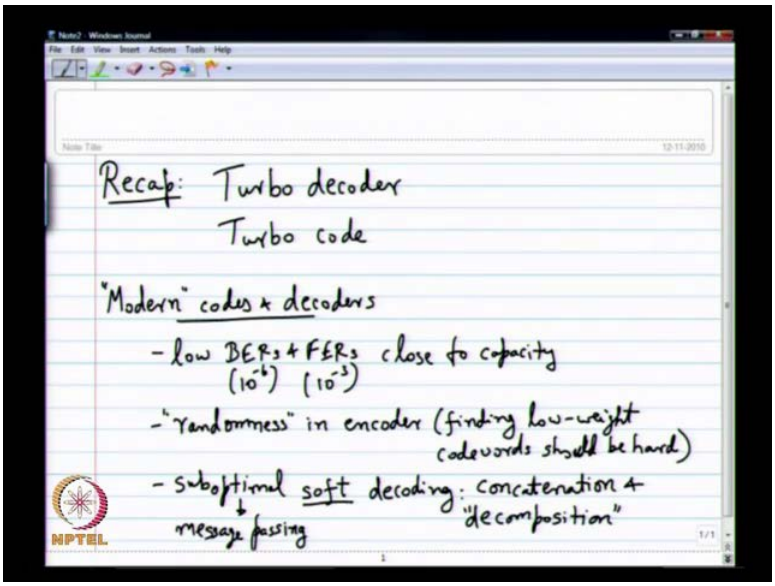
Coding Theory
Prof. Dr. Andrew Thangaraj
Department of Electronics and Communication Engineering
Indian Institute of Technology, Madras

Lecture - 38
Modern Codes

(Refer Slide Time: 00:20)



So, not going to do a big recap the small recap for previous lecture basically we looked at the turbo decoder, before that we saw the turbo code. The thing that I want to emphasize is some kind of generic rule about these modern codes, so called modern codes and decoders. They follow a certain generic construction principles. You cannot say that these are very precise from a theoretical point of view, but there are some generic principles, which are used in the design of these codes and the decoders. They go hand in hand and the goal or what these codes have managed to achieve, which was not possible before was low B E R S and F E R S closed capacity.

So, that is capacity in what channel I should tell you in what channel, in fact for almost any channel that you can think about, any reasonable practical channel people have been able to get very low B E R S and F E R S quite close to capacity. When I say low well low B E R is 10 power minus 6 or lower and low F E R is 10 power minus 3 and lower. So, that is what these codes have accomplished. What are the major guiding principles on the code design side? In fact the principles are very generic, I mean what is the main

idea, what are the main ideas any themes, it looks like. So, the idea of randomness in the encoder is quite important, some kind of randomness in encoder

This should basically make finding low weight code words difficult. So, these are this is are again like I said there is no major theoretical impact here, but just principles that you use rules of thumb. That you use in the design finding low weight code words should be hard. So, that is that should be there and there should be some randomness, which make sure that you do this. Then the other point of randomness is when you when you vary this vary the randomness here.

So, of course there is various possibilities, that is why it is called random. It is not one possibility you should be able to address a lot of codes. So, there should be many codes that are that belong to an ensemble. You should be working with one average member from the ensemble. So, that seems to be like a theme here, but then you cannot just make the entire code random.

If you make it, if you make everything random of course, the code will be good. Of course, finding low weight code words is hard. If you take a random parity check matrix finding low weight code weights is hard, we saw that in the beginning also. Given a parity check matrix it is hard to find low weight code words, but then what is the problem with those codes? Optimal decoders are tough. So, the main impact here is you can do sub optimal soft decoding, soft decoding must be possible the soft is important. How is this made possible in your encoder? This is through ideas of concatenation and decomposition.

So, the main question is why are these message passing decoders sub optimal? Well, the independence assumption is the crucial property, which is violated. So, if you do not violate the independence property at any point in your message passing, then there is no problem you will get to the optimal decoder. In fact you can prove that no problem, but point is you will violate the independence assumption.

So, there is an independence assumption when you do the decoding that is violated. So, sub optimus of decoding, so the encoder should facilitate the sub optimal soft decoding by making possible, by using either concatenation or some kind of decomposition is possible. So, in the L D P C codes for the instance there is there is some. So, let me put decomposition in codes.

So, what I mean by decomposition is there should be a part of the code word, which is decodable by an optimal soft decoder. There should be a part of the code word and that part can be anything, I mean as long as there is a sub code word, so to speak. It is a long code word, which you cannot directly optimally decode, but some part of it should be optimally decodable. Once you have that then you are enabling this decomposition in your encoder.

So, when you do that then you have to do message passing. So, this involves message passing and the important thing in message passing. You should pass only extrinsic information and you cannot make sure that it is extrinsic globally. I mean that was a question that was asked about the stage of their decoding. Also, you have to make sure that it is extrinsic locally for one component decoder in one part of it. What it received should not be send back to the same part in the other decoder.

So, you make sure that locally there is some extrinsic thing maintained globally you do not care. So, if you want to care about global extrinsic all that then you have to take care of the entire M A B decoding. Then you will be again stuck with the difficulty of optimal decoding, your problem will become very difficult. So, hopefully this local thing is clear to you.

So, what you do in your message passing and L D P C codes? For instance you make sure you pass extrinsic instance from each check node only, but if you do the ultimate tree the decoding tree, you remember decoding tree. There might be repetitions at some levels once you go deep enough there might be repetitions. You do not take care of those repetitions in the message. That is passed by some check node up on top, you do not worry about that. If you want to worry about all of that then you will be optimal, but then complexity is going to be huge. So, you give up there some optimality that is why your decoders are sub optimal, but then the great gain is they work. They work quite well, they give you low B E R S and F E R S close to capacity, the M L decoding, I think.

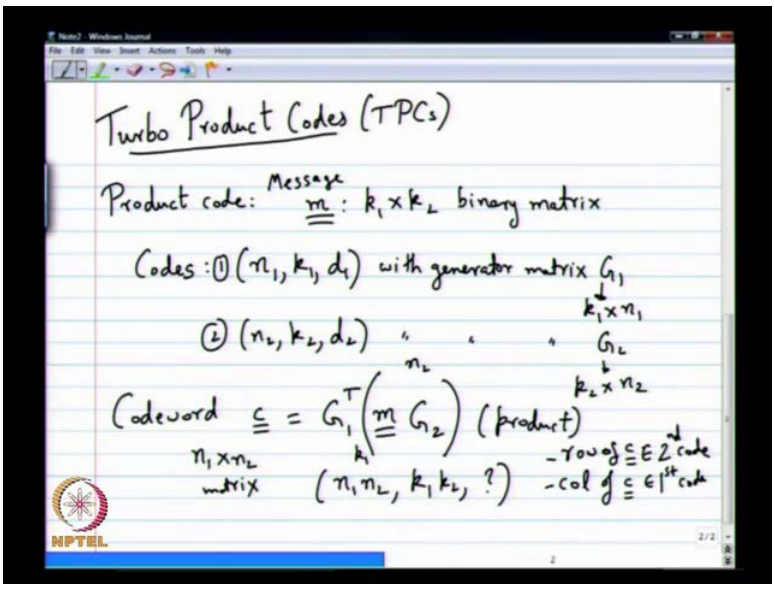
So, again those are things that people are trying to prove. There are various versions people have shown these results for L P D C. Even, turbo codes they have shown that if you can do M L D coding or M V P D coding, you can get to capacity with these codes, those things are again that. Such problems that people have shown. So, it is not totally unknown there are papers such results you would not find in text books, because these

are like ultra theoretical results. This involves a lot of calculations, it is not very interesting. Otherwise, what is interesting in practice is that 10 power minus 3 F E R, that you can get, that is what is interesting. Coding that you are doing if you are able to do an M L decoding optimal thing, then it will reach capacity?

So, any coding such that finding low weight code words is hard. You cannot do very bad codes also just make sure that you are not doing anything that is very easily shown to be bad. Then if you M L, M A P D coding, the rule of thumb, the general folk theorem, so to speak nobody has shown, but everybody believes in that you take a random code random looking code. Of course, for random codes they have shown, but some any design that you come up with. As long as finding low weight code weights is hard you do M L or M A P D coding, you will get to capacity. So, these are all I mean it is not real theorems, but this may be used to guide designers.

So, let me show you few more ideas here there are turbo and L D P C codes are not the only ideas that follows these generic principles. There are other codes that I will show you. The first thing that I am going to show you is what is known as a turbo product code product codes were always known, but this turbo product code is basically the turbo decoding of product codes. So, let me talk about that for a while, it is called turbo product code, so T P C is short.

(Refer Slide Time: 09:05)



So, the idea here is really interesting and these codes have nice structure also. They might be used some in some places, it's very interesting to study them closely. So, what is a product code? A product code is the following. So, it has this, it has the structure your message instead of being a one dimensional vector is now going to be… Now, I am going to put two bars to indicate that it is a k 1 cross k 2 matrix.

Let's say a binary matrix you can also do non binary, but it is not too interesting. So, the message is now k 1 by k 2 binary matrix, this is the message, then you take two codes. So, the code codes there are two of them one is a n 1 comma k 1 comma let us say d 1 code with generator matrix G 1. So, what will be the dimensions of the generator matrix k 1 cross n 1? And then the other code?

There are two of them like I said first code is this guy second code is n 2 comma k 2 comma d 2 with generator matrix G 2 this will be k 2 cross n. So, you have a matrix message it does not matter whether. It is a matrix or a vector, if k is large enough you put it into a matrix form. It is not a big deal bits you can think of it as a matrix or a vector or anything else you want.

So, we are not really deviating from the definition of codes just by going to two dimensions, but these using these two codes is interesting. How they form the code word is also interesting, you make a code word, which is again a matrix. It is going to be an n 1 cross n 2 matrix and how do you do that? You take G 1 transpose multiply with m then you multiply with G 2, is this a valid multiplication? He is happy G 1 transposes is going to be n 1 by k 1 m is k 1 by k 2 G 2 is k 2 by n. You will get a n 1 by n 2 matrix final.

So, you can now think of this is called the product construction. So, this is the product of the two codes. So, first of all dimension and block length is easy right? For the product code what is the dimension block length is of course, n 1 n 2 dimension is k 1 k 2? The interesting question, of course is what is the minimum distance first of all is this linear. You can show linearity quite easily if you have a message m 1 another message m 2. What will happen to m 1 plus m 2? The same as c 1 plus c 2. So, you can show linearity very easily, so clearly if this matrix, if you unwrap it and make it into a vector.

You have a linear code binary linear code n 1 n 2 comma k 1 k 2 comma some minimum distance. Question is what is this minimum distance? What do you think the answer will be? You have to think a little bit. So, remember you can think of this see matrix

multiplication is associative. So, you can pick any two first multiply it and then multiply by the other. So, let us look at this multiplication after you do this multiplication, what happens to every row?

So, this is a bad way of looking at it, so let me do it slightly differently. Let us say we look at this product. So, we look at this two together and then what will this be? This will be a n 1 cross k 2 matrix, n 1 cross k 2 matrix I am multiplying this m 1 cross k 2 matrix by G 2 on the right. So, after I do the multiplication what will be each row of c each row of c? Will be a code word of the second code, is that clear? So, each row of c finally, will be a code word of the second code.

Now, you can do the same thing, since matrix multiplication is associative you can group that side and c does a multiplication of G 1 transpose on the left. That will show you that each column of c is code words of the first code think about this carefully. If you have not seen this before, it is a little bit confusing. What am I doing first, what am I doing next etcetera.

So, it's quite easy to see, if it's whether you do G 1 transpose m first. Then multiply by G 2 or you do m G 2 first, then multiply by G 1 transpose you should get the same answer. That is because matrix multiplication is associative, it take you group it any which way you want you will get the same answer. So, am I doing matrix multiplication? Yeah, of course, I am doing multiplication. So, you do the first two first you get n 1 cross k 2 matrix. Then you multiply by G 2 on the right, what is multiplication on the right? Each row you take and multiply on with G 2 right, that is what you do. So, each row times G 2 you are getting.

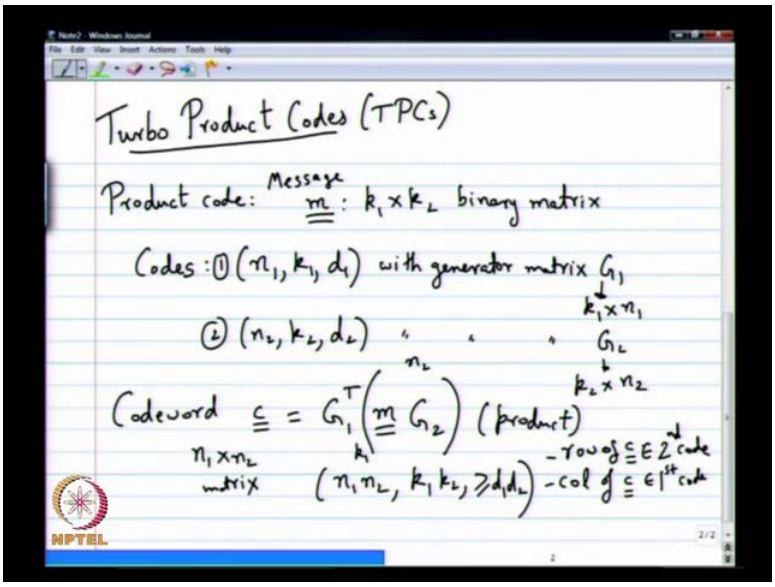
So, ultimately every row of c belongs to the row space of G 2. So, that will mean that every row of c is a code word of the second code, the code word generated by G 2, code generated by G 2. Now, you do the same thing a little bit differently. Instead of thinking of this multiplication this way you do the grouping, this way in which case you get what? k 1 cross n 2 matrix. Now, you are multiplying by G 1 transpose on the left, which means each column of this product m G 2 is multiplying the rows of G 1.

So, you get every column of c becomes a member of the row space of G 1 again. So, it is a bit of a twisted picture you have to think about it, you can do it in so many ways. If you like I am not saying this is the only way to do it pull the transpose out. So, that you get

the columns as rows then you will see it as this thing transpose times G 1. That is probably a better way of seeing it. If you want to see it that way also, basically the final result is that every row of c is a code word of first code. So, it belongs to the first code and every column of c belongs to the second code.

So, row belongs to the second code and the column belongs to the first code, that is fine right? So, if I think of a non zero code matrix c right in each row in there will be at least one row, which will be non zero. That row should have at least d 2 ones. Now, look at those d 2 columns and each of them are non zero code words of the first code, which means each of those columns should have at least d 1 once. So, overall any non zero code word of c will have at least d 1 times d 2 non zeros once. So, that is the idea I am coming up with the code word minimum distance. So, you can have a lower on the minimum distance, this is at least d 1 d 2.

(Refer Slide Time: 17:21)



So, that is the product code like I said the product code was known for a long time. It was not an unknown code, but notice from a modern point of view this is nice except for the randomness. So, G 1 and G 2 are like fixed codes may be you can live with that. It is not random, but then the decomposition that it gives you is very obvious. What is the decomposition every row belongs to the second code? Every column belongs to the first code. So, if you were to come up with a decoder using the modern principles of message

passing decoding, what would you do? What can you do? What are your component decoders? You look at it as rows and columns.

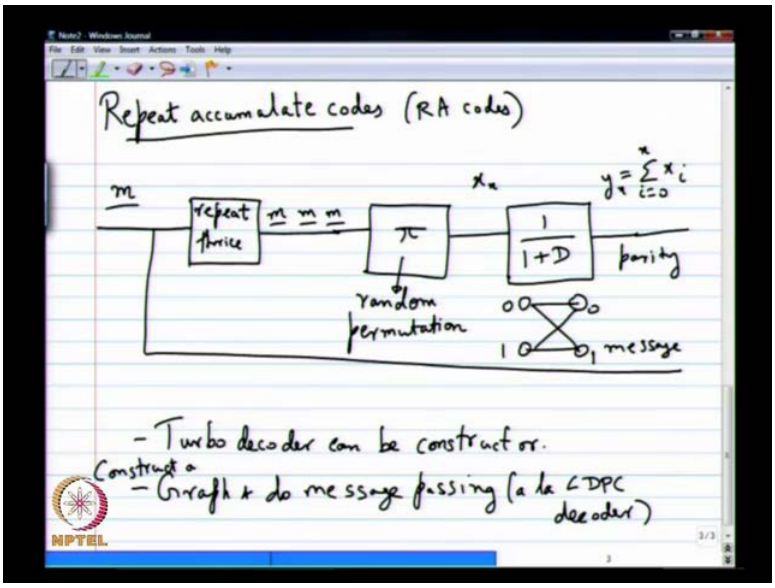
So, one thing you can do is you have a decoder for the second code. May be you have an optimal decoder for the second code, then what will you do? First, you decode all the rows, so after you transmit you receive the received values you put them in a matrix. Then you decode all the rows using your optimal decoder for the second code. Now, you have to gather some extrinsic information and give to the. So, another decoder, which is the column decoder column decoder is an optimal decoder for the first code.

So, how you do the extrinsic is not quite clear. Here, it is a little bit more confusing in fact when people do it they just do a scale down version you know. Its multiplied by some 0.72 reduce the positive feedbacks, so to speak control something. Then you give it to the second decode, the second decoder works on this decodes all the columns. Then the first decoder takes over.

So, you go back and forth and back and forth like this you hope to improve your performance. So, this is another idea is that okay? So, once again the only thing that is kind of missing here is the inter lever and all that. So, it is very common to pick G 1 and G 2 to be generators of hamming codes or extended hamming codes. So, you would take the 16 11, 16 11 is particularly common. So, you take 16 11 hamming code take a general matrix and then hamming codes you can easily decode.

You can also do soft decoding without too much effort. So, you can do that in fact there are also sub optimal soft decoding is possible etcetera. So, all these things that people use the turbo product rule consideration. So, this is one idea, in fact there are other ideas may other modifications to this, which I will skip. Then the next set of codes are what are called repeat accumulate codes, these are particularly simple again.

(Refer Slide Time: 20:00)



I think to me these codes represent the real power of this modern approach repeat accumulate codes or R A codes, the code the encoder is so simple. The decoder is once again, you uses these iterative ideas. So, you will see it is kind of a demonstration for this modern idea. How does, well this modern idea work, so repeat accumulate that is the word I mean that is what indicates this code. So, what are you going to do to your message? You are going to repeat your message bits.

It is a repetition code, which we know is not very good, then you are also going to accumulate. Then using our modern ideas between the repetition and the accumulate. You are going to some random permutation and that is it, that is all you do. Do you do you have a message m it goes through first repetition code? Let us say a 3 comma 1, so let me just do its very common to do three repetition.

So, I will say repeat each bit three times. So, each bit is repeated three times, so this is bit wise. So, when I say repeat thrice message itself is repeated three times. So, you get a code word here. So, may be let us m m m or some permutation of it you might do m 1, m 1, m 1, m 2, m 2, m 2 all these things I think it's something does not matter. Then you have a permutation and this is going to be a random permutation. Then in fact what you do is this accumulate, what is an accumulator? Accumulator is basically something, which does 1 by 1 plus d.

What is 1 by 1 plus d? If x n is the input y n is the output what will be 1 by 1 plus d. So, y n is going to be summation of x n x x i. So, let me do it carefully x i i equals let us say 0 to n. So, it accumulates this i by one plus d is short hand notation. All of you must know this we have done enough D S P to know this. So, this is the accumulator that is it that is your code, this is the repeat accumulate code. So, there is also a systematic version, where your message will also come through by itself.

So, this can be the parity and this is the message. There is also a non systematic way, where you do not send the message explicitly out you only send the parity. So, there is various ways of doing it once again two codes, which are very easy to decode optimally decode. The repetition code can be very easily optimally decoded, there is no problem. There you know complete exact decoders 1 by 1 plus d is a very simple. So, it is a two state, how will the look very easy to come up with a 1 by 1 plus d? It will look like this, so you can write down.

We will come to all that slowly, so it is that looks like this. You can put the transition 1 by 1 plus d is just a two state. So, you can do B C G R very easily, so you have optimal soft decoders for both these things. So, you can construct a turbo decoder you do a first you decode the 1 by 1 plus d, you get some extrinsic information. You do the permutation send it to the repetition encoder decoder, which again does some more improvement of your L L R S. Then you go back and forth back and forth you can keep doing. So, very standard turbo decoder can be considered, it does not add any accumulator is rate one kind of thing, well it does something, which you can decode.

So, I want to point out that is what I want to point out here. So, that is the for instance question he asked is very interesting because 1 by 1 plus d is not doing anything, I mean rate is one, whatever you put in you are say getting. So, but what it gives you is some way of getting soft information an and it adds certain other dimension in the dependency. So, you have repetition, which is very local you know, which is local. You know everything is repeated and this accumulation somehow connects everything together. So, you are able to pass your soft information all around. You know it goes through all around goes through and slowly trickles down from here and there like economics. Ultimately, the poorest guy also benefits that is the idea.

Nobody knows exactly how it is a random permutation in the middle doing, its magic also it works. That shows you the way the these modern codes work, you know what I mean? Just do not pay attention to the actual code, pay attention to your decoder. Make sure that you cannot come up with a very simple low weight code word. So, here also even I mean here coming up with a low weight code word is a little bit non trivial, but it is not too hard. You can come up with a low weight code word, can you? How will you come up with a low weight code word? Let us say that you know the random permutation. You cannot do that see the accumulator is going to make any weight one

See, if you put just one it is A I I R filter, its going to put a lot of ones out. So, you have to be careful. So, its a little bit tricky you have to do it, you might be able to get there. Even, the random permutation it is not too difficult, I am not saying it is difficult, but it is not obvious, that it's plenty. Also, you cannot just come up with low weight code words very easily. So, hence it follows the principle and the turbo code turbo decoder can be constructed. In fact what you can also do is you can construct a graph and do message passing also.

So, you can construct a graph and do message passing, so I have not spoken about this in the class, but it is kind of interesting to think about how you might want to do it, but it is something I want to point out here. So, do message passing this is basically like the L D P C decoder. So, without B C J R that is what I am saying, so one way is to do B C J R. Another way is to you can even do like you can construct a graph, it is easy. In this case we are not doing the doing it explicitly. Here, you can construct a graph and then do local message passing kind of thing. Then that also gives you good for the repeat accumulate coder. Then there are all kinds of variations here, so this is repeating everything thrice.

You might want to repeat something more than the others, that is called irregular repeat accumulate, so you might want to do that. So, what is the distribution with which you do that, which ones do you get repeat twice? How many do you repeat twice? How many do you repeat thrice? How many? So, you can optimize over that you know optimize that and get good irregular repeat accumulate codes, all these things can be tried.

I mean all these variations will give you better and better result. In fact I think I believe people have shown that these codes can achieve capacity on these binary ratio channel,

all even in a W G N, maybe they get to close to capacity, I am not very sure, but not A W G N, but on a ratio channel I believe people have shown that. This construction achieves capacity, if I am not wrong I have to recollect this anyway.

So, there are some optimality results capacity approaching results that people have shown for these codes pass through the accumulator again, because it is only the permutation, which we are sending through the accumulator. What do you mean permutation is sending through? It bits are going through the accumulator permutation does not go through accumulator thing. So, where does the turbo decoding come into it? The repetition code also will do, I mean you will have a block, which decodes the repetition code. That will not be A D C J R, but it will be something simple, but near other block.

No, no definitely not you can use the turbo principles. So, you will have one decoder, which decodes the repetition code and gives out soft output. You have to figure out how to use extrinsic input and the repetition coder. That is very trivial you can just do that without any problems. So, you figure that out and then you have extrinsic input subtract it outs and the extrinsic output. That is what I mean when I say turbo decoder they use the turbo principle in the decoding.

So, you have one B C J R the other will not be a B C J R something other, but then you use the extrinsic subtraction principle. Then do it the reason I called it turbo is its more or less like a one shot decoding I mean using the once. Then you are coming back here and going back that is the idea. So, you can also do something else that is why I said you can construct a graph and do message passing and that also works quite well here. So, I think in this class we have seen both the traditional codes, which culminated in the Reed Salmon codes and in fact convolution codes are also traditional codes. There are interesting differences between the traditional approach.

These modern approaches one thing that these modern approaches, do not have is clear characterization of minimum distances. It’s only a rough characterization, but the advantage that they have is they allow sub optimal soft decoding, which works very well. So, the holy grail, so to speak is to have a perfect union between the two.

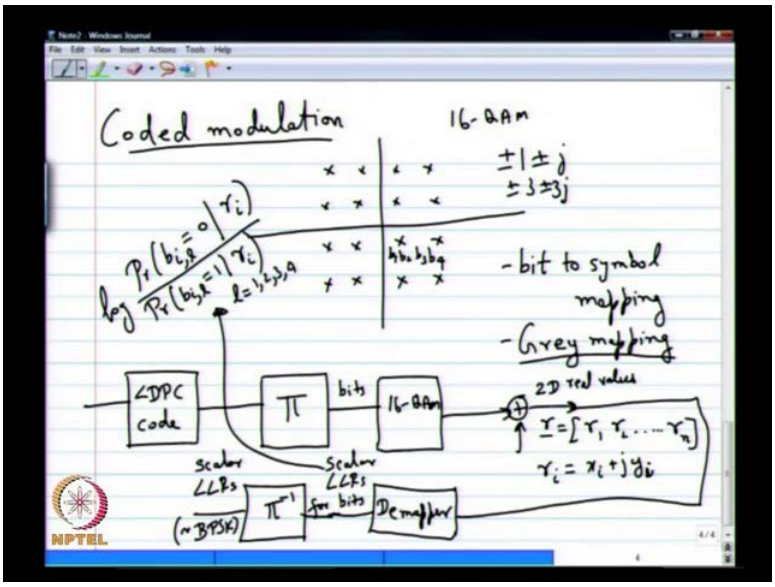
So, you come up with the construction, which has guaranteed minimum distance, which also in a way grows with block length. When you increase block length in your

construction, the minimum distance has to grow may be linearly may be sub polynomial. I mean sub linear polynomial like n power point 2 or n power 0.5 or something root n or something like that. It has to grow with n it cannot be log n, it cannot go to log once it goes to log it turns out. Whatever you do with the decoding will get killed.

So, it has to grow with n can you have that, if you can have that then you should still be able to do optimal soft decoding. You should that construction not be like very arbitrary random construction, where you cannot do soft decoding, you can do soft decoding also. If you also can show minimum distance, then you have made a union of the two. Then you have a nice new code for which you might get awards, who knows.

So, those are those are interesting problems today. So, let us leave this here and we have about twenty minutes. The last item that I want to talk about is higher order modulations. We have been always talking about B P S K and what do you do if you have 16 Q A M 64 Q A M. These are being used today increasingly in many communication systems and you should know how the architecture looks. So, the typical way of doing this is basically the title is coded modulation.

(Refer Slide Time: 31:18)



So, when you say coded modulation well B P S K is also a modulation but people do not consider B P S K as coded modulation. They will say it is just obvious. So, when you say coded modulation it's implicit that you are going to at least 16 Q A M. Something like

that eight symbols must be there in your modulation aspect. Only then it becomes non trivial, otherwise it's very trivial.

So, there are several things here, the first thing is if you look at 16 Q A M you are going to have, let us say the regular Q A M. So, basically this will be 1 plus or minus 1 plus j and then plus or minus 3 plus or minus 3 j this is 16 Q A M. So, the first thing is that you have to worry about is bit to symbol mapping. So, how many different ways you can map symbols to bits here?

This we knew 16 factorial, 16 factorial it's a huge number, you are not going to exhaust all of them. So, the you have to use some thumb rules to design something that is nice. One of the most common things that is used is what is called grey mapping Grey g r e y or e y. So, grey mapping, so here you make sure that adjacent symbols differ only in one bit. So, you can come up with the mapping like that for this.

So, that is most common you can also use other mappings. Nothing stops you from doing that, but you have to show that the other mapping, that you are using is better than Grey mapping, usually it is not. So, let us say we fix Grey mapping it is not a bad deal. I do not know, if can anyone come up with a grey mapping very quick you know how to do that. So, you know how to do that, you put for four at a time and then you add the remaining to four and this particular direction, it can be done very easily. So, you can come up with a grey mapping. So, now how do you use this with the code? So, what is done usually is this is what is usually done. Today and in many standards people do this.

You have an outer code, let us say an L D P C code you can also have a turbo code basically not an outer code. You have an L D P C code, then what do you what people would do is you might want to do a permutation here. So, I will tell you why this permutation we need it you might want to do it may not want to do it. There is some reasons why the permutations might be needed. Then after that you put it into your Grey mapping for 16 Q A M.

So, this is how your encoder looks and then noise gets added. Now, you will have 2 D noise. You are you are putting out complex valued symbols will have complex Gaussian distributions. Let us say two dimensional independent normal distribution each of them with some sigma squared adds up to some 2 sigma squared etcetera. So, you will get complex valued things.

So, basically you will get 2 D real values here. So, your received vector will be basically r equals r 1, r 2 so on till r n. Each r i will be x i plus j y i, now what you need to do is come up with what is called a 16 Q A M d mapper. This d mapper has to be a soft output d mapper. What you need here? See here you have bits, here you need L L R S for bits that is it. Once you have L L R S for bits it does not matter what modulation you are doing.

You can easily separate the modulation part of things with the coding part of this, because what is input to the L D P C code is L L R what is input to the turbo code. It is kind of like L L R . So, L L R is what you input into the thing, but you think you might, they might be like branch matrix, that you have to compute. So, there is a way to incorporate that in the turbo codes. So, that is why I did not go into the turbo code stuff, but usually L D P C codes its quite straight forward to see. What you have to do the L L R S can also be converted into the branch matrix and turbo codes it is not too hard. So, what you imagine sometimes is that this L L R S are actually coming from a B P S K modulation.

Even though it actually went through 16 Q A M and came back, because you are doing permutation and its all looking random. So, what people usually assume is that what comes out here is L L R S. So, you assume that it is approximately from B P S K pi inverse, maybe pi was such that pi squared was 1, these will be scalar L L R. So, these will be scalar, this is all scalar L L R S. So, I will do a quick discussion on how to compute the scalar L L R. It’s quite easy when I say scalar what I mean is each symbol here has 4 bits, b 1 b 2 b 3 b 4

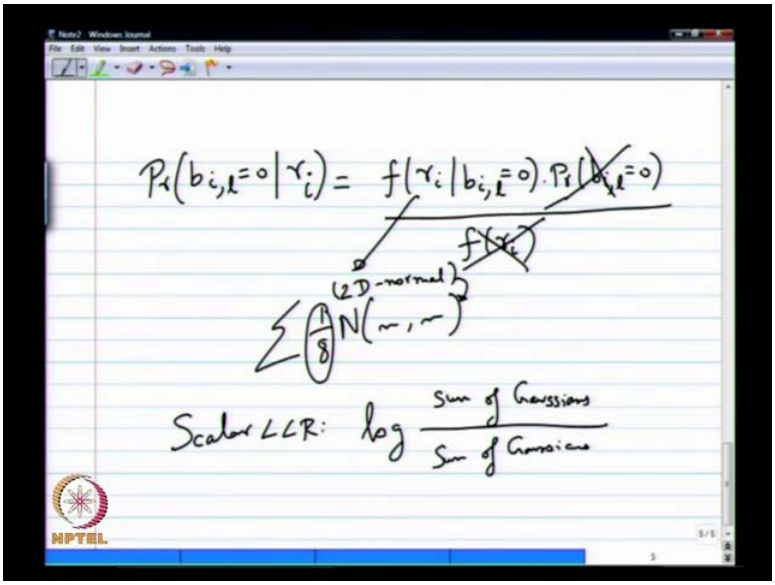
So, this scalar L L R will be what? Will this be this? Will basically be if you look at the i th bit log probability that b i n o b b l equals n o. Let me do that symbol care carefully let us say b i l equals 0 given r l divided by probability that b i l equals 1 given r i. So, that what I mean it is given only that l th no r i. So, l will go from l is 1, 2, 3, 4, so what you do is once you get this L L R S you think of them as roughly coming from B P S K. You do not worry about the modulation, after that I have got my L L R S think of it, as 2 by sigma square into something you know. You assume that is what is happened and then you decode.

So, in fact most decoders like s z r either or max log map what they do not need the sigma squared. So, they only need some number, which tell you a certain confidence about your bit. So, if that number is large it means large and positive, it means that bit was zero.

If it's large and negative, it means the bit was one, that is all you need. Then say the decoder you use it and then you can improve your iterations. So, this is the typical way particularly if you do Grey mapping this is more than good enough. You can simply treat it as like B P S K, because I mean adjacent symbols are vary only in one bit. You do not have to worry about things far away, everything depends only on what is next to you Then you get a B P S K kind of situation, so this is a common approach. Once you get the scalar L L R, so treat it like it is from B P S K and then run your decoder, just like before.

So, the only non trivial step is to compute this de mapping scalar L L R how will you go about computing it? You have to use Bayes rule. So, let me show you how that is done. So, probability that b i l equals 0 given r l is going to be r i it is going to be f of r i given b i l equals 0 times probability. That b i l equals 0 divided by f of r i, is that right?

(Refer Slide Time: 39:19)



Now, when I am going to divide and take ratios etcetera. What is going to happen? This is going to go away and this is going to go away, I do not have to worry about these things. Only thing I have to worry about is f of r i given b i l equals 0.

How do I compute this f of r i given b i l equals 0. So, you go back and look at your constellation given that b i l i is quite irrelevant, for every you do the same thing. So, given that b 1 is 0 how many points are possible in these constellation 8 points? That is totally 16 points you fix any one bit there will be 8 different points. So, given only that b 1 is 0 you will actually have a two dimensional mixture Gaussian, with eight different Gaussians adding up. So, that will be the distribution, so you plug in the received value into that distribution you get that answer.

So, once you find the P D F, what is this P D F? This P D F is going to be summation. Let us say 1 by 8, in fact this 1 by 8 is quite irrelevant, because when you divide it is also going to go away, normal with mean suitable mean. Then some variance this will be a 2 D normal 2 D normal distribution, this is what it will give.

So, you know exactly what the formula is you plug it into the formula you are going to get the value that is it. For B P S K its very simple, because there is only one Gaussian. Actually, it will be a Gaussian here you have multiple Gaussians shifted means adding together. It is just a more messy expression, but theoretically its nothing. This 1 by 8 you do not have to worry about, because anyway when you do L L R the denominator also it will come and it will cancel. So, ultimately for the scalar L L R will look like log sum of Gaussians by sum of Gaussians. So, what is Gaussian now? So, e power something. So, log of e power x plus e power y plus e power z so on. So, again you can use your max log map approximation whenever you have log of e power x plus e power y.

If x is greater than y its going to be much closer to x than y. So, e power x is exponential function if x is greater e power x is going to be much larger than e power y, you can kind of approximate it and drop it. So, you do that and that is also very commonly done in practice and implementations. People will simply look at the in fact they look at the received value and see the closest point in the constellation with a particular point being 0. You look at that distance they simply use that distance instead of numerator then from that you subtract the other distance of closest point, where that bit is one. That is what most people do in practice that is the max log map approximation to this expression, but you can also do the exact expression. It’s not a big deal, do that you get your answer.

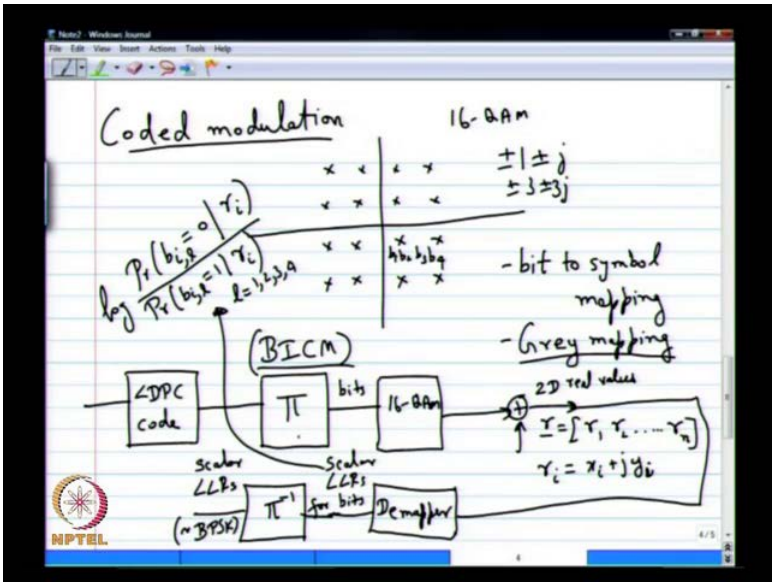
So, this is you know not too much detail, but that is what is done. So, in fact you can you can imagine in grey mapping this is not. So, what I am going to say next is not so useful,

but if you have other mappings this structure is reminiscent of what you can do in turbo. Remember, the 16 Q A M of course, it does not have any redundancy, but it gives you a way of computing scalar L L R some soft output its able to compute, in fact you can use the priors.

So, here I have set the priors to be to cancel out, but if you are getting some extrinsic you can use that in the priors here. So, in fact in some mapping in grey mapping it is not very good, but in some other mapping you have to do iterations between your L D P C decoder and the de mapper, to get best performance. So, you get the de mapper out assuming equal distribution for all the other bits.

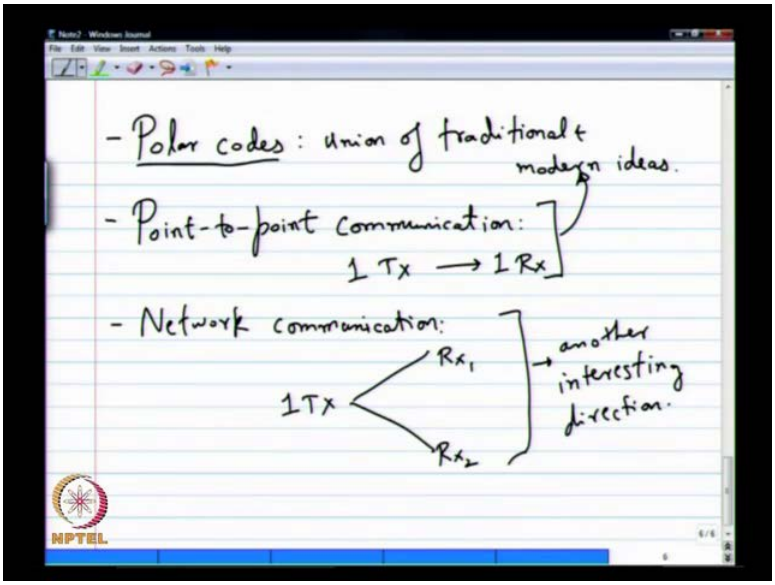
Come to the L D P C code, it gives you some information you pass extrinsic back to de mapper, use that as extrinsic input here. So, you are going to use it an extrinsic input here. Then you go back to the L D P C code and you can do the iterations. So, that will give you some improvement, if you are not doing grey mapping turns out in grey mapping, one can show the gains are very minimal, if you do iterations. So, people do not do it, but a non grey it can be can give you good performance, but Grey mapping is usually the best you know. It gives you pretty good F E R and B E R S, so we will settle for that. This for instance is called B I C M block interleaved coded modulation and this is common commonly used.

(Refer Slide Time: 44:39)



In fact if your L D P C code does not have a regular structure in decrease, all that people do not even do the permutation, but if your degree structure for L D P C codes is in a sequence. It is good to do interleaving to mix it up a little. So, we are down to the last five minutes of the course, so in the last five minutes I am going to talk a little bit more. Everybody is why is he why does not he stop now, I can hear you do not worry about it. The only thing I am going to say is a few things, which are of interest is a very recent ideas, which we will not have time to discuss in detail, but something's that I think are interesting. One idea called polar codes, it's very interesting.

(Refer Slide Time: 45:36)



So, these are in my opinion some kind of a union of traditional and modern ideas. So, you have good minimum distance minimum distance will be roughly like root n. Something it is based on what is called a Reed Muller code. It is an interesting construction and the decoding is quite modern. These for instance you can show explicitly that they will approach capacity for L D P C and other codes.

It is difficult to show that it is a bit painful to show those things these codes under very implementable decoders, will achieve capacity. You can show that they achieve capacity any rate below capacity is achievable. So, it's very good this these are being studied very actively today in research polar codes. Something did not have time to look at, but it is interesting idea another way in which coding is progressing is basically.

So, today many people think that the point to point coding problem is reasonably well solved understood and done, so to speak. Point to point what is point to point communication? You have one transmitter one transmitter to one receiver. Of course, now what is non trivial today is network communication. So, almost all communication that you do today is network cellular network is an example. Wireless networks around your campus is an example, many communications are network communications. So, in network communication there are some quite a few non trivial coding problems.

You do not have to go so far as the cellular network, look at very complicated scenarios. You can look at very simple scenarios like for instance one transmitter talking to two receivers. So, problems like this are called multi terminal problems or network problems. So, doing codes for these kind of problems is a little bit more complicated. This is also another direction in which codes are evolving today. So, another interesting challenge is problems like this, this is a simplest example, but of course there are more complicated examples. Something called the interference channel, which is a little bit more similar to the cell phone etcetera. This is another interesting direction in which you can move.

So, in point to point communication what is left is making a perfect union of these traditional and modern ideas. Once you come up with a nice code, which has good minimum distance. For which you can provably show that it achieves capacity pretty much the point to point problem will end. Only problems that are left are things like wireless, where you do not know the channel ahead of time, some complications in that direction are still left. Maybe I S I is still a little bit open, but mostly point to point communication people know what to do one.

When you have network communication like three transmitters three receivers and they are all interfering. Each of them want to talk to each other that is called an interference network, its being hotly studied today, how to do coding, how to do in fact how to even approach coding is not very well known. So, people are looking at that also. These are some just one slide the last five minutes, what might come in the future. So, that is it, that is the end of the course.