COMPSCI 650 Applied Information Theory Apr 5, 2016

Lecture 18

Instructor: Arya Mazumdar *Scribe:* [redacted]

1 Correcting Errors in Linear Codes

Suppose someone is to send us a message x on an errornous channel. To do so he will first obtain the corresponding codeword c by multiplying x with the generator matrix G:

$$\underset{1\times k}{x}\ \underset{k\times n}{G} = \underset{1\times n}{c} \tag{1}$$

He then sends the codeword c on the channel; however, our channel introduces some error e:

$$c + e = y \tag{2}$$

On the other side of the channel we will recieve y and our objective is to infer the original message x. If we could somehow identify the error e which the channel has introduced, we could obtain the codeword c sent by the sender which directly would lead us to the original message x.

Lets multiply the parity-check matrix H with y:

$$\begin{aligned} Hy^T &= H(c^T + e^T) \\ &= Hc^T + He^T \end{aligned}$$

Hc^T is zero by defintion (c is a codeword if multiplying it by H gives 0), thus:

$$Hy^T = He^T \tag{3}$$

This is called the *syndrome* of the received message.

If for every possible correctable error the channel may produce (e), we obtain a different value for the syndrome (He^T), then we will be able to uniquely identify that error by simply multiplying the parity-check matrix with the received message y. And if we can identify the error of the received message, we will be able to obtain the codeword by flipping the error bits in the received bit stream. More formally, if E is the set of errors which we are able to correct:

$$E = \{\text{set of correctable errors}\} \subseteq \{0,1\}^n$$

$$e_1, e_2 \in E \Rightarrow He_1^T \neq He_2^T$$

or in other words $H(e_1^T + e_2^T) \neq 0$. (Note that + and - are the same in binary).

Example. Error Correcting in Hamming Code

For Hamming Code we have the parity-check matrix:

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{4}$$

Note that all columns in this matrix are different. Suppose there is only one non-zero entry in our error e. Then He^T will simply produce one of the *unique* columns of H. So for Hamming code, the set of correctable errors are the errors with only one non-zero entry:

$$E = \{e \in \{0,1\}^n : \mathrm{wt}(e) = 1\}$$

As you can see, we can identify any one-bit error in Hamming Code. Recall the minimum distance in Hamming code is 3, that is the hamming distance between any two codewords is at least 3. If the minimum distance were any lower, say 2, than a single-bit error could make us be as far away from the original codeword as we are from some other codeword. This would not allow us to distinguish which of the two is the original codeword. However, as long as the minimum distance between any two codewords is 3, any single bit error would still keep us closer to the original codeword and is correctable. To confuse a codeword in Hamming Code, we need at least two errors.

Now what if we want to correct more than one error? In other words, we want the set of correctable errors to be:

$$E = \{e \in \{0,1\}^n : \mathrm{wt}(e) = t\}$$

To do so, we must build our parity-check matrix H in such a way that:

$$\forall e_1, e_2 \in E : H(e_1^T + e_2^T) \neq 0$$

BCH codes can correct t errors as long as t is constant (i.e. does not work if t as a function of n). BCH codes are beyond the scope of this class but you can learn more in next year's Coding Theory course.

2 Correcting Erasures in Linear Codes

In the previous section we investigated decoding messages on channels which introduce errors (i.e. flip codeword bits). In this section we will look into a channel which erases bits. In this channel, each bit could either remain unchanged or be changed to a new symbol `?`.

Question. Suppose we have a code with a minimum codeword distance d. How many bit errors and how many erasures can this code correct?

We'll start with errors. In order to be able to correct an error, the resulting sequence after applying the error on a codeword must remain closer to that codeword than any other codeword. Knowing that the distance between the original codeword and any other is at least d, the maximum amount of error bits we can have is $\lfloor \frac{d-1}{2} \rfloor$. Any more than this and the result will be closer to or as close to another codeword and we will not be able to infer the original codeword.

In the case of erasures, even if we have up to $d-1$ erasures we are still closer to the original codeword than any other. However, any more than that we will make us confuse the codeword with another.

As an example, in Hamming code where the minimum codeword distance is 3, we can correct up to $\lfloor \frac{3-1}{2} \rfloor = 1$ errors and $3 - 1 = 2$ erasures.

2.1 Singleton bound

In Hamming code, any two columns of the parity-check matrix H are linearly independent. If you multiply H by any vector of weight two, which essentially means just summing two columns of H, the result is never zero. However, for Hamming code three columns of the parity-check matrix may be lineary dependent.

In general in an error correcting code with a minimum distance of d, any $d-1$ columns of the parity-check matrix are linearly independent. And to have an error correcting code with minimum distance d, we need to construct a parity-check matrix such that any $d-1$ columns are linearly independent.

Since the size of each column of H is $n-k$, there can be at most $n-k$ column vectors in H which are lineary independant.

$$\begin{aligned} d-1 &\leq n-k \\ d &\leq n-k+1 \\ \frac{d}{n} &\leq 1-\frac{k}{n}+\frac{1}{n} \\ \frac{d}{n} &\leq 1-R+O(1) \end{aligned}$$

This is called the **singleton bound**.

2.2 Correcting Erasures

Lets consider an example. Suppose H is the parity-check matrix for the Hamming code and x is a bit stream we have received on the channel. As shown below, we observe that three bits of the codeword have been erased by the channel. Assuming bits x_2, x_3, x_5, x_6 have been received correctly, our objective is to deduce the value of bits x_1, x_4, x_7.

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\begin{array}{ccccccccc} x = & ? & 1 & 0 & ? & 0 & 0 & ? \\ & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{array}$$

If x is to be a valid codeword we know by definition that $Hx^T = 0$.

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = 0$$

We'll expand this multiplication to obtain the equations below.

$$\begin{cases} x_4 + x_5 + x_6 + x_7 = 0 \\ x_2 + x_3 + x_6 + x_7 = 0 \\ x_1 + x_3 + x_5 + x_7 = 0 \end{cases}$$

We already know the values for bits x_2, x_3, x_5, x_6, so the equations will be reduced to

$$\begin{cases} x_4 + x_7 = 0 \\ x_7 = 1 \\ x_1 + x_7 = 0 \end{cases}$$

We can solve this system of equations by writing it in the matrix format

$$\underbrace{\begin{bmatrix}0 & 1 & 1\\0 & 0 & 1\\1 & 0 & 1\end{bmatrix}}_{H_{unknown}} \quad \underbrace{\begin{bmatrix}x_1\\x4\\x7\end{bmatrix}}_{X_{unknown}} = \quad \underbrace{\begin{bmatrix}0\\1\\0\end{bmatrix}}_{H_{known} \times X_{known}}$$

In other words, multiplying the unknown columns of H by the unknown variables must be equal to multiplying the known columns of H by the known variables. Our decoding objective is to solve this set of linear equations. The linear equation is solvable as long as $H_{unknown}$ is invertable which is the case in this example.

$$\begin{bmatrix}x_1\\x4\\x7\end{bmatrix} = \begin{bmatrix}0 & 1 & 1\\0 & 0 & 1\\1 & 0 & 1\end{bmatrix}^{-1} \begin{bmatrix}0\\1\\0\end{bmatrix}$$

Note that we previously mentioned we can correct two erasures in Hamming code; however, in this example we were able to correct three. This is due to the fact that the three columns erased in this example are linearly independant. This would not always be the case when correcting more than two erasures in Hamming code. For example, if x_1, x_2, x_3 were erased, $H_{unknown}$ would have not been invertable and we would not have been able to decode the message.

We are now able to correct erasures in polynomial time. However, our decoding performance is bottlenecked at $O(n^3)$ by the matrix inversion step, which is unacceptable considering we take n to be very large. We need an easier and faster way for solving the linear equation. Consider our previous example, we can directly find the value of x_7 to be 1 since it is in an equation with only a single variable. We can then remove x_7 from the other equations and find x_1 and x_4 in the same way. Obvisouly this method cannot be applied to all linear equations in general, but we will formalize this procedure in the next section using Tanner Graphs.

3 Iterative Decoding using Tanner Graphs

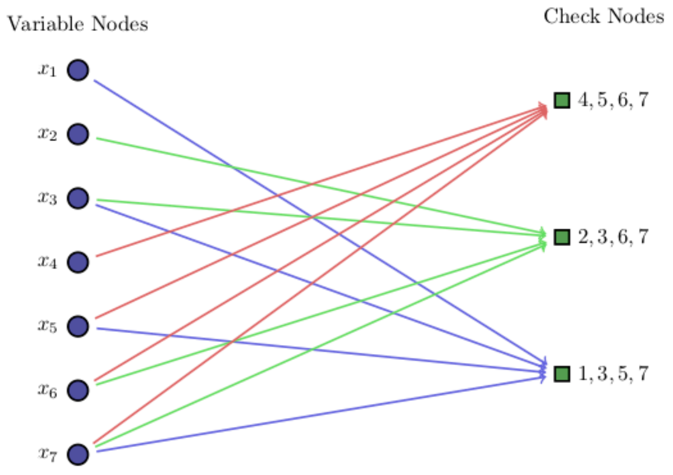
A *Tanner Graph* is a bipartite graph with n variable nodes, one for each variable, on one side of the graph and $n-k$ check nodes on the other. Each check node represents a linear constraint and there will be and edge between a check node and a variable node if the variable is contained within the equation represented by the check node. In other words, you could consider the H parity-check matrix the adjency matrix for the Tanner graph.

The Tanner graph of a code with the parity-check matrix below is shown in Figure 1. A different parity-check matrix will lead to a different Tanner graph representation.

$$H = \begin{bmatrix}0 & 0 & 0 & 1 & 1 & 1 & 1\\0 & 1 & 1 & 0 & 0 & 1 & 1\\1 & 0 & 1 & 0 & 1 & 0 & 1\end{bmatrix}$$

We will use Tanner graphs to solve linear equations such as that obtained in Section 2.2. The procedure is as follows. Define the *Erasure Set* $\mathcal{E}$ to be the subset of variable nodes which are erased and unknown. We look for a check node in the Tanner graph which has exactly one neighbor within the erasure set. That is, a check node which represents a linear constraint containing only one variable. Our iterative decoding procedure can procede as long as we find such a node. Once this check node is found,

Figure 1



we will sum all values of the known neighbors of the check node and place the result as the value of the unknown variable. This unknown variable is now known and is removed from the erasure set. We iteratively continue this procedure until all variable values are known.

Example.

Consider the example in Section 2.2. We will solve the linear equation using Tanner graphs, Figure 2 illustrates each step of the procedure.

The initial Erasure set is:

$$\mathcal{E} = \{x_1, x_4, x_7\}$$

Check node 2 is the only check node with a single neighbor in the erasure set.

Check Node 1 $\rightarrow \{x_4, x_7\}$	✗
Check Node 2 $\rightarrow \{x_7\}$	✓
Check Node 3 $\rightarrow \{x_1, x_7\}$	✗

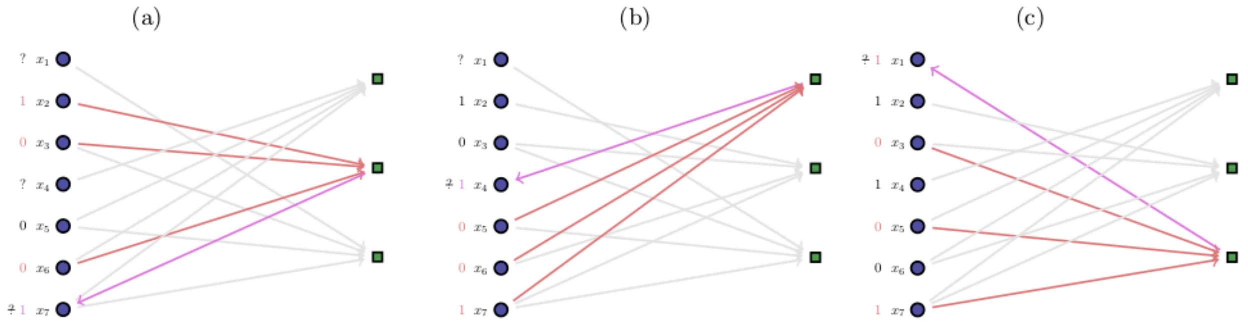
We will sum the values of check node 2's known variables, x_2, x_3, x_6, and place the result 1 as the value of x_7 (Figure 2a). The new erasure set now contains only x_4 and x_7 and both check nodes are neighbors with only one variable of the erasure set. We will apply the same process to these check nodes and obtain 1 as the value for both variables (Figure 2b and Figure 2c).

3.0.1 Stopping Sets and Sparse Graph Codes

To be successful in iteratively solving the linear equation using a Tanner graph, we must be able to find a check node with a single neighbor in the erasure set in every step of the process. We define a *stopping set* to be a subset of variable nodes such that every check node has at least two neighbors within the stopping set. More formally S is a stopping set if:

$$S \subseteq V_{\text{variable nodes}} \ : \ \forall v \in V_{\text{check nodes}}, \ |N(v) \cap S| \geq 2$$

Figure 2



If the erasure set contains a stopping set ($S \subseteq \mathcal{E}$) then the iterative algorithm above will fail. A stopping set will always exist in a large enough set if the graph is dense. However, if the graph is sparse, meaning that every linear equation in the system has only a few variables involved, it is unlikely that we have a stopping set.

Sparse Graph Codes also known as *Low-Density Parity-Check Matrix Codes* are codes where the number of ones in the parity-check matrix is small compared to the number of zeros (the number of ones does not linearly grow with n). The benefit of having Low-Density Parity-Check Matrix codes is being able to use iterative algorithms for decoding.

4 Belief Propagation Decoding

Suppose c is a binary codeword which has been sent over the channel. Also, suppose the channel is probabilistically introducing some errors. In order to decode the message we could find the best probability distribution given the observation y:

$$\arg\max_{c \in C} \ p(c|y)$$

This is called *Maximum A Posteriori* (*MAP*) decoding and it is the best possible decoding. However, doing MAP decoding directly is often infeasible and hard to do. We will look at two different approaches to this problem.

4.1 Maximum Likelihood Decoding

In maximum likelihood decoding, we use the bayes rule to convert the MAP maximization above into

$$\arg\max_{c \in C} \ p(y|c)\frac{p(c)}{p(y)}$$

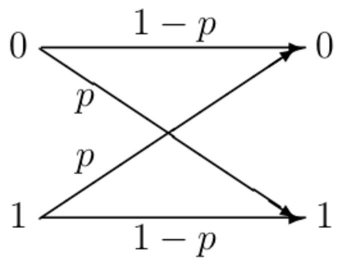
Note that $p(y)$ can be removed from the maximization since it does not depend on c. Assuming a uniform prior distribution for codewords, meaning all codewords are equally likely to occur, we can also remove $p(c)$:

$$\arg\max_{c \in C} \ p(y|c)$$

Given the memoryless property of the channel, this could be computed as:

$$\arg\max_{c \in C} \ \prod_{i=1}^{n} p(y_i|c_i)$$

Figure 3: Binary Symetric Channel



In the case of Binary Symetric Channels, $p(y_i|c_i)$ depends only on whether c_i is equal to y_i or not. The channel flips the value of a bit with probability p, so c_i and y_i would have different values with probability p. If the distance between y and c is denoted by $d(y,c)$, there are $d(y,c)$ bits which differ in total between y and c. So the maximum likelihood is reduced to:

$$\arg\max_{c \in C} \; p^{d(y,c)} \; (1-p)^{n-d(y,c)}$$

$$= \arg\max_{c \in C} \; \left(\frac{p}{1-p}\right)^{d(y,c)} \; (1-p)^n$$

$$= \arg\max_{c \in C} \; \left(\frac{p}{1-p}\right)^{d(y,c)}$$

If $p < \frac{1}{2}$ (which is a reasonable assumption since $p = \frac{1}{2}$ would give us a channel that provides no information) then $\frac{p}{p-1} < 1$ and the maximization above is equivalent to minimizing $d(y,c)$:

$$\arg\min_{c \in C} \; d(y,c)$$

In other words, like we have seen in minimum hamming distance decoding, we find the codeword which has the minimum hamming distance with the received bit stream y.

4.2 Belief Propagation Decoding

In belief propagation, instead of maximum likelihood we do a *bit MAP decoding* (bitwise MAP). In bit MAP decoding, we find the value for each bit i by computing:

$$\arg\max_{x \in \{0,1\}} \; p(c_i = x|y)$$

The i_{th} bit takes the value which has a higher probability of occuring as the i_{th} bit in codewords, given y. As you can see, in bit MAP decoding we are not trying to optimize over the codeword but rather optimize over each bit of the codeword. As a result, the bit stream we end up with in the end may not necessarily be codeword but it often does turn out to be.

To compute the value for each bit i, we must sum over the probabilities of all codewords which have an i_{th} bit of x. So the above maximization would be equivalent to:

$$\arg\max_{x \in \{0,1\}} \sum_{c \in C : c_i = x} p(c|y)$$