Combined Permutation Codes for Synchronization

R. Heymann, H. C. Ferreira, T. G. Swart
Department of Electrical and Electronic Engineering Science
University of Johannesburg
P.O. Box 524
Auckland Park, 2006
South Africa
E-mail: rheymann@uj.ac.za, hcferreira@uj.ac.za. tgswart@uj.ac.za

***Abstract*—A combined code is a code that combines two or more characteristics of other codes. A construction is presented in this paper of permutation codes that are self-synchronizing and able to correct a number of deletion errors per codeword, thus a combined permutation code. Synchronization errors, modelled as deletion(s) and/or insertion(s) of bits or symbols, can be catastrophic if not detected and corrected. Some classes of codes have been proposed that are synchronizable, i.e. they can be used to regain synchronization although the error leading to the loss of synchronization is not corrected. Typically, different classes of codes are needed to correct deletion and/or insertion errors after codeword boundaries have been detected. The codebooks presented in this paper consist of codewords divided into segments. By imposing restrictions on the segments, the codewords are synchronizable. One deletion error can be detected and corrected per segment.**

I. Introduction

Establishing and maintaining synchronization is important in communication systems. A loss of synchronization, even if it is just for a short period, can lead to bursts of errors that may be catastrophic. Synchronization errors can be modelled as deletion(s) and/or insertion(s) of bits or symbols.

The simplest solution to ensure that a sequence can be resynchronized is by using markers (also known as commas). A marker is a known sequence that is periodically placed in the transmitted sequence. No other codeword in the codebook should include this sequence. At the receiver the positions of the markers are used to determine if synchronization has been lost and to resynchronize. Markers for binary codes have been studied in [1] and [2]. In [3] a construction for markers used with permutation codes is given.

A disadvantage of markers is that they add additional redundancy to the sequence. Markers can detect the loss of synchronization and help with recovery, but cannot correct the specific deletion or insertion errors.

The use of watermarks have been proposed as an alternative to markers [4], [5].

Another approach is to construct comma-free codebooks [6]. In a comma-free codebook the codewords are constructed in such a way that the overlap between two codewords does not result in a valid codeword. These constructions are valid for binary and M-ary communication. A formal definition of comma-free codebooks is given in Section II. In [6] an upper-bound for the cardinality of comma-free codes is also given. A construction for maximal comma-free codes is given in [7]. In [8] a suffix construction method is proposed to construct variable length codes with synchronization capability.

Comma-free codebooks can also be used to regain synchronization but cannot correct information bits lost due to synchronization errors.

Prefix codes, a subclass of comma-free codes, have also been suggested to regain synchronization [9], [10]. Every codeword in a codebook starts with a predefined prefix. The prefix does not appear anywhere else in the codeword and can thus be used to determine the start of every codeword.

Many synchronization error correcting techniques have also been investigated. These schemes assume that the boundaries of the codewords are determined by using markers. In [11] constructions for codebooks consisting of M-ary codewords able to correct single deletions are given. A construction to correct two or more deletions using design theory is given in [12].

A permutation code is a code where every symbol of the alphabet occurs exactly once in every codeword (see Section II for more detail). Permutation codes are applied in power line communications (PLC) [13] and flash memory [14].

The use of permutation codes combined with M-ary FSK modulation has been shown to be able to combat different types of noise present in PLC, especially for narrowband PLC [13] in the CENELEC A band. Applications include automatic meter reading and demand side management. Impulse noise, background noise and permanent frequency disturbances can occur in the PLC channel. Combining permutation codes and convolution decoding to correct these errors was proposed in [15].

The problem of synchronization errors when using permutation codes in conjunction with M-FSK modulation has been investigated in [16]. An error-correcting scheme was proposed that could correct a single insertion or deletion error. This scheme was improved on in [17]. A method to correct synchronization errors using tree structures was proposed in [18].

In flash memory, permutation codes are combined with rank modulation [14]. Flash memory consist of floating gate cells which have a discrete number of levels. Every level represents a symbol. It is much more time consuming to erase cells than writing to cells. It is thus important not to overshoot when

charging a cell to a certain level. Using permutation codes with rank modulation eliminates overshoot errors.

In the rest of this paper we focus on permutation codes.

This paper is organised as follows: in Section II formal definitions are given of the most important concepts used in the paper, as well as notations. A construction for self-synchronizing, deletion correcting codebooks is given in Section III. Section IV focuses on the resynchronization and decoding procedure. The work is concluded in Section V.

II. Definitions and notations

Let M be the length of the codewords. Let S_M denote the set of all $M!$ permutations.

Definition 1: A permutation code C of length M is a subset of S_M, where every codeword in C contains M different integers $\{1, 2, \ldots, M\}$ as symbols.

Definition 2: A codebook C is synchronizable if an overlap between any two codewords in the codebook does not result in a valid codeword.

Thus, let $X = x_1x_2 \ldots x_M$ and $Y = y_1y_2 \ldots y_M$ be two, not necessarily unique, permutation codewords from the permutation codebook C. An overlap between these two codewords,

$$x_{j+1}x_{j+2} \ldots x_M y_1 y_2 \ldots y_j \tag{1}$$

should not be a valid codeword if C is synchronizable.

If j in (1) can be any value, $j = 1, 2, ..., M-1$, without producing a codeword from C, then the code is called comma-free.

In [6] the upperbound of the cardinality of a comma-free codebook of words with length n, constructed from an alphabet containing the letters $0, 1, \ldots, \sigma - 1$ is given as:

$$f(\sigma, n) \leq \frac{1}{n} \sum_{d|n} \mu(d) \sigma^{n/d} \tag{2}$$

where the summation is extended over all divisors d of n, and $\mu(d)$ is the Möbius function. A good approximation of this upperbound is given by $f(\sigma, n) \leq \sigma^n/n$ [10].

In this paper only the occurrence of deletion errors will be considered. $X_T = x_1x_2 \ldots x_M$ is transmitted over a channel prone to deletion errors. A deletion error occurs if a transmitted symbol is not received at the receiver, i.e. suppose symbol x_i is deleted then the received codeword will be $X_R = x_1x_2 \ldots x_{i-1}x_{i+1} \ldots x_M$. The received codeword's length will thus be shorter and all the symbols after symbol x_i will move one position to the left.

Deletion errors usually occur with low probability. However, it is important to detect and correct them since they cause a shift in the received sequence and the results can be catastrophic. A more common error is substitution errors, where one symbol is turned into another due to channel noise. This paper will not focus on substitution errors, but since they are much more probable than deletion errors, their effect on the code's ability to synchronize correctly will be taken into account. Since permutation codes are applied in flash memories, only unidirectional substitution errors will be considered to dominate, i.e. symbol values can only decrease due to charge leakage.

III. Codebook construction

If a symbol is deleted from a codeword, then the resulting codeword of length $M-1$ is known as a subword. Every possible such subword for every codeword in C should be unique for it to be able to correct a single deletion error.

In [11], the complete permutation set S_M is divided into M partitions, each consisting of $(M-1)!$ codewords. Each such partition forms a codebook able to correct one deletion per codeword.

The partitions for $M = 4$ from [11] is given in Table I.

TABLE I
S_4 partitioned into 4 codebooks each able to correct one deletion error

1234	2134	1324	2341
2143	3124	1423	1243
3142	4123	2413	1342
4132	1432	2314	4312
3241	2431	3412	4213
4231	3421	4321	3214

The proposed codebook will consist of codewords divided into segments. The different segments will be constructed using the method in [11] to ensure that a deletion can be corrected in every segment. Every sequence in segment 1 will then be combined with every sequence in segment 2 to form the combined permutation codebook.

The construction can be explained step by step as follows:

1) Choose the lengths of each segment. Let l_1 denote the length of segment 1 and l_2 the length of segment 2; $l_1 + l_2 = M$.
2) For segment 1: Construct sequences as in [11], consisting of symbols $(1, 2, \ldots, l_1)$, capable of correcting one deletion error per sequence. The number of possible sequences is $(l_1 - 1)!$
3) For segment 2: Construct sequences, consisting of symbols $(1, 2, \ldots, l_2)$, capable of correcting one deletion error per sequence. Add l_1 to every symbol so that the sequences consist of symbols ranging from $(l_1 + 1, l_1 + 2, \ldots, M)$. The number of possible sequences is $(l_2 - 1)!$
4) Concatenate every sequence of segment 1 with every sequence from segment 2.

Example: Create a synchronizable codebook with $M = 6$.

1) Let the length of segment 1 be equal to the length of segment 2, i.e. $l_1 = l_2 = 3$.
2) There are $(l_1 - 1)! = 2! = 2$ possible sequences: $\{123, 321\}$.
3) Since segment 1 and 2 have equal lengths, the possible sequences for segment 2 are also $\{123, 321\}$. The sequences are translated to $\{456, 654\}$.
4) The sequences from segment 1 and 2 are combined to form the codebook $\{123456, 321456, 123654, 321654\}$.

As further example where $l_1 \neq l_2$, the codebook for $M = 7$, $l_1 = 3$ and $l_2 = 4$ is given in Table II.

TABLE II
EXAMPLE OF A COMMA-FREE CODEBOOK CAPABLE OF CORRECTING 1 DELETION ERROR PER SEGMENT

$M = 7$		
1234567	1235476	1236475
1237465	1236574	1237564
3214567	3215476	3216475
3217465	3216574	3217564

A codebook constructed in this way will be able to resynchronize after a synchronization error, as well as correct one deletion error per segment. The process for resynchronization and error correction is given in the next section.

The cardinality for codebooks constructed in this way is

$$|C| = (l_1 - 1)!(l_2 - 1)! \tag{3}$$

The minimum length of a segment is $l_i \geq 3$. A segment length of 2 will lead to $(2-1)! = 1$ sequence, which would then result in a prefix code. Using a prefix code, where the prefix has a length of 1 or 2 symbols, will result in codebooks with higher cardinalities. However, if errors affect the prefix, the code's ability to resynchronize will also be affected.

To maximize the cardinality, one segment should be kept to a minimum. For example: if $M = 8$ and $l_1 = 4$, then the cardinality would be $|C| = (4-1)!(4-1)! = 36$. However, if $M = 8$ and $l_1 = 3$, then the cardinality increases to $|C| = (3-1)!(5-1)! = 48$.

IV. RESYNCHRONIZATION AND ERROR CORRECTION

The decoding consists of two steps: Firstly the received sequence is resynchronized, i.e. the codeword boundaries will be determined. Secondly, errors in the codewords are corrected. These two steps can either be performed sequentially for every codeword, or the entire sequence can be synchronized before the errors are corrected in the individual codewords. The results will be the same. The two steps will be discussed individually.

A. Resynchronization

Due to the construction of the codebook, it is very easy to determine the start of each codeword if no errors are present, i.e. the symbols unique to segment 1 indicate the start of a codeword. Even if a burst of errors occur, the scheme can always resynchronize again once the channel becomes error-free.

If errors are present, it is more difficult to determine the codeword boundaries. Let $l_1 = l_2 = 3$. Suppose the sequence 123456123456 is transmitted and, due to a deletion error, the sequence 12345613456 is received. An obvious solution will be to use a sliding window approach and, if at least $l_1 - 1$ of the symbols of segment 1 are present in l_1 symbols, then it indicates the start of a codeword.

However, resynchronization should still be possible even if substitution errors occur. (As explained earlier, only unidirectional substitution errors will be considered.) Suppose the sequence 123654123456 is transmitted, $l_1 = 3$, and due to a substitution error the received sequence is 123653123456. A sliding window approach, as described above, will recognize the sequences 123 and 312 as the start of the two codewords.

A resynchronization procedure will be followed that makes provision for four scenarios:

1) A deletion error did not occur in a codeword.
2) A deletion error occurred in the first segment of a codeword.
3) A deletion error occurred in the second segment of a codeword.
4) A deletion error occurred in the first and second segment of a codeword.

Assume that $X_T = x_1 x_2 \dots x_{nM}$ is the transmitted sequence, consisting of n codewords, each of length M. Let l_1 and l_2 be the lengths of segment 1 and segment 2 respectively. S_1 is the set of all the symbols that is allowed in segment 1 and S_2 the set of symbols allowed in segment 2. The received sequence, $Y_R = y_1 y_2 \dots y_k$, may be shorter than the transmitted sequence due to deletion errors. The following steps will be followed in the resynchronization procedure:

1) Let i denote the index in the sequence. Initially $i = 1$.
2) Assume scenario 1 (no deletions): Extract $s_1 = y_i y_{i+1} \dots y_{i+l_1-1}$ and $s_2 = y_{i+l_1} y_{i+l_1+1} \dots y_{i+l_1+l_2-1}$ from the received sequence. A metric is calculated for this scenario, where every symbol in s_1 that is also in S_1 increases the metric by 1. If a symbol is in s_1 and not S_1, then the metric is decreased by 1. Similarly, if a symbol in s_2 is also in S_2, then the metric is increased, otherwise it is decreased.
3) Assume scenario 2 (deletion in segment 1): Extract $s_1 = y_i y_{i+1} \dots y_{i+l_1-2}$ and $s_2 = y_{i+l_1-1} y_{i+l_1} \dots y_{i+l_1+l_2-2}$. Calculate the metric for this scenario as described above.
4) Assume scenario 3 (deletion in segment 2): Extract $s_1 = y_i y_{i+1} \dots y_{i+l_1-1}$ and $s_2 = y_{i+l_1} y_{i+l_1+1} \dots y_{i+l_1+l_2-2}$. Calculate the metric.
5) Assume scenario 4 (deletion in segment 1 and segment 2): Extract $s_1 = y_i y_{i+1} \dots y_{i+l_1-2}$ and $s_2 = y_{i+l_1-1} y_{i+l_1} \dots y_{i+l_1+l_2-3}$. Calculate the metric.
6) The scenario with the highest metric is assumed to be the correct scenario. If scenario 1 has the highest metric, then $i = i+M$. If scenario 2 or 3 have the highest metric, then $i = i + M - 1$. Lastly, if scenario 4 has the highest metric, then $i = i + M - 2$.
7) Repeat the steps until the end of the received sequence has been reached.

Example: The transmitted sequence is 123456123456 and the received sequence is 13456123456. Let $l_1 = l_2 = 3$, $S_1 = \{1, 2, 3\}$ and $S_2 = \{4, 5, 6\}$.

Iteration 1: $i = 1$

Scenario 1: $s_1 = 134$ and $s_2 = 561$ and the metric, m_1, is thus: $m_1 = (2-1) + (2-1) = 2$. Scenario 2: $s_1 = 13$ and

$s_2 = 456$ and thus $m_2 = 2 + 3 = 5$. Scenario 3: $s_1 = 134$ and $s_2 = 56$, $m_3 = (2 - 1) + 2 = 3$. Scenario 4: $s_1 = 13$ and $s_2 = 45$, $m_4 = 2 + 2 = 4$. Scenario 2 has the highest metric and a deletion in the first segment is assumed to have occurred and $i = i+5$. Thus, the next codeword starts at index $i = i + 5$.

Iteration 2: $i = 6$

Scenario 1: $s_1 = 123$ and $s_2 = 456$ and the metric is thus: $m_1 = 3 + 3 = 6$. Scenario 2: $s_1 = 12$ and $s_2 = 345$ and thus $m_2 = 2 + (2 - 1) = 3$. Scenario 3: $s_1 = 123$ and $s_2 = 45$, $m_3 = 3 + 2 = 5$. Scenario 4: $s_1 = 12$ and $s_2 = 34$, $m_4 = 2 + (1 - 1) = 2$. Scenario 1 has the highest metric and thus no errors are assumed to have occurred in this codeword.

Simulations have been done with the codebooks from Section III to determine the effectiveness of the algorithm. The number of codewords, n, that was included in the transmitted sequence was determined by the deletion error probability, p_{del}, to ensure that on average 3000 deletions were simulated. Table III displays the probability of a deletion being detected. For $M = 6$, $l_1 = l_2 = 3$ and for $M = 7$, $l_1 = 3$ and $l_2 = 4$.

TABLE III
PROBABILITY OF A DELETION ERROR BEING DETECTED

Deletion Probability	$M = 6$	$M = 7$
0.09	0.9799	0.9886
0.08	0.9880	0.9928
0.07	0.9920	0.9958
0.06	0.9959	0.9992
0.05	0.9960	0.9993
0.04	0.9973	≈ 1
≤ 0.03	≈ 1	≈ 1

The synchronization scheme can detect, with high probability, up to one deletion error per segment. If this is exceeded, the deletion error will propagate to the next codeword. If the next codeword is an error-free codeword, the deletion error can be detected in the next codeword. Suppose a deletion error occurs in codeword i. Table IV shows the probability of the deletion error being detected in codeword i, the codeword where the error occurred, and the probability of it only being detected in the codeword $i + 1$.

TABLE IV
PERCENTAGE OF DELETION ERRORS CORRECTED IN THE SAME OR NEXT CODEWORD

Deletion Probability	$M = 6$		$M = 7$	
	i	$i+1$	i	$i+1$
0.009	0.9940	0.0060	0.9927	0.0073
0.008	0.9942	0.0058	0.9931	0.0069
0.007	0.9956	0.0044	0.9948	0.0052
0.006	0.9966	0.0034	0.9951	0.0049
0.005	0.9971	0.0029	0.9953	0.0047
0.004	0.9989	0.0011	0.9961	0.0039
0.003	0.9990	0.0010	0.9974	0.0026
0.002	0.9993	0.0007	0.9989	0.0011
0.001	≈ 1	≈ 0	≈ 1	≈ 0

Simulations were also used to determine the effect of unidirectional substitution errors on the resynchronization process. The value of random symbols were decreased. The number of codewords transmitted was determined as explained above, but instead of the deletion probability a substitution probability was used.

If a substitution error is identified by the synchronization algorithm as a deletion error, a misclassification error occurred. Figure 1 shows the probability of misclassification errors.

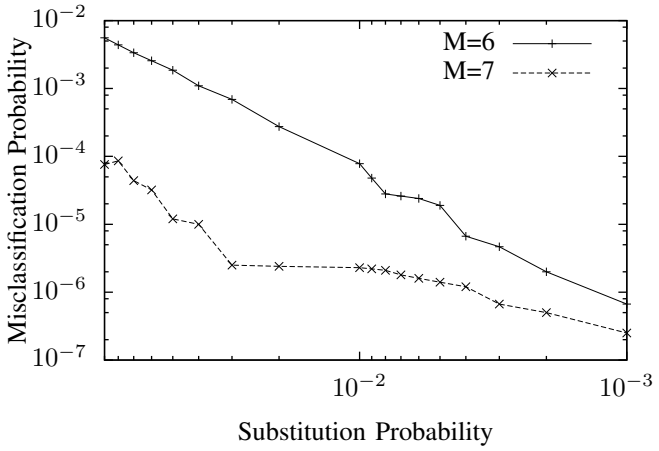

Fig. 1. Probability of misclassification errors

A misclassification error, in this case, is equivalent to the algorithm introducing an insertion error. The algorithm needs to be adjusted to correct misclassification errors. A fifth scenario is introduced: Assume a misclassification error has occurred. A substitution error does not affect the preceding codewords. We assume thus that the next codeword is error free. The algorithm will look at the next codeword. If reversing the decision that a deletion error has occurred in a codeword causes the next codeword to be bounded correctly, then it is assumed that a misclassification error has occurred.

Misclassification errors are more probable for $M = 6$ than for $M = 7$. With a longer segment, more symbols contribute to the metric and thus less misclassification errors occur.

B. Error Correction

Once the boundaries of the codewords have been detected, the deletion errors can be corrected. As mentioned earlier, if a codebook is able to correct deletion errors, every codeword has unique subwords. Since our codebook is constructed of two segments, each segment containing sequences able to correct one deletion, a combination of these sequences form codewords able to correct a deletion error in every segment. All possible subwords of every codeword, due to a single deletion, is inserted into a data structure called a map (or a dictionary). The subwords, due to deletions in every segment, are also included. Every subword uniquely maps to one codeword in our dictionary. This relationship between the subwords and codewords is used to determine the original codeword before an error had occurred.

The use of a map is similar to a look-up table. However, it is much faster and more efficient. Maps store a link between a key (the subword) and a value (the codeword) and uses a hash function to retrieve the information in an efficient manner.

Simulations have been performed to determine the Block Error Rate (BER) after the synchronization and error correction process. The number of codewords transmitted is determined as explained previously. Simulations are repeated many times to get good statistical averages. Codebooks for $M = 6$ and $M = 7$ are used. The codebook for $M = 6$ is used in the simulations with $l_1 = l_2 = 3$. For $M = 7$ two configurations are given, the first with $l_1 = 3$ and $l_2 = 4$, and the second with $l_1 = 4$ and $l_2 = 3$. Substitution errors are not taken into account. The results are given in Figure 2.

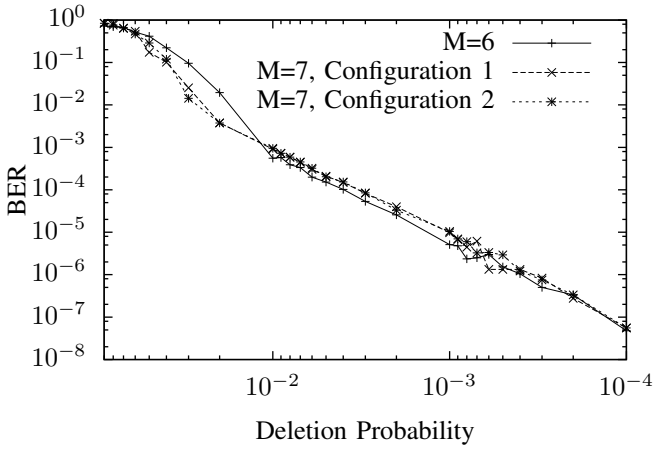

Fig. 2. Decoding performance with deletion errors

When the deletion error probability is high, the probability that the correction capability of the system can be exceeded, is also high. The system is then not able to detect the errors or only detects it in following codewords.

The BER for $M = 7$ decreases earlier than for $M = 6$. With $M = 7$ more symbols are used to determine the metric and it is thus easier to determine which scenario has occurred. The $M = 7$ codewords only contain one more symbol than the $M = 6$ codewords, so the probability that the correction capability is exceeded in the $M = 7$ codebook is not much higher than the $M = 6$ codebook.

V. Conclusion

Applications for permutation codes include flash memories and PLC. A loss of synchronization, modelled as insertion and deletion errors, can be catastrophic for a communication system.

A code construction is presented that can be used to synchronize the received sequence without the use of additional redundancy, such as markers. Every codeword in the codebook is divided into two segments. In addition to the ability to self-synchronize, a deletion error in each segment can be corrected.

VI. Acknowledgment

This material is based upon work supported financially by the National Research Foundation. Any opinion, findings and conclusions or recommendations expressed in this material are those of the author(s) and therefore the NRF does not accept any liability in regard thereto.

References

[1] F. F. Sellers, "Bit loss and gain correction code," *IEEE Trans. Inf. Theory*, vol. 8, no. 1, pp. 35–38, Jan. 1962.

[2] E. N. Gilbert, "Synchronization of binary messages," *IEEE Trans. Inf. Theory*, vol. 6, no. 4, pp. 470–477, Sep. 1960.

[3] T. Shongwe, T. G. Swart, H. C. Ferreira and T. van Trung, "Good synchronization sequences for permutation codes," *IEEE Trans. Commun.*, vol. 60, no. 5, pp. 1204–1208, May 2012.

[4] M. C. Davey and D. J. C. MacKay, "Reliable communication over channels with insertions, deletions and substitutions," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 687–698, Feb. 2001.

[5] R. Heymann and H. C. Ferreira, "Regaining synchronization using neural networks and watermarks," *Int. Symp. Inform. Theory and its Applic.*, Auckland, New Zealand, pp. 1–6, Dec. 2008.

[6] S. W. Golomb, B. Gordon and L. R. Welch, "Comma-free codes," *Canad. J. Mathematics*, vol. 10, pp. 202–209, Feb. 1958.

[7] W. L. Eastman, "On the construction of comma-free codes," *IEEE Trans. Inf. Theory*, vol. 11. no. 2, pp. 263–267, Apr. 1965.

[8] R. A. Scholtz, "Codes with synchronization capability," *IEEE Trans. Inf. Theory*, vol. 12, no. 2, pp. 135–142, Apr. 1966.

[9] C. V. Ramamoorthy and D. W. Tufts, "Reinforced prefixed comma-free codes," *IEEE Trans. Inf. Theory*, vol. 13, no. 3, pp. 366–371, Jul. 1967.

[10] H. Morita, A. J. van Wijngaarden and A. J. H. Vinck, "On the construction of maximal prefix-synchronized codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 2158–2166, Nov. 1996.

[11] V. I. Levenshtein, "On perfect codes in deletion and insertion metric," *Discrete Math. Appl.*, vol. 2, no. 3, pp. 241–258, Jan. 1992.

[12] P. A. H. Bours, "On the construction of perfect deletion-correcting codes using design theory," *Designs, Codes and Cryptography*, vol. 6, no. 1, pp. 5–20, Jul. 1995.

[13] A. J. H. Vinck, "Coded modulation for powerline communications," *Proc. Int. J. Elec. Commun.*, vol. 54, no. 1, pp. 45–49, Jan. 2000.

[14] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank modulation for flash memories," *Proc. IEEE Int. Symp. Inform. Theory*, Toronto, Canada, pp. 1731–1735, Jul. 2008.

[15] H. C. Ferreira, A. J. H. Vinck, T. G. Swart and I. de Beer, "Permutation trellis codes," *IEEE Trans. Commun.*, vol. 53, no. 11, pp. 1782–1789, Nov. 2005.

[16] L. Cheng, T. G. Swart and H. C. Ferreira, "Synchronization using insertion/deletion correcting permutation codes," *Proc. Int. Symp. on Powerline Commun. and its Applic.*, Jeju Island, Korea, pp. 135–140, Apr. 2008.

[17] L. Cheng, T. G. Swart and H. C. Ferreira, "Re-synchronization of permutation codes with Viterbi-like decoding," *Proc. Int. Symp. on Powerline Commun. and its Applic.*, Dresden, Germany, pp. 36–40, Mar. 2009.

[18] R. Heymann and H. C. Ferreira, "Using tree structures to resynchronize permutation codes", *Proc. Int. Symp. on Powerline Commun. and its Applic.*, Rio de Janeiro, Brazil, pp. 108–113, Mar. 2010.