

Reed Solomon Code

Jie Gao*

February 19, 2007

1 Introduction to Coding

Coding is fundamental in communication systems. There are two main categories of coding, source coding and channel coding. Intuitively, source coding refers to data compression. A picture taken by a digital camera in its raw format (bitmap, .bmp file) is large in size. We often immediately perform data compression to reduce its size by using, say, a JPEG format. This is an example of source coding. Channel coding considers the delivery of data through a noisy channel reliably. The channel here refers to a broad notion of communication method, i.e., the media through which data exchange happens. For example, wireless communication uses the air, which is the channel. Or, in the case of storage, when we burn some data on a CD and carry it physically and load the data to a different computer, the CD is the channel. For both cases we can see that the channel is unreliably, or noisy. In other words, data transmitted through the channel might go wrong — a bit is flipped, a data block on a CD is completely erased, etc. Thus channel coding is used to ensure the reliable delivery of data through a noisy channel, by adding duplications or redundancies in the data.

In a communication system source coding and channel coding are often both used. Take the cellular network as an example, here is a general pipeline: the voice from the sender is first compressed in a favorable format (hopefully with reduced size). Then the compressed data goes through an encoder to supplement it with redundancy. The coded information is then delivered through a wireless communication channel to the receiver side. When the receiver receive the data, it might be corrupted in the middle, the receiver then performs decoding to try to recover the original correct data. If the data is largely corrupted such that the decoder is unable to recover it, then this data is thrown away and higher-level layers (e.g., the transport layer) will handle re-transmission. The receiver, once recovered the data, will again de-compress it to its original voice signal, which is what the receiver hears.

In this note we focus on the channel coding problem. We will discuss two widely used coding schemes, block coding and convolutional coding. This note is devoted to block coding scheme, in particular, Reed-Solomon code.

2 Preliminaries

We first just talk about some simple and naive coding schemes. The idea of error detection and error correction codes adds, in some way, certain levels of redundancy to the data. Here are something simple you can do.

2.1 Duplication

Probably the simplest thing to do is just duplicating each bit of the data. For example, suppose the data to be delivered is 01 (2 bits). Now we simply duplicate each bit by say 7 bits. So we have the coded data string

*Department of Computer Science, Stony Brook University, Stony Brook, NY 11794, USA, jgao@cs.sunysb.edu.

as 00000001111111. This new and longer string is delivered to the receiver, during which some bits might be flipped. For example, the receiver receives 00010001101101. The receiver will now sum up each 7 bits, which is about 1,5. If the sum turns out to be below 4, then that bit is considered to be 0; otherwise it is considered to be 1.

Certainly it is an error detection code (a correct data string should have each 7 bits to be exactly the same, anything that violates this property is wrong); but also an error correction code (correct it to the one with minimum flips). The drawback of this code is obvious — it is not efficient as each bit is repeated 7 times. Thus data rate is reduced by 7 times.

2.2 Parity check

Now let's see another simple scheme that is efficient but less powerful. This is however widely used in communication systems. The idea is to append at the end of each data sequence a 'parity bit'. The parity bit is set up such that the number of 1's in the whole data sequence (including the parity bit) is even. This is more efficient than the previous scheme as only 1 extra bit is included. But it is less powerful as it can only detect 1 bit of error and can not correct any error. Essentially, if one of the bit (either a data bit or the parity bit) is flipped during transmission, then the number of 1's will be odd — the receiver will detect that. However, there is no way to know which bit was flipped. Further, 2 bits error or more errors may stay undetected.

2.3 Hamming distances, detection and correction abilities

From the above examples, we can see there is a tradeoff between the number of bits used (the efficiency of the code) v.s. the detection/correction abilities. Here is a way to characterize the detection/correction abilities by examining the hamming distance of the codewords.

Now we assume that each data string is chopped into k bits segments. We code each segment into a string of length n , i.e., with $n - k$ more bits than the original raw data. The data rate is reduced by a factor of n/k . For each input of k -bit string K , we denote by N the coded string of n bits, and name it the *codeword*. The insight of coding is to design how to code the raw data into codewords that are 'special'. In other words, not every n -bit string is a valid codeword (otherwise there is no way to tell whether a received string has error or not since anything is valid). Take the above two examples, in the simple duplication form, the codewords have the form that every 7 bits are exactly the same. In the parity check scheme, each valid codeword has even number of 1's.

Now the ability to detect or correct errors really depends on the Hamming distance between the valid codewords. The hamming distance of 2 n -bit strings, A and B , is the minimum number of flips to turn A to B . It is obvious that the hamming distance of any 2 n -bit strings is at most n . In the example of the parity check scheme, we take a data string of k bits and turn it to a codeword of $n = k + 1$ bits. Now the minimum distance of 2 valid codewords is 2 — if we flip two bits, then we can turn one codeword with even number of 1's to another string of even number of 1's (which is also a valid codeword). Now we can use Hamming distances of codewords to characterize the ability to detect and correct error.

Lemma 2.1. *If the minimum Hamming distance between 2 valid codewords is m , then the scheme can detect $m - 1$ errors.*

Proof: If the minimum number of flips needed to turn one codeword to another codeword is m , then any $m - 1$ errors can not derive a valid code, thus will be detected. \square

A natural way to correct error is to take the codeword that is 'closest' to the input (corrupted) data in Hamming distance. In other words, take the minimum number of flips such that the result is a valid codeword. Now if the number Hamming distance of 2 valid codewords is m and the number of errors is $\lfloor m/2 \rfloor$, then by taking the most similar codeword, we can always correct the error. Thus we have,

Lemma 2.2. *If the minimum Hamming distance between 2 valid codewords is m , then the scheme can correct $\lfloor m/2 \rfloor$ errors.*

3 Reed Solomon Code

Reed Solomon codes are used in a wide variety of commercial applications, most prominently in CDs and DVDs, and in data transmission technologies such as DSL, DVB (Digital Video Broadcasting) and WiMAX (Worldwide Interoperability for Microwave Access). One significant application of Reed Solomon coding was to encode the digital pictures sent back by the Voyager space probe of Uranus. Modern versions of Reed Solomon codes with convolutional coding were and are used on the Mars Pathfinder, Galileo, Mars Exploration Rover, etc.

Reed Solomon code is a scheme of block coding — that is, if during transmission a block of data is missing or completely erased, as long as enough of the remaining blocks are received, there is still hope to recover the data. The channel in which a block of data might be erased is also named the *erasure channel*. And codes designed for erasure channels are also named *erasure codes*. Erasure channels are commonly seen in the applications of Reed Solomon Code. For CDs or DVDs, physical scratches typically destroy one or more data blocks. In storage systems, a hard drive fails with the pattern that a whole block can not be read out. In digital video, it often happens that a data frame is missing or corrupted due to either bursty error or packet drop in the network (because of congestion).

3.1 Mathematical formulation

The key idea behind a Reed Solomon code is that the data encoded is first visualized as a polynomial. The code relies on a fact from linear algebra that states that any k distinct points uniquely determine a polynomial of degree at most $k - 1$.

The polynomial is then encoded by its evaluation at various points, and these values are what is actually sent. During transmission, some of these values may become corrupted. Therefore, more than k points are actually sent. As long as not too many of the points are received incorrectly, the receiver can guess what the original polynomial was, and hence decodes the original data.

To show exactly how this is performed, we first move away from the 0, 1 encoding of data and use a larger alphabet (conceptually using 01 string is the same as using some other alphabet, say, English has an alphabet of 26 elements). For the ease of explanation, suppose we just use the real numbers to represent the data. The input data is a sequence of k real numbers $a_0, a_1, a_2, \dots, a_{k-1}$. We want to deliver this sequence of numbers to the receiver.

Now we define the following polynomial of a variable x with the data as coefficients:

$$f(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \dots + a_{k-1}x^{k-1}.$$

Obviously if we know all the coefficients a_i then this polynomial function of x is determined. On the other hand, with the polynomial determined all the coefficients are determined as well. Thus, if somehow we are able to transmit this polynomial to the receiver, then the receiver will also know the coefficients, which is exactly what we want.

How to send this polynomial function $f(x)$ from the sender to the receiver? Notice that the polynomial $f(x)$ has maximum degree $k - 1$. From algebra we know that if we know k sample points then a degree $k - 1$ polynomial is uniquely determined. Thus we will send some sample points of $f(x)$ to the receiver. Since some data might get lost in the middle, we will send more than $n \geq k$ sample points, hoping that at least k samples get to the receiver correctly.

Now simply, we will send the following n numbers to the receiver:

$$\beta_0, \beta_1, \dots, \beta_{n-1},$$

which are the samples of the polynomial when $x = 0, 1, 2, \dots, n$, the number $\beta_i = f(i)$. Assume that some of the numbers are lost during the transmission, without loss of generality let's assume the first k numbers

$$\beta_0, \beta_1, \dots, \beta_{k-1},$$

are received (the scheme works for any k of the original n numbers). Now let's see how to recover the original k data $a_0, a_1, a_2, \dots, a_{k-1}$.

At this point the receiver has the following information:

$$\begin{aligned} \beta_0 &= f(0) = a_0 \\ \beta_1 &= f(1) = a_0 + a_1 + a_2 + \dots + a_{k-1} \\ \beta_2 &= f(2) = a_0 + a_1 \cdot 2 + a_2 \cdot 2^2 + \dots + a_{k-1} 2^{k-1} \\ &\dots \\ \beta_{k-1} &= f(k-1) = a_0 + a_1 \cdot (k-1) + a_2 \cdot (k-1)^2 + \dots + a_{k-1} (k-1)^{k-1} \end{aligned}$$

Remember that the receiver knows β_i but does not know a_i . In fact the receiver wants to recover a_i . This is not difficult – now we have k linear equations with k variables a_i , and we can solve this linear system to calculate these a_i 's. This is exactly what the decoding algorithm does!

3.2 Properties of Reed Solomon codes

There are a number of nice properties of the Reed Solomon codes.

1. The Reed Solomon code is a type of erasure code. Given the input k numbers a_0, a_1, \dots, a_{k-1} , we encode it into n numbers $\beta_0, \beta_1, \dots, \beta_{n-1}$. Now the sender will send these n numbers to the receiver. If the receiver gets at least k numbers, no matter what k numbers they are, the receiver can always recover the original k numbers a_i . Notice that this is already the best possible — one has to use k numbers to recover the original data of k numbers, since one can not generate information out of nowhere. This property makes Reed Solomon code particularly useful in applications when errors happen in bursts, say a sequence of numbers are lost.
2. Another nice property is that the introduction of redundancy is natural. If one wants more redundancy (thus more reliable), simply send more samples of the polynomial.