UDC 621.391.14

APPLY ERROR PATTERNS TO CORRECT AND ERASE TWO-DIMENSIONAL ITERATED CODES

REN XUN HUAN, V.K. KONOPELKO, V.Yu. TSVIATKOU

Belarusian State University of Informatics and Radioelectronics, Republic of Belarus

Submitted 28 November 2020

Abstract. Two-dimensional iterated codes have large minimum Hamming distance and their complexity might be compared with Turbo codes. Familiar algorithms for iterating codes have low decoding capabilities and very high complexities. In order to ensure the applicability of iterated codes, in this paper, we propose a method for correcting and erasing errors of iterated codes for a two-dimensional (2D) interference channel with Hamming code which provides good performance and acceptable complexity.

Keywords: iterated codes, Hamming code, correct and erase errors.

Introduction

Hamming code or Hamming Distance Code is the best error correcting code we use in most of the communication network and digital systems. In Error control coding, parity check bits are calculated based on the input data. The input data and parity check bits are transmitted across a noisy channel. In the receiver, an ECC decoder is used to detect or correct the errors induced during the transmission. The number of parity bits depends upon the number of information bits. The hamming code uses the relation between redundancy bits and the data bits and this code can be applied to any number of data bits. A powerful ECC usually requires more redundant bits and more complex encoding and decoding processes, which increases the codec overhead [1].

At present, the most successful coding schemes are turbo codes and low-density parity-check codes, since their excellent capability, closely to the Shannon limit. Under some specific requirement (typically, code-rates near to the unity and low error rates required), iterated codes may turn into competitive. In addition, they are naturally advisable for high-speed parallel decoder implementation. Extended Hamming codes are representative constituents of iterated formulas since they are uncomplicated and can be iteratively decoded by fast suboptimal algorithms [2, 3]. More formidable constituent codes cause more powerful schemes but request more complex decoding algorithms, usually avoiding high data-rate performances.

The simplest two-dimensional iterated codes are single parity check (SPC) product codes, guaranteed to correct only one error by inverting the intersection bit in the erroneous row and column [1]. Multidimensional SPC iterated codes can be constructed to improve the error correction capability, but a more complex decoding process is required

In [8] proposed a method correct and erase errors by using the error pattern which is based on the process of calculating the pattern of the $t \times t$ (the error mode can be represented as a t by t matrix which can correct only t errors) type library and applying the identification parameters to recognise the errors. The method of correcting and erasing errors which are proposed in [8] cannot completely include all error patterns that can find through permutations of rows. Based on this defect, we propose an improved scheme.

Linear binary product codes are only considered In the thesis. However, the analytical results and the methods are also easily applied to other non-linear codes.

The rest of the paper is organized as follows. In Section 2, a brief introduction of the iterated codes will be presented. And in section 3, an improved algorithm which used to ameliorate pattern

library for correcting and erasing errors that exist through permutations of rows has proposed. The conclusion will be given in Section 4.

Syndrome (table-lookup) decoding

The use of syndrome for error detection and correction is discussed in [6–8], the standard array and its application to the decoding of linear block codes are presented.

Premeditate a (n,k) linear code with generator matrix *G* and parity-check matrix *H*. Let $v=(v_0,v_1,...,v_{n-1})$ be a codeword that transmitted over a noise channel. Let $r=(r_0,r_1,...,r_{n-1})$ be the received vector at the output of the channel. Because of the channel noise, the received words *r* may be different from *v*. The vector sum $e=r+v=(e_0,e_1,...,e_{n-1})$ is an *n* tuple, where $e_i=1$ is called the error pattern. When *r* is received, the decoder computes the following $(n-k)$ tuple

$$S=r\cdot H^T=(S_0,S_1,...,S_{n-k-1}).$$

Which is called the syndrome of *r*. Then, $S=0$ indicates that *r* is a codeword, and $S\neq 0$ means that produced errors. Therefore, we can employ the value of *S* to determine whether an error has arisen. Every (n,k) linear block code is capable of detecting 2^n-2^k pattern errors, however, it is capable of correcting only 2^{n-k} error patterns. For large *n*, 2^{n-k} is a small fraction of 2^n-2^k. Therefore, the probability of a decoding error is much higher than the eventuality of an undetected error.

A linear block code with $d_{\min}$ can assure to detect any errors less than or equal to $d_{\min}-1$. The theorem confirms the fact that a (n,k) linear code with minimum distance $d_{\min}$ is capable of correcting all the error patterns of $\lfloor (d_{\min}-1)/2 \rfloor$ or fewer errors, but it's not capable of correcting all the error patterns of weight $t+1$. A standard array has an important property that can be used to simpify the decoding process. There is a one-to-one correspondence between a correctable error patternand a syndrmes. Using this one-to-one correspondence relationship,we can form a decoding table,which is much simpler to use than a standard array. This table is either stored or wired in the receiver. The decoding of a received vector consists of three steps:

1. Compute the syndrome *S*.

2. Locate the coset leader e_l whose syndrome is equal to $r\cdot H^T$, Then e_l is assumed to the eroor pattern caused by the channel.

3. Decode the received vector *r* into the codeword $v^*=r+e_l$.

In theory, table-lookup decoding can be applied to any (n,k) linear code. It results in minimum decoding delay and minimum error possibility, however for large information redundancy, the implementation of this decoding scheme is not very reality, and either a major storage or a complicated logic circuitry is needed. Product (iterated) codes has the capability of constructing long, powerful quotes from short component codes. Therefore, our analysis is focused on the two-dimensional iterated codes.

Introduction to iterated code

Iterated codes (or product codes) are serially concatenated codes which were presented by Elias in 1954 [2]. The construction method of iterated codes allows us to construct long, powerful codes from short assembly codes. The concept of iterated codes is simple enough and comparatively efficient for constructing extremely long block codes by using at least two short block codes [3].

For a linear block code, the minimum distance is equal to the minimum codeword weight, which is defined as the number of nonzero symbols in a codeword. The minimum Hamming distance is also used to evaluate the error detection capability of a linear block code. The simplest two-dimensional product codes are single-parity check (SPC) product codes [1]. SPC product codes only guarantee correction of one error. The product codes, whose component codes are Hamming or extended Hamming product codes, are known as Hamming product codes.

The random-error-detecting and random-error-correcting capabilities of code are determined by its minimum distance. Hamming codes have a minimum distance of 3 and therefore are capable of correcting any single error over the span of the code block length. The weight enumerator of Hamming codes is known. Hamming codes are perfect codes and can be decoded easily using a table-lookup scheme. Good codes with a minimum distance of 4 for single-error correction and double-error detection can be acquired by properly shortening the Hamming codes. Hamming codes and their shortened editions have been proverbially used for error control in digital communication and data storage systems in these years owing to their high rates and decoding briefness.

Presume that two component codes $C_1(n_1, k_1, d_1)$ and $C_2(n_2, k_2, d_2)$ are used, where n_1, k_1 and d_1 are codeword width, input data width, and minimum Hamming distance for the code C_1, respectively n_2, k_2 and d_2 are codeword width, input data width, and minimum Hamming distance for the code C_2, separately. Here we use the simple Hamming codes construct the iterated codes, let $v = (1100, 0100, 1011, 1001)$ be a codeword, simultaneously $C_1(7,4,3)$ and $C_2(7,4,3)$ are used. Encoding process of iterated (product) codes shows as Fig. 1.

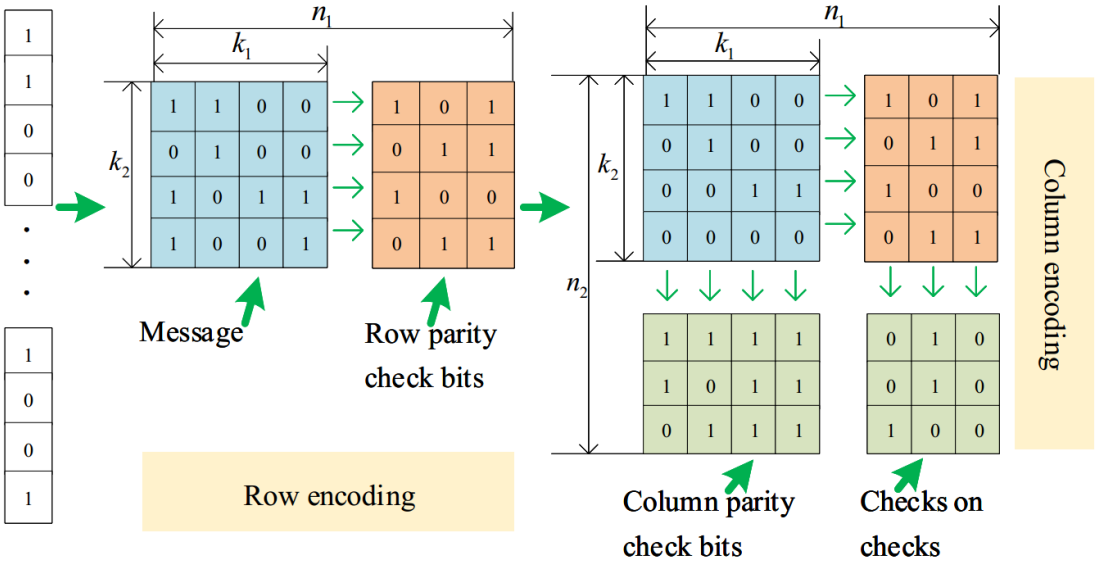

Fig. 1. Iterated codes encoding process

Fig. 1 shows the creation process of two-dimensional iterated codes, the iterated code $C_I((n_1 \times n_2 = 49), (k_1 \times k_2 = 16), (d_1 \times d_2 = 9))$ is constructed from C_1 and C_2 as follows:

The product code C is obtained from the codes C_1 and C_2 in the following manner:

1. Place $k_1 \times k_2$ information bits in an array $k_2 = 4$ rows and $k_1 = 4$ columns.

2. Coding the $k_2 = 4$ rows using the code C_1. Note that the result will be an array of $k_2 = 4$ rows and $n_1 = 7$ columns.

3. Coding the $n_1 = 7$ columns using the code $C_2(7,4,3)$.

Iterated (product) codes have a larger Hamming distance compared to that of the component codes. If the component codes C_1 and C_2 have minimum Hamming distance d_1 and d_2 respectively, then the minimum Hamming distance of the iterated code C_I is the product $d_1 \times d_2$, which greatly increases the error correction capability. Iterated codes can be constructed by a serial concatenation of simple component codes and a row-column block inter-leaver, in which the input sequence is written into the matrix row-wise and read out column-wise. Iterated codes can efficiently correct both random and burst errors. For example, if the received product codeword has errors located in a number of rows not exceeding $(d_2 - 1)/2$ and no errors in other rows, all the errors can be corrected during column decoding.

Algorithm decoding the product codes

Since in 1954, Elias introduced the product code, numerous decoding algorithms for decoding product codes were presented. The most obvious method of decoding is the one suggested by Elias himself in his original work [2]. In Elias's algorithm, the rows in the received message are decoded using a decoder for the code C_1 that decodes up to $\lfloor (d_{1\min})/2 \rfloor$. The columns of the resultant matrix are then decoded using a decoder for the code C_2 that decodes up to $\lfloor (d_{1\min})/2 \rfloor$. It can easily be shown that such a decoder can correct only up $\lfloor (d_1 \times d_2)/4 \rfloor$ [5]. In [8] proposed a new method erase the errors based on the two-step decoding with an error pattern library. Unfortunately, this two-step decoding method fails to correct certain error patterns. Besides, the method of correcting and erasing errors which are proposed cannot completely include all error patterns thast can find through permutations of rows, simultaneously, the algorithm didn't concrete presented how can we erase the error bytes, the disadvantage of the algorithm [6–8] showed in Fig. 2. Based on this defect, analysis of the structure of the error pattern [3], we proposed a more reasonable error correction algorithm. A three-stage pipelined Hamming product code decoding method is proposed, compared to the two-step row-column decoding method, the three-stage pipelined decoding method uses a row status vector and a column status vector to record the conducts of the row and column decoders. Instead of passing only the coded data between row and column decoder, these row and column status vectors are passed between stages to help make decoding decisions.

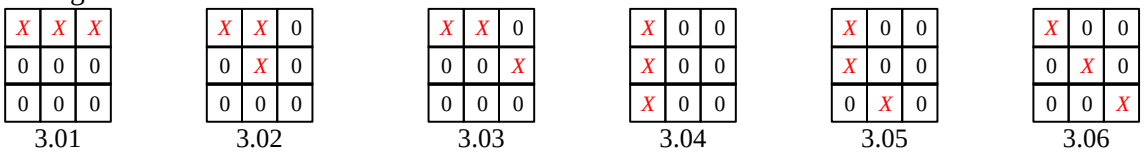

Fig. 2. All of the reduced combination error patterns for $t = 3$ [3]

If we apply Hamming code C_1 that distance is $d_1 = 3$ and Hamming code C_2, $d_2 = 3$ decoding the information that the error's patterns maintained the form as Fig. 2 or their transformer of the row and the column. Obviously, the property of the code itself can correct all of these pattern errors.

To illustrate the error correction process more evidently, Hamming codes are used as row and column component codes. An example of row and column status vectors after the first and second decoding stages shows as Fig. 3.

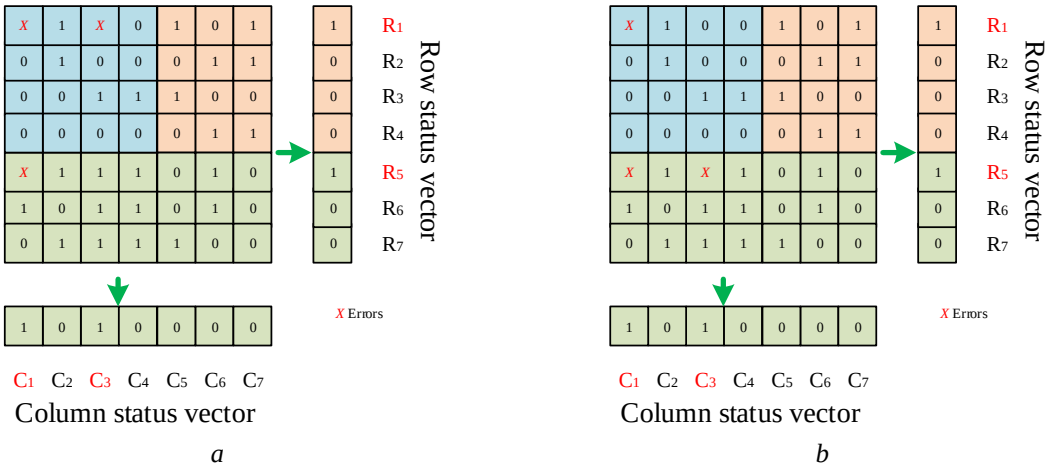

Fig. 3. An example of row and column status vectors after the first and second decoding stages

The simplified row and column status vector implementation can be described as follows.

The *i*-th position in the row status vector is set to «1» when there are detectable errors (regardless of whether the errors can be corrected or not). Then those locations where have only one error (determined by the value of the syndrome) are marked as R_{i_1}, otherwise mark the row as R_i However, if the value of the syndrome is 0, those positions are set as «0».

For the column status vectors, if in the *j*-th column the value of the syndrome is not equal 0, the *j*-th position in column status vector to be set to «1» when an error is detectable but not correctable, and also mark the location of the column as C_{i_1} when an error is correctable (means that only occurs one error), otherwise mark as C_i Fig. 4 shows an example of the row and column status vectors after the first and second stage decoding process.

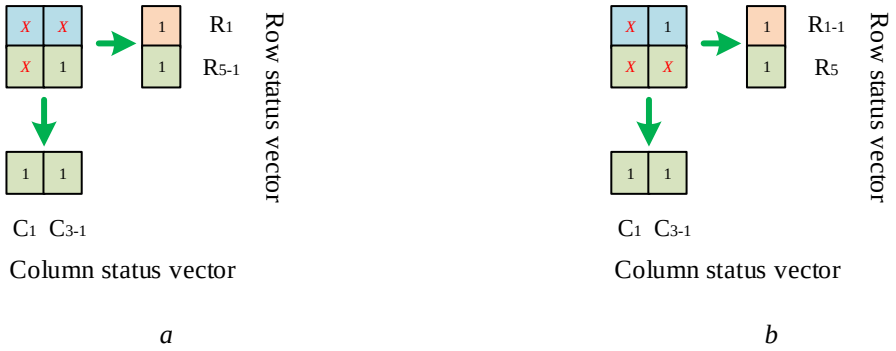

Fig. 4. Example of row and column status vectors after the first and second decoding stages

From the (*a*) and (*b*) of Fig. 4 we can perceive that the errors occurred in the same row (R_1, R_5) and column (C_1, C_3), yet the first graphic shows that there are two errors located in the row of R_1 and only one error located in the row of R_5, and the second graphic just the opposite, so when we erase the first errors of the graphic (*a*) we can apply three steps correct the errors, firstly, if the errors in a row are correctable, the error bit indicated by the syndrome is flipped. The row status vector is set to «0» if the syndrome is zero and «1» if the syndrome is non-zero. In this example we can correct the error which located in the row of R_5 and then correct the column of C_1 and C_3, however, for the graphic (*b*), at first we need to correct the R1and then correct the C_1 and C_3. Therefore we can know that the error pattern of (*a*) and (*b*) are different, the process of erasing is also different. The example is perfect because the Hamming code $d_{\min} = 3$ can correct 1 error and detect 2 errors.

And then we will describe the decoding process of the three-level pipeline Hamming product code. After initializing all state vectors to zero, the steps are as listed below.

The proposed iterative decoding method of two-dimensional Hamming products codes

Input: $r = v + e$
Output: v
Initialization: r_{sta}, $c_{sta} = 0$
While $S \neq 0$ program do.
Step 1. Row decoding of the received encoding matrix. If the error in the row is correctable, the error bit indicated by the syndrome will be reversed. If the syndrome is non-zero, the row state vector is set to «1».
Step 2. Update the column decoding of the matrix. The error correction process is similar to step 1, starting from step 1, use the column error vector and row state vector to calculate the column state vector.
Step 3. After the change from step 2, the matrix is decoded. The syndrome in each row has to be recalculated. If any remaining errors in each row can be corrected, use row syndrome to correct. If errors in each row are still detected but cannot be corrected, use the column status vector in step 2 to indicate which columns need to be corrected.
End while
Return v

Based on the theory of syndrome decoding and the capability of minimum Hamming distance which can correct $(d_{\min}-1)/2$ quantity errors and our proposed method which mark the state of error the row and column, obviously our method can correct permanent errors that are distributed in different rows.

From the analysis above we can derive that the method of three-step decoding can correct all of the random and burst errors that the combinations of error pattern less than 4, and also can correct some of the random and burst errors patterns which equal 5.

In the experiment, we apply our method to correct the error pattern library which proposed in [3], as a result, we can 100 % correct the random and bursts errors *t* less than 3, and for $t=4$ (the quantity of error pattern equal 16) we can correct and erase 93,75 % error patterns, and for $t=5$ (the sum of error pattern equal 34) we can correct and erase 91,17 % error patterns, but for $t=6$ (the sum of error pattern equal 90) we can only correct and erase 74,44 % error patterns, therefore for the error pattern which $t>4$ we need to use the minimum distance of Hamming code which $d_{\min}>3$.

From the process of decoding, we know that not only the encoded data is passed between the row and column decoders, but also the row and column state vectors are passed between stages to help make decoding decisions. We have proposed the method by dividing the encoded information into two transmissions, the reliability of the proposed method depends on both the error detection capability in the first transmission and error correction capability for the iterative decoding method. In the first transmission, the error patterns with single errors in different rows are corrected and the error patterns with two errors in a row are detected. The iterative decoding algorithm can correct up to six-bit errors, furthermore, our method can correct random errors that are distributed in different rows.

Conclusion

In this paper, we analyze methods for decoding the random errors on the basis of product coding, compared with two-step row-column decoding and this method solves the problem of rectangle four error patterns by recording the conduct of the row and column decoders using row and column status vectors. The iterative decoding algorithm can correct up to six-bit errors once the row and column parity check bits are received. Also, our method can correct permanent errors that are distributed in different rows.

References

1. Shu L., Daniel J. // ISBN 0-13-042672-5. 2004. P44–P63.
2. Elias P. // IRE Trans. Inform Theory. 1954. Vol. 4. P. 29–37.
3. Конопелько В.К., Смолякова О.Г. // Докл. БГУИР. 2008. № (7) 37, С. 19–28.
4. Pyndiah R. // IEEE Trans. 1998. Vol. 46. P. 1003–1010.
5. Ericson T. // Computer Science. 1986. № 307. P. 43–57.
6. Конопелько В.К., Смолякова О.Г. // Докл. БГУИР. 2008. № (9) 39, С. 142–153.
7. Конопелько В.К., Хоан Ф.Х. // Докл. БГУИР. 2007. № (8) 38, С. 55–60.
8. Смолякова О.Г., Хоан Ф.Х. // Докл. БГУИР. 2008. № (1) 31, С. 70–75.