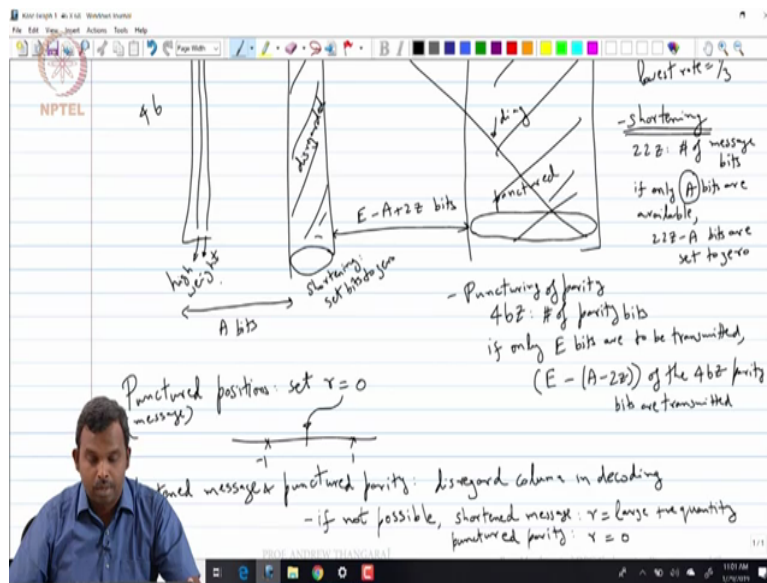


**LDPC and Polar codes in 5G Standard**  
**Professor Andrew Thangaraj**  
**Department of Electrical Engineering**  
**Indian Institute of Technology, Madras**  
**Rate Matching in LDPC Codes using Puncturing and Shortening**

Welcome to this next lecture on LDPC codes, we will talk primarily about rate matching in this lecture, ok. So I am going to describe rate matching a little bit first and then show how it is implemented in the LDPC code here, ok.

(Refer Slide Time: 00:36)



So how is rate matching done, so like I said let us take the base graph 1, this if you remember is a 46 by 68 matrix, it has these forms, so it has let me just draw a small visualization for you, 46 by 68, ok there are this first 22 columns, block columns which correspond to the message, ok so maybe I will put that somewhere here, this is 22 block columns, ok and then the remaining of how many is remaining after this 46 are parity, ok and if you remember there is the stop 4 by 4 which is dual diagonal, ok and then you have the diagonal, right so ok so this one right a little bit of so draw like this, this one is the diagonal part, this is double diagonal, you remember this when I spoke about the structure and the encoding.

So the first two columns, the first and the second block columns out of this you will see these are high weight columns, so high weight meaning there are lot of entries which are not minus 1, ok so in rate matching the first thing that is always done is these two message blocks are punctured, ok. So these two message blocks remember each block corresponds to Z this, the

expansion factor, ok so this is two times  $Z$  bits, ok this is two times  $Z$  bits of message and they will never be transmitted, ok.

So they are always not transmitted, so the number of transmitted blocks, number of message blocks equals 22, number of transmitted blocks equals to 66, ok why 68 minus 2 only 66 are transmitted 2 of the block are message blocks but you do not transmit them but you expect to recover them at the decoder, ok so on the because you do not transmit it does not mean that the decoder should not the decoded, decoder needs to (exp) recover it at the receiver, that is very important, ok so this 22 as the is the actual message blocks and 66 is the transmitted blocks.

So the base rate, the lowest rate is 1 by 3, right so 22 by 66, if you transmit all the 66 blocks, ok. So but usually what happens is further rate matching will be done, ok. So there are multiple ways in which you can do rate matching one is shortening, ok. So let us say  $Z$  is expansion factor 22 is the message blocks, so 22 times  $Z$  is the number of message bits that you can send, ok but in your actual transmission you may not have 22 times  $Z$  bits, ok so you may have only  $A$  bits, if only  $A$  bits are available, ok 22  $Z$  minus  $A$  bits are shortened, are set to zero, so this is the shortening process, alright 22  $Z$  minus  $A$  bits are set to zero.

So now the standard will pick  $Z$  a bit wisely, ok so  $A$  will be the starting point in the standard you will know how many bits you want to send and that in the base graph and the value of  $Z$  will be carefully chosen, so that this 22  $Z$  minus  $A$  is not too large, ok so you do not end up shortening to many bits in your parity check matrix. So there is a logic for how the choice of  $Z$  and the base graph are done etcetera, this is clearly specified in the standard I do not want to go great in details here but 22  $Z$  minus  $A$  bits are set to zero, ok.

So you may have a few shortened positions here, set bits to zero, ok this messages are set to zero, after that you do the encoding, ok so you have the message bits, now all the message bits are populated remember even the punctured bits are actually populated there use to in the calculation of parity except that you do not transmit those bits, ok so you in the encoder you fully use this.

So now the number of parity bits you produce is 46 times  $Z$ , ok so there is further puncturing involved here, puncturing of parity, ok 46  $Z$  is the number of parity bits produce at the encoder, ok but you will not be transmitting all the 46  $Z$  bits, ok so if only  $E$  bits need to be

transmitted, ok you will puncture remember there are  $A$  bits of message out of which two times  $Z$  you already punctured, ok.

So  $E$  will have to be  $A$  minus, ok I am sorry I will right in other way  $E$  bits to be transmitted, so  $E$  minus  $A$  minus  $2Z$ , ok right of the  $2Z$  parity bits are transmitted, ok. So remember once again this is this whole thing is  $A$  bits, ok remember after expansion that is, ok and then her you will send out maybe I should write it down a little bit here, you will send out how many parity bits  $E$  minus  $A$  plus  $2Z$  bits, ok.

So that how you will send out total of  $E$  bits, ok. So all these guys are punctured, these parities are not sent, ok, so this is how rate matching is implemented on the LDPC codes. So several puncturing and shortening, there are two types of puncturing message bit puncturing and parity bit puncturing, message bit puncturing comes on the left most side, the two blocks are always puncture they never transmitted, ok and then there is some message bit shortening, ok so if you do not have enough bits you picked your choice  $Z$  an a from only a finite number of possibilities, so the  $A$  will be slightly smaller than  $2Z$  in most cases and then you set those bits are zero and then you start sending parity is you send as many parity as you need, so that the total number of bits you sent is  $E$ , ok.

So remember  $E$  is the total number of bits that you are sending, ok. So the actual block length that you will be decoding will be  $E$  plus  $2Z$  right, So  $E$  bits you send is that right, so  $E$  bits that you send  $A$  bit is the total number of messages and then  $2Z$  you punctures, so  $A$  minus  $2Z$  is the actual message bits that you will be sending, ok and then  $E$  minus  $A$  plus  $2Z$  parity bits you will be sending together you will be sending  $E$  bits, ok so you send  $E$  bits,  $A$  bits was the message the actual rate is  $A$  by  $E$  if you want to calculate all these things carefully but remember there is some functioning going on, ok so something to keep in mind, ok.

So this is how rate matching is done , one can you can see how to modify the code to make this happens, so it is quite easy to modify the code, so for an instance you do not send the first two bits you simply do not send it, what do you do for the received values, ok supposing you do not send the bits, ok how do you substitute the received values, in the received values you put those values as zero, ok  $r$  equals zero, ok for punctured positions set  $r$  equals zero, ok.

So remember this is BPSK, BPSK is minus 1 and plus 1 is transmitted, ok so  $r$  equal zero is in the middle, so for the punctured position you simply putting  $r$  equal zero, so it is in the middle and you are not using, you are not committing to either side, ok particularly for the

punctured message positions are equal zero is quite crucial. Now the punctured parity on the right side, right the on the diagonal part and the shortening actually what you should do is you should disregard that part of the parity check matrix, this part and the punctured part of the, ok I do not draw this correctly the punctured part of the parity check matrix are to be disregarded, ok so this have to be disregarded in the decoding, ok those parts should not be active, ok.

So depending on how you implement the disregarding might be a bit difficult may be the disregarding of the punctured part, parity part will be easy disregarding of the shortening will be a pain, so in case disregarding is a problem, ok so for shortened positions and punctured parity punctured message positions, punctured parity disregard column in decoding, ok so this is the right thing to do, if disregarding some columns and rows is not possible particularly within a particular block then a good solution is as follows, ok.

So for shortened message you can set received value equals large positive quantity and for punctured parity you can set  $r$  equal zero, ok so want empathize this once again so that it might happen that within a certain block you are transmitting some bits and either shortening some message bits or in the parity block you are puncturing some of the parity bits, ok. So in those cases if within a block of few bits, few columns or rows you cannot disregard in the decoding, nice short hand for a shortcut to simplify the received value to be a large positive quantity for the short hand message bits and set the received value to be zero for the punctured parity, so this will not have effect on the rest of the decode, ok.



So if you dealing with the real numbers I could say something like 20 is large positive value but usually you are going to be quantizing and since you are used quantized values you take  $r$  to be the largest positive value in here a quantized version, so if you are using say 6 bit quantization minus 31 to minus 32 to plus 31 you set  $r$  to be plus 31, ok. So that is one another way of dealing with the punctured parity and shorten message in case you cannot disregard the whole thing, ok.

So there is an why disregarding whole thing may be slightly difficult as there are blocks, you know so within a block maybe a part of it is being transmitted and part of it is being punctured, ok then your coding will becomes too problematic if you want to disregard those kind of columns, so block wise you may want to set the values like this and the whole block may be disregarded otherwise, ok.

So this is the this is rate matching and the so what I am going to do is to illustrate what happens in rate matching, maybe I will show you for one or two candidate possibilities with this E and A, how to do it. The range of values of A and E and the choice of Z are tight down in a strict way in the standard, ok. So it is sort of I do not want to go into that part of the standard in this course, I do not want to talk about how A and E and Z are decided, so I will just pick some value of A and E, how this may not be valid values as per the standard but I will just pick some value of A, some value of E and then show you how this can be done, ok.

So I will pick it in a simple way, so that we do not have some shortening problems here and there, so that something you can do the adjustment for, so this is the essential idea behind puncturing and shortening, so you have to actually program it and then set to your received values suitably depending on how the things happen, ok. So we will not go into great details on to it matching I will just simply show you, point out in the code where are the changes that you might have to make for this purpose, ok.

(Refer Slide Time: 14:45)

```

13- n = nb*2; %number of codeword bits
14
15- Rate = k/n;
16- ERM0 = 10*(ERMod/10);
17- sigma = sqrt(1/(2*Rate*ERM0));
18
19- Nbiterrs = 0; Nblkerrs = 0; Nblocks = 100;
20- for i = 1: Nblocks
21-     msg = randi([0 1],1,k); %generate random k-bit message
22-     msg = zeros(1,k); %all-zero message
23-     %Encoding
24-     cword = zeros(1,n); %all-zero codeword
25
26-     s = 1 - 2 * cword; %BPSK bit to symbol conversion
27-     r = s + sigma * randn(1,n); %AWGN channel
28-     %Puncturing of message
29-     r(1:2) = 0;
30
31-     %Soft-decision, iterative message-passing layered decoding
32-     L = rr; %total belief
33-     itr = 0; %iteration number
34-     R = zeros(2len,r); %storage for row processing
35-     while itr < Maxitrs
36-         R1 = 0;
37-         for lyr = 1:mb
38-             t1 = 0; %number of non -1 in row-lyr
39-             for col = find(R(lyr,:) == -1)
40-                 *t1 = *t1 + 1;



```

PROF. ANDREW THANGARAJ  
@THANGARAJ

Rate Matching in LDPC Codes using Puncturing and Shortening

So let us go ahead and do that, ok so for instance here when you look at the code word, ok when you look at r, ok so I the first 2 into Z are not transmitted, right. So you should not be transmitting that first 2 into Z, ok so one way to do the rate matching is, so I wish I do right matching, puncturing of message, ok. You simply said r of 1 colon 2 into Z equals zero, ok so I have punctured it, I have know it actually transmitted it but that just and the messages just all zero you might want to set it up as the actual encoding even if you actual encode, you simply set the received values to 2 into Z, ok.

(Refer Slide Time: 15:39)





```
10- treg = zeros(max(sum(s == -1,2)),2); %register storage for minsum
11
12- k = (nb-nb)*z; %number of message bits
13- n = nb*z; %number of codeword bits
14
15- Rate = k/(n-2*z);
16- RNNo = 10*(RNMod/10);
17- sigma = sqrt(1/(2*Rate*RNNo));
18
19- Nblttrrs = 0; Nblkerrs = 0; Nblocks = 100;
20- for i = 1: Nblocks
21-     msg = randi([0 1],1,k); %generate random k-bit message
22-     msg = zeros(1,k); %all-zero message
23-     %Encoding
24-     cword = zeros(1,n); %all-zero codeword
25
26-     s = 1 - 2 * cword; %BPSK bit to symbol conversion
27-     r = s + sigma * randn(1,n); %AWGN channel
28-     %Puncturing of message
29-     r(1:2*z) = 0;
30
31-     %Soft-decision, iterative message-passing layered decoding
32-     L = r; %total belief
33-     itr = 0; %iteration number
34-     R = zeros(1:n,2); %storage for row processing
35-     while itr < MaxIttrs
36-         Ri = 0;
37-         for lyr = 1:nb
```

PROF. ANDREW THANGARAJ  
IIT MADRAS  
Rate Matching in LDPC Codes using Puncturing and Shortening

So that is something, so now remember this needs an adjustment in the rate also, so was the number of code words bits for you it is not  $n$  be into  $Z$ , so in the rate you have to set this is  $n$  minus  $2$  into  $Z$ , ok so that is one little adjustments that is one needs to make, ok. So that is the first little thing as you know, this will make my rate  $1$  by  $3$  exactly and but I have set the first thing to be punctured, ok.

(Refer Slide Time: 16:13)



```
40-     ti = ti + 1;
41-     Ri = Ri + 1;
42-     %Subtraction
43-     L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z) - R(Ri, 1);
44-     %Row alignment and store in treg
45-     treg(ti, 1) = mul_sh(L((col-1)*z+1:col*z), B(lyr, col));
46-
47-
48-     %minsum on treg: ti x z
49-     for il = 1:z %treg(1:ti, il)
50-         [min1, pos] = min(abs(treg(1:ti, il))); %first minimum
51-         min2 = min(abs(treg(1:pos-1 pos+1:ti, il))); %second minimum
52-         S = sign(treg(1:ti, il));
53-         parity = prod(S);
54-         treg(1:ti, il) = min1; %absolute value for all
55-         treg(pos, il) = min2; %absolute value for min1 position
56-         treg(1:ti, il) = parity*S.*treg(1:ti, il); %assign signs
57-     end
58-     %column alignment, addition and store in R
59-     Ri = Ri - ti; %reset the storage counter
60-     ti = 0;
61-     for col = find(B(lyr, :) == -1)
62-         Ri = Ri + 1;
63-         ti = ti + 1;
64-         %Column alignment
65-         R(Ri, 1) = mul_sh(treg(ti, 1), z - B(lyr, col));
66-         %Addition
67-         L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z) + R(Ri, 1);
```

PROF. ANDREW THANGARAJ  
IIT MADRAS  
Rate Matching in LDPC Codes using Puncturing and Shortening

So now in addition instead of number of code word bits if you are, if say you are the message the number of blocks you actually intent to transmit is lesser than that then you have to make further adjustments here, right so how would you do the adjustment, so you have to look at this  $n$  is number of code words bits you have introduced the additional variable  $E$  here, ok

and once you compute the code word you will only transmit up to E, ok the code word here that you will transmit will stop at E, ok when stops at E you will add r and then for the remaining r you can either set the large positive value that we spoke about or make the code change here, so that you will not go till the end of all the columns, the B that you will use will stop at the particular n be the that you want, ok.

So it will not go through the end of the matrix, it will stop somewhere before, ok so that is the small change you want to make, I am going to skip making the change in the code here it is a little bit (may) it is not too difficult but one each to be little bit careful here, so wherever you use B, you will not go till the end of the column you will stop a little bit before, ok. Hopefully that is clear to you that is something that you have to do in the rate matching, ok.

(Refer Slide Time: 17:31)



The slide displays a MATLAB script for LDPC encoding and decoding. The script includes the following key sections:

- Parameters:** Rate, Message length, and Block size are defined.
- Encoding:** A random k-bit message is generated, converted to a binary vector, and multiplied by a generator matrix G to produce a code word.
- Puncturing and Shortening:** A subset of bits in the code word is set to zero based on a puncturing pattern.
- Decoding:** An iterative message-passing layered decoding process is implemented using the Soft-Decision Iterative Message-Passing (SD-MPP) algorithm. It involves calculating the total belief and iteratively refining the message estimates.

At the bottom of the slide, the text reads: "PROF. ANDREW THANGARAJ" and "Rate Matching in LDPC Codes using Puncturing and Shortening".

So you disregard the remaining of the parity bits, so I will leave this as an exercise for you, it is easy enough to try, you have the n B but here you have a different E, ok that will correspond to a smaller n B, ok everything else you keep the same and you do not transmit beyond E and you in the decoder you do not use anything beyond the E, ok so that is the couple of small little changes that one needs to make, ok. So that is the rate matching part, ok.

(Refer Slide Time: 17:57)





```
10- treg = zeros(max(sum(B == -1,2),2)); %register storage for minsum
11
12- k = (nb-nb)*z; %number of message bits
13- n = nb*z; %number of codeword bits
14
15- Rate = k/(n-2*z);
16- EbNo = 10*(EbNoB/10);
17- sigma = sqrt(1/(2*Rate*EbNo));
18
19- Nbitserrs = 0; Nblkerrs = 0; Nblocks = 100;
20- for i = 1: Nblocks
21- %msg = randi([0 1],1,k); %generate random k-bit message
22- msg = zeros(1,k); %all-zero message
23- %Encoding
24- cword = zeros(1,n); %all-zero codeword
25
26- s = 1 - 2 * cword; %BPSK bit to symbol conversion
27- r = s + sigma * randn(1,n); %AWGN channel
28- %Puncturing of message
29- r(1:2*z) = 0;
30
31- %Soft-decision, iterative message-passing layered decoding
32- L = r; %total belief
33- itr = 0; %iteration number
34- R = zeros(2len,z); %storage for row processing
35- while itr < Maxitrs
36- Ri = 0;
37- for lyr = 1:mb
38- ti = 0; %number of non -1 in row-lyr
39- for col = find(R(lyr,:) == -1)
40- ti = ti + 1;
41- Ri = Ri + 1;
42- %Subtraction
43- L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z) - R(Ri,1);
44- %Row alignment and store in treg
45- treg(ti,1) = mul_sh(L((col-1)*z+1:col*z),R(lyr,col));
46- end
47- %minsum on treg: ti x z
48- for il = 1:z %treg(1:ti,il)
49- [min1,pos] = min(abs(treg(1:ti,il))); %first minimum
50- min2 = min(abs(treg(1:pos-1 pos+1:ti,il))); %second minimum
51- S = sign(treg(1:ti,il));
52- parity = prod(S);
53- treg(1:ti,il) = min1; %absolute value for all
54- treg(pos,il) = min2; %absolute value for min1 position
55- treg(1:ti,il) = parity*S.*treg(1:ti,il); %assign signs
56- end
57- %column alignment, addition and store in R
58- Ri = Ri - ti; %reset the storage counter
```

PROF. ANDREW THANGARAJ  
IIT MADRAS  
Rate Matching in LDPC Codes using Puncturing and Shortening

The next two things I mention that one needs to do is convert this code into fix point, ok we will do that it is not very difficult to do.

(Refer Slide Time: 18:01)



```
31- %Soft-decision, iterative message-passing layered decoding
32- L = r; %total belief
33- itr = 0; %iteration number
34- R = zeros(2len,z); %storage for row processing
35- while itr < Maxitrs
36- Ri = 0;
37- for lyr = 1:mb
38- ti = 0; %number of non -1 in row-lyr
39- for col = find(R(lyr,:) == -1)
40- ti = ti + 1;
41- Ri = Ri + 1;
42- %Subtraction
43- L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z) - R(Ri,1);
44- %Row alignment and store in treg
45- treg(ti,1) = mul_sh(L((col-1)*z+1:col*z),R(lyr,col));
46- end
47- %minsum on treg: ti x z
48- for il = 1:z %treg(1:ti,il)
49- [min1,pos] = min(abs(treg(1:ti,il))); %first minimum
50- min2 = min(abs(treg(1:pos-1 pos+1:ti,il))); %second minimum
51- S = sign(treg(1:ti,il));
52- parity = prod(S);
53- treg(1:ti,il) = min1; %absolute value for all
54- treg(pos,il) = min2; %absolute value for min1 position
55- treg(1:ti,il) = parity*S.*treg(1:ti,il); %assign signs
56- end
57- %column alignment, addition and store in R
58- Ri = Ri - ti; %reset the storage counter
```

PROF. ANDREW THANGARAJ  
IIT MADRAS  
Rate Matching in LDPC Codes using Puncturing and Shortening

And then introduced the offset here, ok so we will do these two things in the next lecture, ok and with that we will close the LDPC codes chapter.