02 Information theory

02.04 Special encodings

In this lecture we introduce constant-length redundant encodings which make use of extra bits in order to grant to the code the capability of detecting and/or correcting some of the errors that may affect the code words.

A. Error models

The error model is based on two key assumptions (**Slide 2**): the bits in a word are affected by independent errors, each bit has a probability *p* of being affected by an error.

From statistics we know that the joint probability of two independent events A and B is given by the product of their probabilities. In symbols:

Prob (AB) = Prob(A) * Prob(B)

Since the probability is defined in the [0,1] interval, the joint occurrence probability of two events is always less or equal than the probability of each of them.

The probability of a bit to be correct is the complement of its error probability: 1-*p*. A word of *n* bits is correct if and only if all its bits are correct. Hence, the probability of a word to be correct is given by the product of the correctness probabilities of its *n* bits: $(1-p)^n$.

The word error probability is the complement of the probability of the word to be correct (Equation 1 in Slide 2) which can be approximated by *np* when *p* is very small (as it usually is).

Slide 2 also reports the expressions of the probabilities of a word of *n* bits to be affected by errors of a given multiplicity, from 1 to *n*. What is relevant is that the probability of the word to be affected by errors of multiplicity *k* is dominated by p^k, so that the probability decreases for increasing values of *k*. In particular, any word is much more likely to be affected by a single error than by 2 or more simultaneous errors. That's the reason why single errors are the main target of any error-detection/error-correction techniques (**Slide 3**).

Given 2 words of the same length, w1 and w2, the number of bits which take different values in the two words is called *Hamming distance* between w1 and w2 (**Slide 4**). For instance, the Hamming distance between w1=1100 and w2=1010 is 2. The Hamming distance of a code is the minimum Hamming distance between any pair of code words. Consider, for instance, a code composed of the following code words: 00, 01, 10, 11. Its Hamming distance is 1. Now consider a different code with code words 000, 011, 101, 110. Its Hamming distance is 2.

Any irredundant code has Hamming distance 1 and it has no error-detection capabilities (**Slide 5**). In fact, single error may transform a code word into a different one which still belongs to the code, making it impossible to detect the error. The minimum Hamming distance of an error detecting code is 2. In order to detect up to *e* simultaneous errors we need a code with Hamming distance *e*+1.

If we want not only to detect e errors, but also to correct them, we need to increase the Hamming distance of the code to 2e+1. Consider, in fact, a codeword w1. If the code has Hamming distance 2e+1, all other code words differ from w1 for at least 2e+1 bits. Now assume w1 is affected by e simultaneous errors. The erroneous version of w1 has distance e from w1, while it has at least distance e+1 from all other code words. Hence, from the erroneous version of w1 we can obtain the original one by taking the closest code word.

Slide 6 provides some definitions and the classification of error-detection codes (EDC) and error-correction codes (ECC). Both EDC and ECC are said to be *separable* codes if any code word is composed of *r* information bits and *m* control bits with assigned positions. Replication codes (**Slide 7**) represent trivial examples of EDC and ECC.

B. Parity code

The simplest parity code makes use of a single control bit, called *parity bit,* which is added to the *r* information bits to build a codeword of length *n=r*+1. The value of the parity bit is determined is such a way that the codeword contains an even number of 1's (**Slide 8**). For instance, if we add a parity to an irredundant code with *r*=2 we obtain the following codewords: 00**0**, 01**1**, 10**1**, 11**0**. The parity bit grants to the code the capability of detecting single errors. In fact, any single error affecting a codeword produces an odd number of 1's in it, so that the resulting word can be recognized as not belonging to the code.
Notice that parity bits are highly efficient, in that they allow us to build a 1-EDC by adding only 1 control bit regardless of the number of information bits we have.

Assume we have a stream of *L* bits to be transferred across a noisy channel and that we want to add parity bits to make the transmission more reliable. To this purpose, we split the bit stream in chunks of *r* bits and we add a parity bit to each chunk. Thanks to the control bit, the receiver can check the parity of each received word of *n=r*+1 bits and ask for retransmission whenever an error is detected.

The question is: what's the ideal value of *r* which provides a good protection at the minimum cost? The answer is non trivial because of the following conflicting observations:

- the larger the word, the smaller the marginal cost of the parity bit;
- the larger the word, the higher the probability of having two or more errors affecting the same word and the lower the protection provided by the parity bit;
- the larger the word, the higher the probability of having an error which requires retransmission;
- the larger the word, the higher the cost of each retransmission.

Given the length of the stream (L), the bit error probability (p), and the chunk length (r), **Slides 9 and 10** show how to compute the probability of receiving a correct word (*P_w_correct*), the probability of receiving a word affected by one or more errors (*P_w_error*), the probability of receiving a word affected by exactly one error (*P_w_1error*), the probability of receiving a word affected by multiple errors (*P_w_Merror*), the number of chunks (*N_words0*), the average number of retransmissions per word (*N_reTx*), the overall number of words transmitted including the retransmissions (*N_words*), and the overall number of bits transmitted across the channel (*L_total*). All these parameters allow us to evaluate the impact of r in terms of *overhead* (i.e., the percentage of extra bits sent across the channel) and in terms of *failure probability* (i.e., the probability of receiving a word containing 2 or more errors which are not detected by the code).

A spreadsheet is provided together with the lecture notes to allow you to evaluate the trade-off between overhead and failure probability. **Slide 11** plots the results (as functions of *r*) for L=1000 an p=0.001.

C. Hamming code

The 1-EDC based on a single parity bit has Hamming distance 2. To achieve higher values of the Hamming distance we need to add more independent parity bits (**Slide 12**). If we have a word with *m* independent parity bits, the configuration of all the parity bits is called *syndrome*. A parity code provides the possibility of correcting the errors which affect a word if the syndrome of its parity bits uniquely identifies the error.

The Hamming codes are a family of ECCs using the minimum number of control bits required to obtain Hamming distance 3 in order to correct single errors. According with the Hamming rule reported in **Slide 13**, the minimum number of control bits required to correct single errors on a word with *r* information bits is the minimum integer number *m* such that 2^m is greater or equal than *r+m*+1. In fact, the 2^m configurations of the *m* control bits must encode all possible single errors affecting one of the *n=r+m* bits, plus the error-free case. The value of *m* corresponding to the first 5 values of *r* is reported in the table of Slide 13.

Notice that the Hamming rule establishes is not a constructive rule, in that it establishes the number of control bits to be used, but it doesn't tell us how to use them. A constructive rule is provided in **Slide 14**.

We represent a generic codeword as an array of bits, the positions of which are numbered starting from 1. In Slide 14 you can find a pictorial representation of the array of bits, with the positions expressed in binary notation. All the positions which are powers of 2 (namely, 1, 2, 4, 8, 16, and so on) are used to represent parity bits. All other bits can be used to represent information.

Now assume that we want to build an Hamming code starting from an irredundant constant-length code of *r*=5 bits. The 5 information bits can be mapped on the array of bits of Slide 14 skipping all the positions reserved to parity bits. Namely, the 5 information bits can be mapped in positions 3, 5, 6, 7, and 9. The array of slide 14 is constructed in such a way that the control bits required to grant error-correction capabilities to

the code are exactly the ones that we skipped when mapping the information bits. In fact, in our example we skipped 4 control bits, which are exactly the parity bits predicted by the Hamming rule of Slide 13 for m=5.

What we need at this point is a rule for computing the value of the control bits while guaranteeing their independence. Such a rule is driven by the binary encodings of the positions of the bits: the generic parity bit, placed in position 2^k, is computed over the bits the positions of which are encoded with a 1 in position k. For instance, the first parity bit, in position $1=2^0$, is computed over bits 3, 5, 7, and 9. Similarly, the second parity bit, in position $2=2^1$, is computed over bits 3, 6, and 7.

Slide 15 shows how to use the parity bits to detect and correct single errors. The example of Slide 15 refers to a 6-bit Hamming code with 3 control bits. If the codeword contains no errors, its syndrome is 000. If the codeword contains 1 error, the syndrome encodes its position, so that it can be corrected by inverting the corresponding bit. Finally, if the codeword is affected by more than 1 error, the he provides a wrong information and the error cannot be corrected. Even worse, since there is no way of distinguishing the case in which the error cannot be corrected, in case of multiple errors we might endup using the wrong word as it was correct.