PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 39, NO. 1, PP. 73-83 (1995)

AN INTERESTING AND EFFECTIVE CODE FOR ERROR-CORRECTING IN MASS-STORAGE DEVICES[1]

András SZEPESSY

Department of Measurement and Instrument Engineering
Technical University of Budapest
Budapest, XI. Műegyetem rkp. 9. Building R, I.113.
Postal Address: H-1521 Budapest, Hungary
E-Mail: Szepessy@mmt.bme.hu

Received: March 5, 1995

Abstract

We present an interesting version of error correcting codes that makes use of the idea of multi-dimensional coding and is able to make very large data total error-free even if the ratio of the number of erroneous characters to the total amount of characters is about $0.1 - 0.2$. The encoding and decoding mechanisms are very simple to do. Because of these properties it can be used when transferring very large data through a noisy channel or storing in a shoddy memory-device even if the data is required to be transferred or stored intactly, which means to obtain it after decoding total error-free. Noisy channels can be e.g. phone lines transferring facsimile data (DOONG, 1992), and as a 'shoddy' memory-device can be regarded any kind of standard-quality ones when counting on the worst case in a system to accomplish the requirement of high-reliability (e.g. a stable storage device), mainly when using cheaper kinds of mass storage devices (e.g. Hard Disks, Optical Disks).

Introduction

Let us suppose that a transient failure has broken a given part of the bits of data stored in a memory, in a way that every bit (or character) broke independent from each other (Random Errors) with a probability bit. The ratio of the number of corrupted bits to the amount of every bit is called Bit Error Ratio, or BER. Let us call BER_0 the Raw Bit Error Ratio and BER_∞ the value of BER after passing the whole error-correcting method. We can correct a very large data total error-free when using an error-correcting code which decreases the value of BER to such a value that

$$P_{\text{error-free}} = (1 - BER_\infty)^{\text{number of bits}} \approx 1 - BER_\infty \cdot (\text{number of bits}),$$

[1]This work was supported by Hungarian Scientific Foundation Grant OTKA T-760 and supervised by Prof. Endre Selényi.

that is, the probability of getting the data total error-free will have a value very close to zero. The task of error-correcting becomes so very difficult. A method offering a possible solution is considered in this paper.

A Solution – Multi-Dimensional Coding

Unfortunately, solutions using a great deal of codewords independent of each other cannot be used if there is a need of correcting the whole data error-free by a high value of the Bit Error Ratio of Random Errors. The reason for this is that every bit is a member of exactly one codeword which means if a bit is protected by a codeword that cannot be corrected there is no way to correct it any more. A solution for that is to protect every bit by more than one codeword, and possibly every codeword should protect another group of bits than the other ones, which means that the members of any pair of the codewords should contain maximum one common bit. (Of course there are also kinds of codes whose characters consist of more than one bit. By these the same is valid for characters as we mentioned for bits.) This is called interleaving or multi-dimensional coding technique. It is to say that now we focus on the area of Random Error Correcting Ability, insipid of the fact that interleaving technique is generally used against Burst-Errors occurring in the data.

Our code scheme makes use of the idea of a repetitive algorithm used e.g. in CDROMs (KURTZ, 1987), (RAO, 1987). It has got a good deal of versions, but there are two general ones related as follows. One of these was developed to correct erasures only. The other one corrects simple errors, too. *The basic idea in both versions was to use the possibly simplest kind of error correcting codes, which means a code that need not be able to correct more than one error per codewords, and to create a multi-dimensional code with them.* Another important question was of course to find the code filling the above requirement and *having a minimal value of redundancy.* So we have chosen the simple 'one parity bit per binary codewords' for erasures correction, and the $(n, n-2)$ Reed–Solomon code for simple-error-correction. The number of bits per one character in the latter case depends on the upper bound of the miscorrections rate required to reach a great error correction capability. (A miscorrection means the event when the decoder observes an error in a codeword, then deciding to be able to correct this error, it does try to do that. When done, it believes in the success of correction, although the codeword has remained erroneous, or moreover, it has become wronger than before. The miscorrection rate means the probability of a miscorrection after observing an error, and it is a very important parameter of a code.) (See *Fig. 1*).

Definition of the Erasure Correcting Version

The code is a Block Code, that is, it handles blocks of the data of $(X \cdot Y)$ bits size. A Data Block can be imagined as a $\mathbf{B}$ matrix consisting Y rows and X columns. While encoding it gets P extra columns containing the parity bits of the block ($\mathbf{B}'$).

Redundancy

There are more possible ways to define redundancy of a code. Now we define it as the ratio of amount of the redundant bits (in this version parity bits) to that of the information bits. The redundancy defined in such a way has a value of P/X.

The Encoding Procedure

Let us call $B(y,x)$ the information bit existing in the yth row and xth column. After the encoding procedure every information bit will be contained by exactly P codewords or, as we will say, every information bit will be encoded in P directions. Every codeword contains X information bits and a parity bit. Let us call $P(y,p)$ the parity bit standing in the yth row and pth parity-column of the matrix $\mathbf{B}'$. In the encoding procedure this $P(y,p)$ will be given a value as follows:

$$P(y,p) = B(y,1) \oplus B(y+p,2) \oplus \ldots \oplus B(y+(x-1)\cdot p, x), \qquad (1)$$

where the operator $\oplus$ means a modulo 2 addition (Exclusive Or), and the operator $+$ a modulo Y addition. This latter one guarantees that the row-index remains in the interval $[0, Y-1]$. So is the encoding procedure fully-defined.

We should add one more important rule to the above ones. It is worth choosing an (X,Y,P) parameter-set that any two codewords contain *maximum one* common bit.

Notice

The above distribution of the bits among the codewords is very simple to realise in practice, and it is also the simplest way to understand the logic of this code. Nevertheless we suggest using another kind of distribution, i.e. a stochastical one. It means that for every codeword it is drawn

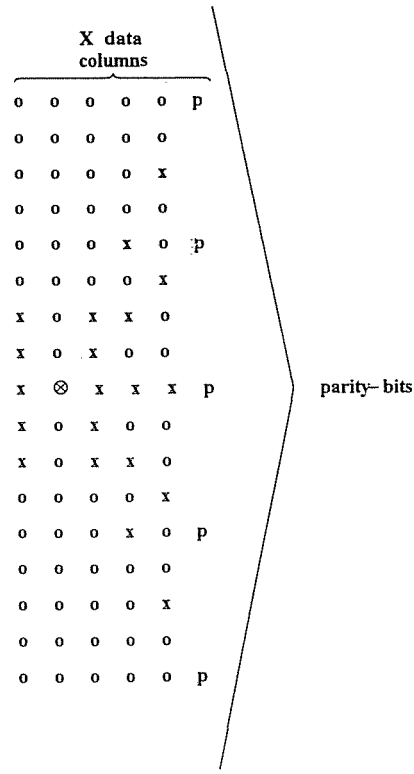

Fig. 1. The x's mean the bits of that codewords which contain the bit marked with the $\otimes$. The p's mean the parity bits of these codewords. We have placed every of them onto the line of their own codeword

stochastically which bits it will contain. In this way there are three rules to be accomplished:

I. Every codeword consists of X information bits and one parity bit.

II. $P(y,p)$ is the modulo 2 sum of the information bits of the codeword called $CW(y,p)$.

III. If $p1 = p2$, then $CW(y1,p1)$ and $CW(y2,p2)$ contain no common bit, otherwise they contain maximum one common bit.

Of course, we can use a pseudo-random distribution, too, and what is more the same one for every block of the data. In this way we get also a simple distribution procedure. We have no place here to explain its

reason, so we simply say that following this way, we get a code protected much better against some kinds of errors. We can also see that the non-stochastical version defined before the stochastical one accomplishes the rules I–III, too, so we can regard the non-stochastical version as a special case of the other one.

The Decoding Procedure

In all of the P directions we pass the following step called a *sweep*:

We correct every codeword of this direction p, if and only if it contains exactly one error. Since we have supposed there are erasure-type errors only, it can always be seen if a given codeword is correctable. Having seen every codeword of the actual direction (or expressed more illustrated having swept over all of them) we go ahead attempting correction in the next direction. If we have swept over every codewords in every direction (or otherwise expressed we have passed a *complete sweep* over the block), we begin this procedure with the first direction again. We stop the procedure only if no error has been corrected in a complete sweep passed.

One could ask the question: 'Is it worth passing more than one complete sweep over the block?' The answer: 'Certainly it is!' Let us suppose there has been observed more than one erased bit in $CW(y, p1)$ while the nth complete sweeping over the block, so it could not be corrected. Then there exists the possibility that all of these erased bits except exactly one will be corrected in the other directions. By the next attempt for correction of $CW(y, p1)$ we will find one erasure and we can already correct it. Let us call BER_n the value of BER after passing the nth sweep over the block.

Error Correcting Capability of the Above Code

As in the abstract of this article we said the test results showed that even very high values of the Bit Error Ratio (BER_0 was about $0.1 - 0.2$) can be decreased a lot. The tests were run by a program written in Pascal that generates errors in a data block and tries to correct them. Because of the very small value of BER_∞ we can get a valuable result only when running the program on a very large block of data. This is quite a slow way. Fortunately, it has become possible to create a relatively simple program in order to simulate these tests, that is, to compute the values of BER_n theoretically. Unfortunately, we have no place to write down the algorithm in these pages, but it is to say that the results computed by this program are very close in value to those generated by the tests, particularly when using the stochastical distribution of bits among codewords (see *Table 1* later).

If we have chosen a block large enough to be tested, we get a series of the values of BER_n decreasing monotonously and having a bottom bound while increasing the value of n to infinite. This bottom bound shows the ratio of the number of uncorrectable bits to the total amount of bits existing in the block. In other words it is equal to the probability of that event when a given bit of a block is uncorrectably erased.

Since the value of this probability depends on the value of X and P only, we can call it $BER(X, P)$. Using this $BER(X, P)$ we can compute for every value of Y (which means for every possible size of a block) the probability of the decoder being able to correct a given block completely.

Now here are some complete test results. Two of them have been made with the same parameters ($X = 16; P = 4$). The only difference between them is that one was made with a pseudo-random distribution of bits among codewords while the other one was made with the non-random distribution related in *Eq.* (1). We have also computed the results theoretically. These computed ones are shown in the middle column of *Table 1* between those of the tests in order to compare them. It can be seen that, firstly, the computed results are much closer to the results of the test using random distribution than to those of the other one; secondly, even the value of $BER_n(X, P)$ was decreasing faster when using a random distribution.

$$\begin{aligned} X &= 16, \\ P &= 4, \\ Y &= 2.56 \cdot 10^7, \\ BER_0 &= 0.1. \end{aligned}$$

The third test was made with another value of P, namely 8, which means that every bit is encoded in eight directions. Unfortunately, we have place here only for supplying the final result. This was $BER(16.8) = 1.22 \cdot 10^{-9}$.

It is worth noticing that by the most combinations of (X, P) tested by us a very simple rule has been accomplished to compute the value of $BER(X, P)$.

$$BER(X, P) = BER_0^{P+1}$$

where BER_0 means of course the Bit Error Ratio before the beginning of the error correction procedure.

There are even more interesting facts to observe. First, the value of $BER(X, P)$ does not depend on that of X. Really, if X has not a too large value (i.e. larger than a given bound), it has nearly no effect on the value of $BER(X, P)$. (So it can be called $BER(P)$.) Nevertheless there are no marvels on earth; it does have an effect on the speed of decreasing of BER_n,

Table 1

	Pseudo-random distribution of bits among codewords	Computed results	Non-random distribution of bits among codewords
1st complete sweep:			
1st direction:	8.146 E-02	8.147 E-02	8.148 E-02
2nd direction:	6.096 E-02	6.098 E-02	6.098 E-02
3rd direction:	3.962 E-02	3.962 E-02	3.985 E-02
4th direction:	2.027 E-02	2.017 E-02	2.103 E-02
2nd complete sweep:			
1st direction:	9.679 E-03	9.462 E-03	1.083 E-02
2nd direction:	3.496 E-03	3.241 E-03	4.787 E-03
3rd direction:	9.736 E-04	8.232 E-04	1.912 E-03
4th direction:	2.492 E-04	1.966 E-04	7.268 E-04
3rd complete sweep:			
1st direction:	6.811 E-05	5.500 E-05	2.661 E-04
2nd direction:	2.555 E-05	2.207 E-05	1.014 E-04
3rd direction:	1.461 E-05	1.362 E-05	4.322 E-05
4th direction:	1.166 E-05	1.137 E-05	2.302 E-05
4th complete sweep:			
1st direction:	1.091 E-05	1.079 E-05	1.543 E-05
2nd direction:	1.069 E-05	1.063 E-05	1.228 E-05
3rd direction:	1.062 E-05	1.059 E-05	1.134 E-05
4th direction:	1.060 E-05	1.058 E-05	1.093 E-05
5th complete sweep:			
1st direction:	1.059 E-05	1.058 E-05	1.080 E-05
2nd direction:	1.059 E-05	1.057 E-05	1.075 E-05
3rd direction:	1.059 E-05	1.057 E-05	1.073 E-05
4th direction:	1.059 E-05	1.057 E-05	1.073 E-05

and if it has a value greater than a given bound then the decreasing of BER_n (increasing n) will stop at a bound higher than $BER(P)$. Examining this fact nearly the same results can be obtained using both methods testing or computing.

Secondly, we should observe that the value of $BER(X, P)$ decreases exponentially while decreasing that of BER_0. The reason for this exponential behaviour is the following:

Let us imagine the simplest kind of uncorrectable error pattern, i.e. an erased information bit having all of its parity bits erased, too (*Fig. 2*).

The probability of finding this pattern examined a given information bit has a value of exactly BER_0^{P+1}, because all the $(P+1)$ bits examined must be found erased knowing that the event of an erasure of a bit has been independent of that of the other ones. Of course, there are more

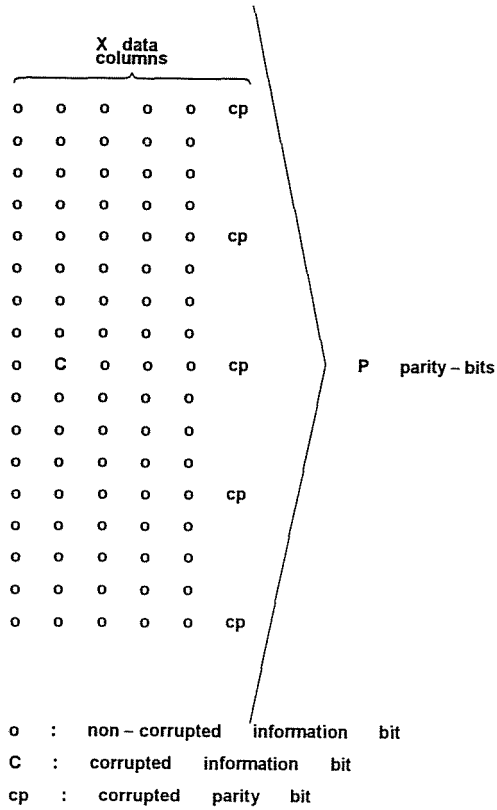

Fig. 2.

complicated uncorrectable error patterns, too, but the tests passed over have shown the above one to determine the value of $BER(X, P)$.

Error Escape Rate

Supposing that there are erasure-type errors only in the data the probability of the event that the decoder regards an erroneous-remained block is equal to zero after the decoding as an error-free one.

Definition of the Simple Error Correcting Version

The structure of blocks and the encoding mechanism is nearly the same as those in the erasure correcting version. The only differences are first we encode on characters (which means groups of bits of a determined size e.g. 8 or 16 bits) instead of single bits, secondly, as mentioned before, we have chosen the $(n, n-2)$ Reed–Solomon code as an encoding-decoding mechanism instead of the $(n, n-1)$ binary one. Therefore there exist two check characters instead of one parity bit per codewords. Otherwise we use the same encoding procedure (see *Fig. 3*).

The Decoding Procedure

The decoding procedure is almost the same as that of the erasure correcting version except one big difference: Since the errors existing in the data are simple errors (that is, not erasures) we cannot see if a given character is erroneous or clear. Thus we have to try to correct every codeword of the direction whose sweeping over the data we are passing. Attempting correction of a codeword five kinds of outcome can occur.

- The codeword is error-free. It remains intact.
- The codeword has one erroneous character. The decoder passes it over the correcting mechanism of the code and so it becomes error-free.
- The codeword has more than one erroneous characters in itself, but the decoder believes that it is error-free. The codeword remains intact (i.e. erroneous).
- The codeword has more than one erroneous characters in itself, but for the decoder it seems to have only one erroneous one in it. The attempt at correction results in a miscorrection, thus the codeword becomes as incorrect or even more incorrect than before.
- The same as the previous one, but the decoder recognizes that the codeword cannot be corrected at this time, so it remains intact.

The possible events are the same as those by the erasure correcting version except the third and the fourth ones. If the miscorrection rate of the code has a value of zero, these both events could not happen. So when choosing a code having a miscorrection rate small enough and supposing that the check-character-pair of a codeword becomes erroneous with the same probability as any one of its information characters, we can use the same programs as before in order to get the values BER_n. And of course we get almost the same results. (BER means here of course the Character Error Ratio.)

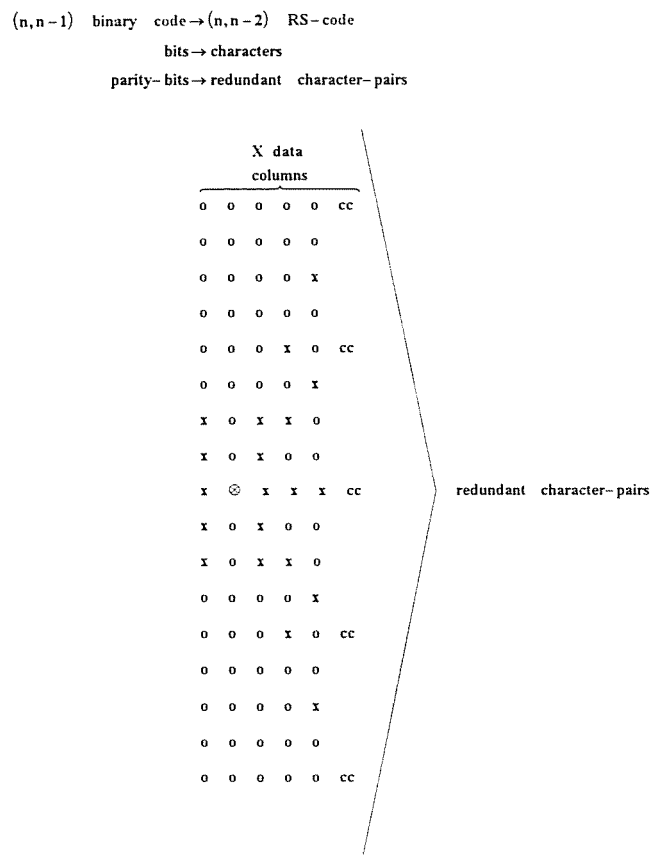

Fig. 3. The x's mean the bits of that codewords which contain the bit marked with the $\otimes$. The cc's mean the redundant chech-charcter pairs of these codewords

As mentioned before, the value of the miscorrection rate can be reduced when using a code with longer characters. (E.g. characters containing 16 bits instead of only 8.) Nevertheless it is not a catastrophe if we use a code having a miscorrection rate not so close to zero. In this case the value of BER_n will be decreasing to that of $BER(X, Y)$ a little slowlier, and indeed, even the value of $BER(X, Y)$ can grow. In spite of these facts it is not an unusable solution.

Error Escape Rate

We suggest that before encoding you should pass the data through a CRC encoder. When having done so, the result of the decoding can be controlled, and the error escape rate will be a function of the quality of the applicated CRC code. In this way there will not be any connection between the error escape rate of the block and the miscorrection rate of the Reed–Solomon code used in it.

Conclusion

We presented an interesting version of error correcting codes that makes use of the idea of multi-dimensional coding and is able to make very large data total error-free even if the ratio of the number of erroneous characters to the total amount of characters is about $0.1 - 0.2$. The encoding and decoding mechanism is very simple to do. Because of these properties it can be used when transferring very large data through a noisy channel or storing in a shoddy memory even if the data is required to be transferred or stored intactly, which means to obtain it after decoding total error-free. We have developed a method to compute the value of error correction capability in many cases and also tested a lot of them.

References

1. KURTZ, C.: Development of a High-Capacity Performance Optical Storage System, *Dig. 8th IEEE Symposium on Mass Storage Systems* (May 1987), pp. 107–111.
2. RAO, T. R. N. – FUJIWARA, E.: Error-Control Coding for Computer Systems, 6.2.3. Prentice Hall International.
3. DOONG, – WEIFANG, D. – HUIZHU, LU. – HEDRICK, G. E.: Interleaving Technique for Block Coding of Black-and-White Facsimile Data, Applied Computing: Technological Challenges of the 1990's, *Proc 92 ACM SIGAPP Symp Appl Comput SAC 92.* pp. 37–45, (1991).