


LDPC and Polar codes in 5G standards
Professor Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology Madras
Successive Cancellation List Decoding

Hello and welcome to this lecture on successive cancellation list decoding of polar codes. A quick word I wanted to tell you about terminology here, so I have been describing the successive cancellation decoding using the binary tree, so Erdal Arıkan in his original paper did not use the binary tree representation, he described it just using basic location and matrix and all that, so traditionally this version of description of the success of cancellation decoding is called success of cancellation decoding.

(Refer Slide Time: 0:51)




Polar Codes

Successive Cancellation List Decoding

Successive Cancellation (SC)
binary tree → Simplified Successive Cancellation (SSC)
Simplified Successive Cancellation List (SSCL)

List decoding: 40's, 50's



PROF. ANDREW THANGARAJ
IIT MADRAS

Successive Cancellation List Decoding


So if you use the binary tree representation and use that language the terminology is called simplified successive cancellation decoding, so like I said successive cancellation is the 1st decoder that was proposed by Arıkan so you can abbreviate as SC, simplified successive cancellation is the one that we describe with the binary tree, okay. So that is called SSC and today we will see simplified successive cancellation list decoding which is SSCL okay. So I am simply calling it successive cancellation list decoding, the word simplified more or less just refers to the fact that you are using binary try and the language of tree traversal to describe successive cancellation decoding but list is new okay, so what is this list decoding?

What does it do? And how it works in the context of polar codes is what we will primarily describe in this lecture, okay? So a brief warning for you the idea of list decoding is very old okay, so list decoding is really old decoding it goes back to 40s and 50s, late 40s and early

50s some of the pioneers of information theory use the notion of list decoding, so it is not of natural that people tried list decoding in the context of polar codes when they wanted to improve performance okay.


So let us see how that works and we will describe that in this okay and of course list decoding is usually not that successful in many codes, so even in LDPC codes it is sort of difficult to make list decoding work but polar codes because of the sequential nature are sort of very amenable to list decoding and one can do a lot of modifications to make it more interesting and efficient okay so we will not look at all the details but at least we will describe how the list decoder works from a high-level okay.

(Refer Slide Time: 3:09)



Successive cancellation list decoding

- SC decoding: needs improvement in performance *1 dB (or) so poorer*
- List decoding: produce a list of possible codewords
 - Instead of producing a single codeword estimate
 - 4 or 8
- How to choose from list? *in Physical Layer*
 - Use Cyclic Redundancy Checks (CRCs) *used for error detection*
- Adds complexity but provides vital improvement in performance
- Good candidate for implementation



PROF. ANDREW THANGARAJ
IIT MADRAS

Successive Cancellation List Decoding

Okay so what is the overview here so like we said a successive cancellation decoding works really well it is soft input soft output actually but we end up ignoring the soft output we make hard decisions but it works quite well. It turns out when you compare it with other codes of the same length it is not quite equal in performance, so it is some improvement in performance so it is just not good enough it is about maybe 1dB or so lesser in performance so 1dB or so poorer okay so this is just a rough numbers give you something to think about, so people few years back were looking for some ways to improve the performance okay.

So one interesting idea in this direction was list decoding okay, so what is list decoding, so usually when you think of a decoder it takes the received vector and puts out one code word, right? That is how you think of a decoder, so that is how decoder is supposed to work. List decoder will not put out just one code word as the output, it will put out a list of possible code

word as output. It sort of says this received word looks close enough to multiple code words and I am not sure if I want to decide between them, so I will give you all the code words okay, so in the communication chain what do you do?

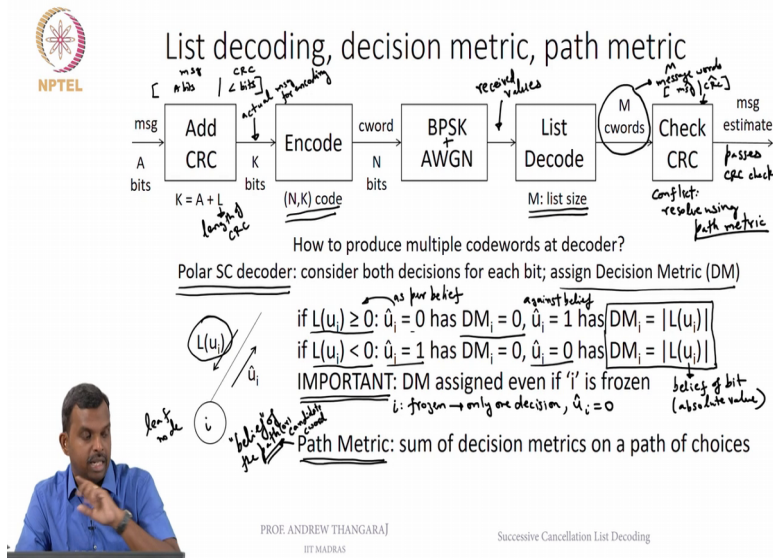
The decoder gives you multiple code words, so of course you have to do something, so usually what people do is use Cyclic Redundancy Checks, so it turns out CRC is another code that is used for error detection. It has very good error detection properties, it is very simple to use it has a very efficient single I mean shift register implementation for both the encoding side and for the detection side, so it is simple to implement and it is very good error detection properties okay.

So Cyclic Redundancy Checks has been traditionally used in communication system for long time okay so in the physical layer any message you send will always have a CRCs bit padded to it okay, so now what the higher layers will do is they will look at the decoded code word from the physical layer and check if the CRCs is satisfied. If the CRCs is not satisfied the higher layer traditionally will sort of realize that there is an error okay and then it will ask for a retransmission so all these ARQs and other schemes at the high-layers, when I say higher above the physical layer, so maybe at the...even at the TCP sort of player, so whatever higher layer you can think of... So in that layer the CRCs is used okay.

So when you do list decoding you sort of use the CRCs in physical layer okay, so how do you do it in the physical layer, in the physical layer itself it for the CRCs you have your decoder now producing a list may be 4, maybe 8 code words you check whether any of these 8 code words pass the CRC check or not okay. If they pass the CRC check then you except that code word alone otherwise none of them past the CRC check you are in trouble and if more than one pass the CRC check also you are in trouble you have to then pick something by some choice but usually as it turns out in polar codes usually that does not happen, so usually you have one among in the list as being the valid code word and that will pass the CRC okay.

So this observation was quite crucial in making polar codes successful in practice. It of course add complexity you will have to run multiple decoders more or less but it gives that that critically improvement in performance which makes it possible, so most of the decoders I expect for the 5G polar codes to be successful cancellation list decoders they will have a list and then and they will improvement the performance and that. Just as a high-level overview. Now let us go into detail on how the list decoder works, how do you make it work, et cetera.

(Refer Slide Time: 7:10)



So this is the picture and it is a good picture to keep in mind that there are a few terms here we will come to decision metric and path metric later but for now let us focus on the list decoding okay. So you have a message which is A bits okay and the 1st step is to add CRC okay so you add CRC. This L is the length of the CRC bits, so the way it will happen is you have the message which is A bits and to which the CRC will be added and this will L bits. Computing the CRC is standard and for instance the standard documents completely describe how to computer the CRC, how to run the shift register et cetera and how to add it to the message bits, so you take the A bits at the L bits you get K bits okay, so this is the actual message which is encrypt okay so this K bits is the actual message for encoding.

So you do not encode the raw bits that are received into the encoder, you add a CRC to it and after that you encode and like I mentioned the CRC addition always happens and this is very crucial for the next layer to work with the physical layer and the layer above it is the link layer, the link layer will look at the CRC definitely, so that it is very important so the CRC is there but you can also use CRC in the physical layer, nothing stops you from doing that as well, okay. So you encode, encode let us say it is N, K code N, K polar code maybe okay then you get a code word which is N bits after that you do BPSK plus AWGN exactly like before you get the received value here.

So this is easy enough to say you will get N received value and you do list decoder, so we will see how to do the list decoding later on but for now let us say we assume we do a list decoding okay and the list sizes is M let us say okay so the list decoder produces M code words not just one code word M code words. What you do here is remember the code words

will also be N bits long from there you can find the M message words, right? So from where you find the M message words which each of which will contain a message cap and a CRC cap okay. So you check if this satisfied CRC okay. Now you will get M of these each will be maybe a different message different CRC.

You check whether they satisfy the CRC the parity check conditions, CRC is usually a check condition you can quickly check using a shift register. The code word or the message word which actually passes the CRC is put out as the message estimate, so this passes CRC check, okay? So of course there are these corner cases of what happens if none of the code words pass the CRC. In that case you have to pick one and there can be ways to pick one I will tell you that later on and also it might happen that more than one passes the CRC even there you have to pick between these code word, it turns out each of these candidates has a metric associated with it which is called the path metrics.

So I will describe how this path metrics are chosen and based on the metric you can decide, so if there is any conflicts you resolve using path metric. I will talk about path metric and how to deal with that later on okay, so this part metric is an important notion in this list decoding. It serves multiple purposes, it keeps the decoding complexity in control and also gives you an indication of how good a particular candidate code word is okay so you put out M code words not all of them may be equally likely or something like that you may not have the same believe on each of them okay.

So you can also put out of live on each code word, so this path is like is like a belief on each code word okay so that is how to think about it. So how you produce multiple code words of the decoder? In the polar successive cancellation decoder this is actually relatively simple and straightforward okay the reason is you are sequential, right? So start from the received word you go and make a decision on the 1st bit okay may be it is a frozen bit. If it is frozen bit then there is only one decision, so it is going to be 0 there is no problem, so slowly eventually you will hit a message bit and you have to make a decision based on its belief 0 or 1.

Now you are committing to one direction there you are making that decision as 0 or 1 that would be an erroneous choice okay. Even if it is an erroneous choice may be your decoder will eventually correct it but the final code word you put out will always be in error because you made that 1 mistake there. So this can happen in any of the positions not necessarily the 1st position but this can happen okay, so when you make a decision in the sequential decoder

you can make an error. If you make an error you are not allowing yourself any sort of way to go and correct it okay.

So now what the list decoder does is, whenever you have to make a decision like that you sort of keep 2 paths open in your decoder instead of only decoding the one path which is according to the belief you make both decisions okay so you allow for a decision 0, so the decoder proceeds into paths after that at a decision point either 0 or 1 okay and what you do also is you assign a decision metric to every decision you make okay, so when you have a certain belief for the bit and you make a decision, your decision may be according to the belief or opposite from the belief okay.

So this is what is discussed here, so if L of u_i , so how do you make a decision is belief is coming into the leaf node, this is a leaf node let us say i^{th} leaf node and then you put out a U_i cap, how do you that usually? If L of u_i is greater than or equal to 0 you are going to say u_i is 0, right? Okay, so if you have made a decision as per the belief then you assign a decision metrics of 0 okay then you say there is no penalty in my decision. I got a certain belief I made a decision exactly like I should, so there is no penalty or there is no metrics.

Metrics is also like a penalty you can think of it okay so on the other hand if L of u_i is greater than or equal to 0 and you decided u_i cap as 1 okay so this is like the other path are divergent path sort of okay. So this is against belief and whenever you go against belief you put a penalty okay, so these are the 2 cases where you put a penalty or a decision metric you add a decision metric to that decision if you went against belief. Same thing happens here if L of u_i is less than 0 this is as per belief, this is against belief.

When it is as per belief you put a decision metric of 0 you do not penalise yourself on that path but on the path when you made a opposite decision you penalised yourself. And what is the penalty? Penalty is the belief of the bit absolute value. Okay, so the critical thing to realize is, this is an important fact when you do this decision metrics, so supposing i is frozen, so this is what you do if i is not frozen. If i is frozen then what you do? You always decide, so you do not make 2 decisions when i is frozen okay so only one decision u_i cap is equal to 0 but even when i is frozen you have a L of u_i okay remember that.

Even when i is frozen you have L of u_i okay and if your decision of u_i cap equal to 0 is against your belief it may happen that L of u_i is actually negative okay even though it is a frozen bit, the decoder might have come through some path and it may tell you that frozen

that is actually 1 okay it might do that but remember the frozen bit were chosen so that the least reliable bits of frozen, so they can be in error with quite a high probability.

So whenever they are in error and belief is opposite you have to penalise even that frozen bit decision okay, so that keeps the whole thing correct and all that so you know which decoder is doing good which decoder already had a heavy penalty and so on okay. So this is very important to understand so that this metric is assigned even if i is frozen. The decision has not changed it is always $u_i \text{ cap } 0$ for the frozen bit but you will penalise if your L of u_i is negative so you will add the absolute value of that to this decision, so that will be a metric. And what is path metric?

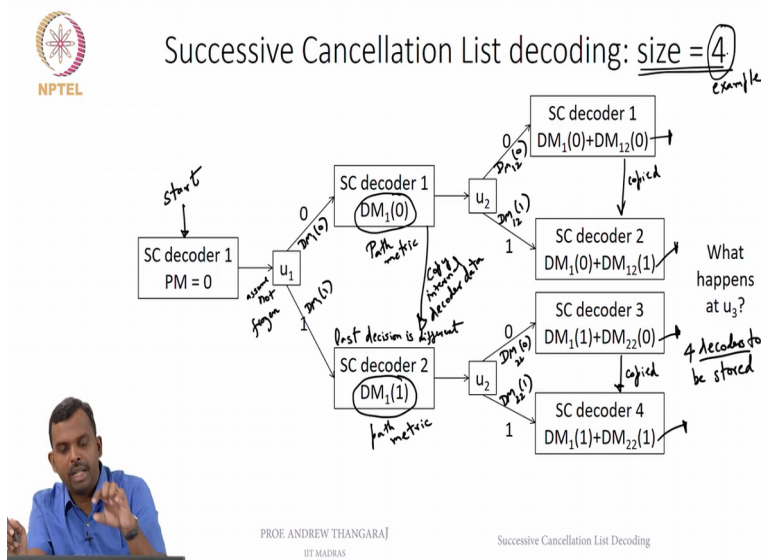
Decision metric is the metric on each decision, path metric is sum of all the decisions you made on a particular path okay so like a said every point you can make multiple decisions may be okay. Then on a path if you made every decision there is a penalty that is associated with the decision on every path you add all those penalties and the total penalty is the path metric okay.

So the path metric is sort of like a belief of the path okay path remember corresponds to candidate code word okay so the list decoder for short multiple code words a particular path is particular candidate code word and the path metric is crucial because it is like the belief in your code word, how confident are you that the code word is correct, if the path metric is very low okay so it is sort of like opposite of 'belief' I put belief in quotes that if the path metric is very low then your penalties very low and each means the belief is very high okay this is sort of the opposite okay when I say belief keep these quotes in mind it is the opposite of belief so if the path metric is low then you are very confident more confident about the code word.

If the path metric is higher you are less confident about the code word okay, so this is how path metric can be used to resolve conflict okay, so when you check CRC supposing you find none of the message candidates are actually passing CRC then what will you do? Among the candidates you pick the one with the lowest path metric okay, so that resolves conflict in the output of the list decoder when you want to put out one single output and if multiple candidates messages satisfy the CRC check once again you can pick the one with the lower metric okay so that is one way of resolving conflicting.

So hopefully this is clear to you this is how the list decoder generally works and now I mentioned it in the context to the polar successive cancellation decoder you can do this for any other code also. If you have a sequential decoder working for any code you can always penalised like this and you can always have a list and can do this. The fact that it works really well and polar coding is an important observation and I do not think there is real proof for these things but people have observed it and they use it for improving performance okay, so this is how the list decoder works, you make multiple decisions on the code word bit depending on the beliefs and you might go against the belief you put a penalty if you go against the belief and keep track of a path metric which gives you the confidence level on the particular path or the candidate code word. So this is of the list decoder generally works okay.

(Refer Slide Time: 18:59)



Now let us look at it in the context of, let us look at it more carefully in terms of paths, so this view of the path is very important and also restrictions on the size of the list is very important okay, so usually when you do list decoding you restrict the size of the list, so you are not allowed to put more than 4 candidate code words let us say 4 is just an example okay so I will put an example here you can have 8, you can have 16, you can have 10, you can have 12, you can have 7 whatever number you like 4 is one example I have put okay.

So this puts some restrictions on how make decisions you can possibly make, right? I kept saying for every decision you can either decide whether it is 0 or 1 okay but if you keep on doing it the number of possible paths will quickly diverge okay at the 1st place it will diverge into 2, right? So that is what shown here that initially you start with a single successive cancellation decoder, so this is your initial starting point and you have a path metric of 0 you

are not making any decisions you are starting from the root you are going down, left down, left down, left you just have one decoder you are not making any decisions.

Till you reach u_1 okay, u_1 is the 1st decision you make okay. Remember u_1 might be frozen, if it is frozen you will not make this decision okay, so I am assuming this is not frozen, assume not frozen okay, so if it is frozen you will continue to be in one path okay remember that, so I am assuming u_1 and u_2 and all is not frozen I am not freezing anything, so you will start diverging even in u_1 but in case u_1 is frozen, right? you will continue only in one path but your path metric may not be 0 anymore okay it may be nonzero okay so we will look at the more general situation soon enough but for now let us continue with this simple picture to see how the thing evolves okay.

So you come to u_1 assuming it is not frozen, so you make 2 decisions 0 and 1 okay, 0 will have some penalty which I am calling DM_1 of 0 okay it gets added to the path metrics, so the old path metrics becomes DM_1 of 0. Remember this as always the path metric okay the sum of all the decisions metrics, one of these will be 0 okay so one of these will be 0 depending on what the sign of the L of u_1 but the other will be nonzero okay so you have 2 single successive cancellations decoders now, so initially you had only one decoder and remember every decoder cost you memory and storage.

So every decoder has a state you have 2 store all those L values the belief that are flowing in and the U cap values, things that are flowing back and the states okay, so you have to store all of them and when you make a copy of this decoder you have to copy all that information up to that point all the information that you had you have to copy okay so that happens here, so this is copied, so you copy internal decoder data okay so this is very important okay so you have to copy this and except for the last decision okay, so the last decision alone is different, right?

Except for the U_1 which was decided as 0 in this SC decoder 1 this is SC decoder 2, u_1 would have been decided as 1 okay so you will have 2 different decoders, 2 different path metrics but most of it will be same okay so up to that point it is the same okay only there it differs okay. The next you go on to u_2 again I am assuming this u_2 is not frozen then what you do? You again divert it into 2 okay so you make both decisions 0 and 1 and this path metric one more decision metric will be added, so like I said this is decision metric one of 0, this is decision metric one of one likewise this I will call decision metric on the 1st decoder

for the 2nd bit when you make a decision 0 okay likewise here decision metric in the 1st decoder still for the 2nd bit when you make decision one okay.

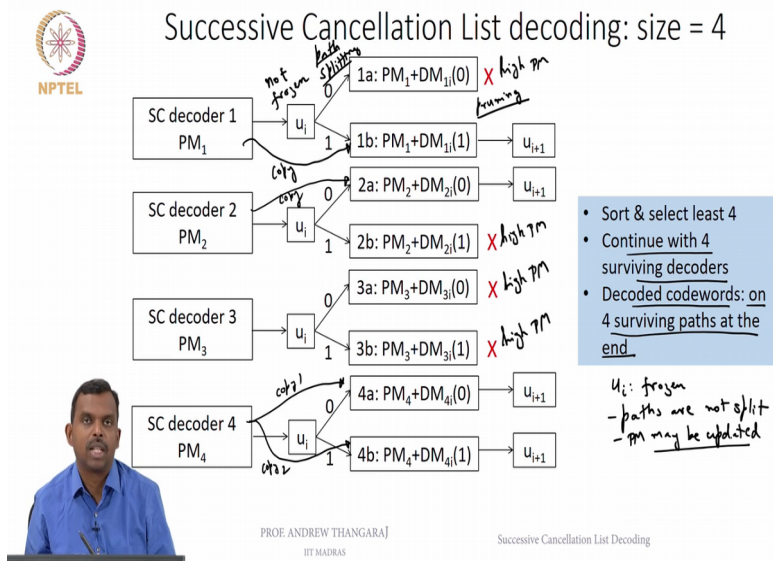
So this decision metric gets added to this, this decision metric gets added to this okay again one of this will be 0, the other will be the absolute value of the belief that came in okay so you get added here okay so later on when we see an example we will sort of explicitly sketch this okay. So likewise here this is DM₂₂ I think M in a subscript DM₂₂₀ DM₂₂ of 1 okay so these are again decision metric that gets added to the earlier path metric get the overall path metric.

Remember again this was copied up to the u₁ point and likewise this was also copied, these are all copies okay so you copy the 1st decoder to get the 2nd decoder, the 2nd decoder will sort of be renamed as the 3rd decoder and you will copy to get the 4th decoder okay. So remember these 4 path metrics have to be maintained and 4 decoders have to be maintained to be stored okay, so when I say decoders are stored all the internal information, the L values that is flowing from top to bottom u cap value that are flowing from bottom to top and the states have to be copied and retained in this 4 decoders okay.

So what do we do at u₃? What happens in the next okay? So we kept making 2 - 2 decisions for every decision okay now u₃ also if you make 2 decisions in each decoder you will have 2 paths and that becomes 8 paths okay but I want only the size of 4 I do not want 8 paths okay, so what you do at that point is you thrown away 4 paths okay you do not go to 8 paths you throw away the 4 least reliable paths or the 4 paths which have the higher penalty you throw them away okay, so that you go back 4 okay.

So this is a slightly tricky thing if you think about it you are growing the decoder 1st and then you are pruning at so you have 4 decoders here 3 also after the decision give 3 eventually you will have 4 decoders but in the intermediate path you have 8 decoders then you throw away 4 of them and you have only 4 decoders, so this is how the list decoder proceeds I am going to show that in the next slide but hopefully this is clear, so you get a feel for how it happens I am not showing the internals of each of this decoders they sort of proceed in the same way only when they make decisions they have this kind of divergences.

(Refer Slide Time: 25:47)



So this is what happens in size equal 4 decoders in general, so this is a more general picture at a particular node u_i okay, so you have 4 decoders coming in okay, so before the decision on i you have 4 decoders and 4 path metrics PM_1 , PM_2 , PM_3 , PM_4 okay there will be 4 path metrics then you coming to u_i , right? And then you make a decision it could be 0 or 1, so again once again I am assuming not frozen, if it is frozen if the i^{th} bit is frozen there is no problem okay the decision is always 0 4 remains 4 okay if it is frozen I do not show anything more 4 remains 4 I will only have to update the path metrics depending on the sign of L of u_i okay if L of u_i is positive there is no update to do just same path metric.

If it is negative then you add the absolute value of L of u_i to this, so if it is frozen that is what you do frozen is very easy nothing much to do, if it is not frozen then you have to make both decisions on each u_i okay 0 and 1, 0 and 1 okay so you end up getting 8 different path metrics here you throw away 4 of them these are the high path metrics okay so you sort and select the least 4. The higher 4 path metrics are thrown away and you keep only the lower 4 okay, so now if you think carefully about it this involves more copying and moving around of data between the decoders okay, so we have seen decoder 1 alone moves here okay so this will be copied here okay, so what happens here?

This one is copied there this is okay, so both of these are retained and only one way, now what happens here this SC decoder 3 is totally deleted it did not survive at all it made some other mistakes on the other hand this is copied twice okay maybe I will show it from the same starting point so that you see where it goes copy 1, copy 2 okay so you make 2 copies so that

it still maintains after the splitting and the pruning okay, so lot of other things can happen okay.

So it may happen that such things do not happen all the decoder survive one from each survive but that is little bit unlikely there will always be a situation where one splits into 2 all of that will happen, so all of that involves moving data between these decoders because each of these decoders were sort of functioning with their your own tree okay each of them have their own tree with their own L matrices and this u cap matrices you to move them around to make sure that this whole thing will work okay, so this is the overall thing so you continue with 4 surviving decoders all the time.

So what are the decoded code words these are the decisions made on the 4 surviving paths at the end, at the end you will have 4 surviving paths whatever insurance you made on them are the u caps that have that you check CRC on that then you proceed okay. So this is in essence how the list decoder works, once again I have not talked about the frozen, if u_i is frozen paths are not split okay so this operation of making 2 decisions is called path splitting, so this is called path splitting okay and this is pruning okay so you split the path and expand it into 8 or 2 times the original size and then you prune it down to bring it back to M.

So if u_i is frozen paths are not split, path metrics may be updated okay so the updating of the path metrics will happen but paths are not split if u_i is frozen, so hopefully this is clear to you so this is how the list decoder proceeds and when we implement it you will see that there are some lots of (())(29:56) take that is little bit more complicated than the successive cancellation decoder which we could implement very easily but this is just essentially making multiple copies of that and working with them okay.

So that is it that is the end of this lecture on successive cancellation list decoding. In the next lecture we will talk a little bit about the implementation of this, how we go about implementing it? What are the data structure, et cetera and then finally we will write some Matlab code for this okay and that is more or less the end of the course and after that I will have some conclusions where I will look at performance and all okay, thank you very much.