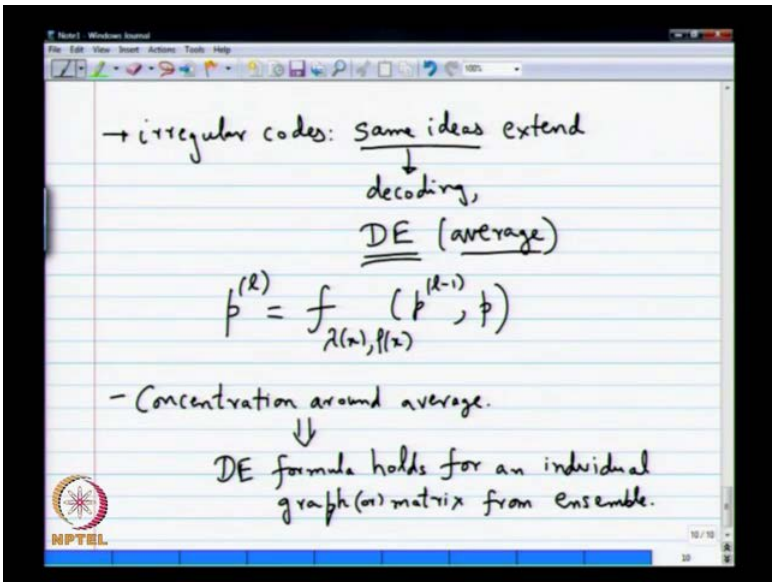
Coding Theory
Prof. Dr. Andrew Thangaraj
Department of Electronics and Communication Engineering
Indian Institute of Technology, Madras

Lecture - 27
Optimized Irregular LPDC Codes, Soft Message Passing Decoders

(Refer Slide Time: 00:15)



So, let us begin, so this is the main recap of what we did in previous lecture, so we extended ideas from regular to irregular codes. The decoding extents in the almost trivial way the analysis not quite in a trivial way, but it also extents you have a density evolution formula that is the main idea. There is the density evolution formula which goes which is parameterized by lambda of x rho of x in terms of which gives you the probability that error that.

There is an error in the message passed on a particular edge on a random edge, so to speak from one iteration to the other. So, you can track this, now you can again analyze the properties of this density evolution formula, so you are from and we did some analysis and found whether it was monotonically decreasing in p, whether it was monotonically decreasing in p l minus 1. All those things will be true, but before we do all that how we reconcile this notion of average, so when you average over all this codes how do we know that this formula is good for any particular code.

That is something that you have to answer, so what turns out, so you can you can prove something known as concentration result around average, what does the concentration around average means. So, if you take for instance the number of errors in the particular iteration number of errors that are there in the messages in a particular iteration across different codes from the same on sample it turns out its concentrated around the average number of errors.

So, what does that mean when you show a concentration around average with probability close to one the density evolution formula will hold for any one graph that you pick from the on sample. So, that is what it means I do not want to go into more technical details than that. Basically, once you show a concentration around average result this implies d formula holds for holds for an individual graph or matrix.

So, this will have some meaning in practice, what it means even though you have in your analysis you have averaged over all possible graphs, we are only computing the average. We have a concentration around the average result which means the average is still meaning full for any one randomly chosen graph from the on sample. It is like saying in a class the average marks was some say 60 percent. Then, I also have a concentration around the average result which says with the number of people between 55 and 65 is may be the fraction of people between 55 and 65 let us say 0.98 or something.

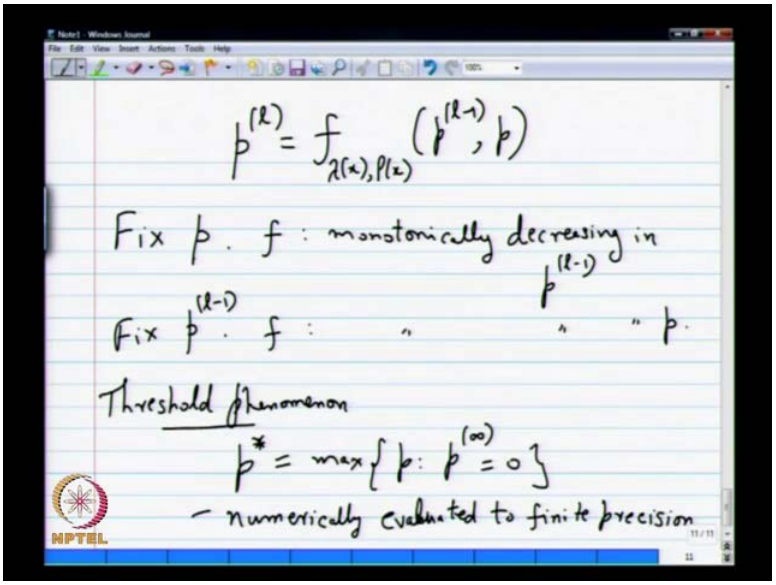
It means the average is very meaning full for every individual person, so if you ask a particular person's mark, you can give that mark correct to plus or minus 5 marks which would be what average itself and say and it will be right almost 98 percent of the time. So, that is what this means concentration around average, so it is useful tool when you analyze a very complex algorithm.

So, when you analyze a complex algorithm in a particular situation you may not be able to analyze it what do you do average out all the situations may be the analyses become simpler. You can you will get some IID variables some random variables will become IID when your average over all these things. Once you do that analysis is easy, but then the average may not be meaning full in practice, but if you have a concentration around average result, then that analysis average analysis is still meaning full in practice.

So, it is a useful trick to remember in case in the future you are analyzing the algorithm as complicated as decoding message passing decoding on huge graphs for example. So,

the density evolution formula is meaningful and it works in the practice, but remember the formula is a bit complex. So, it is much more complex than the regular case regular case at least you have just one expression here. You have some summation over average over lambda i and then average over rho j when you have to take the powers individually and all that. So, the analysis will be a little bit more meant, the expressions will be a little bit more complex nevertheless many of the properties extend without any problem.

(Refer Slide Time: 05:19)

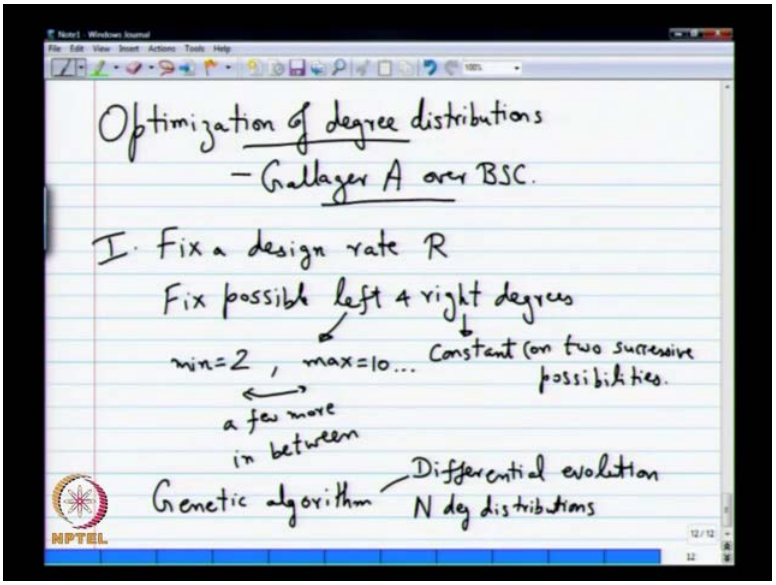


So, what kind of properties are we looking at now in the d e formula, so if you fix p or fix p right p l is a well fix p f is monotonically decreasing in p l minus 1. You can show using some analysis same thing is true for p l minus 1, also you fix p l minus 1 f will be monotonically decreasing in p. So, both of those will be true and then you can also show the threshold phenomena there exists a p star which is maximum of p such that p infinity equals 0.

So, it will be great if you have a close form expression for p star, but usually you would not have a close form expression for p star. So, you have to estimated by numerical methods, so what would be a one numerical method of estimating? So, we will start with p equal 0 and see if what happens to p infinity, p infinity will clearly be 0's. Then, you increase p by say 0.0001, so how many ever precision you want if you want a precision of 10 power minus 6.

So, increase it by 10 power minus 6, then compute the compute p infinity how you will compute p infinity you have to iteratively evaluate this formula till it goes to 0. Once it goes to 0, it will remain at 0, so it evaluate this formula till it goes to 0, then see if it p infinite is 0. So, if it is still 0 then you increment p again, so you can numerically evaluate this to arbitrary precision to some finite precision. That is good enough in practice, we would not have anything better than finite precision, so evaluate. Now, you can have your grand optimization program, what is your grand optimization program, you want to fix for a, this is the optimization of degree distribution.

(Refer Slide Time: 08:10)



Remember, what this optimization is for it is for Gallager A over BSC that is very important because the density evolution formula is for Gallager A over BSC wherever we do density evolution that is the exact optimization you are doing. So, it is not for anything else a code that you are degree distribution that you get for Gallager A over BSC might be very bad over some other channel and some other decoding. So, there is no guarantee that the same code will work over the other things also this is important.

So, what will you do the first thing to do is to fix a design rate R, this is version 1, I will call it version 1 and also I will give you a version 2 which might be slightly different. You fix a design rate R, so this in turns fixes that is what fixes anything, but it provides constraints on l i possible degrees you can have. So, it is very common to also fix possible left and right degrees, otherwise your problem becomes really more complex

and you may not till need to do that and in fact the most cases you will fix right degree to be constant or near constant or two possibilities.

I mean two successive possibilities that is what I mean by near constant, so do not ask me why and all that these are justified. Based on experience, people find that this is good enough for optimization, so for a particular rate r really a fix a right degree to be constant and fix left degrees. How you fix possible left degrees is you will always fix the minimum to be 2 and you will fix some maximum say will be 10 or something. So, of some maximum and then you fix a few in between few more in between, so this this is again once again why because it works in practice.

So, you even you fix a maximum as a 10 or 20 or 30, you do not allow all the degrees from 2 to 20, then your optimization will become so complex, your space becomes very big. So, you can have and to minimize your space to which you are searching, so these are ways of doing that without getting hurt in practice. So, that is the idea, so once you do this, you will have several possible degree distributions, so you can you can in fact find exhaustively all the degree distributions and try to search over them.

That is not usually done usually what people do is some kind of a genetic algorithm what are these genetic algorithms that there is these notion of differential. There is one algorithm called differential evolution this is what is usually used it also abbreviates as d e, but this is very different from density evolution. So, differential evolution what they would do I will describe very roughly what you do is you will fix some capital n degree distributions.

So, if you first pick some n degree distributions at random n is going to be large think of does at least 100, 1,000 may be huge number of random degree distributions with satisfy the rate constraint. These other constraints that I have put on it you pick according to that and then what you do with n thresholds how do you compute the n thresholds you have a numerical method for doing it.

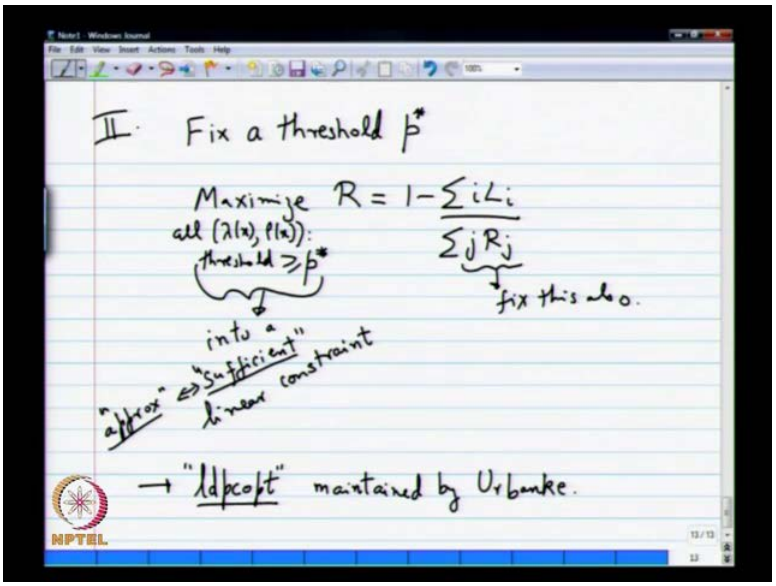
So, you compute the n thresholds, so one among these will have the best threshold right so you pick that best possible guy. Then, you evolve all your n minus 1 using the best possible one, it is like you have some genes, you know best possible guy has some good genes. You mix it with that guy kind of things, so it is a genetic in that way, so you do at

roughly, so how do you mix one way to mix is to kind of join these two distributions in the space.

Then, take some convex combination etc. some you combine the best distribution and every other candidate distribution to get a new distribution that is the next evolution stage. You also keep the best degree distribution by itself then you evolve everything itself and then what do you do you have n evolved in distributions for who you once again compute thresholds. Among them, there will be a best guy, then evolve and evolve there are no guarantees that you have done well.

It just works in practice and very complex optimization problems such methods work well if you do some optimization courses later on you can learn about many more genetically algorithms in this. So, when problem becomes very complex and you would not have, you use this kind of methods. So, this is one way of doing it, I cannot really answer any questions about this because you have to implement to see how it works. I have students implement this and it works you get good degree solutions about this. This is a rough idea like I said this is one way of doing it, there is also another way which is which gives you which gives you some rough linear programs and these methods are also quite useful.

(Refer Slide Time: 14:08)



I am not going to exactly describe it, but this is how it works so what do you do is think you would fix a threshold say some p 0 let us say p star. So, we will fix a threshold p star

some threshold probability and then you maximize rate the design rate over all lambda of x rho of x such that such that threshold is greater than or equal to p star. So, this is a kind of dual way of doing it what did we do before. We fix star and then we maximized p star over all lambda effects low effects that satisfy the rate constraint.

Now, we are doing the other way, so you want all the lambda effects low effects you want to consider those which have threshold at least p star. Among those, you try an maximized rate nice thing about doing this is remember rate is going to be 1 minus submission i l i by submission j r j. So, what people do is usually they fix this part also and there is a smart way of converting this constrain into a linear constrain into a sufficient linear constrain not even a sufficient. Let us just say approximate linear constriction some kind of constrain which is linear in per to being etcetera.

So, that is a smart way to do this, so what is the advantage if you all this you basically get a linear probe. So, that is the big advantage when you do this other program, so the objective function itself it is linearized by fixing the denominator. So, it becomes linear in the l i's and then you have work with this constrain, so this is a difficult constrain to work with, how do I know the lambda effects and row effects which gives you a threshold greater then equal to p star.

I mean that I do not know, but you can kind of look at the density of illusion of expression very closely and linearized there are ways of doing it, but like I said its only sufficient not even sufficient. I think maybe even that is not proved its only approximate may be its just approximate some approximate linear constrains which guaranteed that the threshold likely to be greater than to p star. Then, you do your linear program to optimize, so this also complex it is not as easy and I think this is not as successful as the density of illusion differential illusion method differential.

I think is much for successful, but this also pretty good you get some thresholds with this for instance what will happen is if allow your left degrees to become very large for instance 1,000 and 2,000 and 3,000 left degree. Then, the linear program works quit good, but if you want to get the best possible code for a left degree of 10 or 20 which is what is usually in practice. Then, differential evaluation works much better the approximation method does not work that good.

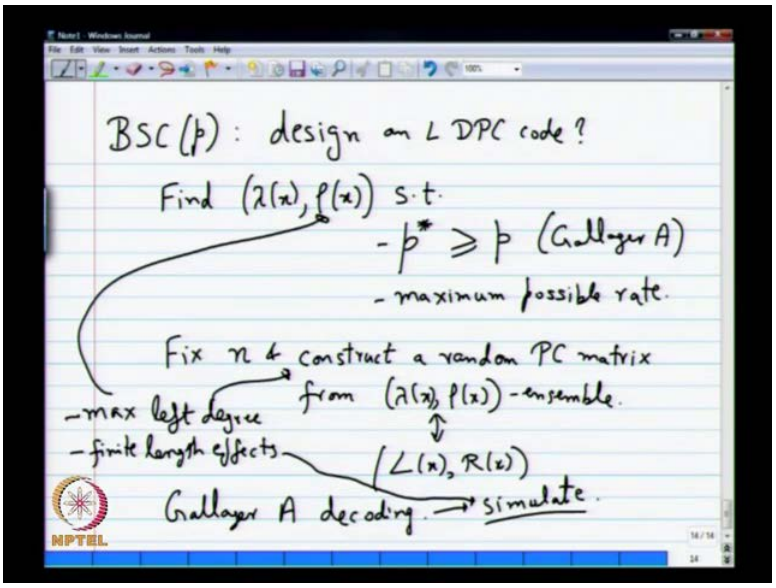
That is a good it is a good rule of come to remember, so that is it I mean there is not too much information in the that i told you about the optimization and it is just something that you have to try I mean if you have to do it. You have to try, but until recently Rudiger Orbonkey was maintaining website out of his home page called LDPC opt.

If you go and search for LDPC opt on google, you will hit at and he had a huge database of optimal degree distributions for different channels different decoders and different rates it was that, but as it turns out in the last few weeks it is been down. So, I don't know why I think I am going to come up again in the few more weeks or months or something it is not going to happen immediately, but eventually I think it is will come up. So, this comes up then you do not have to do all this programming, so optimization do not have to do to know the good best degree distribution you just simply go to the this web page.

You will get this this maintain the Orbonkey and his group, but they will hopefully fix it and get it upon running if it is not, then go into trouble. I do not know if somebody has is there is mirror database, then that will be great, but I do not know if they have, I am sorry something has happened, it can I continue.

Now, this is really dramatic we have lights back again, so that is why I want to stop about Gallager, so by now I think what you should have an impression of this some kind of rough idea of how to work that LDPC codes and practice. How do you work if you want to design an LDPC codes suppose your face to the channel you start with the channel, let us say BSC of p and the probability of error is given to you.

(Refer Slide Time: 19:43)



Channel transaction probability of error is given to you how will you design an LDPC code nobody says design an LDPC codes what will you do, what will be the first step regular or irregular. So, let us say I want the best possible code like some sense, so suppose say LDPC opt is available the database is available.

Somebody has a database of database of all process degree distributions the best degree distributions for a particular threshold or the best degree distribution for a particular rate as well. Then, what will you do, you go find lambda f x rho f x such that what two conditions need to be satisfied what the two conditions are. Threshold is greater than or equal to p because I know then code has a good chance of working and then what else is needed after maximize rate.

So, you guys if you delay if you have n is 1 million right your data has going to be may be data rate is gone to be 1 mbps or something. By the time you process and finest decoding 1 million some 10 million bits would have already come to you know you want to keep there somewhere I do not know where do you keep there all these problems are rise. So, people do not tolerate such long block lengths and practice maybe be your block length can be at most let say 5,000.

So, that is a reasonable number today 5,000 it is be reason may be 2,000 and 3,000 its reasonable somewhere on thousands may be is reasonable for you the what do what will happen is when actually stimulate. Suppose, your was let us say 0.04 and you thought

maybe I can use threshold which is 0.039 pick a code with threshold point no 0.041, let us say so little above slightly above, but your block length now is 2,000.

What will happen when you stimulate it will not be anywhere close to the performance that you want, you would not get 10 per minus 3. If per rate or something near the probability of error points 0.04, so then what you should do should back off little bit more in the threshold 0.01 is not enough, so you go to 0.05 or something depending on your block length. So, that process you have to iterate till all the approximations you made with the analysis make sense in practice. Today is the best of my knowledge, there is no theoretical way of doing it, so you have to do simulation. So, you have to do repeatedly simulations and refine your degree distribution choice.

So, another thing that will happen is when you have any n equals 2,000, you cannot have huge left degrees. So, if the rate is half n equals 2,000 means 1,000 is the maximum left degree. You can have you cannot have more than 1,000, but even if it is just 1,000 is even if it is even 100 with 2,000 or 1,000 by 2,000, matrix with column degrees with being 100, it is impossible to put it down.

So, that you do not have length four cycles, so in the constructions face also you will have trouble, so may not be able to let your left degree go very large, it can 2,000. It can be maybe 10 or 20 or something. So, depends on how good are your constructions methods, so all those constraints will come in practice maximum left degree. That is an important limit you have to look at, so maximum left degree you will look at and finite block length effects maximum left degree improves your construction finite block length f x of x of simulation.

Both these will affect your choice of lambdas of x, so you cannot just have these to constraints. You will have other practical constraints depending on how you have implemented the construction of the random parity check matrix from the ensemble and how stimulation works for finite both of these are very crucial. So, another thing some other things like this will play a big role in practice, so one more thing which is very crucial when you have degree 2 bit notes.

So, it turns out even though average over all possible degrees and we have a average formula for distribution and everything is great but in practice the degree two notes are very unrealizable if you have a degree 2 bit notes that is bit is much more likely to be an

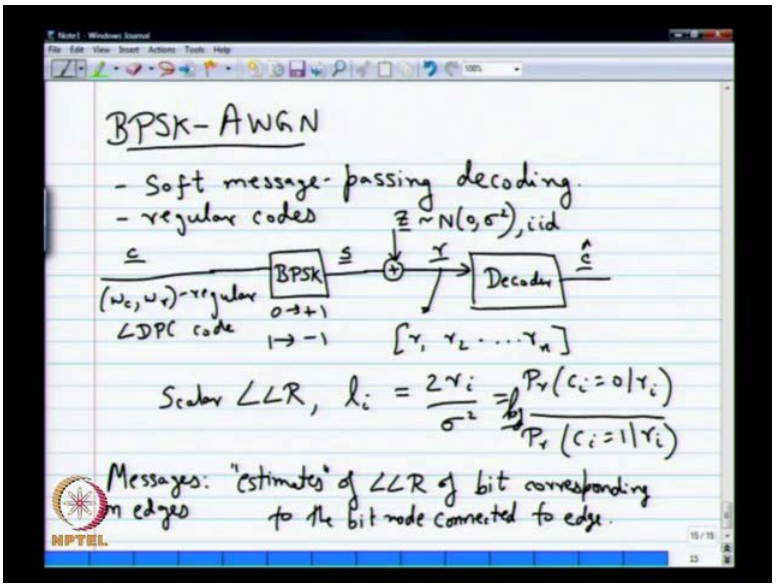
error when compare to other bit notes. So, that happen in practice all the time, but another thing that will happen all the time is unless you have degree 2 bit notes you would not have good threshold particular rate is to do not have degree 2 bit notes. You would not have thresholds, so these are interested, so these are interesting things that many people do not have a theoretical interpolation.

So, you just know this is two in practice nobody knows why it is has to be that. There are some reasons you can think about it is not clear, so there is like a balance you have to do how many degrees notes you want to put that is this is good thresholds, but then threshold is good. The bit note itself will have trouble in in correcting degree two only two check notes, so what people do is smart trick of making all degree w notes as parity bits.

So, you design your matrix such that you can make all the degree 2 bit notes as parity bits. So, what happens when you decode even the parity bits are in error, it does not matter violate anything else the message bits will be correct you something on your message part validate you will be fine. So, those are several that that is one of tricks, there are so many tricks that you have to do construction to make things work very nicely. That is all several papers are there such you will see such papers in past 15 years many methods are close. So, what we are going to do next is stop with BSC and Gallager and move on to UGMBSK and soft decoding.

So, what we will do next, so this is good time to ask me on general ideas on irregular codes and what happens in practice any questions anybody questions. So, I mean unless you program, this things you will not really learn all the tricks and this is sufficient to program this things you are not going to, but it is good to know this. I think it is an exciting an interesting problem, so let us finish with this and move on to BPSKAWGN.

(Refer Slide Time: 30:12)



Soft message passing decode, so many of the principles that we saw before will once again extend but the fact that it is soft makes things a little bit more complicated. Many of the things that are very simple in Gallager will not be so simple any more, but never the less the essential ideas will nicely extend without any problem.

So, you do not have to worry about it if you have a good feels for what is happening in Gallager you will also understand what is happening in the soft message passing decoding, but there are some intercross which will complicate matters little bit. So, for one we are passing just one bit back and forth. Now, what will be we passing, we passing technically real numbers or more than one bit definitely we have precision we will be passing lot of bits. So, analysis becomes a little bit more involved, other than that it is a its pretty much straight forward extension works in very much like a Gallager experiment.

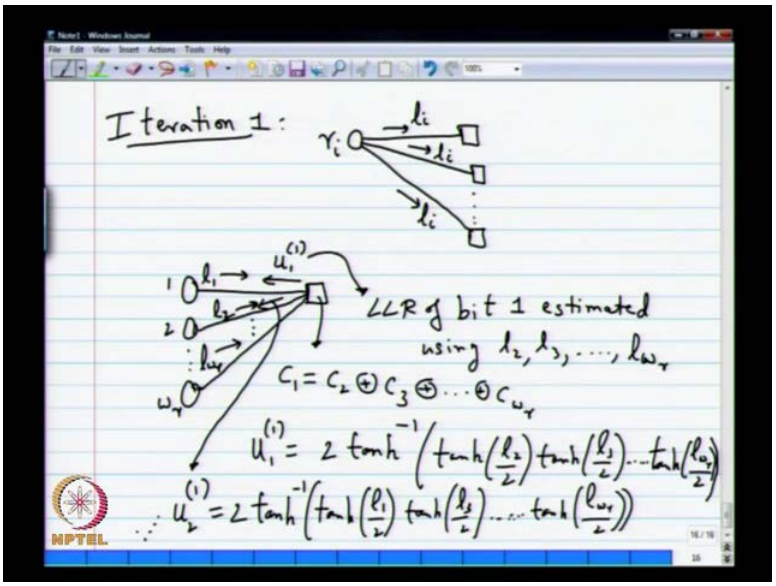
So, let us do the set up of the problem, so you have a code words c from and let us say so I will once again stick to regular codes to start with and then the extensions to irregular also will follow in a very obvious way. So, you have a code words c from AWCR regular LDPC code. So, this is first this goes through BPSK and BPSK as before will be 0 going to plus 1 and one going to minus 1 I have a symbol vector R which Gaussian noise which is distributed 0 mean sigma square IID gets added.

You get the received vector R and you want to run a decoder on this to produce say as recap this is the general idea. This R is going to be R 1 r 2 R n and what will be important or what is called soft information is usually the l l r or the scalar l l r small l sub which will be 2 r i by sigma square. This is the probability that c i equals 0 given r i divided by log, forgot the log.

So, this is the most important and critical change other then that the principles is the same you have some channel information that is received and loaded on to the bit nodes what is the channel information. Now, it is not just one bit it is the scalar l l r in the first iteration what will the bit notes do they will pass the scalar l l r to all the check notes that are connected to it. What will the check notes do they will receive l l r from all the bits that they are connected to and pass updated l l r's back.

You will also maintain the extrinsic nature of the messages how you will maintain an extrinsic nature the same things will not be used when the check note passes back. Then, what will you do at the bit node again, you will collect all the new l l r's that are coming in then use the extrinsic send back extrinsic to all these case. That is the essential principle, so let me write down how it works, I write down the description of the decoder and then we will think about analyzing it etc, so this is how it works and iteration 1.

(Refer Slide Time: 36:04)



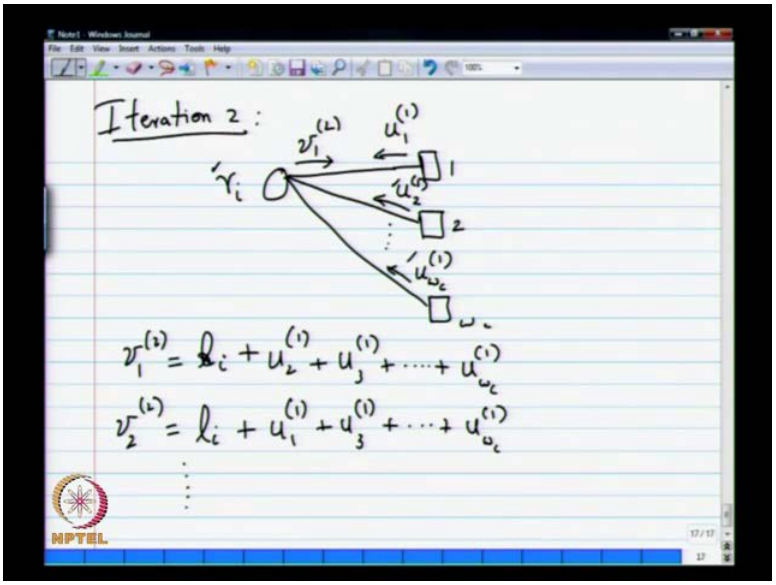
The bit node, let us say i th bit node which received r i will send, remember I am looking at regular LDPC codes, so how many check notes will I have w c check notes. So, this

will simply pass l i 2 all the check notes that are connected to it and what will the check notes do in the first iteration. So, let me write that down so it this check note would have received l 2 through l w r it will also have received l 1, but it would not use it and it has to compute.

Now, what I have been calling as u 1 just u 1, u 1 1, so what is this u 1 1 suppose to be so these are bits one to w r I am just numbering them as 1 to w r for convenience. So, this u 1 1 is going to be l l r of bit 1 estimated using l 2 l 3 so on till l we already know how to do this. I quickly write down the formula, you will see what I mean remember the bit node knows c 1 to be equal to what c 2 x or c 3 x or so on till CWR and the individual log likely hood ratios are known l 2 l 3 l w r. How will you compute the extensor log likely hood ratio for c 1, now that two tan hyperbolic formula is there?

So, usually you can compute u 1 1 to be 2 tan hyperbolic inverse is there inverse of tan hyperbolic l 2 by 2 tan hyperbolic l 3 by 2 so on till tan hyperbolic l w r, so that is the formula as simple as that what will I do for u 1 2 what will this be what is u 1 2. It is the message that is going in that edge u 2 1, I am sorry instead of l 2, you will put l 1 that is it. So, you will have two tan hyperbolic inverse tan hyperbolic l 1 by 2 tan hyperbolic l three by 2 so on till tan hyperbolic l w r by 2 that is it and so on. I do not have to write what the other messages are that is what will happen, so what is going to happen in iteration 2 that is quite important iteration 2.

(Refer Slide Time: 40:06)



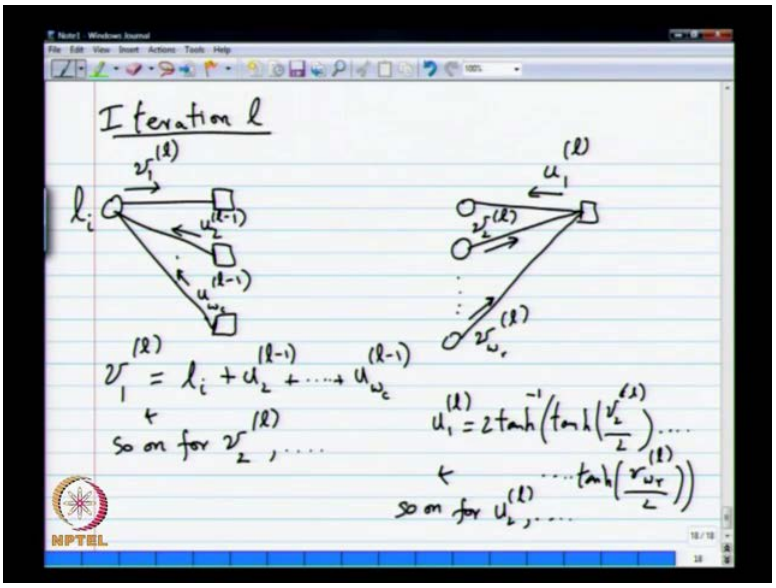
In a particular bit node, you have r i of course and then you now have prior information coming in from other check nodes. So, maybe I number this one to w c what comes in from here is u 1 2, again I am numbering for convenience its not to I would not be doing it consistently. Then, this also will be coming in u 1 1 and I want to send out v 2 1, how will I do this, so what is happening at the bit node bit node is received and estimate of the l l r from the channel which is l i.

Then, now its receiving w c other estimates from each of the check nodes that is connected to and as far as the bit node is concerned all these estimates are independent estimates. Now, based on all that the bit node knows it has to send back some estimate to the first check node which is extrinsic to that check node what all is extrinsic to that check node. This guy is extrinsic this guy is extrinsic this guy is extrinsic and all these are l l r's and they are all independent, so what can I do simply, add them all up and send it to bit node that is all.

So, you send v 1 2 to be l i of course l, I will always be there plus u 2 1 plus u 3 1 plus so on till u w c 1 what will you let v 1 v 2 to be same L i plus u 1 1 u 3 1 plus so on till u w 0 and so on. So, once you understood the principles of Gallager, this is also very similar so you do not you not doing anything different except that the updates are now l l r's. So, the formulas are will come from probability, once you know the assumption formulas will directly come but in Gallager we had one artificial way of creating this v 1 1.

So, square artificial I mean it was just some add hock kind of method everything agrees then you make it be otherwise you do not make it R. So, here it is a method that will come from probability itself, once you decide it is a l l r right there will be certain way to compute it. There is no other way to do it probabilities, once you assume independence probably calculations will tell you the answers it is a l l r. They all they all add and they go to their other row is that, now I can formalize both this iteration and write down iteration l.

(Refer Slide Time: 43:05)



There will be two steps, one from the bit node to the check nodes u 2 l minus 1 u w c l minus 1 and what goes out b 1 l may be this may be some I should say l i here because I am using l also for the iteration. Hopefully, it is not confusing to you and so on for other for similar for the other things. Then, what do you do at the check node v 2 l v w r l i want to send back u 1 l, how do you compute u 1 l two tan hyperbolic inverse v 2 l by 2 tan hyperbolic v l w r by 2 and so on for u 2 l etcetera.

So, that is it as simple as that and there are all efficient ways of implementing it which I have not gone into great details here for instance. So, you never do these multiplication actually, so what do you do is you can write this formula as c bring this two down here becomes u by 2. Then, take tan hyperbolic on both sides you get tan hyperbolic of u 1 l by 2 equals product of several tan hyperbolic. Then, you do logarithms on both sides, so you get log 10 hyperbolic of u 1 l equals some of log tan hyperbolic of all these things. It turns out log tan hyperbolic in absolute value you can always invert very easily, so taking log you know, so you have taken absolute value the inverse is itself.

So, you all you need is just one non-linear function which is log tan hyperbolic and then you can write u 1 l as f of summation of several f. So, there is very simple way of implementing this which I am not going into here, but it is possible to implement this in a very simple fraction same here with the left hand side. Also, you do not have to keep doing these summations several times you add up everything and then subtract one after

the other similar thing. You can do on the check node also like you did on the check node you can do on the bit node.

So, all of these can be implemented in a very nice and simple way, but essentially this is the idea, so this is soft message passing decoding in BBSKWGN and this is in fact the major selling points for LDPC codes. So, today the way technologies is you can get soft and information from the channel on every bit and you want to have a efficient way of using this soft information you saw that this MAP coders.

They are what like theoretically, but there is no easy way of efficiently implementing, so bottle neck is nicely solved by this kind of a decoder. You see it is easy to use the soft information, you iteratively improving it one integration after the other. You are using more and more code words from your parity check matrix and exhorts of code words from the parity check matrix in a way that is specified locally by the tanner graphs. So, you go on the computation graph and then that is how it is specified you do it smartly using that. Then, it works really good none of the operations are that complex and in fact in practice even if you do like 5 or 10 iterations 5 iterations even you get very good performance.

So, you get a good great coding in etcetera, so that is the soft message passing decoding that is it so as simple nothing more to do anybody can implement this. In fact, you can write like 35, 36 line mat lab program for that, you do not need I think more on that it is very easy to write code for simple letters where you will get good you will get good performance.

So, the next important step is density evolution right it is fine enough, I have a algorithm I can of course use regular codes for a particular designed rate. I can use 3, 6 if you want to rate half. I will get some performance, but once again you will notice that there will be a gap between the capacity of capacity at the particular rate and the threshold that are obtained by regular codes.

So, you have to do irregular codes, so when you do that you need a good density evolution now density evolution is much more complicated here, why is it complicated here? So, the messages passed are not bits anymore, so you cannot just say probability of a bit what should you track instead the PDF of the random variable which denotes the message.

So, it becomes more complicated because of that fact, so you are not tracking just one probability you are tracking an entire PDF, of course you cannot track the entire PDF in practice numerically. So, you will approximately you have been it you will track a long enough discrete PDF and that will give you a density evolution algorithm. So, we will stop here for today and we will see how to do density evolution in the next lecture.