Posted by permission

A Survey of the Theory of Error-Correcting Codes

by Francis Yein Chei Fung

The theory of error-correcting codes arises from the following problem: What is a good way to send a message across a noisy channel? The investigation of this problem has many manifestations in communications and computer technology. Perhaps the most familiar application is the spectacularly successful Compact Disc player, which uses Reed-Solomon codes to deliver dramatically clear and error-free sound by actually correcting errors that occur from dust and scratches.

Introduction

Since the very beginning of communication, there have always been errors and a need to deal with them. Human language is quite redundant, as an attempt to chat across a crackling phone line will convince you; you can probably understand a friend's message even if you only hear about half of it, as long as you do not miss the whole last half! (And if you do, you can ask your friend to repeat it.) Since the advent of electronic transmission in the form of telegraph and radio, people have been sending messages with other symbols, the dots and dashes of Morse code for instance, and the key problem has been to discover ways of cleverly introducing redundancy into the transmission so that even if it is garbled, one may be able to fully recover the message, or at least ask for a retransmission.

Let us begin with a quick example. Since computers are great users of binary information (1's and 0's), we shall send our messages in binary. Suppose we want to send a message which is either 1 (meaning "yes," for example) or 0 (meaning "no"), across a "channel" (e.g., a telephone line, a radio transmitter, or a CD laser). The problem is that the channel may be noisy (because of lightning, scratches, etc.); in other words, there is a certain probability that a 1 will be flipped to a 0 and vice versa. What should we do?

The first idea is just to send our message twice; if we want the other end to know that we meant 1, we send the "codeword" 11. If there is some noise, and the other end receives 01 instead, then the other end knows that an error occurred, and (if the circumstances permit) can ask for a retransmission. So this "repetition code," which takes 1 and encodes it as 11, and 0 as 00, allows us to detect one error. For some applications this is good enough.

However, if we get 01, we know that one error occurred, but was the first entry garbled from a 1, or the second entry garbled from a 0? There is no way of telling, since 01 is "as close to" 00 as it is to 11 (in the sense that it differs in the same number of places from each codeword).

Now instead of sending just two 1's, let us send three! Then the two codewords are 111 and 000. This time, if just one error occurs and we receive 101, we can still decode the received string (or "vector") by a "majority vote" and recover the correct 1. Therefore this code can not only detect, but also correct, one error, because 101 is closer to 111 than to any other codeword (in this case, 000 is the only other candidate). This notion of distance is the key to error correction; in constructing a code, we want to add our redundancy in such a way as to make codewords as far apart from each other as possible, without lengthening the message more than necessary.

History

The history of coding theory is inextricably bound to the development of transmission technology. For instance, the telegraph required an encoding scheme to transform English words into dots and dashes (the reason for Morse code). However, until around the 1940's, error detection (rather than correction) was sought after. Only with the advent of digital computing technology did the decoding of error-correcting codes become practical.

Error-correcting codes were first discovered abstractly in 1945 when Claude Shannon proved a theorem (described later) which states that, even with a noisy channel, there exist ways to encode messages in such a way that they have an arbitrarily good chance of being transmitted safely, provided that one does not exceed the "capacity" of the channel by trying to transmit too much information too quickly.

The first constructions of error-correcting codes, however, come from rather curious sources. In 1947, at Bell Labs, a mathematician named Richard Hamming became fed up with a computer which could detect errors in his input during weekend runs, but would then just dump the program, wasting the entire run. He devised ways to encode the input so that the computer could correct isolated errors and continue running, and his inquiries led him to discover what are now called Hamming codes.

Soon after, Marcel Golay generalized Hamming's construction, and constructed codes that corrected one error per word which used p symbols (rather than just 0 and 1) for p prime. He also constructed two very remarkable codes that correct multiple errors, and that now bear his name.

However, perhaps most curiously, one of the very same Golay codes appeared a few years earlier, in a Finnish magazine dedicated to betting on soccer games! We shall elaborate on this connection later. For more on this interesting history (and on the material to follow), see the introductory book by Thompson [7], and also Conway-Sloane [2], van Lint [4], Pless [5], and the references in these sources.

Basics

The formal definition of a code is this: a code of length n is a subset of all possible strings, or *vectors*, of n symbols (chosen from some alphabet, which will often be {0,1}), which has the property that every two members of the subset (i.e., codewords) differ in at least some fixed number of places d (the *minimum distance* of the code). If the number of codewords (which equals the number of possible messages) is M, then we shall say that it is an (n,M,d) code. Also, a code on {0,1} is called a *binary* code. So in our original example, {00,11} is a binary code of length 2, with 2 codewords and minimum distance 2. Similarly, {000,111} is a code of length 3 with minimum distance 3. We would like to have n small (for speed), M large (for efficiency), and d large (for reliability). These goals of course conflict, and a large part of coding theory is constructing codes that strike a balance between these three, and that are also practical to use.

For notation, the number of places in which two vectors differ is usually referred to as the *Hamming distance* between the two vectors. The *weight* of a vector is the number of non-0 entries in it (that is, its Hamming distance from the all 0's vector). Using mod 2 arithmetic (that is, 1+1=0) to add two vectors coordinate by coordinate (no carrying), we see that the Hamming distance between two vectors is equal to the weight of their difference. For example, 100110101 has weight 5, and

$$010010010-110100111 = 100110101,$$

so 010010010 and 110100111 have Hamming distance 5.

So if a codeword gets garbled, then the number of errors is exactly the Hamming distance of the received vector from the original codeword. In order that we detect the errors, the received vector must not land on another codeword. This is why a large minimum distance is so important; if there are fewer than d errors then we can at least detect the error. Also, if there are there are strictly fewer than $d/2$ errors, then the received vector is still closer to the original codeword than to any other.

Decoding

After we use a code to transmit our messages, we have to decode the vector which is received at the other end. The most intuitive way is roughly just to see which codeword is closest to the received vector. As long as your channel has more than 50% accuracy for each bit sent, then the best bet is the codeword which differs from the received vector in the fewest places, that is, with the smallest Hamming distance from it. This strategy is called *maximum-likelihood decoding*. (Notice that if, say, there is a 90% chance that each bit is wrong, then most of the received vector is likely to be garbled; but in this case, we can just reverse every symbol of the received vector and consider the error rate to be 10%.)

Now we can formally define "error-correcting;" a code is said to correct *e* errors if, whenever at most *e* errors occur when transmitting one codeword, this decoding process w ill always yield the correct message. So by what we said in the previous section,

a code with minimum distance d is $\lfloor (d-1)/2 \rfloor$ error-correcting.

This method of comparing our received vector to every codeword and selecting the closest one is theoretically the most reliable way of decoding. It is of course not very efficient, especially for big codes; a large part of coding theory, not explored any further here, deals with finding codes that can be decoded efficiently and implementing decoding schemes for them.

Constructing Some Codes

The double repetition code has minimum distance 2 and so can only detect one error, and yet it doubles the length of the message sent. Clearly there is room for improvement. If we merely wish to detect one error when transmitting the message 100, we do not need to send the whole message twice. We can detect one error by adding a "parity bit" as follows: just append a 0 or a 1 in such a way that the resulting codeword has an even number of 1's. For example,

$$100 \longrightarrow 100\ 1 \text{ and } 110 \longrightarrow 110\ 0$$

In this manner, we impose parity on the codeword. If any single error occurs (say 1001 becomes 1101), it will change the number of 1's by one and thus throw off the parity. So when we perform a "parity check" on the received vector by adding all the coordinates, we will detect the error. But we cannot tell which digit is wrong, since all single errors have the same effect.

Notice that, given a message of *m* symbols, this process creates a (*m*+1,2^*m*,2) code. This idea of making "parity checks" is the precursor to more sophisticated codes.

A generalization of this process yields the Hamming codes. Hamming studied the problem of trying to find a way to correct a single error efficiently, using as few extra bits as possible. Hamming's strategy was to impose parity on parts of the codeword interleaved in such a way as to cover the maximum amount of ground.

He looked at this situation. Suppose you have a message of *m* binary symbols, and you want to construct a code of length *n* by adding *k*=*n*-*m* "parity bits." Every parity bit will make a certain part of the codeword, plus itself, sum to 0; checking that these positions do in fact sum to 0 will be called a "parity check." How many parity bits do we need to ensure that our code corrects single errors?

To answer this, suppose we transmit our codeword. Then when we receive a length *n* vector, we perform *k* parity checks. If we received our codeword error-free, then by design all the parity checks sum to 0. For our code to detect an error, some parity check must fail whenever an error occurs. For our code to *correct* an error, each single error, which could occur in any of the *n* positions, must disturb the set of values of the parity checks in a different way, and they must all be distinct from the result of checking the correct codeword.

All in all, the total number of distinct sets of parity check values, 2^*k*, must be at least *n*+1. So Hamming searched for codes of length 2^*k*-1, which would require *k* check digits.

We demonstrate with an example of encoding, which takes any 4-bit message and encodes it into a 7-digit codeword.

Suppose we want to encode the word 1100. Then we construct the codeword (with symbols called *X*1, ..., *X*7) by putting the four "message bits" into the 3rd, 5th, 6th, and 7th slots, like so:

X_1	X_2	X_3	X_4	X_5	X_6	X_7
_	_	1	_	1	0	0

Now to fill in the rest of the positions with "parity checks," we fill in the 1st slot with the digit that will make the sum of the 1st, 3rd, 5th, and 7th entries equal to 0. This implies

$$X_1 + 1 + 1 + 0 = 0, \text{ so}$$
$$X_1 = 0$$

The 2nd slot ensures that the sum of the 2nd, 3rd, 6th, and 7th entries is 0, that is,

$$X_2 + 1 + 0 + 0 = 0, \text{ so}$$
$$X_2 = 1$$

Finally the 4th slot is chosen to make the 4th, 5th, 6th, and 7th entries add up to 0, so

$$X_4 = 1$$

Thus our codeword for 1100 is 0111100.

Hamming's ingenious selection of which coordinates to add together comes from the fact that binary expansions of 1, 3, 5, and 7 (i.e., 001, 011, 101, and 111) all have a 1 in the rightmost position. Similarly, 2, 3, 6, and 7 all have a 1 in the next-to-last position in their binary expansions. Using this list, we put our message bits into those positions which are not powers of 2, and then fill in the other positions by checking parity. The reason for this ordering comes from the fact that if a single error occurs, say in the 6th position, then that one error will upset just those parity checks that contain X6 in them, so

$$X_1 + X_3 + X_5 + X_7 = 0$$
$$X_2 + X_3 + X_6 + X_7 = 1$$
$$X_4 + X_5 + X_6 + X_7 = 1$$

and the values of the parity checks give the binary expansion of 6 (110), the error position! Theoretically, of course, we could arrange that the first four positions contained the message, and the last three were the check digits (or any other ordering); this is one example of how one ordering can possess certain practical advantages over another.

We now derive some basic facts about this code. The procedure above generates a length 7 codeword for each length 4 message, so there are 2^4 = 16 codewords out of 2^7 = 128 vectors of length 7. Also, we find that the minimum distance between any two codewords is 3. If you change exactly one message bit, that affects at least two parity checks (because message bits are not located at powers of 2); if you change two, then each one affects at least two parity checks, and not exactly the same ones because the position numbers have different binary expansions. If you change more than two message bits, the codewords are already at least at distance 3!

We have constructed the (7,2^4,3) Hamming code. In general, a binary Hamming code of length 2^*k*-1 will have *k* check digits and thus 2^*k*-1-*k* message positions left over, so it will be a $\left(2^k-1, 2^{2^k-1-k}, 3\right)$ code.

We now prove that these codes (and all of the codes above) have the property that the sum of any two codewords is another codeword, that is, they are *linear*. This extra structure allows us to use the tools of linear algebra to study such codes.

First note that repetition codes are linear, since the sum of any two messages is another valid message when we add codewords coordinatewise.

One can also see this easily for the parity codes, because the equations "even + even = even" and "even + odd = odd" still hold for the weights of the codewords. For instance,

100 1+110 0 = 010 1.

The only time you lose any 1's is when two 1's in the same position cancel, and this does not affect parity.

A little tweaking of this argument shows that the Hamming codes (which just come from overlapping parity checks, after all) are also linear.

As we said, this linearity condition is quite useful. For instance, one can show that in a linear (*n*1,*M*,*d*) binary code, *M* is always of the form 2^*m*, and we can arrange that *m* of the positions are message bits. In fact, we can even arrange that the code is obtained by specifying some sequence of parity checks on the messages, just as in the Hamming codes. We will not dwell on these or the many other advantages of using linear algebra to study codes, except to say that many properties of codes become quite transparent by the application of this basic tool of mathematics.

Shannon's Information Theorem

Clearly, the idea of a code is intimately tied with the idea of how much information the code carries. It makes sense that if a codeword has *n* bits of 0's and 1's, and the message itself has only *m* bits, then the rate R of the code is

$$R = \frac{m}{n}$$

This rough statement can be extended to nonlinear codes where the number of codewords *M* is not always of the form 2^*m* by defining the rate as

$$R = \frac{\log_2 M}{n}$$

There is a fundamental theorem of Shannon which guarantees the existence of codes that handle a given noisy channel. Given a channel with probability p of sending each bit correctly, we can concoct a quantity called the *capacity* $C(p)$ of the channel, which is roughly the maximum amount of information that can be sent across it in a unit amount of time. In fact,

$$C(p) = 1 + p\log_2(p) + (1-p)\log_2(1-p)$$

Then Shannon's theorem states that for any rate R strictly less than the capacity $C(p)$ and any $e>0$, there exists some code with a sufficiently large length and with rate at least R, such that the probability that an error will occur in the message is less than e. This means that we can achieve, with long enough codes, communication that is as safe as we like. These codes do not have to be linear, and the proof does not construct them; all we know is that they exist. One challenge of coding theory is to construct such codes.

Perfect Codes

Some codes have the very nice property of being "as tight as possible" inside of the space of possible vectors. More precisely, the collection of all vectors that differ in at most r places from a codeword forms a "ball" of radius r around it. If our code has length n, then the ball of radius r will have $\sum_{i=0}^{k}\binom{n}{i}$ vectors in it, because there are $\binom{n}{i}$ vectors at distance i from a given vector. Certainly if r is less than or equal to the number of errors e the code corrects, then any two balls are disjoint.

Now, we ask, what codes have the property that the balls of radius e not only are disjoint, but pack the space completely? This question has been satisfactorily answered. All possible parameters (n, M, d) of perfect codes are known. They correspond to the linear perfect codes; that is, certain trivial cases, the Hamming codes (and analogues over other finite fields), and two remarkable Golay codes. The only other perfect codes are certan nonlinear codes with the same parameters as the Hamming codes.

First, notice that every repetition code of odd length is perfect, because any vector either has more 1's than 0's (in which case it is strictly closer to the all 1's vector), or vice versa---no ties! At the other extreme, the code consisting of all possible vectors of a given length is also perfect (with e=0). Finally the code with only one codeword is of course perfect.

One can show that the Hamming codes we constructed above are also perfect. In fact, in a Hamming code of length 2^k-1, a ball of radius 1 around a codeword contains 1+(2^k-1) vectors, 1 for the codeword itself and one for each position. But there are 2^{2^k-1-k} codewords, so these balls contain

$$2^{2^k-1-k}2^k = 2^{2^k-1}$$

vectors; thus the balls of radius 1 exactly exhaust all vectors of length 2^k-1 and the code is perfect.

In addition, the special binary code on 23 symbols, with parameters (23, 2^12, 7) discovered by Golay, is perfect (and it corrects 3 errors!). These, coupled with certain nonlinear codes that have the same parameters as the Hamming codes, are all of the binary perfect codes. If we allow more symbols in our alphabet than just 0 and 1, then we get analogues of the Hamming codes, and another Golay code of length 11, this time on three letters (say 0, +, and -) and with parameters (11, 3^6, 5).

This completes the list of all linear perfect codes.

The Finnish Football-Pool Connection

It is this "ternary" Golay code which was first discovered by a Finn who was determining good strategies for betting on blocks of 11 soccer games. Here, one places a bet by predicting a Win, Lose, or Tie for all 11 games, and as long as you do not miss more than two of them, you get a payoff. If a group gets together in a "pool" and makes multiple bets to "cover all the options" (so that no matter what the outcome, somebody's bet comes within 2 of the actual outcome), then the codewords of a 2-error-correcting perfect code provide a very nice option; the balls around its codewords fill all of the space, with none left over.

It was in this vein that the ternary Golay code was first constructed; its discover, Juhani Virtakallio, exhibited it merely as a good betting system for football-pools, and its 729 codewords appeared in the football-pool magazine *Veikkaaja*. For more on this, see Barg's article [1].

Reed-Solomon Codes and CD players

Compact Disc technology makes use of so-called "Reed-Solomon" codes to encode a binary representation of music into an error-resistant format on aluminum discs (the process used to convert music into binary format is itself interesting and uses the "Nyquist sampling theorem" which guarantees that an audio signal can be reconstructed from sufficiently many samples, provided the signal has a bounded frequency). There are two particular codes used; both use alphabets of 2^8 symbols, and they are of length 28 and 32, and both can correct 2 errors per codeword. The main sources of error being scratches and dust (which generate "bursts" of errors), a clever "interleaving" of the two codes allows the CD player to correct over 4000 (!) consecutive errors. For more information on this, one can consult the (understandable) original technical report by Hoeve et. al. [3].

Further Remarks

We have of course merely scratched the surface of coding theory. Many other families of codes exist, with diverse properties and constructions. The implementation of coding and decoding algorithms presents its own subtleties. In addition, coding theory both gives and takes from the whole of pure mathematics. For instance, the class of Goppa codes arises from deep theorems of algebraic geometry (and have some better parameters than any previous codes!). And the two Golay codes are very special; further study of them yields many interesting objects, in particular the so-called Mathieu groups, which are "sporadic simple groups," a special sphere packing in 24 dimensions (the Leech lattice), and even more sporadic simple groups. For more on codes and these connections, see the introductory book by Thompson [7] and Conway's impressive book [2] (which contains far more on many things).

The real-world applications go far beyond the CD player to all of computing and transmission technology, including hard disk drives, satellite communications, and digital television. There is much more to be said about the powerful and important theory of error-correcting codes.

References

[1] Barg, Alexander. "At the Dawn of the Theory of Codes," The Mathematical Intelligencer, Vol. 15 (1993), No. 1, pp. 20--26.

[2] Conway, J. H., and N. J. A. S. Sloane. Sphere Packings, Lattices, and Groups, 2nd ed., Springer-Verlag, 1993.

[3] Hoeve, H., J. Timmerman, and L. B. Vries. "Error Correction and Concealment in the Compact Disc System," Philips Technical Review, Vol. 40 (1980), No. 6, pp. 166-172.

[4] van Lint, J. H. Introduction to Coding Theory, 2nd ed., Springer-Verlag, 1992.

[5] Pless, Vera. Introduction to Error-Correcting Codes, 2nd ed., Wiley, 1989.

[6] Sloane, N. J. A. S. A Short Course on Error-Correcting Codes, Springer-Verlag, 1975.

[7] Thompson, Thomas M. From Error-Correcting Codes Through Sphere Packings To Simple Groups, Mathematical Association of America, 1983.

Francis Yein Chei Fung is a second-year graduate student in mathematics at Princeton University, working under Prof. John H. Conway. He grew up in Manhattan, Kansas (the Little Apple) and did his undergraduate studies at Kansas State University. His fields of interest include representation theory, group theory, and algebraic topology.

Please direct inquiries to fycfung@math.princeton.edu. (dead link)

Appeared in volume 1 in 1994 Tangents: Harvard-Radcliff Math Bulletin.

---

posted 5-13-2004 wdj