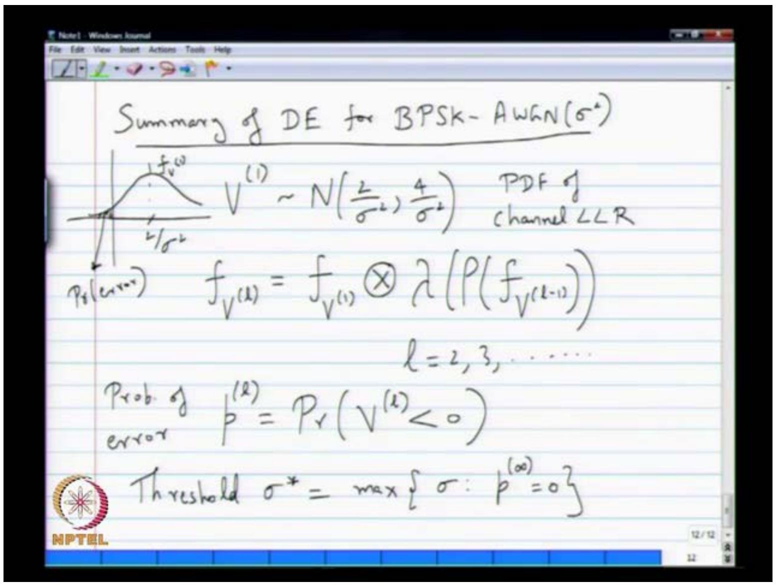
Coding Theory
Prof. Dr. Andrew Thangaraj
Department of Electronics and Communication Engineering
Indian Institute of Technology, Madras

Lecture - 29
LDPC Codes in Practice

(Refer Slide Time: 00:26)



Summary of DE for BPSK-AWGN(σ^2)

$f_{V^{(1)}}$
$2/\sigma^2$
Pr(error)

$V^{(1)} \sim N\left(\frac{2}{\sigma^2}, \frac{4}{\sigma^2}\right)$ PDF of channel LLR

$$f_{V^{(l)}} = f_{V^{(1)}} \otimes \lambda\left(\rho\left(f_{V^{(l-1)}}\right)\right)$$

$l = 2, 3, \ldots\ldots$

Prob. of error $p^{(l)} = \Pr\left(V^{(l)} < 0\right)$

Threshold $\sigma^* = \max\{\sigma : p^{(\infty)} = 0\}$

So, let us do a so the last thing we saw in the previous lecture was this idea of density of illusion for soft message passing. Let me just quickly summarize that. Remember this is for B P S K, A W G N and A W G N is parameterized by channel noise sigma square. So, what you start with the first iteration, where v 1 is distributed as normal mean 2 by sigma square variants, 4 by sigma square. So, this is the P D F of the channel L L R, L L R it is the P D F of channel L L R. How do you find the P D F of V 2? It's some formula and its basically f v 1 convolved with this function lambda then this function row of f let me write p l in terms of l minus 1. So, that is better l minus 1 you run this from l equals 2, 3 onwards

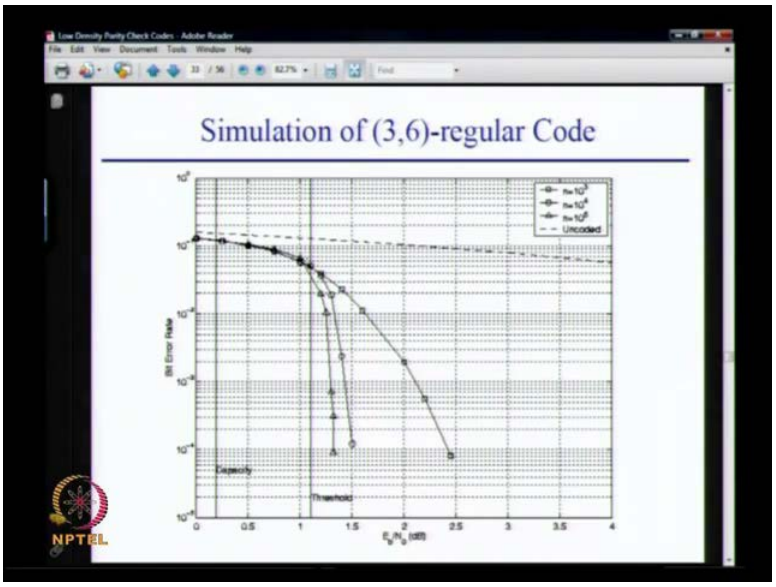
So, remember how this P D F look, this mean 2 by sigma square and variants 4 by sigma square. It will look something like this, we are not quite like this. So, those P D F will look like this, what is the probability of n r in this P D F in this picture this area? It is a probability of error. You can even do with q it will q of one by sigma that should be q of 1 by sigma minus, because it multiplies by 2 by sigma square, suddenly probability of error cannot be different, that is should be q of 1 by sigma. So, that will be the probability of error.

So, in general also after iterations what is the probability of error corresponding to a particular P D F real. So, that will define as p l p l is the probability of error. Remember, all my messages are supposed to be log likelihood ratios for the corresponding bits on an edge. On an edge you have a bit node on one side every message is an L L R for that bit. I know that bit is already zero, I am restricting myself to the all zero code word. So, if I have an error that bit must have become must go to one, which means the message must be negative. If my bit is zero and the symbol is plus one my message has to be positive for no error and negative for error. That is how you set this area was the probability of error.

So, probability of error for the l th iteration is simply probability that v l is less than 0. So, the integration I mean the area under the curve on the left side it is a probability of error. So, as you do iterations this is basically f v 1 what do you expect f V 2 to look like? Hopefully, it will get compressed on the left side. Then and the p could may be shift to the right. As you keep iterating more and more what do you expect your P D F to look the probability of error hopefully is lesser.

Then the pick is shifting to the right continuously, eventually if for l tending to infinity. You have to assume you have to get zero probability of error. Then what should have happen, there should be nothing on the negative side, it should have totally gone up to the right. So, that is the that is the idea. So, based on this definition of probability of error you can define a threshold sigma star, which is maximum sigma, such that p infinity is 0. So, this is the threshold that I have plotted here.

(Refer Slide Time: 05:55)

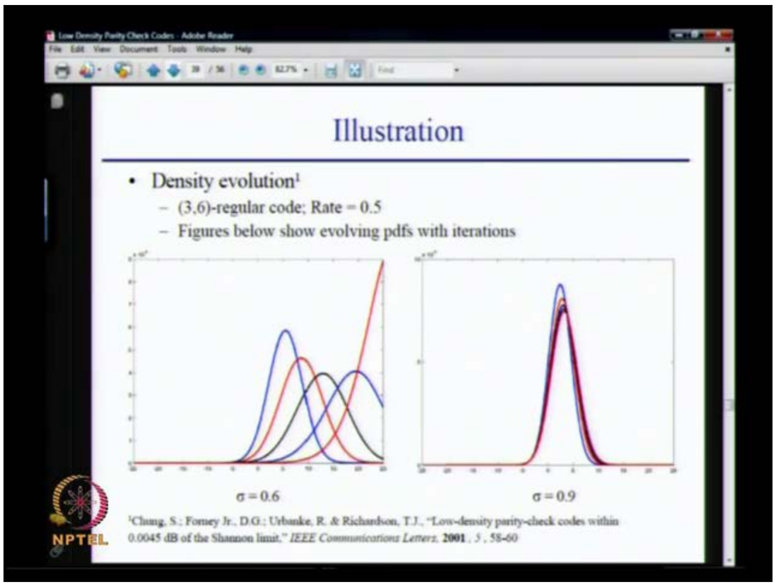

Simulation of (3,6)-regular Code

This threshold line some 1.1 d b for 3, 6 regular codes. That is the 1.1 d b I will get a sigma star. I can always convert a sigma star into an u d over n naught 1 by 2 are sigma square is the formula. So, r is equal to half here you plug it in you will get a u d over n naught of 1.1 d b, so that is the threshold that I have plotted here. So, how will you compute this threshold? You really need this density allegation formula. So, we start with the low enough sigma lets much lower. So, that b 1 P D F, which is mostly to the right.

Then you do a few iterations of a density evolution, you are going to get zero probability of error. Then what do you do keep increasing your sigma slowly till you gets your point where the P D F is not really continuing to move to the right. It gets stuck somewhere if the P D F gets stuck somewhere. Then it is not going to move to the right and you are going to get a non zero probability of error. So, that is what you look for in density evolution.

So, it is not as simple as B S C any more, where you could just see whether the number is tending to zero or not. You have to actually track a entire function and see if it has any area on the negative side, that is what you have to see. So, little bit more complicated, but it can be done. I mean it is you can do it with good precision and you will get your answer. I mean it is also hopefully it is also clear to you how the iteration is starting. If you put l equals to you only need f v 1 and f v 1 you know already. So, it starts and proceeds like that. So, maybe I have some pictures or do not know maybe I have maybe I do not have, not very sure. So, here are some pictures.

(Refer Slide Time: 08:02)


Low Density Parity Check Codes - Adobe Reader
File Edit View Document Tools Window Help
NPTEL

Illustration

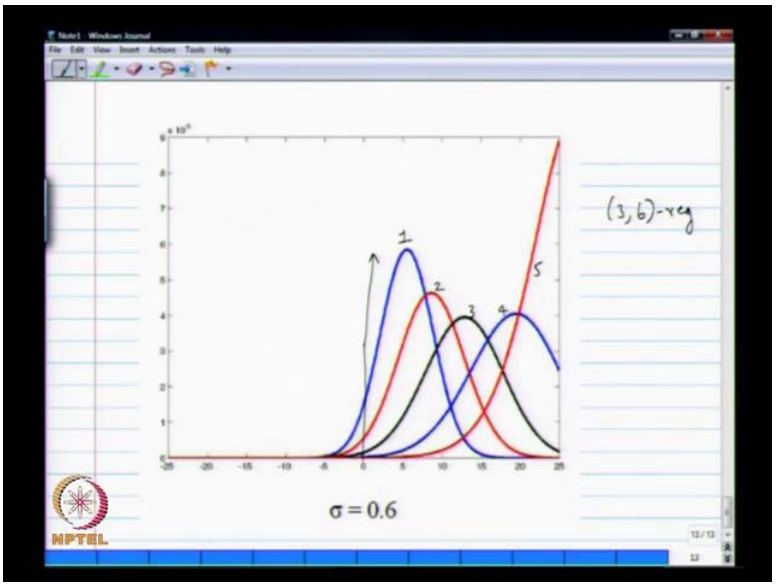
- Density evolution[1]
  - (3,6)-regular code; Rate = 0.5
  - Figures below show evolving pdfs with iterations

$\sigma = 0.6$ $\sigma = 0.9$

[1]Chung, S.; Forney Jr., D.G.; Urbanke, R. & Richardson, T.J., "Low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Communications Letters*, **2001**, 5, 58-60

So, these are actually some density evolutions that I, did you know for a 3,6 regular code for sigma equals 0.6 that is shown on the left side. Sigma equals 0.9 that shown on the right side. So, on the left side you are seeing, so the blue is the original the first blue that you see on the left side, that is the first L L R, P D F that you start with that is the P D F of the channel L L R, I do one iteration. So, this is I did not think I can write on top of this right, but this will like fifty six page document. I can copy and paste, so let me just do this. So, that is better, so this you do this and I maybe form this.

(Refer Slide Time: 09:15)



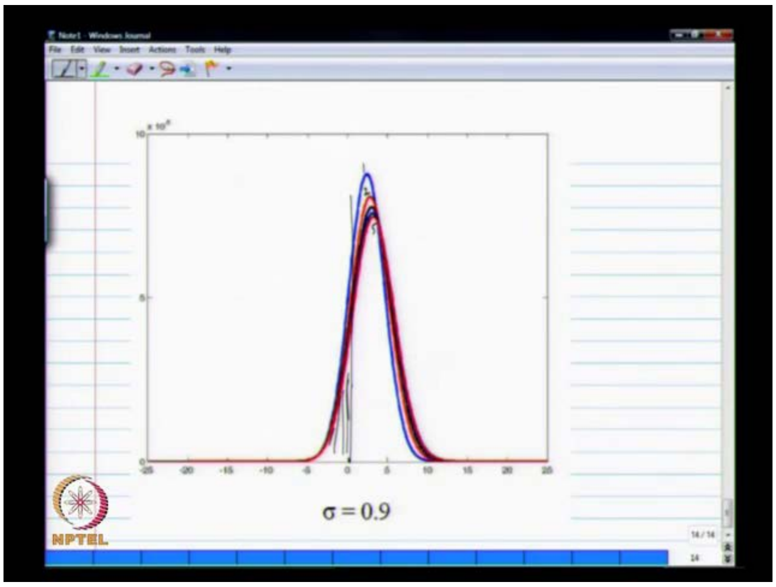
So this is first iteration this is second iteration, this is third iteration, this is fourth iteration fifth iteration. So, you can see what is happening. So, initially you have remember this is my x axis. So, initially you have quite a bit of area to the left of the y axis. Then after you do first iteration it is shifted to the right. It is compressed on the left side. The second iteration, third iteration is even further now. Finally, by the fourth or fifth iteration, you are saying it is pretty much 0. So, this happens its sigma as 0.6. So, this is once again for the 3, 6 regular code, so that is rate half

So, in terms of I think if you think in terms of ah we cannot quite see resolution here, but think in term of resolution minus 25 into 25 is good enough. So, you are thinking in mean, so you have to think a lot think about a lot of things will numerically implement this. I am looking at the P D F of some random variable what should be the range that I should keep track of? I cannot keep track of an infinite range has to be some finite length only. So, here I am keeping track only between minus 25 and plus 25 and that seems to be good enough.

So, it is a good number to remember minus 25 into 25 is a good range of L L R. L L R being 25 is a large number, you know may that is pretty much enough. Then resolution also matters minus 25 into 25. I cannot store a real valued function you know I mean I cannot real number function, all I cannot store I would take resolution. So, this might been have like ten bits or so, I might have taken divided this into about a thousand intervals and then everything in the interval.

Some resolution you have to take finite resolution it looks quite, when you cannot see these steps, maybe a little bit if you assume. Maybe, when it goes up you can say some minor thing, but the resolution is quite big quite good enough. So, that is quite small and since this sigma 0.6. Then the next thing sigma 0.9 let me do the same thing.

(Refer Slide Time: 11:44)

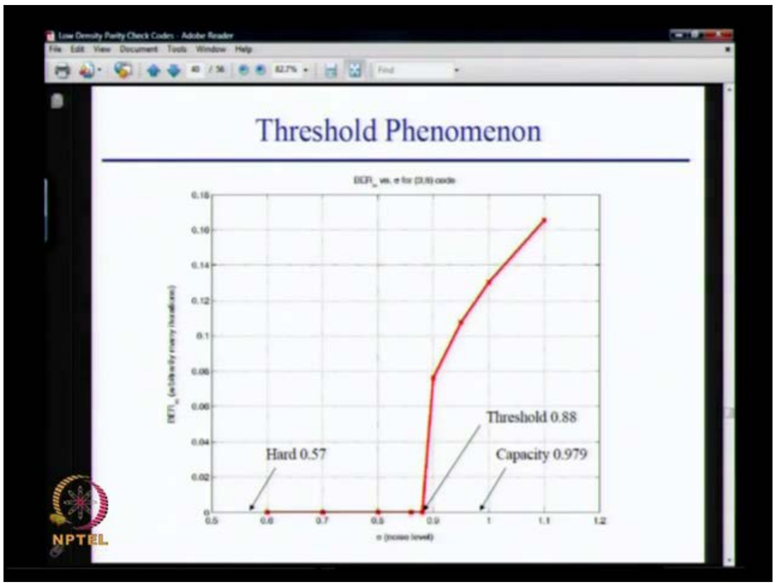


So, you can see the probability of error I think its 10 power minus 3. So, if you wondering about the y axis that is 10 power minus 3, it’s not 8 or something. So, it is it is quite big. So, when your noise is very large your decoder is not going to give you zero probability of error. So, it is beyond the threshold, so when the noise has gone above the threshold its not going to give you zero probability of error. That you can see in this picture, that is your y axis this you as you keep doing more and more iteration 1, 2, so on.

Till like fifth iteration you have pretty much reach like a fix point of your iteration. You take a function you go through the iteration, you get the same function back keep on doing it. You hit a fix point you are not going to improve beyond that and that gives you a finite probability of error. So, based on these two runs you can know that sigma star lies between 0.6 and 0.9 for rate half codes. I could have think and can be found now by repeating this step over and over again. Now, you say maybe its 0.8, 0.7 actual number will come based on that.

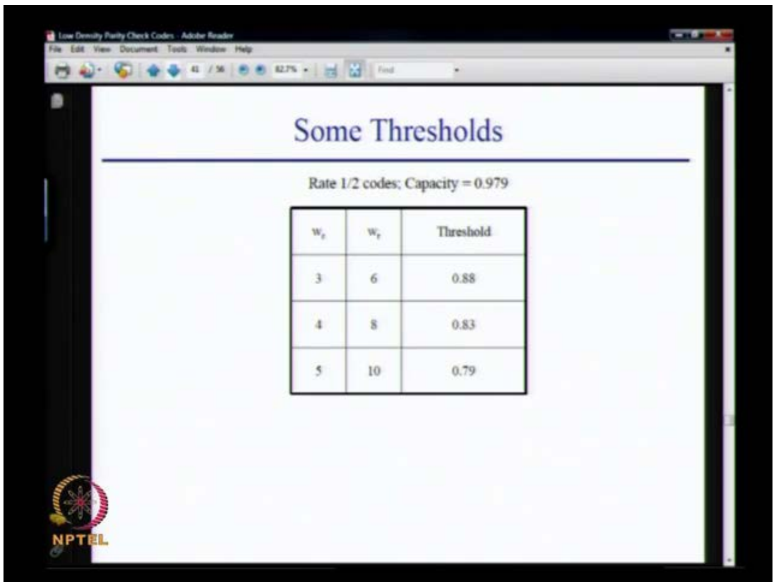
So, to find sigma star you have to repeatedly run density evolution for multiple iterations several times, only then you will get sigma star. You will have to compute several sigma stars when you are trying to optimize the degree distribution. So, to have a very good efficient density evolution is very important. That is useful when you design L D P C codes, think that is probably the last slide that is interesting in this.

(Refer Slide Time: 13:31)



So, this is a picture for something, then some thresholds. So, these are nice things to see.

(Refer Slide Time: 13:40)



Some Thresholds

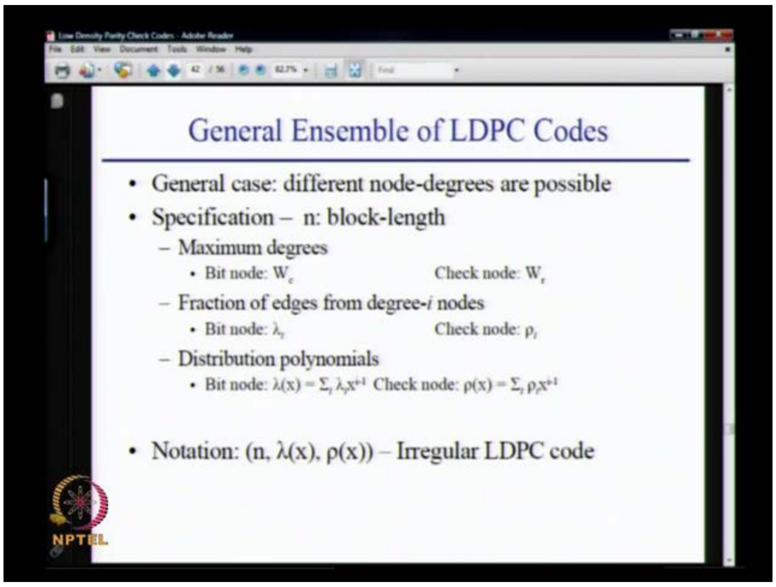
Rate 1/2 codes; Capacity = 0.979

w_c	w_r	Threshold
3	6	0.88
4	8	0.83
5	10	0.79

Here are some threshold for rate half codes the capacity is 0.979. So, this is B P S K A W G N, so capacity sigma star is 0.979, what do I mean by capacity? Again, what is capacity again? So, when I say capacity is 0.979, what does it mean? It means for any sigma less than 0.979 , I can have a rate half code for long enough block length. That gives me arbitrarily low probability of error 10 power minus 6, whatever. So, that is what capacity means. Now, threshold what is threshold mean for 3,6 regular codes for any sigma less than 0.88. If I go to a long enough n long enough block length my probability of error will go down to 0, that is what it means.

So, the gap between threshold and capacity is non 0.88 and 0.979 you have not come up to capacity. You can do better designs, but you are stuck at 0.88 that is because of regular 3, 6. If you go to 4, 8 what happens? You get 0.83, which is worse than 0.88. So, in the Gallagher a decoder we saw that 4,8 was better than 3,6 in D P S K, W G N soft message passing decoding 3, 6 is the best weight half regular code 4,8 is poorer 5,10 is even poorer as you keep going further, further it will be even worse and worse. So, based on just regular codes, if you were to ask what is the best weight half code, it is going to be 3, 6. Maybe, you make sure you mug up all these numbers, because some number like this might I want to ask you a question, do not scared, so that is that okay?

(Refer Slide Time: 15:40)

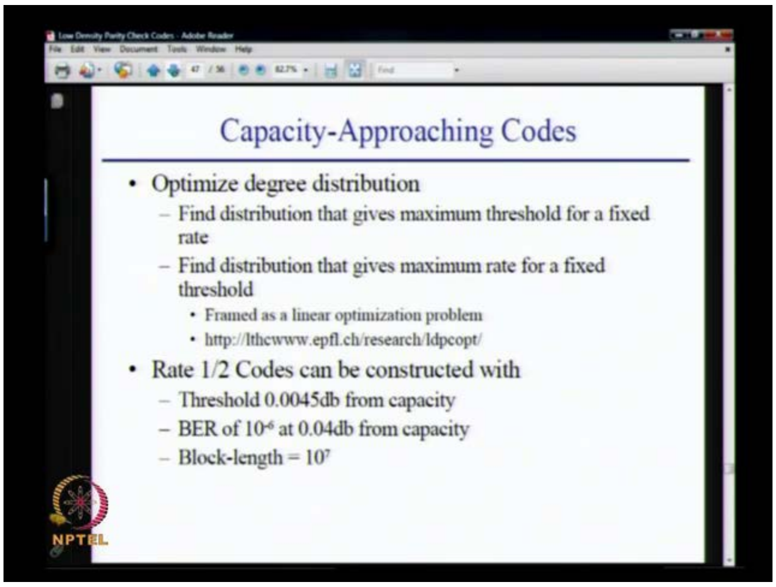


General Ensemble of LDPC Codes

- General case: different node-degrees are possible
- Specification – n: block-length
  - Maximum degrees
    - Bit node: W_c Check node: W_r
  - Fraction of edges from degree-*i* nodes
    - Bit node: λ_i Check node: ρ_i
  - Distribution polynomials
    - Bit node: $\lambda(x) = \Sigma_i \lambda_i x^{i-1}$ Check node: $\rho(x) = \Sigma_i \rho_i x^{i-1}$
- Notation: $(n, \lambda(x), \rho(x))$ – Irregular LDPC code

Now, let us this is all some slides on irregular codes.

(Refer Slide Time: 15:50)

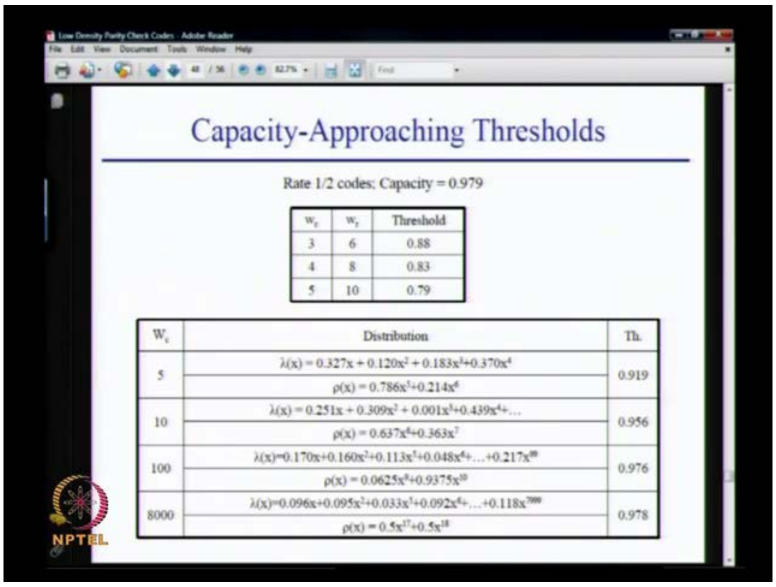


Capacity-Approaching Codes

- Optimize degree distribution
  - Find distribution that gives maximum threshold for a fixed rate
  - Find distribution that gives maximum rate for a fixed threshold
    - Framed as a linear optimization problem
    - http://lthcwww.epfl.ch/research/ldpcopt/
- Rate 1/2 Codes can be constructed with
  - Threshold 0.0045db from capacity
  - BER of 10^{-6} at 0.04db from capacity
  - Block-length = 10^7

So, how do you get to capacity is a question. So, this looks like there is a gap, how do you get the capacity with a regular codes with regular L D P C codes in terms of, you have to optimize the degree distribution regular is not good enough. You have to optimize and get better degree distributions. Like I said there are rate half codes, whose threshold is 0.0045 d b away from capacity. It works quite well. So 0.04 d b from capacity, so that is what this slide shows.

(Refer Slide Time: 16:24)



Capacity-Approaching Thresholds

Rate 1/2 codes; Capacity = 0.979

w_c	w_r	Threshold
3	6	0.88
4	8	0.83
5	10	0.79

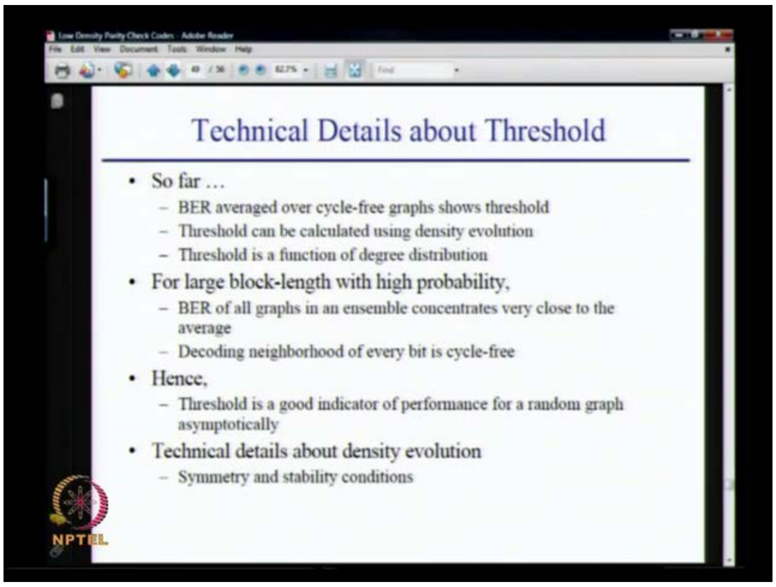
W_c	Distribution	Th.
5	$\lambda(x) = 0.327x + 0.120x^2 + 0.183x^3+0.370x^4$	0.919
	$\rho(x) = 0.786x^5+0.214x^6$	
10	$\lambda(x) = 0.251x + 0.309x^2 + 0.001x^3+0.439x^4+\ldots$	0.956
	$\rho(x) = 0.637x^6+0.363x^7$	
100	$\lambda(x)=0.170x+0.160x^2+0.113x^3+0.048x^4+\ldots+0.217x^{99}$	0.976
	$\rho(x) = 0.0625x^9+0.9375x^{10}$	
8000	$\lambda(x)=0.096x+0.095x^2+0.033x^3+0.092x^4+\ldots+0.118x^{7999}$	0.978
	$\rho(x) = 0.5x^{17}+0.5x^{18}$	

Here are some degree distributions just to compare with regular and irregular. So, if you look at the first one that comes here it has a threshold of 0.919, which is already better than the 3,6 regular code. Look at the degrees on the right degree, we have just 6 and 7. There are only two different degrees, one of them is 6, the other is 7. Remember, in row f x we put x power j minus 1 I mean lambda affects we put x power i minus 1. So, the degree minus 1 is what you what in the affects the 6 and 7. Just two different degrees and on the left hand side you have many more.

You have 2, 3, 4 and 5 just with a little bit playing around you have got already 0.919. Then you have other degrees and W c 10 means, when I say this W c this is the maximum left node degree, that is what this is. This is the maximum left node degree W c is the maximum left node degree for the regular codes. Its only 10 if you allow a degree left node degree of 10 you already get 0.956 and that quite close to 0.979 and these are very fairly old distributions.

Maybe, today there are better distributions people might have found better ones may, if you have program that keeps optimizing degree distributions. You can just let it run for a few months. You know how does it hurt it keep on running every week, you might get a new distribution if you see a better threshold. So, keep on doing it does not stop, so the last one is bit of claritical interest, but things the set things like the second one with maximum degree 10. These are really interesting to the distributions. So, very simple not too complex. You can easily construct them right and these gives you a very good threshold.

(Refer Slide Time: 18:15)

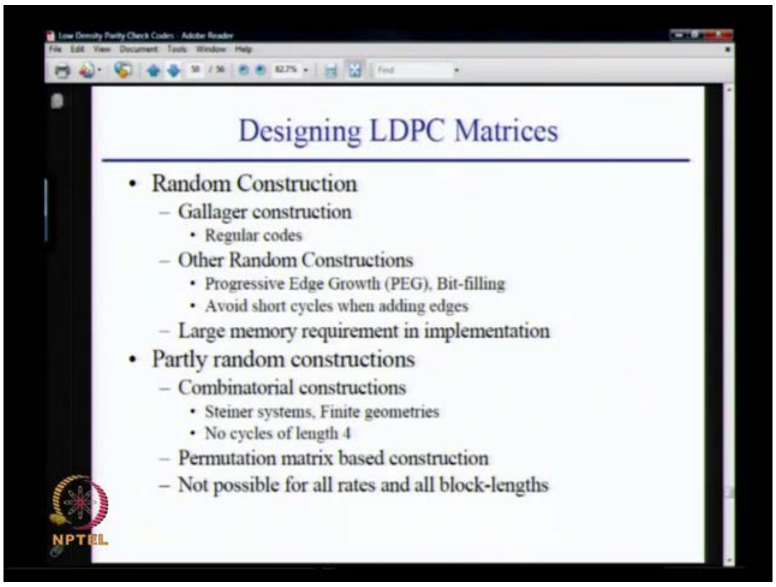


Technical Details about Threshold

- So far …
  - BER averaged over cycle-free graphs shows threshold
  - Threshold can be calculated using density evolution
  - Threshold is a function of degree distribution
- For large block-length with high probability,
  - BER of all graphs in an ensemble concentrates very close to the average
  - Decoding neighborhood of every bit is cycle-free
- Hence,
  - Threshold is a good indicator of performance for a random graph asymptotically
- Technical details about density evolution
  - Symmetry and stability conditions

So, something some technical details, which are not so important.

(Refer Slide Time: 18:22)

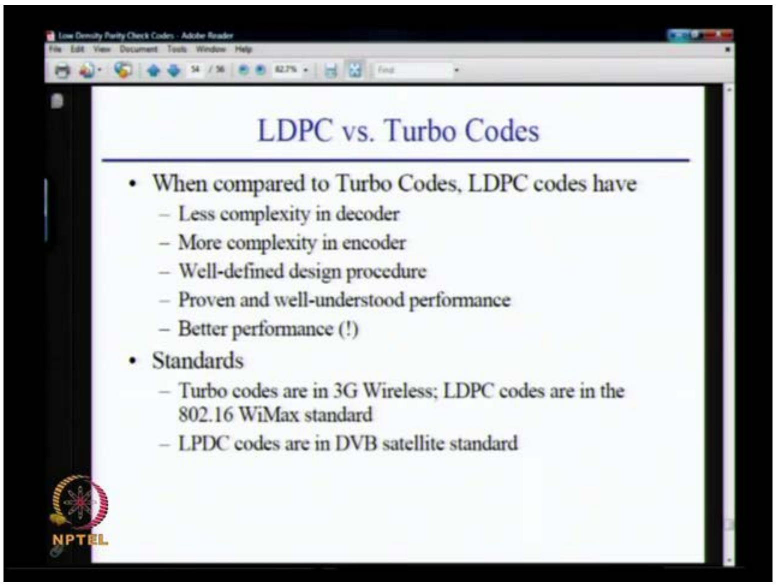


Designing LDPC Matrices

- Random Construction
  - Gallager construction
    - Regular codes
  - Other Random Constructions
    - Progressive Edge Growth (PEG), Bit-filling
    - Avoid short cycles when adding edges
  - Large memory requirement in implementation
- Partly random constructions
  - Combinatorial constructions
    - Steiner systems, Finite geometries
    - No cycles of length 4
  - Permutation matrix based construction
  - Not possible for all rates and all block-lengths

So, like I said there are there are several construction most of them are most of the constructions are random there are some algorithms. We do not have time to look at those algorithms, you can do a search from the internet. You will find lots of algorithms and ideas on how to optimize the random construction. You have a certain degree distribution you respect that. Then on top of that how do you assign the edges? So, that your B E R curve is better. I mean there are lots of research that was done. People have some algorithms in particular, there is an algorithm called ace. So, it is a very popular and successful algorithm, which you can try to use. So, there are other methods also, I am not going into that. So, let us just keep this, so that is it.

(Refer Slide Time: 19:25)



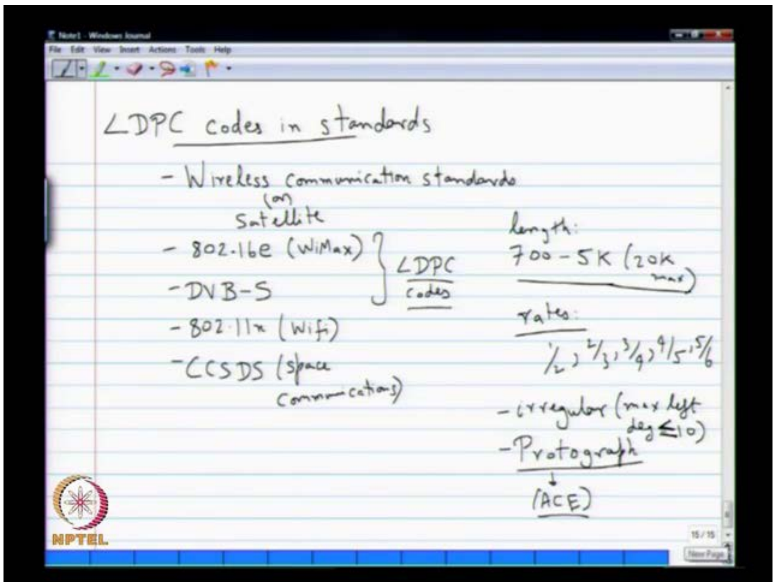
LDPC vs. Turbo Codes

- When compared to Turbo Codes, LDPC codes have
  - Less complexity in decoder
  - More complexity in encoder
  - Well-defined design procedure
  - Proven and well-understood performance
  - Better performance (!)
- Standards
  - Turbo codes are in 3G Wireless; LDPC codes are in the 802.16 WiMax standard
  - LPDC codes are in DVB satellite standard

Let's just stop, there the slides. So, are there any questions on how I got these curves? These kind of these kind of behavior. So, for instance the existence of such a threshold. You can show theoretically that for this evolution process, it looks like a fancy evolution process, but there are some analysis that you can do even for these things. It is not really that difficult, but once you understand what the process are you can do some analysis. Its little bit involved, but not too involved. Show that if the channels satisfy some conditions a threshold, like this will always exist there will be a sharp point. It will be a sigma star before, which probability of error will go to 0 as l tends to infinity after, which it will not go to 0. You can show that there will a sharp point were changes.

So, that is something you can show theoretically and lots of other theoretical analysis possible for this process, but we do not have time to do any of those things in great detail. If there are no more questions we can try and see try to move on to the next topic. So, before we move on to the next topic. Maybe, I can mention one last time one final tidbit of information. Maybe, we will come back to this later also, but L D P C codes in this standards.

(Refer Slide Time: 21:01)



LDPC codes in standards

- Wireless communication standards (or Satellite)
- 802.16e (WiMax)
- DVB-S

$\Big\}$ LDPC codes

- 802.11n (Wifi)
- CCSDS (space communications)

length: 700 - 5K (20K max)

rates: $1/2, 2/3, 3/4, 4/5, 5/6$

- irregular (max left deg ≤ 10)
- Protograph $\downarrow$ (ACE)

So, mostly I am going be when I say standards I am going to be talking about, wireless communication standards, maybe satellite also. So, there's something called the 802.16 e standard. This is also the Wi max standard this has L D P C codes and this D V B, I believe D V B S is digital video broadcast D V B S also has L D P C codes. So, these two guys have L D P C codes there are also others for instance the I believe the Wifi also the latest thing 802.11 n Wi fi is it? Wi fi 11 n I think this also has L D P C codes all these have L D P C codes specified.

So, the type of L D P C codes that these things have I mean the theory is pretty much similar. I mean we need degree distribution, which is also should a little bit decent to handle. You cannot say 8000 as 1 of the degrees, it is a little bit difficult to construct those kind of things. So, usually the block length here is a few thousands. So, it will start at around seven hundred or so and go all the way up to say 5 K or so, even another standard, that has a C C S D S, which is a space communication standard satellite and all. So, this things have it.

So, the block length in this range maybe. Maybe, it goes up to 20 K max nothing below that. So, you cannot have too two large degree most of these codes, this is length like this typical rates. You will see are you start at half it its very seldom that you go below half half 2 by 3 3 by 4 4 by 5. These are standard numbers people like small rational numbers. These are standard that you will see. So, this is the sequence that most standards will follow 1 by 2, 2 by 3, 3 by 4, 4 by 5, 5 by 6 outside of coding. When you go talk to hard core communication engineers. This are the numbers that they can easily understand. So, if you say 0.813 then confuse about what to do just I mean see y.

I mean when you design a system you have to worry about clock rate. How the coding rate affects the clock rate etcetera. So, if you think about how these things work. So, you are going to have some message data rate, which is going to be a constant depending on how we are channel changes. You might have to use a rate half code at sometime a rate 5 by 6 code. If your channel is very good you are going to use rate 5 by 6 code.

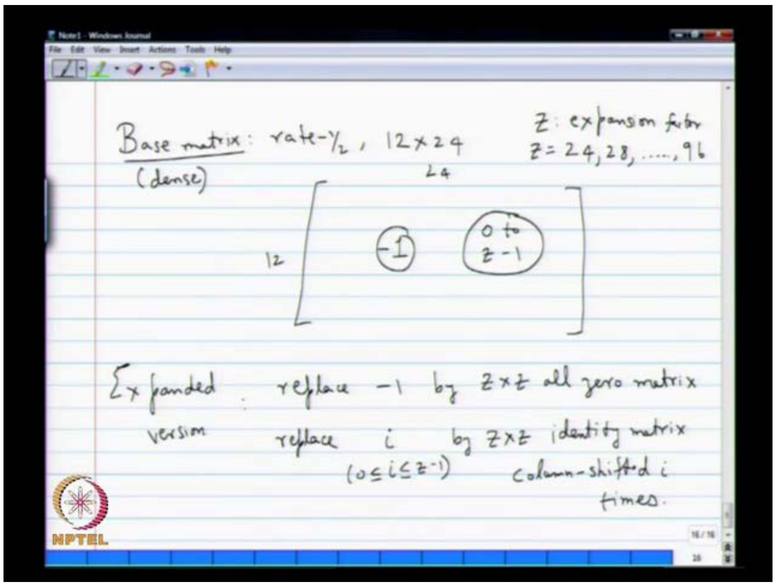
If the channel is very bad you want to maybe use a rate half code, but for both cases you want to keep the message rate the same. Then what has to happen? Symbol rate has to go up for the rate half code. You will have to go down for rate 5 by 6 code. So, you have to have different clocks and manage them. You cannot expect people to divide clock by 813 or something it’s a difficult to do it.

So, these are easy things to do. So, that is why people use these kind of numbers, so these are rates. Then the most of the codes are irregular the degree distribution is designed according to density evolution criteria. The construction is usually what is called a protograph construction. So, let me explain this protograph a little bit more in detail. So, what they would do is they will have a degree distribution maximum degree will be 10 or something for the left nodes over the for a bit nodes. It will be about 10 beyond 10 I have not seen too much. So, it will be ten or lesser than ten maximum left degree 10 maybe even less than or equal to 10 much lesser than 10 also is possible.

So, in the construction they will use this idea of a protograph. So, instead of constructing a large random matrix with putting one's arbitrarily, where ever you want. What they would do is they would do it in two steps. They will construct a smaller matrix, which is reasonably random. Then they will expand it to a larger matrix and that expansion will be very regular. So, I will tell you how that will work and the motivation for this to simplify, the V L S I implementation. If you have to store a large matrix with a sparse ones. If it is very large I mean for length 5 K and all its going to become a large matrix today. V L S I you can do it, but not at low power. If you also want low power, where do you want low power always want low power.

So, always good was wireless devices these are going to be hand held's maybe even, otherwise if it is not, if it is even in the base station. You might you conserve power conserving power is a good thing today. So, for all those reasons protographs are preferred. Of course, you have to show that doing this is as good as constructing a random graph with the full matrix. So, people show those results also, the construction is usually use this idea called ace. I forget what the expansion is also something to with edges and all that. So, A C E is a popular construction for constructing these kind f things. So, it is something that is used to. So, what is the protograph the basic idea is the following you construct what is known as a base matrix.

(Refer Slide Time: 27:19)



Base matrix: rate-$1/2$, 12×24

(dense)

z: expansion factor

$z = 24, 28, \ldots, 96$

$$\begin{array}{cc} & 24 \\ 12 & \begin{bmatrix} \quad \textcircled{-1} \quad \boxed{\begin{matrix} 0 \text{ to} \\ z-1 \end{matrix}} \quad \end{bmatrix} \end{array}$$

Expanded version : replace -1 by $z \times z$ all zero matrix

replace i $(0 \leq i \leq z-1)$ by $z \times z$ identity matrix column-shifted i times.

Suppose, let's pick some numbers to make it specific. Then we will see the generalization is not too hard. So, maybe the best matrix for the rate half codes. I believe this is the this is the y max standard. This is what we use the A C E matrix is a 12 by 24 matrix. So, 12 by 24 very small, it can be stored very efficiently, but the base matrix will be dense. It will not be sparse its 12 by 24 and the base matrix will have three types of entries.

So, I will also fix some z, which is called the expansion factor in the y max standards z text values 32, 36 or actually 24 it starts with 24 24 square is 798 its not correct what is 576. So, that is the smallest 28. So, on it all goes all the way up to 96 in steps of four. So, for a fixed expansion factor. You will have a base matrix for every expansion factor, you will have this matrix the base matrix will have three types of entries some entries will be 0. Some entries will be minus 1 I should not say 0 does not matter some entries will be minus 1. Other entries will be between 0 to z minus 1.

So, remember it is a 12 by 24 matrix some entries will be minus 1. Other entries will be 0 to z minus 1 this is the base matrix. When you expand to get the expanded version you replace minus 1 by z by z all zero matrix. You replace i for i between 0 and z minus 1 by z by z identity matrix column shifted i times column shifted either to the right or the left.

It depends on the standard, some standards might want you to shift to the left some standards might want you to shift to the right, but both are basically equivalent. Whether, you shift to the left or right, if you do it enough times, it becomes the other shift. So, it does not matter, some shifted version the identity matrix. So, you have a small 12 by 24 matrix, which depending on the expansion factor is going to be blown up into a larger matrix. So, if you pick z equals ninety six you are going to get block length of 2, 3, 0, 4. So, it is a fairly large matrix, but you started with a small base matrix, it is just 12 by 24. So, you can go all the up to 2 K with that for different z you define a different base matrix and this is what you do.

So, primary reason for this is V L S I complexity when decreasing the complexity in the in the hardware implementation, which might be, which is good. So, for instance if you think of actually a communication system in total think of your cell phone, it has maybe just one chip. These days there are cell phones with one chip and that in one chip what fraction. Do you think will be signal processing software for communication? Any guesses? Very small believe me all the signal processing that you do for communications all your entire communication thing is probably going to be inside a small. Maybe, a centimeter by centimeter maximum, it cannot be more that, maybe even less than that.

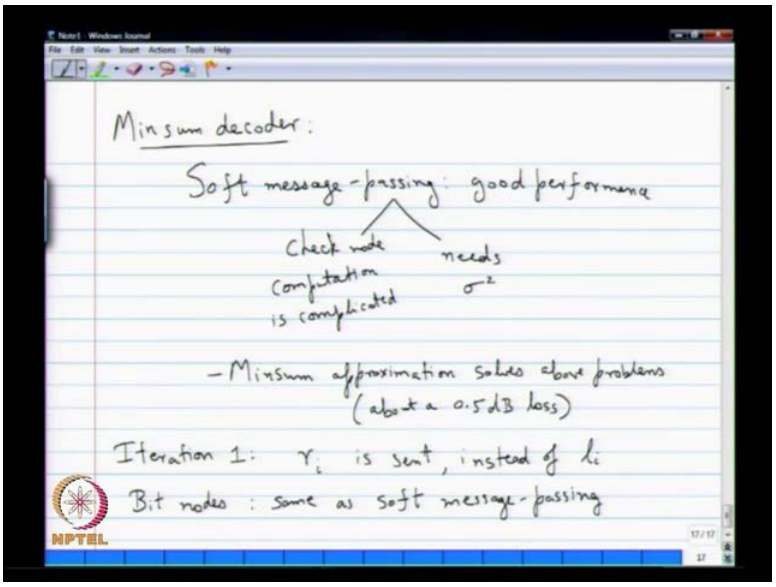
So, it is going to be a few millimeters by few millimeters in area that is all you will get for communication algorithms, everything else will be what all applications. I mean that is what people want in the cell phone who wants communication should be free. You want applications, you want to able to M p 3 playing this and for you mean to fancy all kind of what? Facebook, Twitter whatever you want everything else in your cell phone, you do not want communication.

Communication should be small that is the that is the entire idea and advancements had become much smaller. So, within that millimeter by millimeter area mean few millimeters by few millimeters area a maximum of one millimeter square, will be given for the error correctable code. So, you would not get more than that. So, you getting like a millimeter square kind of area. So, it is little bit more, I do not know, but it is really tiny. That is what I am trying to point out in today way. Today's system are evolving I you cannot say I want a huge area for my error control code. So, nobody will pay you money for that wanted, but cannot make money. So, in that context it really helps to simplify the matrix specification. This specification can take some area and terms of memory also can be significant.

So, you have to decrease all that and this is pretty useful for that. So, that is the main motivation for doing things like this. Couple of other things I should point out when it comes to implementation. So, the encoder complexity is something that is something that should trouble you now. So, because if you have something like Gaussian elimination, you are going to get now dense matrix again, what is the point in simplifying and making a your parity check matrix parse, when anyway you have to store a dense generator matrix.

So, turns out even there because of the protograph structure there are lot of simplification are lot of simplifications. So, the protograph structure is useful reasons and the performance is not bad at all it does not suffer. Because, of this regular structure, you get a good structure. So, that is one comment I wanted to make and the other comment I want to make about L d soft decoders is this idea of a minsum decoder.

(Refer Slide Time: 33:50)



Min sum decoder:

Soft message-passing: good performance

Check node computation is complicated

needs σ^2

- Minsum approximation solves above problems (about a 0.5 dB loss)

Iteration 1: r_i is sent, instead of l_i

Bit nodes: same as soft message-passing

So, you will hear this a lot if you look at implementation, we looked at soft message passing decoding though. So, the soft message passing decoder I mean it is definitely an approximate decoder it is not very accurate, because of cycles and all that right, but still if you look at it in terms of being some kind of an ideal decoder, is in soft message passing is some of the best things, you can do in mean practice. You cannot really do much better than that it is has good performance, but in terms of complexity there is one there are couple of problems in terms of implementation and complexity, there are two problems.

So, one problem is the check node computation is intense. So, computation is complicated, it involves a look up. Then add a adder then another look up, then you have to split the sine out. So, it can be implemented it is not very bad in today's, but still it is complicated, maybe you want to simplify it. See, if it can be simplify the other problem is needs sigma square. Does it sigma square or not yes the first process first step needs sigma square. If you are passing L L R you have to multiply by 2 by sigma square to get the get the log likelihood ratio from the received value r.

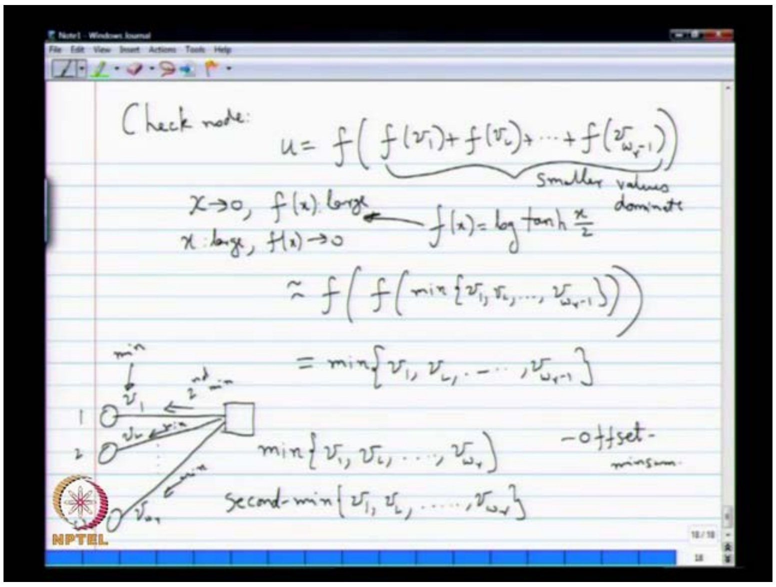
So, there are some this it is mean one might say, you can estimate sigma square at the receiver how hard is it to estimate the noise variance is. You might be able to do that there might be several ways of doing it, but in the other hand there might be cases, where you may want to spent time estimating it. Maybe, you can do without estimating it, who knows there might be some methods of doing it.

So, both of these are solved by the minsum decoder. So, the minsum approximation solves both of the above problems without taking too much of a hit with about these are all just numbers both are 0.5 d b loss. So, compare to the soft message passing decoder, which you can implement with the exact sigma square. With the knowledge of sigma square the minsum decoder, which does not need sigma square will suffer about a 0.5 d b loss, if 0.5 d b is not a big deal for you.

So, you might say I have again a lot by estimating the noise variance. Maybe, I will estimate only who knows you know I mean let me not take that risk. I will simply take the minsum decoder and if once again I mean it again depends on how you construct your code. How you design your degree distribution etcetera and etcetera and all those things are not very well studied, but people based on extensive simulations are found that, if you take a very standard degree distribution and construct your code well.

Then the minsum approximation will not hurt you by more than 0.5 d b. So, particularly the y max codes are pretty good with these, so minsum decoder good. So, what is the basic idea in the minsum decoder the bit node computation is exactly the same. There's no change in the bit node computation, but except that in the first iteration you will simply send R i. You would not send the log likelihood ratio. So, you will simply send R i itself. So, that is the first step iteration one R i is sent instead of instead of what L i. So, you do not need sigma square bit nodes same as soft message passing. So, what you do in the check nodes is, so take a look at this check node computation

(Refer Slide Time: 38:24)



Check node:

$$u = f\Big(\underbrace{f(v_1) + f(v_2) + \cdots + f(v_{w_r-1})}_{\text{Smaller values dominate}}\Big)$$

$x \to 0$, $f(x)$: large $\leftarrow f(x) = \log \tanh \frac{x}{2}$

x: large, $f(x) \to 0$

$$\approx f\big(f(\min\{v_1, v_2, \ldots, v_{w_r-1}\})\big)$$

$$= \min\{v_1, v_2, \ldots, v_{w_r-1}\}$$

$\min\{v_1, v_2, \ldots, v_{w_r}\}$ -offset-

minsum

second-min$\{v_1, v_2, \ldots, v_{w_r}\}$

I will try and motivate how you do the approximation. So, remember the u that you have in the check node is f of b, f of b 1 plus f of b 2 plus one tell f of b something I do not know. So, we let say w r minus 1 right this is the computation that you doing this is for the magnitude of course, for the sine you have another computation going on in parallel. So, if you look at the function f of x it is log tan hyperbolic x by 2, so tan hyperbolic x by 2 near 0 if x is close to 0 it is simply becomes equal to x. So, that is it is almost linear in the begin close to x. So, if x is very small what will happen if you take log of it remember it is all absolute value up to it take log, it will make a absolute value, so all positive.

So, what will happen to a log of small number, it is going to become very big. On the other hand if x is very large tan hyperbolic x by 2 is going to close to one. Then you take log of that you are going to get 0. So, it in this in this computation the smaller values dominated in particular. The least value will have the largest component, so what you do is you make this approximation. This is the same as f of f of min of this guys, remember all these minimum is absolute value. We have to take absolute value everywhere. So, you simply approximate the sum by f of the minimum of these things, that is the idea. So, the observation is this f of x x tending to zero f tending to zero oh my goodness something has happened x tends to 0 f of x is large and x is large, means f of x is tending to 0.

Based on that you simply make this approximation you take the minimum in absolute value of these things and do f of f. So, what will happen when you do f of remember f is it is own inverse will simply get equal to minimum of V 1 V 2 to V w r minus 1. So, this is the minsum approximation of course, the sine equation will carry on as before. There will no change in the sine, but in you do not need an f. So, you simply replace f of f of by simply the minimum. There are some modifications here turns out minimum is slightly bad approximation. So, people usually offset it by something there is like a 0.75 that you multiply by just like that.

So, you do not ask me why you know you just do it, it works, it works better than before. So, you do something like that some offset is done usually to undo the effect of this min, but let me not go into that, but minsum basically the idea is this. So, look at what will happen at the check node there is a very easy computation. That you could do you have a check node. You have W r guys connected to this and you are getting V 1 V 2 all the way to V w r. What you have to compute is basically two things you have to compute the minimum and the second minimum. You just compute two things you compute the minimum and the second minimum, what is second minimum the next least value.

So, second minimum is basically I do not know, why I did, so much is basically minimum of this, after you thrown away the least one second minimum. Then what you do? How do you pass back messages? Suppose, V 1 was minimum. suppose this is minimum what will you do? You will send the second minimum back. This is all absolute value only. The sin you have to anyway take care of sin will have to computed again, what will you send for everything else, minimum on the edge. That you got the least value you send the second least value, because you have to not consider that. The extrinsic is what you have to send. So, you send the second least value on everything else you send the least.

So, it is a very easy computation that you can do at the check nodes, if you are doing minsum. So, this is this is one implementation idea of course, offset minsum is another idea. So, many other ideas like this. So, another implementation idea that people usually uses to change the schedule.

So, right now the schedule is usually well when I say schedule of message passing messages go from bit node to check node. Then you processing of the check node messages go back from check node to bit node etcetera, so if you go back and remember a long time back, when I introduce the M A P decoder. It is basically the ultimate M A P decoder can be thought of as dealing with parity check by parity check. The approximation that is done in message passing decoder is you go from one parity check to the other depending on your neighborhood. Then what you do you do not account for the dependencies when you combine the information with different parity checks.

Now, how you will go to parity check to parity check is called the schedule. The message passing decoder specifies the certain schedule. So, there are ways to modify the schedule now. So, people are found through experiments that other schedules are better. Maybe, you do not have to strictly follow the message passing schedules particularly for protograph codes

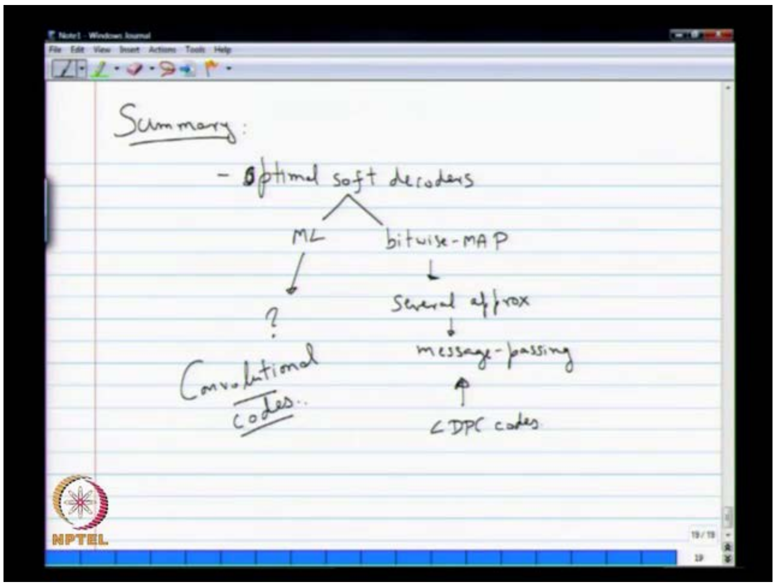
If you design them carefully enough there is schedule I am not able to recall the exact name right now. So, you I forget the name that they give for this, but anyway. So, there are other schedules that they the people use, where they will process one of the rows first. Then they will go to another subset of the rows and then they go to another subset. Actually, they will process one subset of the rows update all the columns do a column processing. Then they will go to another subset of the rows and update all the columns again. Then will go to another subset of the rows and update all the columns again. They will repeat that as suppose to doing all the rows all the columns, all the rows, all the columns.

So, they will use different schedules and that turns out to provide by the performance and it is also simpler. So, ideas like this are used and these things there are lots of papers on this. You can search if you do a Google search you will find, so many hits for L D P C implementations. So, all these are practical ideas and they have they have their value as well. When it comes to standards and implementation you have to pay attention to that.

Any questions on this minsum idea? Instead of L i you should use r, because once you go do minimum scaling does not matter. See, the reason why you can drop the 2 by sigma square is whether I am scaling only in the beginning right even in the soft message passing decoder. I am multiplying by R i by 2 by sigma square, which is a positive number. Now, if I do minimum in the check node I do not have to scale everything by 2 sigma square, you simply drop the scaling, any r i is good enough. So, I do not have to scale at all.

So, R i itself can be used in the bit node that is the idea, because I am doing this f of f of all that I have to take care of the 2 by sigma square. If I do not do that, I am only doing minimum or some operation, which does not involve in any non-linearity. In different way it is easy to just do R i itself, that is the main selling point also. So, the some three minutes left and we have to really make a big jump next. So, let me just quickly summarize in this three minutes.

(Refer Slide Time: 47:11)



So, I think it is maybe it is a bit if a repetition here. So, what we have been doing after the classical part after the Reid's hollow man codes and all that. Let me summarize that a little bit we looked at optical soft decoders. It is very interesting to me that the M L s decoder does not have many approximation. In practice that are implemented the soft M L does not seem to have many approximations, I do not know why. I mean it might interesting to do that, because I will tell you why it is might be interesting, but on the other hand the bitwise M A P has several approximations in particular the nicest approximation is the message passing idea. The design of L D P C codes beautifully feeds into this picture.

So, this is a kind of big picture to remember for a code. We saw that to get good coding game you have to do optimal soft decoding really you cannot do optimal soft decoding for large codes in increase K and N. You also have to increase K and N to get good coding gain. So, when you increase K and N you cannot do optimal soft decoding, but this bitwise M A P particularly. When you interpreted as going parity by parity with an approximation seems to be a powerful idea. Message passing is a nice way of thinking about it and L D P C codes are kind of made for this approximation. So, you do that you get a good implementation.

So, the problem the problem here is if you go to very low rates rate like 1 by 10 or 1 by 8 or something. Then what happens is your parity check matrix really becomes large and the bitwise M A P decoder. Looking at parities is much more inefficient than looking at the M L decoder directly. What is the M L decoder do the soft M L decoder? It does correlations with the codes code words, only if your code rate is very low. Your number of code words is smaller.

So, maybe you should try and approximate M L and to the best of my knowledge. I have not seen any ideas to approximate M L in fact. So, this naturally leads us to some other idea where the M L decoder might be efficient. So, this efficient or not is there a code, where M L decoder is really efficient, turns out the answer is convolutional codes. So, there are these codes called convolutional codes, where the code words themselves can be very easily describe, so that the correlation between the code words can be done very efficiently. The M L decoder becomes nice and that is what we are going to see next and that is a big jump. It is very different of what we have been seeing now. So, I will stop now and tomorrow we will start with convolutional codes.