertion/deletion errors is the task of the framing function. The isolation of the errored codeword received attention by Sellers [2].

Comma freedom [6] is sometimes also utilised to achieve the delineation. Once the framing has been achieved, the codes based on the Levenshtein distance metric can be used to do the error correction.

In this paper, a different view will be presented on the framing aspects when considering codewords based on the Levenshtein metric. The basic assumption is that synchronous communications is used. In this regard, packetised communication will also be considered “synchronous” as long as the number of codewords contained in a frame/packet is long compared to the codeword size.

In Section 2, the effects of using a fixed length decoding window will be highlighted. These effects will be addressed through coding means in Section 3 by modifying the Levenshtein distance.

2 EFFECTS OF USING A FIXED LENGTH DECODING WINDOW

In this section, the mechanisms normally involved when a fixed length decoding window is used together with a block channel code of length \( n \) will be presented. We assume that codewords are sent synchronously without any markers between consecutive codewords.

Let \( s \) indicate the symbol insertion and/or deletion correcting ability of a codebook \( Q \). In this paper, only block codes are considered with \( n \) symbols per codeword \( x \). This study assumes no additive errors are present.

Definition

Insertion and deletion errors introduced by channel disturbances are referred to as Type I errors.

Rule 1

The fixed length decoding window always output a codeword of length \( n \), irrespective of what happened in the received symbol stream.

From the above rule, it is easy to see that should a single deletion error occur (of Type I) in a codeword, that the decoding window will still output \( n \) symbols, i.e. the \( n-1 \) bits from the codeword after the single deletion, as well...
as a “borrowed” symbol from the next codeword. The
same is true for an insertion error, i.e. the fixed length
decoding window will still output $n$ symbols, i.e. the
original $n$ symbols together with the inserted symbol and
a “loss” of the last symbol. In both cases, the last symbol
in the received codeword is affected.

**Definition**

Within the scope of a codeword, the net result of $p$
deletions in a codeword is due to $i$ insertions and $d$
deletions with $p = d - i$, $i > d$, and $d, i, p \geq 0$.

**Definition**

Within the scope of a codeword, the net result of $p$
insertions in a codeword is due to $i$ insertions and $d$
deletions with $p = i - d$, $i > d$ and $d, i, p \geq 0$.

From these definitions, the following propositions follow:

**PROPOSITION 1**

Consider a fixed length decoding window located within
the boundaries of a codeword. A net result of $p$
deletions (or insertions) of Type I will result in $p$
insertions (or deletions) in the output of the fixed length decoding
window.

**Proof**

This follows easily from the fact that $s$ deletions
(insertions) in the symbol stream will not change the
decoding length of the fixed length decoding window.
Therefore, $n$ consecutive symbols will still be framed due
to Rule 1, with the result that the output will include
$p$ “borrowed” (deleted) symbols.

From this proposition, it is clear that in addition to
insertion/deletion errors introduced by the channel, any
decoding scheme based on a fixed length decoding
window will introduce additional errors.

**Definition**

The number of insertions/deletions introduced as a result
of using a fixed length decoding window is referred to as
Type II errors.

**PROPOSITION 2**

Let there be $i$ insertions and $d$ deletions of Type I with
$p = \text{abs}(i - d) > 0$. If a fixed length decoding window is
used, the error correction layer must be able to correct
$(p + i + d)$ insertion/deletion errors.

**Proof**

The codebook must be able to correct the $(i + d)$-
insertion/deletion errors in the underlying symbol stream,
as well as the $p$ additional insertions or deletions due to
the use of the fixed length decoding window.

Following directly from this proposition, the implications
for any coding scheme to be designed are the following:

**PROPOSITION 3**

A block codebook $Q$ designed for use with a fixed length
decoding window to correct $i$ insertions and $d$ deletions of
Type I must have the following minimum Levenshtein
distance property:

$$LD(Q) \geq 2(p + i + d) + 1 \quad (1)$$

**Proof**

The codebook must be able to correct $l = (p + i + d)$
errors. For a codebook to correct any combination of $l$
insertions/deletions, the Levenshtein distance of the
codebook must be $LD(Q) \geq 2l + 1$. Substituting the value
of $l$ proves the proposition.

From this it follows that a codebook designed for use
with a fixed length decoding window and to correct a
single insertion or deletion (of Type I), must have a
minimum Levenshtein distance of 5.

**PROPOSITION 4**

If the net result $p$ of insertions/deletions of Type I in the
symbol stream is zero (within the boundaries of a
codeword), the fixed length decoding window will not
misframe.

**Proof**

A zero net result of $i$-insertions or $d$-deletions in the
underlying symbol stream means that $\text{abs}(i - d) = \text{abs}
(d - i) = 0$. This implies that the same number of insertions
deletions occurred within the boundaries of a
codeword in the symbol stream. The codeword length
therefore does not change, with the result that the framing
is preserved in the fixed length decoding window. The start of
the next codeword will subsequently be at the
right position.

3. MODIFYING THE LEVENSHTEIN DISTANCE

In the previous section we showed that Type I
insertion/deletion channel errors are often followed by
Type II insertion/deletion decoding errors when a fixed
length decoding window is used. This is similar to the
error propagation phenomenon often found in the additive
error correction and line coding fields. The existence of
Type II errors places a higher constraint on the distance
properties of code designed to correct insertions/deletions. Therefore, if a channel introduces
$s = 1$ insertion/deletion errors, and a fixed length
decoding window will be used, the code must be
designed to handle at least $s = 2$ insertion/deletion errors.
The rates for $s > 1$ codes are unfortunately very low as
was shown by Levenshtein [7,8].
However, the errors introduced by the fixed length decoding window are not random and are predictable. As is often the case in information theory, this predictability can be exploited to construct codes specifically suited to this “channel”. If one assumes that symbols are transmitted from left to right, random channel errors (of Type I) will be followed by Type II errors in the last symbol positions. The possibility exists of using this knowledge in order to increase the rates of codes.

**Definition**

The following three definitions redefine the Levenshtein distance to take Type II errors into consideration.

**Proposition 5**

The \( p = \text{abs}(i - d) \) errors of Type II are always introduced at the last \( p \) symbol positions of the codeword into which the Type I errors are introduced when a fixed length decoding window is used.

**Proof**

In the underlying symbol stream, \( s = i + d \) insertion/deletion errors are made while the decoding window introduces \( p = \text{abs}(i - d) \) additional errors. These \( p \) errors are either consecutive insertions or deletions. In the case of deletions, they are “borrowed” symbols from the next codeword; in the case of insertions, this result in the deletion of the last \( p \) symbols. In either case, they affect the last \( p \) symbols.

The definition of the Levenshtein distance does not take the known Type II error position into consideration, because it is defined for the stricter case of random insertions/deletions. For this purpose, it is necessary to redefine the Levenshtein distance to take Type II errors into consideration.

The following three definitions redefine the Levenshtein distance to incorporate the existence of Type II insertions/deletions.

**Definition**

A decoding insertion (\( I_\sigma \)) is defined as the random insertion of a symbol in a codeword at positions \( 1 \) to \( n \) with a subsequent deletion of the symbol at position \( n \).

**Definition**

A decoding deletion (\( D_\sigma \)) is defined as the random deletion of a symbol in a codeword at positions \( 1 \) to \( n \) with a subsequent insertion of the symbol at position \( n \).

Note that the symbol at position \( n \) can be “lost” to the next codeword or be “borrowed” from the next codeword. In either case, that symbol must be assumed to be random. A decoding insertion or deletion is counted as one operation. Together, the Type I and Type II errors constitutes a single error operation. Now the Levenshtein distance can be defined in a broader sense.

**Definition**

The Levenshtein Decoding Distance (\( \text{LD}_\sigma \)) is defined as the minimum number of decoding insertions and/or decoding deletions to transform one codeword to another assuming fixed length codewords.

It is now necessary to prove that the Levenshtein Decoding Distance is in fact a metric. The steps to prove this will be done through the following lemmas.

**Lemma 1**

For all codewords \( x \in Q \), \( \text{LD}_\sigma (x, x) = 0 \).

**Proof**

No decoding insertions or deletions are needed to transform a codeword to itself, with the result that \( \text{LD}_\sigma (x, x) = 0 \).

**Lemma 2**

For all codewords \( x, y \in Q \) and \( x \neq y \), \( \text{LD}_\sigma (x, y) > 0 \).

**Proof**

Codewords \( x \) and \( y \) will at least differ in one position. Hence at least one decoding insertion or decoding deletion will be required to transform codeword \( x \) into \( y \).

**Lemma 3**

For all codewords \( x, y \in Q \), \( \text{LD}_\sigma (x, y) = \text{LD}_\sigma (y, x) \).

**Proof**

By applying the inverse set of operations (decoding deletions and/or decoding insertions) to transform \( x \) to \( y \), one can transform \( y \) back to \( x \).

**Lemma 4**

For all \( x, y, z \in Q \), \( \text{LD}_\sigma (x, z) \leq \text{LD}_\sigma (x, y) + \text{LD}_\sigma (y, z) \).

**Proof**

A distance defined in terms of operations is always a “true” distance. Suppose we have a set of operations \( S = \{ \sigma_1, \sigma_2, \ldots, \sigma_n \} \) on words of length \( n \); define \( \text{LD}_\sigma (x, y) \) as the length \( R \) of the shortest sequence of operations

\[
\sigma_{i_1} \sigma_{i_2} \ldots \sigma_{i_R} \quad (2)
\]

that transforms \( x \) into \( y \). Now if \( \text{LD}_\sigma (x, z) = R \), and \( \text{LD}_\sigma (z, y) = S \), then there are sequence

\[
\sigma_{i_1} \sigma_{i_2} \ldots \sigma_{i_R} : x \rightarrow z, \quad (3)
\]

and

\[
\sigma_{i_R} \sigma_{i_{R-1}} \ldots \sigma_{i_1} : z \rightarrow y. \quad (4)
\]
As a consequence, the sequence of operations
\[
\sigma_{j_s} \sigma_{j_{s-1}} \cdots \sigma_{j_2} \sigma_{j_1} \cdots \sigma_i
\]
transforms \( x \) into \( y \); its length is \( R + S \). So \( \text{LD}_d(x, y) \), the length of the shortest such sequence, is at most \( R + S \).

**Proposition 6**
The Levenshtein Decoding Distance is a true distance metric.

**Proof**
This directly follows from Lemmas 1 to 4.

**Proposition 7**
For a codebook \( Q \) to be \( s \)-insertion/deletion correcting (Type I) when used together with a fixed length decoding window, the minimum Levenshtein Decoding Distance, must be \( \text{LD}_d(Q) \geq 2s + 1 \).

**Proof**
A Levenshtein Decoding Distance between codewords \( x \) and \( y \) of \( 2s \) means that a combination of \( 2s \) decoding insertions and/or decoding deletions is necessary to transform codeword \( x \) into \( y \). This implies that a combination of \( s \) decoding insertions and/or decoding deletions in codeword \( x \) will result in the same error codeword than \( s \) decoding insertions and/or decoding deletions in codeword \( y \). Because of the same "error codeword" for both codewords \( x \) and \( y \), it will be impossible to decide to which codeword the "error codeword" belongs. Therefore, the Levenshtein Decoding Distance must be larger than \( 2s \) to correct \( s \) decoding insertion/deletion errors.

**Proposition 8**
For any codewords \( x \) and \( y \) in \( Q \), the relationship between the Levenshtein Distance and the Levenshtein Decoding Distance is given by
\[
\text{LD}_d(x, y) \leq \text{LD}(x, y).
\]

**Proof**
The Levenshtein Decoding distance requires less insertions/deletions to transform \( x \) into \( y \) because the end-of-codeword deletions/insertions are already included in the distance count. The Levenshtein Distance must include the opposite operations to the insertions/deletions in order to keep the codeword length constant.

As an example, consider the codewords \( x = 000000 \) and \( y = 001100 \). For these codewords, \( \text{LD}(x, y) = 4 \) and \( \text{LD}_d(x, y) = 2 \).

**Proposition 9**
The minimum Hamming distance for a code \( Q \) with \( \text{LD}_d(Q) \geq 2s + 1 \) is given by
\[
d_{\text{min}}(Q) \geq s + 1.
\]

**Proof**
Assume that the minimum Hamming distance of the codebook \( Q \) is \( d_{\text{min}} \geq s \).

Let \( x \neq y \), and \( d_{\text{min}}(x, y) \geq s \). The maximum number of decoding deletions and decoding insertions required to transform \( x \) to \( y \) will be \( 2s \). Therefore the Levenshtein Decoding Distance for codewords \( x \) and \( y \) will be \( \text{LD}_d(x, y) \geq 2s \). However, codebook \( Q \) is an \( s \)-correcting codebook with \( \text{LD}_d(Q) \geq 2s + 1 \). Therefore \( Q \) cannot contain both \( x \) and \( y \).

An algorithm was designed to search for codes based on the Levenshtein Decoding Distance (correcting \( s = 1 \) Type I errors) has a slightly higher rate than that of \( s = 2 \) codes, but smaller than \( s = 1 \) codes (based on the Levenshtein distance).
4. CONCLUSIONS

In this paper it was shown that the use of a fixed length decoding window introduces Type II insertion/deletion errors. It was further shown that codes based on the Levenshtein Distance are not optimal for communications where fixed length decoding windows are used. A modification to the Levenshtein Distance metric was proposed, i.e. the Levenshtein Decoding Distance metric, based on the notion of decoding insertions and deletions. Some properties of codes based on this modified metric were shown and results given.

5. REFERENCES


