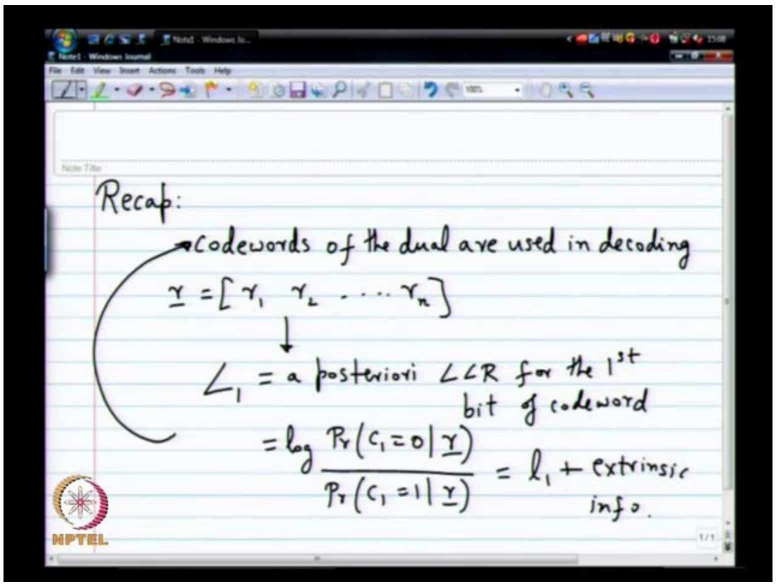
Coding Theory

Dr. Andrew Thangaraj
Department of Electronics and Communication Engineering
Indian Institute of Technology, Madras

Lecture - 23
LDPC Codes

(Refer Slide Time: 00:15)



Recap:

→ codewords of the dual are used in decoding

$\underline{r} = [r_1 \; r_2 \; \ldots \; r_n]$

$\downarrow$

L_1 = a posteriori LLR for the 1st bit of codeword

$$= \log \frac{\Pr(c_1 = 0 \mid \underline{r})}{\Pr(c_1 = 1 \mid \underline{r})} = l_1 + \text{extrinsic info.}$$

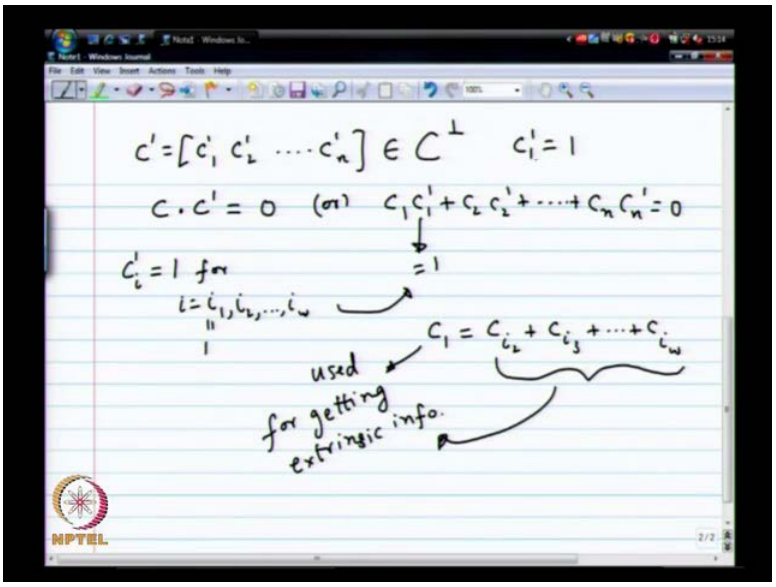
So, let us once again begin with a quick recap of what we have been talking about. So, in the area of soft decoders, I was talking about how you can use the code words of the dual cord to do decoding. So, that is main idea, that idea I think is central to understanding this many decoders that we will describe from now on. Code words of the dual are used to decode, so this is very important.

So, not necessarily the lineally independent code words in the parity check matrix, but any code word of the dual can give you information about say 1 particular bit of the code word that you are interested. So, usually if you think of a code, so if you think of a received vector r so you have r 1, r 2 through r n. And suppose you want to somehow from this vector r get let us say the a posteriori LLR for the first bit. So, you are interested in terms like capital l.

So, this is basically a posteriori probability for LLR, it is not just probability a posteriori LLR for the first bit of the code word. We saw a description how this can be expressed in terms of the individual scalar LLR. So, this is like the vector LLR, so this is basically log of probability that the first bit is zero, given the entire received vector r divided by probability that the first bit is 1 given the entire received vector r. So, you can write an expression involving the small l i for a capital l 1 and I describe how to do that accurately and that is a big nasty expression.

A useful way of simplifying that is to look at code words of the dual. So, each code word of the dual, which involves bit 1. What do I mean by saying involves bit 1 in the first position it should it be 1. So, if you have any code word of the dual, which has a 1 in the first position that can be used to get some extrinsic information about the first bit. So, the idea was to write this in terms of l 1 plus extrinsic. And then you split this as extrinsic information over every code word of the dual, which involves the first bit or which checks the first bit that is the idea.

(Refer Slide Time: 03:33)



$c' = [c'_1\ c'_2\ \cdots\ c'_n] \in C^{\perp}$ $\quad c'_1 = 1$

$c \cdot c' = 0$ (or) $\underset{\substack{\downarrow \\ =1}}{c_1 c'_1} + c_2 c'_2 + \cdots + c_n c'_n = 0$

$c'_i = 1$ for $i = \overset{1}{\overset{\shortparallel}{i_1}}, i_2, \ldots, i_w$

$c_1 = \underbrace{c_{i_2} + c_{i_3} + \cdots + c_{i_w}}$

used for getting extrinsic info.

So, if you have a code word of the dual let us say c prime, c prime 1, c prime 2, c prime n is the code word of the dual. Then we know that the transmitted code word c dot c pot was 0 or what is true c 1, c 1 prime plus c 2 c 2 prime plus so on till c n prime is zero. Now remember c n prime has to be equal to 1, so this k is equal to 1. So, I have c 1 plus then c 1 c prime will be 1 in certain positions, it will 0 in some other positions.

So, maybe you can say c prime i equals 1 for i equals i 1, i 2 so on. So, let say some i w, so w ones in c prime of course, i 1 is equal to 1 so the remaining ones are also 1. So, if you plug this into this equation, you would get c 1 equals what c i 2 x or c i 3 x or so on till c i w. So, every code word of the dual, which has c prime 1 equal to 1 gives you a parity check in equation involving c 1. So, this can be used for getting extrinsic information.

So, this is a bit of a crucial idea and any decoders, so you have the entire code word you have you have the entire code received vector r. Of course, I can split my a posteriori LLR as the intrinsic l 1 plus some extrinsic information and this extrinsic information is going to be a big complicated expression. If you want to evaluate it exactly, but approximate evaluation usually involve looking at code words of the dual, which have c 1 prime equal to 1, the first position should be 1 at the in the dual.

If that is the case then if you look at the condition that satisfied between the code and the dual is you know c dot c prime has to be 0 that gives you some equation involving c 1. So, if you use this to provide extrinsic information, it is going to be a simpler operation to deal with through the poster looking at the entire expression and trying to evaluate it, you can try and attack it this way.

Now, the idea is to look at multiple code words in the dual and see when you get independent extrinsic information as long as you keep getting independent extrinsic information, you can keep adding the extrinsic LLR's. The trouble is you cannot really keep track of everything there; eventually you will also get some dependent information how do you deal with that, that is the question. So, the various approximate decoders to various standard tricks are to simply assume independence even if it is not there. So, that this is the basic general principle behind approximate iterative decoders for at least l d p c codes.

So, in general also this is the philosophy, this is very useful to keep in mind. So, the only kind of remaining detail here if you are going to do this, you know how to get this extrinsic information. Given this equations, it is the two time hyperbolic inverse and hyperbolic products. It is very straight forward to get the extrinsic information from here. The only remaining information is from the dual, what are the code words from the dual that I should consider and in what sequence, where which one should I consider first, which one I should consider next etcetera.

So, that is essentially the problem that will be that is left here. So, this is where we kind of stop for the soft decoders, but remember I mean this can be actually done for any decoder need not necessarily be soft or anything. So, even if you had hard so even if you are received vector r is just bits over the binary symmetric channel, you are getting only bits. Maybe you cannot think of a log likelihood ratio, in fact you can think of that also, but maybe you do not want to think of a log likelihood ration.

But nevertheless you can use these equations for doing decoding. You mean for the binary symmetric channel because this is a valid equation for any channel. So, if you look at this bit, the bits on the right hand side they will give you some information about the first bit. So, the principle to remember is you want dual code words of low weight, why do we need low weight?

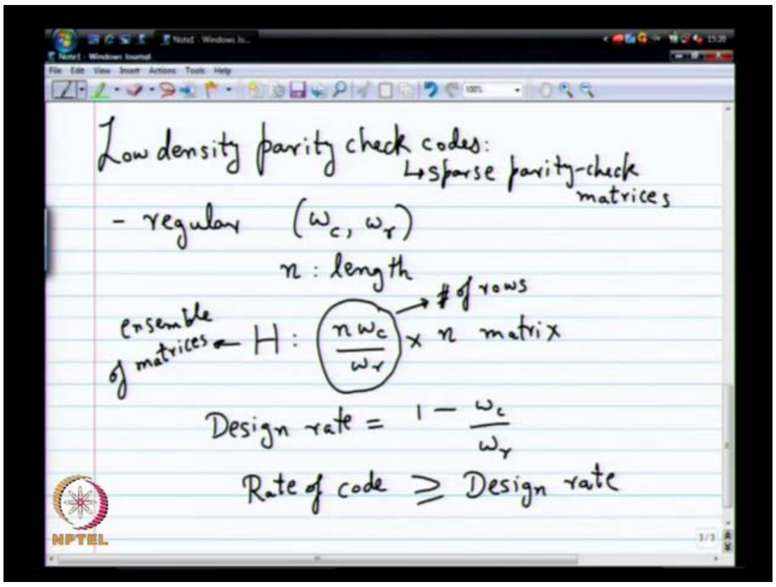
Student: ((Refer Time: 08:32))

So, then this extrinsic information is more meaningful. So, as you keep increasing the weight, extrinsic information will be 0, weight become very large extrinsic information is 0 and you do not get anything. You want it to be non 0 then if it has to be non 0, it is much more likely to be non 0 if you have lower weight that the main idea here. So, you want low weight dual code words and like I pointed out, you can use it for any decoding.

Even if you have a hard decision decoder you can use such things that is the other point may be, which I did not emphasis. So, height in the contacts of soft decoders, but if you have hard decoders, you can use this equation in the hard decode. Then the third crucial point is independence, so you want we do not want the dual code words to overlap in any other position other than at the first position.

So, this i ones are the i 2 to i w, if you look at another code word they should not be the same i 1 can be 1, but there should be no overlap. There is no overlap then you will have independent information and you can keep on adding without any problem. So, these are general principles which are quite useful to keep in mind. So, when I describe the decoder if you keep this principles in mind some of the steps will make sense otherwise, it will seem like it is just an ad of decoder that people have come up with. So, the next thing we say was definition of low density parity check codes.

(Refer Slide Time: 10:12)



Low density parity check codes:

↳ sparse parity-check matrices

- regular (w_c, w_r)

n : length

# of rows

ensemble of matrices ← $H : \boxed{\frac{n w_c}{w_r}} \times n$ matrix

Design rate $= 1 - \frac{w_c}{w_r}$

Rate of code $\geq$ Design rate

So, in particular I spoke about regular low density parity check codes. So, these are parameterized by 2 quantities w c, which is the column weight and w r is the constant row weight. So, these codes basically have sparse parity check matrixes and in particular 1 type of code is the regular code, where each column has w c ones and each row has w r ones. So, your block length is n then your parity check matrix will actually be an n times w c by w r times across n matrix by n matrix.

So, this will be the number the rows, this quantity here is a number of rows you can show that it is not too difficult. And there are many ways of constructing it one method is Gallegan construction, which works if w r itself divides n, there is a way of splitting the matrix into w r parts and picking each one in an obvious way and then doing arbitrary permutations. So, crucial thing to remember as this is not unique so it is in fact, if you say a regular LDPC code you are not talking about one code. So, you are about in fact and ensemble of codes, this actually an ensemble of matrixes.

So, we will see later that ensemble of matrixes shared some properties particularly under certain types of decoding this have some good properties. So, it makes sense think two of them together as one ensemble. So, from a design point of view important issue as use of what value should you take for w r and w c etcetera? One parameter there is the design rate just simply 1 minus w c y w r, it turns out the rate of the code is always greater than or equal to this design rate.

How did this come over? The number of rows is of course n w c by w r which means the rank is less than or equal to n w c by w r rank is n minus k. So, k is greater than or equal to n minus n w c by w r. So, rate is greater than equal to 1 minus w c by w s. So,

Student: ((Refer Time: 13:24))

So, the question was basically can the design rate be negative? From purely matrix point of view yes it can be negative, but from a code point of view maybe it does not make too much sense to have it as negative usually.

Student: ((Refer Time: 13:47))

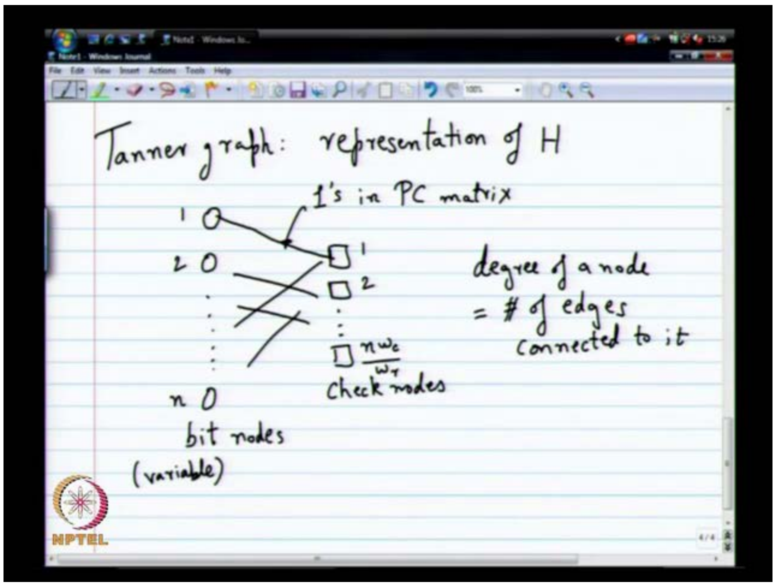
So, if your rate is usually positive right.

Student: ((Refer Time: 13:53))

Yes actually it is also possible so that is the other part of the issue. Even if your design rate is negative the rank can at most be n at least mean not at most be n, it cannot be greater than n. So, it might end up being invertible matrix so in which case code is just a trivial all 0 code, that is the only danger or maybe the rank is less than n, in which case the actual rate might still be positive. So, that is why this what design rate is a bit misleading, that is the best thing we can come up with. But what will happen if your w c and w r are small? Let say for instance 3 and 6, w c is 3, w r is 6; you construct a random matrix like that.

In the Gallegan construction, you will see that there will exactly 2 lineally dependent rows nothing more than 2 will be lineally dependent. So, you can see very easily in the construction, where we construct there will be at least 2 rows, which are lineally dependent on the others. So, if you think about it you can add up everything in the first section, you get the all ones in the next section also you get all ones etcetera. So, that is for w r equals w c equals 3 so you get 2, which are lineally dependent nothing else will be lineally dependent.

(Refer Slide Time: 15:26)



You can see that also mostly in a random construction, it is very difficult to get lineally dependent rows. So, the next crucial idea is this notion of a tanner graph, tanner graph is a graphical representation of parity check matrix. So, the idea is I have a bi-parted graph on the left nodes; the left nodes are called bit nodes. These are also sometimes called variable nodes, but we will use bit nodes almost exclusively. So, you can in fact number them if you like from 1 to n, there will be n of them.

And on the right side you have nodes that are called check nodes. If you number them from 1 on you will get for the regular codes, you will get n w c by w r of them and the edges you put basically are, they represent ones in the parity check matrix. So, if you go to the first row, the first row will contain ones in certain bit positions, you simply connect all those bit positions to think so that is the tanner graph.

Some things to keep in mind here each node is said to have a certain degree in this graphical representation. So, what is the degree? Degree of node is this is equal equals the number of edges incident on it or connected to it. So, what will be the degree of a bit node in a tanner graph it will be equal to w c. What will be the degree of the check node, it will be w r. So, that is something that you can see very easily. So, based on the tanner graph you can come up with something known as a socket construction for regular ensemble where, for each node you have as many sockets as its degree.

So, on the left side you would have w c sockets for each node, on the right side you would have w r sockets foe each node. In total you would have n times w c sockets on both the left side and the right side. If you add up everything and then you define a permutation from permutation of n w c numbers from 1 to n w c that tells you, a way of connecting the edges and it maintains the degree constraint. Every node will have every left node will have exactly degree w c, every right node will have exactly degree w r.

The only confusion there is 2 nodes, 1 bit node and 1 check node might have more than 1 edge, so you should have some way of resolving that. There are many ways of resolving it, for instance you could say if there are more than one edges I may drop one edge or something like that. So, you can come up with some something like that it does not really matter or another thing to do is to say I will only look at those permutations, which do not give me multiple edges between 2 nodes, you could say that.

If you say that then it will map properly to a parity check matrix otherwise it will it will map to multiple parity check matrixes or something, but it is not a big deal you do not have to worry about it. You can show that an overwhelmingly large number of permutations will not give you multiple edges or does not really matter in practice. So, that is about constructions Gallegan's construction and the socket construction. So, why the tanner graph? So, you might say parity check matrix itself has all these information, why do I need to go to the tanner graph?

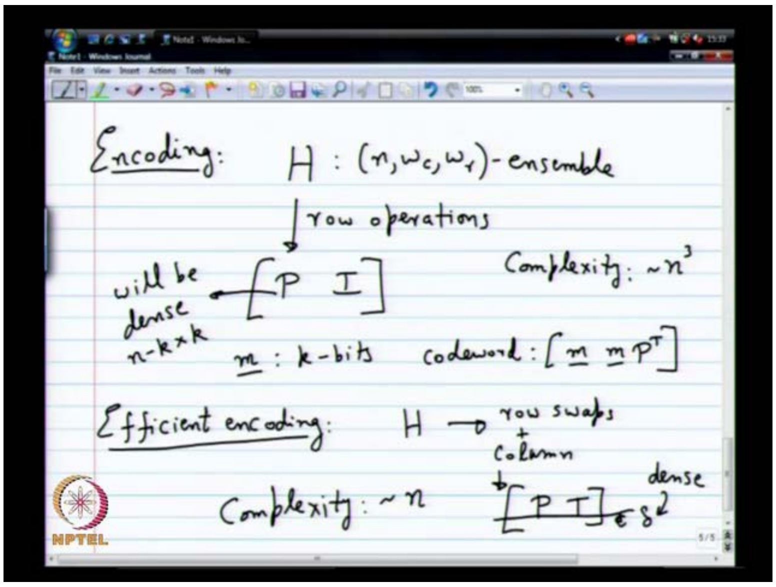
So, it turns the decoder that you will be considering for LDPC codes, which works really-really well and gives provides all capacity achieving performance etcetera is described usually on the tanner graph. So, you remember I was talking about sub optimal decoders just a little while ago I was saying, only thing that you have to specify is what code words of the dual are you looking at and in what sequence.

So, that is what is you are specified any way in a in a sub optimal decoder we are going to do. We are going to look at one code word of the dual at the time. So, which code was do you want you look at and then what sequence etcetera has to be specified, it tends out that tanner graph specifies helps in the specification very efficient way. So, that is the main utility of the tanner graph the decoders are described very nicely in the tanner graph.

Other than that it is it is just an entity of course, today there is a general area called inference in graphical models. So, it is a very popular area other people are doing on it. It has connections to statistical mechanics, statistical physics and information theory. So, many other connections of that of course, the graph itself have a lot of importance. Not saying it does not have importance, but for us at least in the first course we will be just concerned with how it not the parity check matrix and how it specifies the sequence set, which you are looking at code words of the dual in the decode.

So, that is the thing that will be interesting to us. So, we have seen the definition of low density parity check codes, we have seen how to construct them. So, at least technically we have seen how to construct them you can write a Mat lab program or a C program to do the construction given a w c and a w r. Then the next the most crucial step is a decoding, but before that lets quickly see encoding and get rid of it and then we will see what how decoding needs to work on.

(Refer Slide Time: 21:39)



Encoding: $H : (n, w_c, w_r)$-ensemble

$$\begin{tikzcd} H \arrow[d, "\text{row operations}"] \\ {[P \;\; I]} \end{tikzcd}$$

will be dense $n-k \times k$ $\leftarrow$ $[P \;\; I]$ Complexity: $\sim n^3$

$\underline{m}$: k-bits codeword: $[\underline{m} \;\; \underline{m}P^T]$

Efficient encoding: $H \rightarrow$ row swaps + column $\rightarrow$ $[P \;\; T] \in \delta^2$ dense

Complexity: $\sim n$

So, encoding is really not nothing much to say about encoders I will mention them very briefly. So, you have a parity check matrix from the let us say h is from the n w c w r ensemble. I should also point out that there are many more LDPC codes then just regular codes. So, there are all kinds of sparse matrixes that do not respect regularity and there is a lot of benefit and going to regular codes also.

Codes that are not regular, but for now we will see a regular codes for a while and then later on will move to irregular codes. So, suppose you have a parity check matrix from n w c w r ensemble, how will you go about designing encode of other code, what will you do? What is one yes one is to find the generate matrix, exactly that is that is the method. So, how will you find the generator matrix? So, you have to convert the parity check matrix to the standard form, the systematic form. So, you do row operations and convert the parity check matrix to systematic form, what form would that be let us say p i.

And after this you can use the matrix speed to do encoding. So, how will you do encoding, if you have a message m which is k bits, you are going to say the code word is m and then m p transpose so that is the encoding. So, this gives you a valid code word and you can do this. The only trick here is only problem should I speak here is how complex is this encoding. So, how many operations does it take as a function of n for instance.

You would like it to be a linear function n for every code word that you are putting out you would like to do the same amount of work. So, you do not want to do a constant amount of work, you do not want to do too much more work, but it turns out when you do row operations, this sparse character will be lost. So, was row operations involve adding a lot of rows you have just w r rows, which might be a very small, but w r ones in each row, which might be a very small number, but when you keep doing row operations, you are going to increase that number a lot.

So, this p in general will be a dense matrix, it will be dense, but it is not too scary because what is the size of P, size of P is simply n minus k cross k. So, at most it is going to be N Square or something so there will be n squared ones maximum number of ones that are in p can be about N Square. So, the complexity of this encoder can go as much as n cube exactly something like that. So, roughly n k in complexity we will say s of this and encoding so it turns out can do something smarter.

So, instead of doing row operations where your adding rows this is what costs increase in the weight. If you only do row swaps and column swaps then you are not increasing the weight of any row, you are not changing the sparse character of the matrix. So, there efficient algorithms to do only row swaps and column swaps and convert parity check matrix to some other form, which looks similar to this. So, there is efficient encoding methods first thing you can try is, there was a question about maybe you can change the design of h 2 for some constraints on it.

We will come to that soon enough, but before that one way to fix this n cube complexity is to say that I will do from h row swaps and column swaps. And then maybe I will get a matrix of the form some p and then some t where this t is some triangular matrix. It may not be fully triangular may be some small part of it some small delta part of it might be dense for some reason. To make it fully triangular you might have to live with a small dense part for the matrix, but except for that small dense part most of the other parts were done purely by row swaps and column swaps so the weight did not increase in some significant fashion.

So, in fact you can do this and you can show that the complexity here can be utmost linear so this would have complexity. So, when I say complexity I am talking about the number of operations so how many bit resource you have to do to finish the encoding. So, it is going to be of the order of n power 3 here. So, you can do this smartly there are papers that have been written, you can write programs based on those papers and do it very-very smartly and mark this delta really-really small.

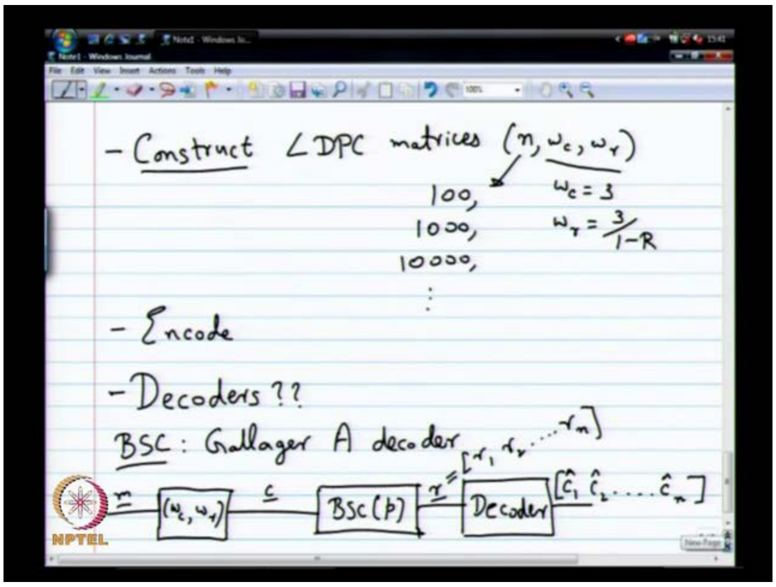
So, except for a very small tiny part of your matrix everything else is sparse and you have this p n upper triangular nature. So, it is very-very easy to do an inverse in this format and you can do encoding with this also. You can put your message here find the last bit then do a back calculation since only a small part is dense the back calculation will be done very quickly. So, that is the idea, you can do this and then the complexity here becomes roughly n.

So, almost linear that is one way to solve the problem at the encoding side, the other problem is what was suggested. The other suggestion was maybe you can design h in a very smart way to make encoding easier. So, that is actually what is done in many standards today, the standard except LDPC codes, which have a very simple encoder for the simple encoder based on the structure of the parity check matrix. So, it is a little bit more complicated maybe towards the end I will describe how that is done.

So, for now this is a better situation to keep in mind. The main idea is given a parity check matrix any way you can come up with an encoder, if you do not care too much about having a little bit more complexity, n cube this is not so scary, it is still polynomial you can do it very easily that is not a problem. But even if you are worried about linear complexity, there are methods very standard methods which I am not describing here, but they do exist which will help you simplify this decoder encoder because of the sparse nature.

So, you are doing some smart swaps and column swaps to get the results. So, this is all I want to say about encoding as you can imagine encoding is a simple problem. So, it is not really that complicated then we saw before never spent too much time discussing encoders because it is reasonably simple. We can do it often relatively much more complex problem is a decoder, which is what we have to spend more time on this. Any questions on this, everybody is particularly after I said it is not important, nobody is going to ask questions on something like this.

(Refer Slide Time: 29:37)



- Construct LDPC matrices (n, w_c, w_r)

100, 1000, 10000, $\vdots$

$w_c = 3$

$w_r = \frac{3}{1-R}$

- Encode

- Decoders ??

BSC : Gallager A decoder

$$n \rightarrow \boxed{(w_c, w_r)} \xrightarrow{c} \boxed{BSC(p)} \xrightarrow{r = [r_1\ r_2 \cdots r_n]} \boxed{\text{Decoder}} \rightarrow [\hat{c}_1\ \hat{c}_2 \ldots \hat{c}_n]$$

That is all I wanted to say about encoding so what can we do now? So, if you want to summarize we can construct LDPC matrices and what kind of LDPC matrixes n w r w c w r, let us say n w c w r we know how to construct. And usually as you can imagine, you would keep these 2 constraint, this 2 very small or when I say constraint its always independent of n that is what I mean when I say constraint.

So, these would be relatively small a very standard choice would be to put w c equals 3 and w r would be 3 by 1 minus rate, the rate that you want. So, this is a this is a very standard choice and then you would increase n, n would let say hundred to start off with, then if you see that is not good enough you might want to go to thousand, then you might want to go to ten thousand etcetera. So, this is how you construct and try out these codes. So, when I say construct you are not going to construct all possible matrixes out there, some random instance of this ensemble.

So, you are going to just pick the column weights randomly, the row weights randomly and hope that you get 1 random number that is ensemble, which is good. So, that is the idea so you have to write computer programs for this not just sitting down with a pen and paper and constructing this. It is no way which you can do this right 10000 with 3 n it not going to work not going to be able to do that, but still it will be efficient. As in many of these matrixes are sparse and many computer programs many I mean even Mat lab have lot of algorithm for spares matrixes for storing them.

You do not have to for instance if it is a 5000 by 10000 matrixes that is a lot of entries, but you do not have to store every single entry you know it is sparse. So, you only show store the locations and the values of the non zero entries so storages efficient and all that. So, you have to all that in your program only then it will work otherwise you need some 16 Giga 32 Giga machine of ram to make things to working may not be efficient.

So, it is easy to write programs given that you are careful with memory in some high level ray, if you are careful it is ok. The next thing is we can we can encode there is no problem using some standard ideas. So, of course, the most crucial part is decoders so what is it about these low density parity check matrixes is that, there is some there should be something. So, popular some crucial property that they have, that that gives you a very efficient decoders is the main study here.

So, what I am going to do is basically I am going to describe decoders for the binary symmetric channel first. There are various approaches you can take to this describing the decoders, we can do it in so many different ways, but what I found to work very well based on past years to first describe the other binary symmetric channel. It will be a sub optimal decoder; it will use the principle of looking code words of the dual through the tanner graph and then getting some information about every bit.

So, it will be bitwise, it will be suboptimal and it will use the tanner graph. So, those are the generic properties and the first decoder I am going to describe was actually given by Gallagher himself in the first in his thesis it is called Gallagher A decoder for the BSC.

When I say Gallagher A it is automatically assumed that it is for the BSC. So, what kind of pictures are we looking at now binary symmetric channel and the Gallagher A decoder.

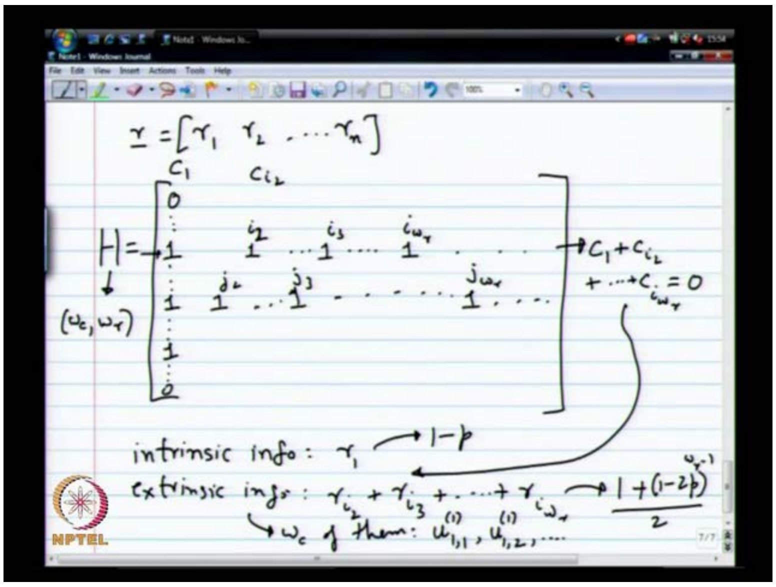
So, you have a message m which gets encoded by a let us say w c w r LDPC code. Regular LDPC code you get a code word c it goes through a binary symmetric channel, which let us say a transition probability p then you are going to run this decoder on the receive vector r, which will now be bits. So, all of these things will be bits will be running your decoder on this. And this will produce c 1 cap, c 2 cap so on. Like I said all these things will be bitwise and it will work like this.

So, this decoder is clearly what is called hard decision decoder, it is not a soft decision decoder, will see soft decision decoder soon enough after we finish this, after we analyze this and understand it will move towards soft decision decoders as well. There are so many a general principle behind this decoder it will takes a lot of time to describe it also. We have to go through it very slowly.

So, this is a setup so when I say a receive vector r now I am going to say r 1, r 2 through r n, but each r a is a bit 0 or 1 obtained when a code word c was transmitted through a binary symmetric channel with transition probability. So, what is interesting here is when Gallagher wrote his thesis; he did not mentioned tanner graphs. So, his description of the decoder was always through the parity check matrix.

So, what I am going to do next is to first describe the decoder through the parity check matrix, then look at the corresponding tanner graph interpretation. And you will see that the tanner graph interpretation is very natural and nice and can be extended very nicely, it gives you a feel for what is happening. So, even the parity check matrix is probably good, but the tanner graph makes it much better to understand what happens?

(Refer Slide Time: 36:10)



$\underline{r} = [r_1 \ r_2 \ \dots \ r_n]$

$H = $ with columns c_1, c_{i_2}; row $1 \ \dots \ \overset{i_2}{1} \dots \overset{i_3}{1} \dots \overset{i_{w_r}}{1} \dots \rightarrow c_1 + c_{i_2} + \dots + c_{i_{w_r}} = 0$; row $1 \ \overset{j_2}{1} \dots \overset{j_3}{1} \dots \overset{j_{w_r}}{1} \dots$; first column $0, \dots, 1, \dots, 1, \dots, 1, \dots, 0$

$H \downarrow (w_c, w_r)$

intrinsic info : $r_1 \rightarrow 1-p$

extrinsic info : $r_{i_2} + r_{i_3} + \dots + r_{i_{w_r}} \rightarrow \frac{1 + (1-2p)^{w_r - 1}}{2}$

$\hookrightarrow w_c$ of them: $u^{(1)}_{1,1}, u^{(1)}_{1,2}, \dots$

So, what have given now we are given a receive vector r, r 1, r 2, r n and then we know that the code word that was transmitted satisfied several parity check equations given by the parity check matrix h. What else do we know this comes from a w c w r ensemble, so if I want to decode the first bit c 1, remember when I do bitwise decoding there is intrinsic and extrinsic information?

What is the intrinsic information that I have about c 1 its r 1, r 1 is a bit itself c 1 is equal to r 1 with probability 1 minus p and it is not r 1 with probability p and p is less than half. So, with higher probability it is going to be equal to r 1. So, the intrinsic information is simply r 1 so the bit r 1 itself is the intrinsic information. And then now I have to worry about what kind of extrinsic information I can get about c 1 from the code words of the dual. And what do I need, I need code words from the dual which have a one at the first position just by staring at the parity check matrix.

How many code words can I come up with immediately let us say in the first round, first round what are the parity checks that you can come up with immediately, w c of them you can come up with. What is the w c of them wherever you have ones at the first column? So, I know on the first column there will be a bunch of zeros and then there will be a 1, then there will be another 1, then there will be another 1. In my illustrations I will assume w c is 3 for us just to keep it simple.

So, you have three ones here that will be in general w c ones and those w c parity checks those corresponding rows will definitely give you extrinsic information about c 1. So, let us say the first one that you take here has a 1 in position let us say i 1, i 2 then it has a 1 in position i 3 so on till 1 n position i. It should be actually w r minus w r then they lefts 0 l for it, there are w r ones like that.

And what information does this give me, this thing is telling me c 1 x or c i 2 x or so on till c i w r was actually equal to 0. So, from this equation what extrinsic information can I get about c 1 from r i to r i w r is the question. So, remember these are all hard decisions now, when it was soft decision I had to worry about probability and all that. When it is hard decisions I cannot do anything more, the only thing I can say is extrinsic information from this equation for instance is simply there is another estimate for c 1, which is simply r i 2 x or r i 3 x or r i w r.

The intrinsic information is giving me one estimate for c 1, which is r 1 itself. The first extrinsic information I am looking at was parity check am looking at gives me some one more estimate for the same bit c 1. It is telling me the c 1 can also be equal to r i 2 plus r i 3 plus so on till r i w r. So, remember c 1 was equal to r 1 with probability 1 minus p, what the corresponding probability here is, with what probability will c 1 is equal to this k.

Student: We will have to take all combinations ((Refer Time: 40:48))

I am now xoring w r minus 1 other bits which could which each of them could be in error with probability p independently. If an even number of them became an error, then my estimate is going to be ok, but if an odd number of them are in error, then my estimate is wrong. So, the only probability of the look at this out of w r minus 1 bit, what is the probability that an odd number of them will become will be in error given that each one of them is independently in error with probability p.

So, that is a calculation you can do with binomial expression and so on. You will see that the probability will work out something 1 minus 1 minus 2 p power w r minus 1 by 2. So, this is a probability of correctness of this. This is simply 1 minus p, is it 1 I think it is plus 1, this should be plus no. So, you want an even number of errors so the odds should go away. So, I should add its correct p 0 should be 1 looking at probability of correct decision. So, this is the probability which with this estimate is correct. Now, I am going to look at the next row that I have here.

Remember I have w c rows now what will happen in the next row so this next row will now have let say 1 in the j 2, then some other 1 here j 3 so on till the last 1 maybe is in j w r. So, I have deliberately put this one's even in this picture in a certain way what is this that I have done. I am picking them to be different from i 2, i 3 i omega r so that is the crucial idea here. Now if the second equation had ones in positions which did not overlap with the ones of the previous ones, previous check that I used.

Then that equation that I get will give me an estimate which is statistically independent from both the previous estimates they had. r 1 is clearly statistically independent of all this guys then the next estimate law should be statistically independent. So, likewise I will get how many estimates in the first round immediately I will get w c estimates. So, that is the hats he first steps in the decoding. So, you could call it you could call it the row iteration.

This is the first half iteration looks at rows of the parity check matrix and generates estimates for the bit. Then now the next half of the iteration will try to process all these estimates and come up with some better final estimate for the bit that is the split alright. So, this step is clear right the row step is clear you look at each row, then try and come up with an estimate for the particular bit that you are interested in c 1 yes. So, that is the next question.

So, of course, now so the question was should you construct h so that this condition is satisfied yes absolutely? Definitely it would help right. So, second row should not overlap the first row here in any position. So, that is one of the crucial constructions crucial conditions imposed on the construction. So, the three rows that you have corresponding to the first position or in general the w c rows you have in the first position will definitely overlap in the first position cannot do anything about it, but they should not overlap anywhere else.

So, that the estimates you are getting end up being independent and that is nice. So, as we go along as we do the decoding you will see there are some nice conditions like this that will come. We have to enforce them in the construction. So, far we just said construction is every matrix will pick, but maybe you should not pick every matrix. You should enforce some smart condition like this to get better decoders. So, the row step is clear right.

So, let me describe the steps kind of informally and then finally, I will write it down, when I write it down I will write the complete expression, but you will see that that is much easier to write in the tanner graph then in the parity check matrix. Parity check matrix is little bit more confusing to write. So, row step is clear. Now, the next thing to remember is what I will do for c 2.

Just look at the corresponding column for c 2 and repeat the same steps for c 3 again do the same thing, c 4 again do the same thing. So, in fact you can do this step for all the bits at the same time. If you like if you have enough memory you can do this operation in parallel. In fact you can also save a lot of these operations. So, if you look at c I 2 you know, that also involves the computation of the same parity check.

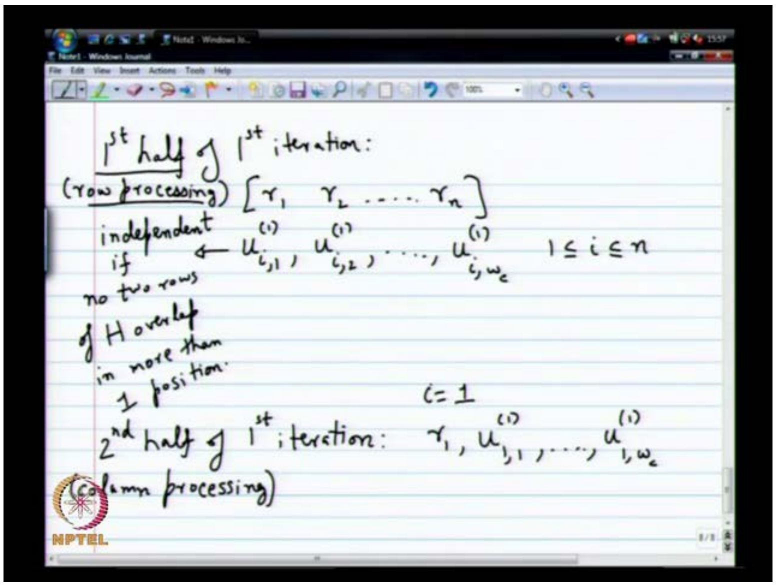
So, you can do that computation first and then update all the bits that it was involved in for this. You can do it very smartly if you like or you can do it sequentially bit after bit evaluate all the checks and figure out what happens. So, there are various ways if implementing it that is the detail, but you have to do it for all the bits alright. So, I have been saying that this is like the first half of the first iteration. Then you have the second half of the second iteration and then there will be multiple iterations.

So, you will see what happens there later is it. So, the end of the first half of the first iteration you have w c estimates or how many estimates exactly w c plus 1 estimate for each bit is it. We have, w c plus 1 estimate what is the plus 1 the intrinsic estimate. So, you have w c extrinsic estimates w c extrinsic estimates and you have 1 intrinsic estimate for each bit at the end of the first iteration. So, the extrinsic 1 is very easy. So, each of these estimates you might want to call them something.

So, for instance you might want to call them. So, we will come to that later if you like, but maybe it is useful. So, let me say each estimate w c of them these a things are maybe we have denote them as u what is a good notation man. So, I want to put 1 here to indicate iteration and then 1 here to indicate the bit position and this is the extrinsic information. So, this is not u 2 still u 1. So, this is how I will denote the estimates that I am getting, the intrinsic estimates is simply r 1.

The first extrinsic estimate I am getting about the first bit in the first iteration is u super script is for the iteration number. In the subscript there are two of them. The first subscribe denotes the bit position itself the second subscribe denotes the number of the extrinsic information number of the row. First one will give you the first ones the second one will ((Refer Time: 48:42)).

(Refer Slide Time: 48:49)



1st half of 1st iteration:

(row processing) $[r_1 \quad r_2 \quad \cdots \quad r_n]$

independent if no two rows of H overlap in more than 1 position. $\leftarrow u^{(1)}_{i,1}, u^{(1)}_{i,2}, \ldots, u^{(1)}_{i,w_c} \quad 1 \leq i \leq n$

$i=1$

2nd half of 1st iteration: $r_1, u^{(1)}_{1,1}, \ldots, u^{(1)}_{1,w_c}$

(column processing)

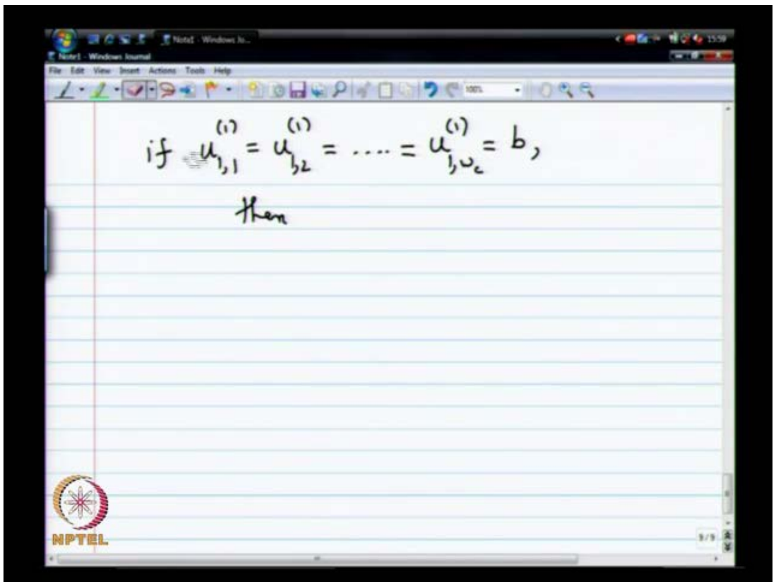
So, at the end of the first half of first iteration of course, you have the intrinsic estimates r 1 r 2 r n. Then you also have all these extrinsic estimates right u 1 i 1 u 1 i 2 all the way to u 1 i w c for I from 1 to n. If you took care of the constraint, what is the constraint the rows of the parity check matrix do not overlap in more than one positions. So, that is the generic constraint, if the rows of the parity check matrix do not overlap in more than 1 position, then all these estimates will be in fact independent for each weight.

So, these things are independent if no 2 rows of h overlap in more than one position. When I say overlap it basically overlaps in ones of course, they will overlap in 0 so we do not care they do not more than one position. So, if you take care of this constraint then this will be independent. So, in the first half of the first is always called row processing. It is clearly the rows of the parity check matrix are involved so it is called row processing. The second half is called the column processing. So, let us see what you can do in the second half, so which is also called column processing. So, what can we do?

So, let us look at the first position, we will fix i equals 1 we of course, have r 1. Then you have u 1 1 so on till u 1 1 w c. So, let us just empirically at this situation which of this estimates is most reliable r 1. So, you cannot make anything better than r 1, so r 1 is definitely a best estimate any xoring you do will definitely be poorer than 1 minus p in success probability. You can you can show that if you want I mean its intuitively kind of clear, more than 1 bit it is going to be bad.

It depends on the weight of course, if w r is equal to 2, then I guess you will get 1 more equal reliability, but it can never be greater than r 1 usually w r is not going to be 2 w c is going to be 3, w r will be some 6 or something. So, it is going to be larger and it will always be poorer. So, you have to in your column processing rely a little bit more on r 1 than you rely on the other things. So, you can now come up several ad hoc methods, the method thus described as Gallagher is the following. So, what I am going to describe next is the Gallagher A processing for columns second half of the first iteration.

(Refer Slide Time: 52:57)

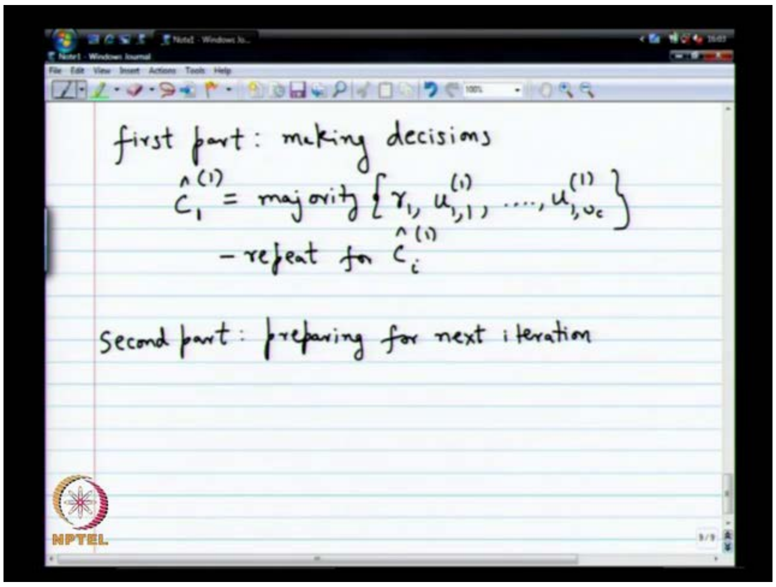


if $u^{(1)}_{1,1} = u^{(1)}_{1,2} = \dots = u^{(1)}_{1,w_c} = b,$

then

So, you will do is if u 1 1 1 equals u 1 1 2 and they are all equal, all these estimates are let say equal to some b. So, let me just make sure I get this absolutely right because it can be it is a bit dicey to explain it, so I should do it very carefully. So, let me just maybe step back a little bit before I do this and say let me step back before, so let me not describe this first. So, there are 2 things that you have to do first. So, in the column processing there are 2 parts.

(Refer Slide Time: 54:12)



first part : making decisions

$\hat{c}_1^{(1)} = \text{majority}\{r_1, u_{1,1}^{(1)}, \ldots, u_{1,w_c}^{(1)}\}$

– repeat for $\hat{c}_i^{(1)}$

Second part : preparing for next iteration

So, there are 2 parts to column processing, the first part is the easy part. First part is making decision, which is very easy so this part is easy. And then the second part is preparing for the next iteration, so this part is a little bit trickier. So, what is started to describe was actually the second part, but before that I should tell you what the first part is only then will be happy.

So, let us say c had 1 1 is some kind of decision we make on the first bit. So, this can simply be majority, this is one way of doing it, you can say this is simply majority of r 1 u 1 1 etcetera you could do this. So, this is not too difficult or if you want you can give some more emphasis to r 1 you can say if r 1, I would give more weight age to r 1 etcetera. So, you can do this in multiple ways it turns out this is not very crucial so the making decisions is very easy.

So, you are a paid for c i 1 so that is the first part you simply make a decision, but if you want stop at the first iteration believe me your decoding will not be very successful. So, you would not you are not going to get much more. The crucial thing is to doing is the second part by which you do multiple iterations. So, for doing the multiple iterations with the matrix, I will try and describe it only a first shot at the description it is going to be confusing to you, but then later on I will do the same description with a tanner graph.

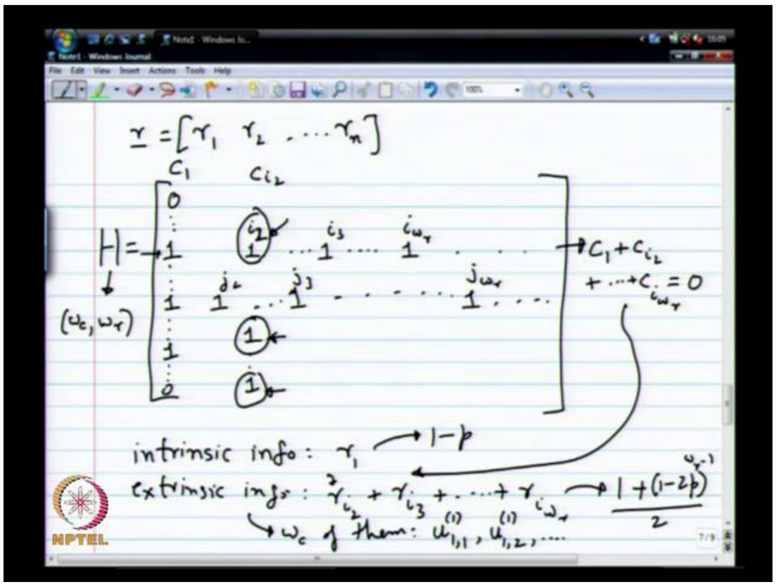
And you will see in the tanner graph that the description becomes very smooth and simple. So, even if you do not understand it stay with me for a while, I will quickly go through the description here in the matrix point of view because I think you should say it once. So, that you get a feel for what is what is really happening. So, in the second part when you prepare for the next iteration, remember what has to happen in the first half of the next iteration.

You have to do row processing, which means each row should now have some estimate of what the connected bit was. In the each row, how do you do the processing? You simply take the Rs and you add it up, but the R itself was used in the first iteration. In the second iteration you have to do something slightly better, you cannot repeat the same r. If you repeat the same r, you will simply be doing the same thing again and again you would not get anything new.

So, what you have to do is, in this instead of these r s you have to use better estimates for c i 2. In the first iteration I would have got some estimate for c i 2, which used r i 2 and some other information. So, that estimate I have to use here that is the main idea. So, I have to use some better estimate for c i 2 then simply r i 2. Initially I used it because that is what I got from the channel I did not have anything else so I used it initially. In the first half of the next iteration I should use a better estimate for i 2 c i 2 in this in each of these equations?

So, the duty of the second part of the second half of the first iteration is it to simply come up with what estimates to use for each bit in the next step. So, turns out you should not use the same estimate for every single row. So, remember if you go back and look at this initially in the first iteration in the row processing, you simply used r 1 for every row that is even was not involved in.

(Refer Slide Time: 57:12)


r = [r1 r2 .... rn]
H =
(wc, wr)
→ c1 + ci2 + ... + ciwr = 0
intrinsic info : r1 → 1−p
extrinsic info : ri2 + ri3 + ... + riwr → (1 + (1−2p)^(wr−1))/2
wc of them: u1,1(1), u1,2(1), ...

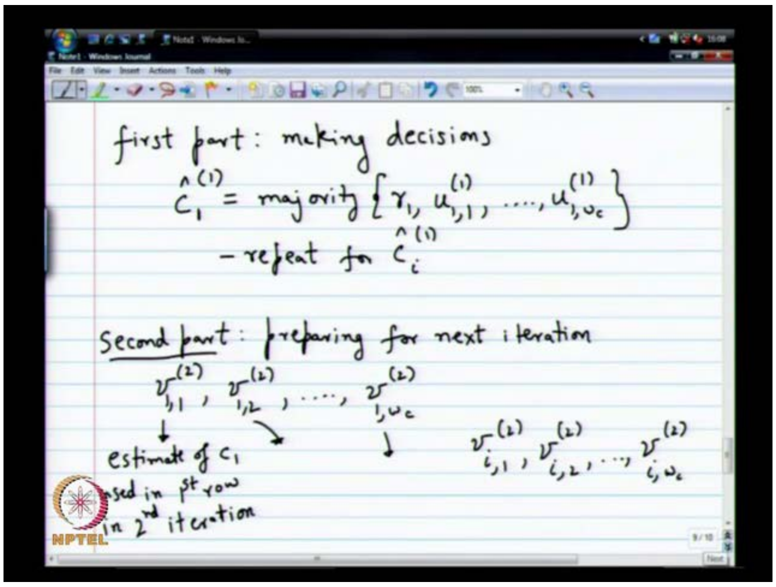
So, let me go to i 2 for instance, maybe i 2 was involved in three rows. You used the same r I 2 for each of these 3 rows as an estimate. It turns out in the next iteration you have to use a different estimate here, and a different estimate here, and a different estimate here. So, it is not a good idea to use the same estimate of c i 2 in all the three rows. In the first row you have to use an estimate a different of c i 2, the second row you have to use another estimate, third row you have to use yet another estimate in the first iteration.

If i 2 was involved in three rows, you use the same estimate r i 2 for each of those rows. In the next iteration you have to use different estimates, the reason is I will tell you why you have to do that. Suppose you want to figure out what better estimate to use for c i 2 in this particular row, you have to use r i 2 then you have to use the estimates that you got from this second row. And this third row only you cannot use the estimate you got from this particular row.

Once again the reason is that will directly kill your independence assumptions. So, you want to keep successive estimates as independent as possible. See suppose you use the estimate you got from this once again in a loop inside, you will be using the same thing over and over again and you are passing it inside the same parity check. So, you will have a lot of repetitions and it is not a good idea. So, this part is not very clear from the matrix point of view, but if you look at it from the tanner graph point of view, it will be much clearer.

So, it is not a good idea to use the same estimate for each row, you have to use different estimates for each row. So, it turns out beginning with the next iteration each bit position will have 1 intrinsic information and it will have w c updated information, that is used in the row processing. It will have w c of them not just 1 r i 2, you will use 1 estimate for the first row, another estimate for the second row so no till the w c t h row you will use another estimate.

(Refer Slide Time: 01:00:51)



first part: making decisions

$$\hat{c}_1^{(1)} = \text{majority}\{r_1, u_{1,1}^{(1)}, \ldots, u_{1,w_c}^{(1)}\}$$

$-$ repeat for $\hat{c}_i^{(1)}$

<u>Second part</u>: preparing for next iteration

$$v_{1,1}^{(2)}, v_{1,2}^{(2)}, \ldots, v_{1,w_c}^{(2)}$$

$\downarrow$

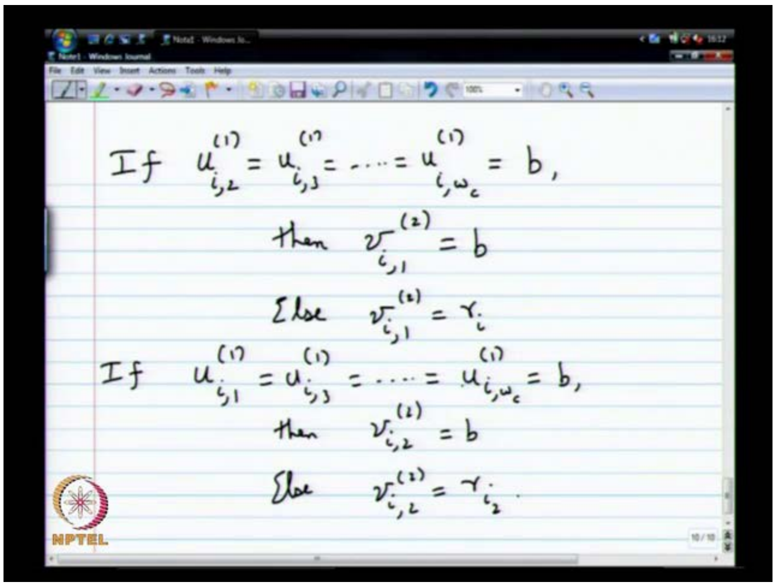
estimate of c_1 used in 1st row in 2nd iteration

$$v_{i,1}^{(2)}, v_{i,2}^{(2)}, \ldots, v_{i,w_c}^{(2)}$$

So, those estimates we will denote as v what v 2 1 1, v 2 1 2 so on till v 2 1 w c. What is this guy, estimate of c 1 used in first row in second iteration. What is this one similarly? Estimate of c 1 used in the second row in which it appears during the second iteration so likewise you have to keep repeating. So, in general you will also have what v i 1 2, v i 2 so on till 2 i w c. So, you will have w c different estimates for the each bit.

So, the question in the second part of the second half of the first iteration is to figure out all these guys, you have to figure out what estimate to use in each row in the next iteration. So, finally, when you mean when I put everything together maybe it will be a little bit clearer. So, let us just do this describe it when I will come and say how it is used. So, how do you decide this is crucial, so the way you decide v i 1 so I will just describe how you describe v i 1.

(Refer Slide Time: 01:02:54)



If $u_{i,2}^{(1)} = u_{i,3}^{(1)} = \dots = u_{i,w_c}^{(1)} = b$,

then $v_{i,1}^{(2)} = b$

Else $v_{i,1}^{(2)} = r_i$

If $u_{i,1}^{(1)} = u_{i,3}^{(1)} = \dots = u_{i,w_c}^{(1)} = b$,

then $v_{i,2}^{(2)} = b$

Else $v_{i,2}^{(2)} = r_{i_2}$.

So I will just describe how you describe v i 1 if u 1 i 2 equals u 1 i 3 equals so on till u 1 i w c equals some b then you will set v i 1 2 to be equal to that b else you will simply set v I 1 2 to be r i itself. So, remember i t h bit got an intrinsic estimate, which was r i itself and then it got w c extrinsic estimates, it got the first estimate from the first row, second estimate from the second row and so on. Now to update the estimate for the first row what are we doing?

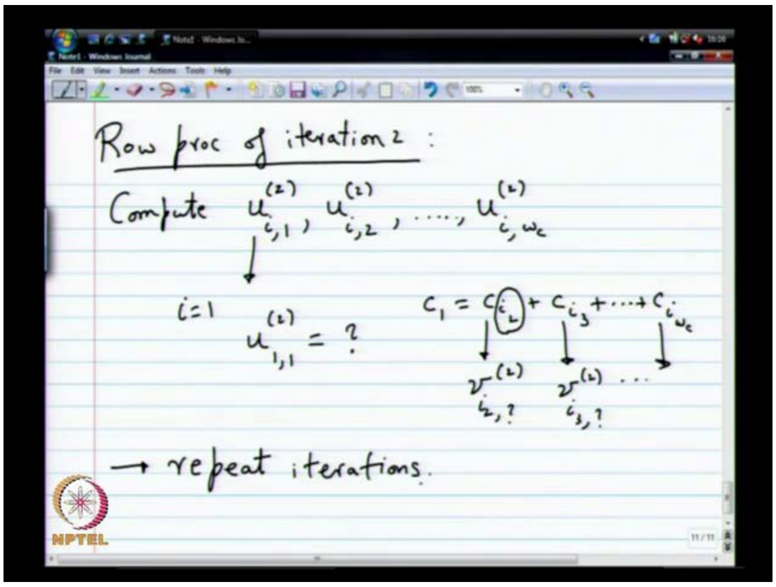
We are looking the estimates from second row, third row onwards. We do not look at the estimate that you got from the first row itself, we look at the estimate we got from the second row third row. If all those estimates agree and they are pointing in direction which is b then you simply say the estimate I will use for my first row was that b. In case there is any disagreement what do you do, you simply go back and rely on the intrinsic information for that that particular value.

So, that is the idea here so if you want you can go back and look at this picture it will be a little bit clearer here. So, remember if you now look at i 2 t h position, you will get one intrinsic information r i 2 and then you will get one extrinsic information from this row, another from this row, another from this row. In the updated version for this row what am I going to look at? I am going to look at these two extrinsic information if they agree what will I do, I will set that test the updated value for i 2 in this row.

If they disagree then I will say I will simply reuse r i 2 that is the idea because r i 2 anyway was a good estimate. So, that is Gallagher A decoding algorithm, which is what Gallagher A decoding algorithm does its not best storage, not the greatest or anything. So, it just an ad hoc method that is what its works a lot. There are some principles here, which are very good. It works very well that is what I mean. So, hopefully this is clear so what will I do for v i 2, if u i 1 equals u i 3 and so on till u i w c equals a particular bit then I will set v i 2 2 to be equal to p else this will again be r i 2.

If you want we can go back once again to this picture, what is actually happening here is? For the i 2 t h bit if I want to update this guy, I will see the estimate that I got from the first row and the estimate that I got from the third row. If they agree I will set the new value for this to be has agreed by value else I will simply reuse r i 2. That is it, I mean that is all we are done I have described the entire decoder, what will you do in the next iteration? May be I should just describe the row processing for the second iteration or let put r i 2 yes, it is just r m.

(Refer Slide Time: 01:07:27)



Row proc of iteration 2 :

Compute $u^{(2)}_{i,1}, u^{(2)}_{i,2}, \dots, u^{(2)}_{i,w_c}$

$i=1$ $\quad u^{(2)}_{1,1} = ?$

$c_1 = \boxed{c_{i_2}} + c_{i_3} + \dots + c_{i_{w_c}}$

$v^{(2)}_{i_2,?} \quad v^{(2)}_{i_3,?} \quad \dots$

→ repeat iterations.

So, maybe I should describe row processing of iteration 2, what will happen in the row processing for iteration 2? If you want to look at i t h bit, you will compute u, u 2 i 1 u 2 i 2 so on till u 2 i w c, how will you compute this guy. So, it is a little bit tricky. So, you will use new estimates for each bit corresponding to that particular row. So, for instance this one maybe so originally it might have been connected or something so you will have to connect.

Use the new estimates so let me set i equal to 1 for instance if I did that u 1 1 2 how will this be computed. So, remember what was my equation c 1 equals c i 2 plus c i 3 plus so on till c i w c. Now, for i 2 t h bit i i originally I used r i 2. Now I will be using v i 2, I do not know the number exactly, but it will be using something. What is that number? Why did I put a question mark there? So, I have to figure out in which row connected to i 2 1 will occur so that a bit more tricky, you get top write in the matrix properly only then you will understand.

So, just the numbering may not be the same, but some v 2 i 2 comma something has to be used their, maybe it is 1, maybe it is 2, may be its 3. I do not know depends on how the rows take up, so likewise here I will use v i 3. I will put a question mark some guy here. Similarly, here also the main point here is the first row connected to the first bit will have will this connected to i 2, i 3 so on. Now, this i two t h bit does not have 1 estimate r i 2, it has several estimates for each row i will the corresponding estimates suitably and compute my new estimates.

So, I have a way of going from v to u then what will I do, I can also go from u to v then I can keep on repeating this iterations. So, of course, the question is when to stop. So, of course, I mean all these are in fact several people published papers saying when to stop when all this was active ten years ago. So, you can do that now, you know that now nobody will publish it. So, one condition is you have estimates for all the bits, but nothing is forcing them to be a code word.

So, one simple thing to do is compute h times estimated word transpose that being 0, you simply stop, you got a code word, you stop that is one idea, but that may not happen, you may end up in a situation where you never get a code work. So, what happens in practices there is always a limitation based on your complexities, you cannot do more than let us say ten iterations. So, you do for a while and then just stop you know just desired I will output whatever I get after ten iterations for instance.

So, that is the way to do it. So, we are really getting close to the end of this lecture. So, I will stop now and then we will see the similar description on the tanner graph. So, when you will see then picture becomes much more clear and easy etcetera. And the other thing to keep in mind is I was talking about using one code word of the dual using multiple code words from the dual, but it looks like in the way I describe it you are using only those code words of the dual that are in the parity check matrix itself.

So, stare at it and tell me if that correct or wrong, that is actually wrong using all kinds of other code words also, but does it clear to you that I am using more code words than those in the dual itself. Why is that exactly because see I think about it so I am using several XORs and those XORs is happening kind of in the way and avoiding the same row. You know that that is the kind of idea to keep in mind. So, I will come in explain that also, it will be very-very clear in the tanner graph.

In the matrix it a little bit more confusing, but you can still see it you can see how more code words of the dual are being used then that are there in the matrix itself and this multiple iteration is keen doing that. So, that is what helps you do that. So, I will stop here for now. Take a break and pick up from here.