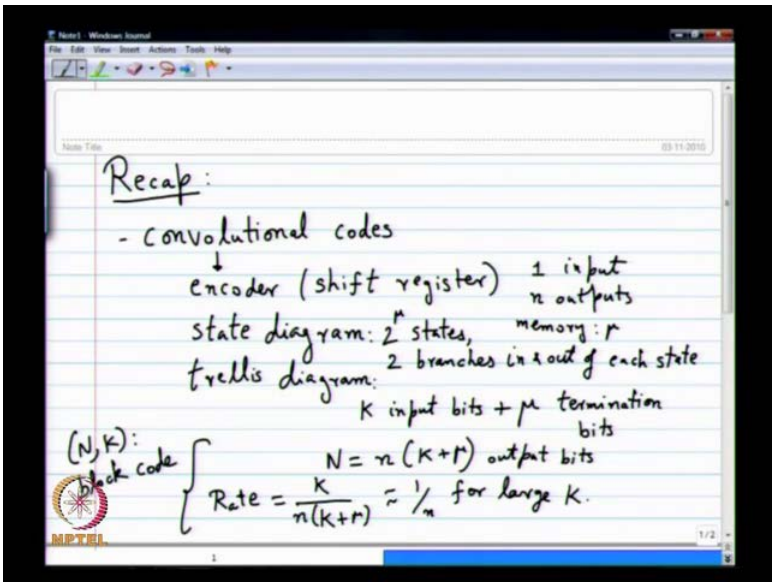
Coding Theory
Prof. Dr. Andrew Thangaraj
Department of Electronics and Communication Engineering
Indian Institute of Technology, Madras

Lecture - 31
Viterbi Decoding of Convolutional Codes

(Refer Slide Time: 00:15)



Let us do a recap of where we are. We have been talking about is convolutional codes. So, the way we have been approaching this, is to first talk about the encoder, the encoder is nothing but a shift register, right? So shift register based encoder, I will give you very simple example. I showed how you can clock widths in and then clock widths out. You get a shift register as the encoder. Then there are various representations for the encoder some reason windows wants to update what should I do? I cannot cancel, postpone.

So, I thought this task bar was supposed to be locked. I guess the update is so crucial that does not care about it, it's still locked. The encoder and we spoke about various representations for the encoder, one was the state diagram and the only drawback in the state diagram was the time axis was missing.

So, a better representation turns out to be, what I pointed out is a trellis diagram. Usually the shift register will have one input, one input stream. Let us say some n outputs and the number of the flip flops is also known as a memory. So, memory is going to be some mu.

If you make a state diagram with this shift register, this shift register encoder you are going to have how many states?

Student: ((Student: 02:32))

Two power mu states. How many branches out of each state?

Student: Two power mu.

How many branches out of each state?

Student: Two.

Two branches out of each state. How many branches into each state?

Student: Two.

Two branches in and out of each state. So it has to be that way in and out of each state. Where will the n play a role? Basically, each branch will be labeled with one bit input and the stroke and n bits as output. That is the branch labeling. Trellis diagram, since we are explicitly have time playing a role, we have to first specify the number of input bits. We usually keep it as, let us say some k input bits plus so is that enough? Can we stop there? No, we have to terminate the state back to the all zero state. For that you need mu termination bits. So, the number of output bits N will be equal to what small n times k plus mu. We are right number of output bits will be small n times k plus mu.

So the actual rate k by n times k plus mu, if k becomes very large, roughly 1 by n for large k so one by n is known as the design rate of the encoder. The actual code rate will be different. That is the description of convolutional codes, so the way we are looking at it once you fix the number of input bits as k. Look at all the output bits you got. Can I think of this code as a block code? For every k bits of message, I am getting n bits of output. If I go ahead and exhaust all the two power k possibilities, I will get two power k code words N bit code words. So actually in this description it was also an N comma k block code. There is nothing wrong with thinking about it that way, also. You can think about it as an n comma k block code.

You can in fact come up with generator matrix parity, check matrix all that you can do there is nothing wrong with that, except that we are describing this block code in a very

simple way using simply mu d flip flops in a shift register implementation of the encoder. You do not have to describe it in any other complicated way. It is a very simple description for the encoder itself.

So, essentially you can also think of it as a block code. There is nothing wrong with that except that it is traditionally called a convolutional code because of obvious reasons, this is a shift register. So it has an impulse response. You can think of the output as being a convolution of input and the impulse response of the shift register. For those reasons it is called convolutional code, but in actual operation it will always be a block code. So will it be a linear block code? Yes. It will be a linear block code because it is a linear shift register. So add two inputs, if you extort the two inputs the corresponding code words will also extort. It is all every operation is linear there right, every code word bit was written as a linear combination of the inputs, some inputs. So, does not matter how else you think about it. It will be a linear code. This is just a different way thinking about that problem.

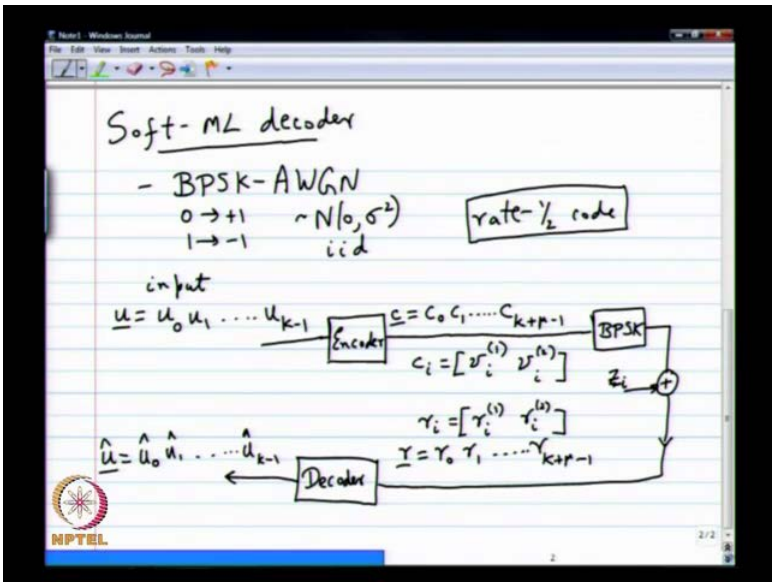
The Trellis representation on the other hand, has a complexity of two power mu case was referred to as the complexity of the Trellis is one measure of complexity. It is a most common one, 2 power mu. The number of states that you have at any given stage is the complexity of the Trellis. It turns out the Trellis plays a key role in decoding. I will describe why. It is easy to see why. It will, but because of that the complexity of the decoder is limited by mu.

So, even if you increase k to a very large number one thousand or two thousand you only have a complexity of 2 power mu and not two power k. You remember that was our fear in when we did m L decoding. Complexity always become two power k and if k becomes thousand then you have no hope of implementing it because of the Trellis for any k, you only have a complexity of two power mu and the decoder also becomes only two power mu complex. That is the major attraction of convolutional codes.

Otherwise if you look at it in terms of minimum distance, these codes will not be that great. In fact more than the minimum distance, I will point out later the number of minimum distance code words is fairly large for convolutional codes. That is the problem when you implement them. So, as block codes they may not be that great, but

since they have a decoder they have very interesting characteristics, which we will see later on.

(Refer Slide Time: 08:40)



So, the next thing I am going to point out to start with is the decoding. The decoder, the m L decoder. So we will talk about the soft m L decoder. For the soft m L decoding as usual we will be using BPSK m L region. Remember, BPSK is 0 going to plus 1 and 1 going to minus 1 AWGN will be noise distributed as normal iid, right. That is our standard assumption. There is also a hard decision m L decoder in which case the channel will be a binary symmetric channel. So that is what we will be using, so that is what we have used as a standard for hard decision decoding. I am not going to describe that. It is quite simple towards the end may be I will point out how that works, but we will begin with the soft m L decoder.

So, there is lots of notation that needs to be built up. Finally, the decoder is very simple and you will intuitively get a very easy idea of what is happening as we go along, but initially we need a lot of notation to make sure that we do not slip up with the description of the decoder. Let us do that notation, so my input u is going to be, I am going to start with a shall, we start with 0 or 1.

Zero, u 0, u one all the way to u k minus 1, then I am going to have some termination that is I will drop that, so if needed we will come to it later. For now these are the input bits. These are going to get encoded into a code word. This is the convolutional encoder.

I have this habit of drawing the box first and then writing inside. So what I write inside tends to get really crammed. Anyway, think it is clear enough so the encoder is going to put out now capital N outputs. Now, I am going to group this outputs into, so again so we will always fix a rate half code. If you have a rate 1 by 4 code or 1 by 5 code it is easy to extend this. So we will have a rate half code.

In each for each input bit there will be two output bits. One of them will we will denote as let us say some what we will be doing v 1 0 and the other is v 2 0. Let us do that so the output will be the code word. May be I will call it c so the code word is going to be. I will call it as c 0, c 1 so on till c. It is going to be k plus mu minus 1. It will go all the way till that and each c i, is actually what it is going to be v 1 i and v 2 i. It is actually a vector, is that, so this is the output. Then as usual you have BPSK and then you have the AWGN.

So what we get here likewise, I am going to call as the vector r and this is going to be r 0, r 1 all the way to r k plus mu minus 1. Then each r i is what it is going to be two things, it is going to be r i 1, r i 2 corresponding to what was received for each thing. Then I think we will stop there. That is enough notations. So, this is the, this is the set up. Given this vector r, I have to now do a decoding. Windows is insisting that it will update. So, what we do?

Student: Sir, postpone remind me four hours.

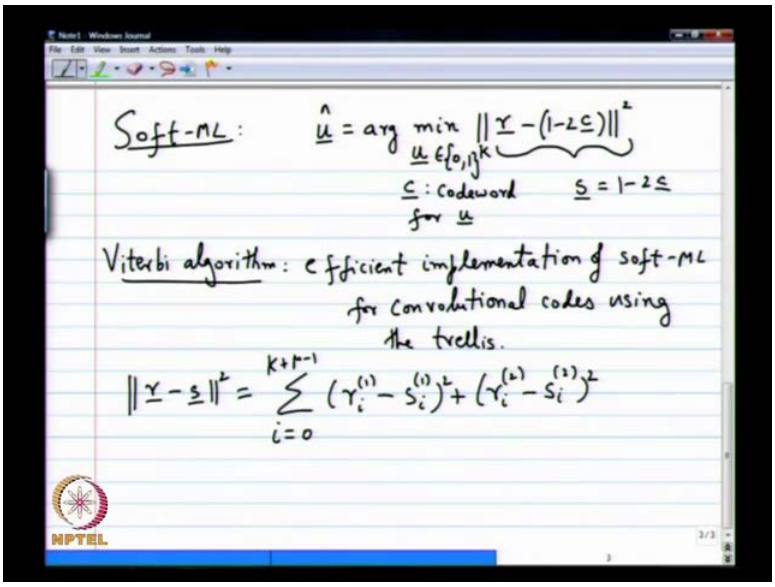
Four hours, everybody knows exactly how to do that. It is been ages since I have used windows actively. So actually I have never used to windows actually, except for this lecture, of course. Is it going to go down on its own or what the task bar it would not?

Student: Sir let us focus on the.

Ok, there is some resident windows experts. So, the decoder is going to put out u hat. That is going to be once again u 0 hat, u 1 hat u k minus 1 hat. There is a tendency to get lost in the notation, then some of you might get lost in the notation, but when they hang on there is some simple description towards the end. So, when you are doing soft m L decoding, what you have to do it is very clear. What you have to do right in BPSK AWGN and when you have to do soft m L decoding, you have to find that code word which is closest in a certain distance. What distance? The Euclidean distance. So I have

to compute that and find the closest. The task is pretty clear and we can write that down. We can write that down without too much of effort.

(Refer Slide Time: 14:53)



So, soft m L decoder it is going to do this. It is going to say, u hat is argument of minimum. Here, is where the notation is going to get a bit messy. So I will say u over all u what the Euclidean distance between r and u. U it is not u. So, I have to do a lot of conversion. I am going to write it as 1 minus 2 v square is that, so u will belong to all 0 1 k and v will be c. I am sorry, I have been using c. Sorry, so c is what we should think of as c, c is the code word corresponding to u. It is a code word for u. That is the notation is that, so the task clearly is to compute this Euclidean distance for all possible inputs. How many Euclidean distances will you have 2 power k.

It looks like you will have to compare compute all of them first and then compare with each other and find the minimum. That is the brute for soft m L task. That we saw before even for the linear codes is hard it is not going to be very easy to do. It is going to be exponentially complex 2 power k complex. It turns out because of the Trellis, you can do it sequentially as oppose to in total. This task of doing the minimization can be done sequentially. That is the idea in the in the decoder. So, the efficient implementation of soft m L decoder for convolutional codes is called the viterbi algorithm. After the person who discovered it, out of all the person who came up with it first and we will describe

that next. This is basically efficient implementation of soft m L for convolutional codes using the Trellis obviously.

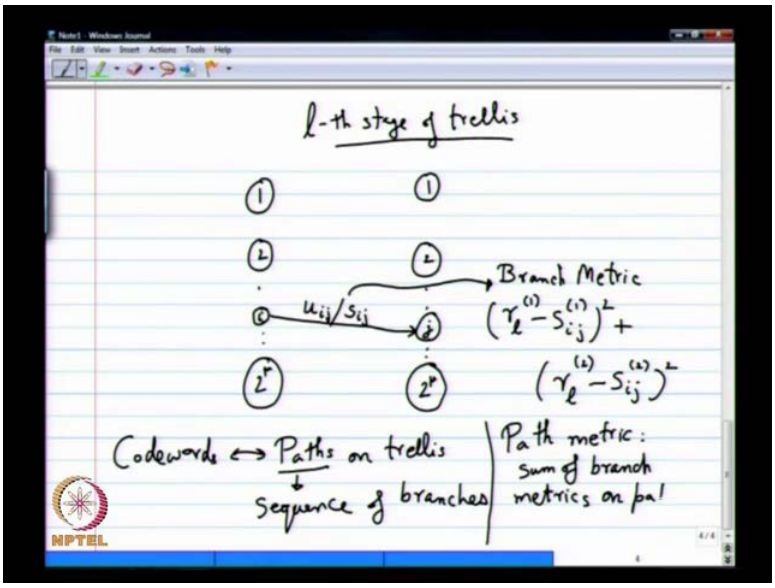
We will begin by looking at this expression r minus 1 minus 2 c. This 1 minus 2 c is a bit of a pain to carry over, so I will simply use s equals 1 minus 2 c. This s expression, we will use this is basically summation i equals 0 to k plus mu minus 1. What should I put inside the summation? Remember, r i is actually a vector and s i is also a vector. It has two entries because I have assumed the rate half codes. If a rate half code, if it is one by n then it will have n entries. That does not matter, so I will, since I assumed rate half this is the slightly simpler situation. So, you will have r i 1 minus s i 1 square plus r i 2 minus s i 2 square. So it is that is the expression.

So, the key idea is you can you can compute this using a summation, of course. The computation can be done sequentially. It is obvious, right. So summation one after the other, it turns out you can also do the minimization sequentially so that is the trick in viterbi algorithm. The minimization also can be done at every step and you do not have to remember too many code words. You only have to know two power mu code words.

As long as you keep track of two power mu code words maintain a list of two power mu code words it turns out the global minimum will occur within this two power mu. You can be happy about the answer you get so that is the central high level idea. How it works is easy to describe. I will describe it and we would not see regress proves, but I think you will see clearly that works. It is a nice idea it will work out very well is that okay?

So, the first thing we will use to translate this into Trellis languages. We will distribute this summation over stages of the Trellis. This is the very natural way to do that. Remember the i th input is associated with the i th stage. So the i th output r i is also associated with the i th stage. These si’s will where, will the si’s be in the i th stage of the Trellis. They will be on the branches; the branches will have the si’s, possible si’s. So you go on each branch and then write down what you compute here. Then you have translated what you have to do into the Trellis. The only thing I have to do next is show you how the minimization can also be done. If you do this, over the Trellis, and that will be the end of it.

(Refer Slide Time: 23:07)



So, let us do the first translation on the i th stage of the Trellis, so if you look at the i th stage of trellis. You will have how many states one, two all the way down to two power mu states. This side also you would have one, two all the way down to two power mu states. There will be branches I will take, I will take some kind of a generic branch. May be from, I do not know where, can we take a generic branch. Somewhere from here to here, then may be it goes from, so maybe that is not a good idea. So let us just take let us take some state i to state j is that i to j. This time I was smart, I wrote first and then drew the circle. Let us take a branch that goes from here to here. What will be the, what should be written on the top of the branch. There will be the label for the branch right inputs slash output.

So I, since it is i to j may be we call it u, u I do not know what should we call it ij slash sij. So, I mean this is just notation. I think it is clear what I mean, but it is good to write it down also, is it so remember uij is going to be just one bit sij will be. Well, I am thinking sij as 1 minus 2 times the BPSK encoded form. It will be two a length two vector because of a rate half code. If it is rate one by n code then it will be length n vector with plus ones minus ones depending on what the actual output. There was, is that alright? So what I will do is, I will associate with this branch something called as a branch metric.

What will be so this, I is occurring in too many places right, so maybe I will make this lth stage of the Trellis. Sorry, for that the l th stage of the Trellis you are looking at state i

to state j as transition the branch metric will then be r l 1 minus sij. What one square plus r l 2 minus sij 2 square. This will be the branch metric associated with that particular branch in the l th stage. You have to think of this branch metric as some kind of a weight on that branch, or some toll money that you have to pay somebody who is sitting there to go on this branch to take this root. We have to pay that money. So, some cost associated with or may be the distance that you have to travel on that branch, something some cost associated with that with that branch or edge or whatever you might want to call. Why did I define it this way?

Student: Sir, the optimizing function will be.

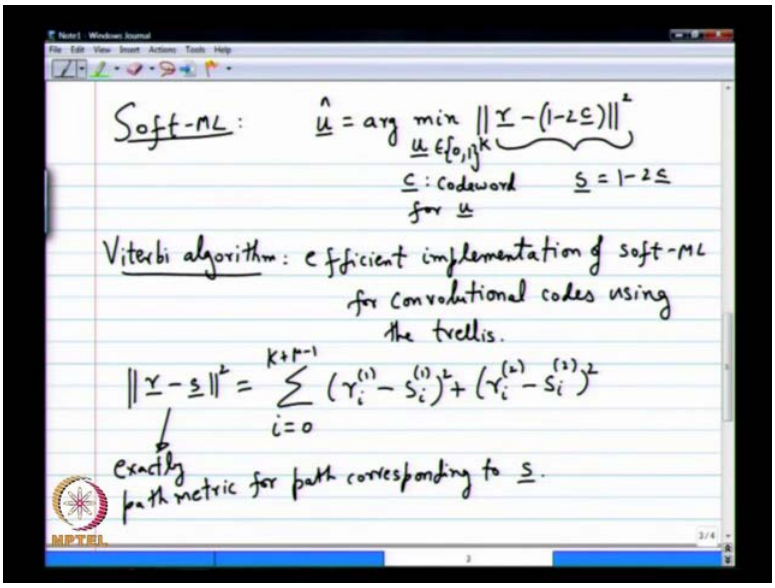
So, I am trying to minimize the sum of terms like this over all possible code words. If I have a code word on which this branch was actually used, this will be a term that will participate in the summation. That is the idea. I am trying to write down all those summation terms in terms of the branches. I think that is, the that is the first step. So we have some kind of a branch metric. We can deal with it is that alright.

So, the goal now is so the soft m L task of finding the least the code word, it is closest in terms of Euclidean distance can be written in terms of the Trellis. What you have to do for that is you have to first use this idea that code words correspond to paths on the Trellis. What is a path? Path is basically a sequence of branches. What do I have for each branch? I have a branch metric, so what can I do for each path? I can total all those branch metrics on those, on that path and get something called a path metric. That is called the path metric. I will define the path metric sum of branch metrics on path. Now, this path metric has a very real meaning for us, what is the path metric?

Student: ((Refer Time: 26:03))

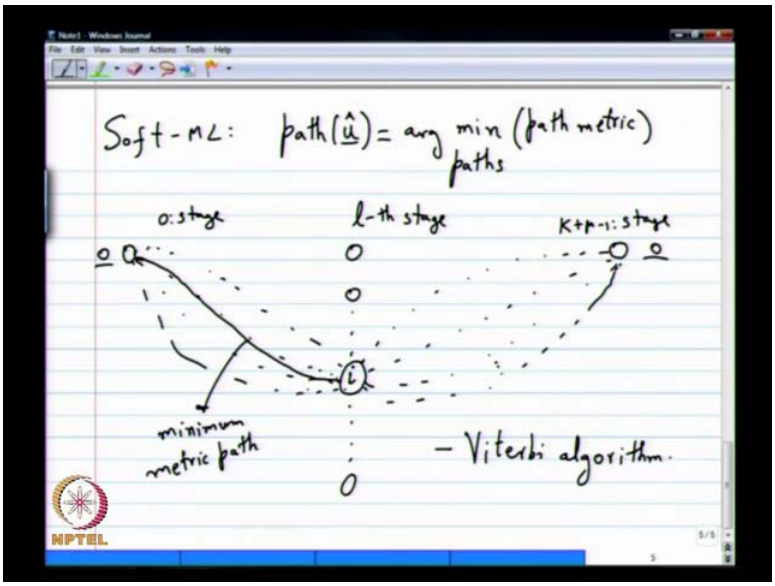
It is exactly this term, it is exactly this term, this is exactly the path metric path metric for.

(Refer Slide Time: 26:06)



That chosen code path for path corresponding to the code word to s. s is a valid well symbol sequence not really a code word sequence, S is the symbol vector. It has, it corresponds to a path on the Trellis. The branch path metric of that path exactly equals this r minus s square. That is what I have done with this trellis representation. My task of doing soft m L decoding of finding that code word which is closest in Euclidean distance from the received word, received vector is exactly the same as finding that path with least path metric on the Trellis. It is a very simple one to one correspondence and that is the first idea in the viterbi algorithm.

(Refer Slide Time: 27:14)



So, the soft m L can now be equivalently written as u cap. Remember as the u cap also defines a path on the Trellis. I have to be careful here when I do notation. Path of u cap equals argument of minimization over all paths of what, the path metric. Remember the inputs sequence also defines clearly a path on the Trellis. I can think of the path corresponding to my decoded input sequence that will be the argument of the minimization over all paths of the path metric on the trellis.

Remember the metrics change for each received word. Once you get a received word you have to compute the metrics and kind of load it on that Trellis, not Trellis with those weights. You have to find the minimum path if you get another received vector. What will happen, all these weights will change. You have to re-compute the weights, so how many computations of weights do you have to do?

Student: Sir, it is going to a two.

For each, for decoding each code words, for each branch. You have to decode, you have to compute one for each branch. In fact you can simplify the computation. It is not you, can do some optimization, but anyway one for each branch how many branches are there. Two times two power mu times k plus mu. So clearly it is linear in k and exponentially mu. It is not exponentially in k once it goes exponentially in k. You have to worry about it. It is not going exponentially in k. It is only going linearly in k. So computing the branch metrics is not a problem, but if you want to compute path metric for every possible path then there will be a problem.

Once again you go back to the same two power k. If you, because there are two power k paths and for each path if you want to keep doing this computation again, you are not going to go anywhere. So the trick to efficiently do it within 2 power mu is to only compute path metrics for some paths. You will keep track of only roughly two power mu parts. As you go along the Trellis and it turns out the minimum the best path will always be within this 2 power mu.

So, the question is which are these two power mu paths and how should I keep track of them. I will give you a simple argument. It is easy to see why this will be true. You can also prove it recursively if you like. Remember, I started at the all zero state. This is the zero state. I also ended at the all zero states. This happened at stage zero, zero th stage. This will be finally, at the k plus mu minus one th stage so that is fine.

So, if I look at the i th stage or l th stage, I am going to have so let us say, so remember there are is going to be something to be on this side or that side. Let us say I pick up this side. There will be a bunch of states. Let us focus on some particular state, let us focus on, let us say state i on the l th stage. Now, if you look at paths from the all zero state at the beginning to this state i in the l th stage there will be several paths. Let us say, let us draw small bunch of them. I am going to draw them dotted. There will be several paths to state i in the lth stage and once again from state i to the all zero state, there will be once again a lot of paths. Let us do that will be lot of paths. I am just drawing a representative sample of some paths, but there will be lots of them. I want to keep track of all of them that is the issue.

So, if I want to find the lowest path metric path from state zero to starting at state zero and ending at state zero. My argument is at this state i, I only have to keep track of that path which has the minimum metric up to state i, I do not have to keep track of all of these paths that are ending here. It is enough if I keep track of that one path, which has minimum metric up to that point. That is the idea, I am doing it sequentially. I supposed to doing it totally at this l th stage. I only worry about the minimum path metric up to that point, but I have to do it state wise. I cannot just do it totally state wise for the state i there will be one path which is the minimum metric path. I can only keep this and throw away all the other paths from zero to state i. Why is that okay?

The reason is if the overall final best path that you have where to go through state i in the lth stage? It will necessarily take this path on the left hand side. It cannot take anything else the reason is, from here it would have taken something to go there and if you took some other path. Clearly you can drop that and go to the minimum weight path and get a lower weight path. For this simple reason because you are starting at zero and ending at zero and you are looking at the ith stage. You only have to keep track of that minimum metric path from the zeroth state to the ith state.

So, let me repeat that argument once again. I will do it just in words. I am not going to write it down. If the minimum overall final greatest path, I mean the minimum metric path that you have where to go through state i in the l th stage. Then from zero to i, it would have definitely taken the minimum the partial minimum weight path. If it for instance the argument is by contradiction, if you say if it did not take that path then what you do? You keep the remaining part the same then here you flip from the path that it

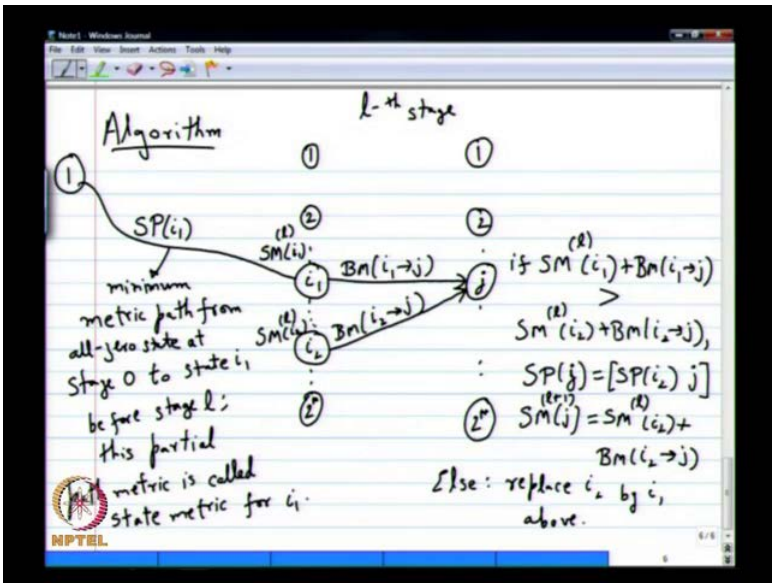
took to this minimum weight path and you would have decreased metric. That is a contradiction. So you cannot have anything else. It splits this way and the crucial reason why it really works is you are starting at zero and ending at zero. If we are not doing that then it would not work. You are starting at one point and ending at one point. So, because of that, this works in a nice way.

So I have to now do this for each state. Each state I have to keep track of only one path and that is only two power mu paths as I go along the Trellis. How you do that efficiently is the viterbi algorithm, how you do that is the viterbi algorithm. It is a no big deal in efficiency that already comes down to this efficiency. This is the idea behind the viterbi algorithm.

Of course the original paper of viterbi it is not described like this, it is something else. It is actually a proof technique used for proving something. It is strange it was not even an algorithm method point so the important person who played a key role and making it an algorithm was David Forney. In case his name, if I have not mentioned, but I think it is good to know that David Forney played a key role in writing it down as an algorithm and explaining it to everybody as an algorithm.

But I think in the world of dynamic programming and other areas it was known long before it is not invented by anybody here so that is the idea. So, that is the, that is the basic task. Let me write down a step by step kind of an algorithmic description or pseudo code for the viterbi algorithm. Only for the lth stage what you do in the lth stage. I will write that down and then you will see how you can repeat it. It is very easy to generalize form them.

(Refer Slide Time: 35:23)



So, some kind of algorithm, so the, so what happens in the l th stage have states 1 to 2 power mu. Then you also have this is the l th stage, each out of each state you are going to have two branches coming out and into each state you have two branches coming in. So, let me, take let me take the j th state here may be its two branches coming in are from j 1 and j 2 i 1 and i 2 sorry, i 1 and i 2, so there are two branches coming in.

So this will have one branch metric that is maybe we can call it B m i 1 to j and this will be another branch metric B m i 2 to j now in my algorithm upto the l minus 1 th stage. I have already done something, so which means for state i 1 I will be remembering one particular path and that path will have a partial path metric up to that point i 1. That is called the state metric for this state i 1. That is what called the state metric. Let me write that down you has the all zero state.

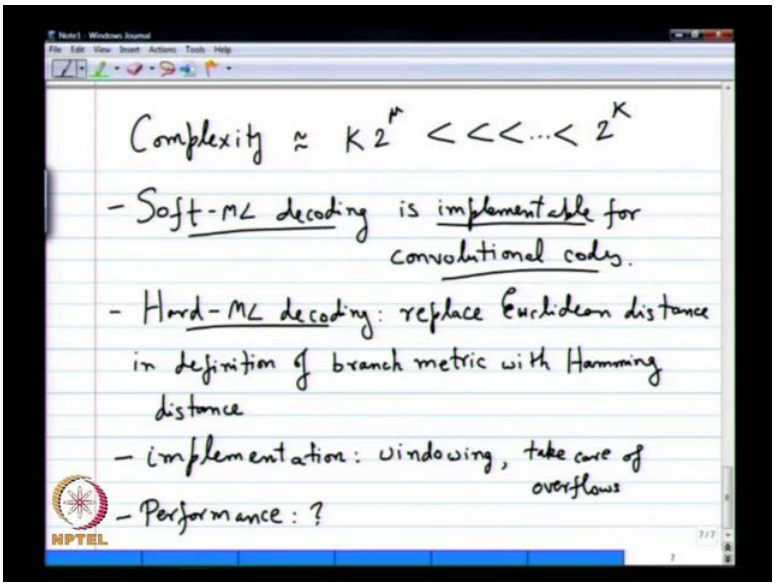
Well I am starting from one here so that is the problem, so let us say one the zero th state. This will be like a minimum metric path from all zero state at stage zero to state i 1 before stage L, it is a long word description for what is may be easy to describe easy for you to understand. That will have a partial path metric, so that metric is called the state metric for i 1. Minimum path metric path, so this partial path metric is called state metric for i 1. We need some notation, so this minimum path I will denote as what it is usually called the survivor path. I will call it S P at the S P for i 1.

It is usually called the survivor path the minimum metric path from all zero state to i 1. The metric of that path is called the state metric for i 1. I have to also index it by L, so I will put L on top so both these case will have state metrics. Both of these states will have state metrics and the branches will also have branch metrics. So, the task for this algorithm is to compute, the survivor path for j after the l th stage and it is corresponding state metric that is the only task. Once I described that, you can simply repeat the tasks stage after stage for every state and you are done. So how is that done it is done like this it is very easy so if S m l of i 1 plus B mi 1 to j is greater than S m l i 2 plus B mi 2 to j. What should I do?

Student: I two.

So, you make the survivor path of i j as survivor path of i 2 and append toward may be j. The last one will be some pj itself. You come up with this path and then add the j to it. Then state metric for j is after l plus 1 is basically what S m l of i 2 plus the branch metric i 2 j. Else what do you do? Else replace, i 2 by i 1. So the reason why I can do that is the justification, I gave before. Once I fix a particular state and a particular stage and I start at all zero and end it at all zero, I only have to keep track of the minimum way to get there. From there I can worry about what I do later on so this is the idea.

(Refer Slide Time: 41:52)



So, like I said the complexity roughly is k times two power mu, which is much smaller than two power k. It is really doable, if you have just even if you have six bits for the

state, 64 states thousand k equals thousand. You can easily do it thousand times 2 power 6 is nothing on the other hand 2 power thousand is something you cannot even imagine. So it is very big. That is the, that is the idea here.

So, what is the basic summary here soft m L decoding, decoding is implementable for convolutional codes. This is the, this is the major step and the Trellis played a crucial role. The other thing is the complexity of the Trellis because we have the definition as the convolutional code, we fix the complexity of the Trellis ahead of time. So we fix mu is six or four or five initially when we design the encoder itself explicitly, we fix it. For instance if you do the other way round, if you start with the generator matrix parity check matrix and then try and go to a shift register implementation. That is very hard it is not very easy to do those things, but since we start with an explicit shift register implementation everything becomes really simple.

So, convolutional codes are some of the most popular codes around, one of your cell phones have it, so it is used in several different applications. All the way from outer space to magnetic disk drives may be not some other things it is used everywhere. Basically, this idea of viterbi algorithm is there all over the place of course viterbi became a millionaire, started quite common other things.

So, my brief comments on the hard m L decoding, if your channel model became the binary symmetric channel and you have to now do hard m L decoding, what will you change in this algorithm description? It turns out the only thing you have to change is the definition of this branch metric. The branch metric definition here involves a Euclidean distance computation what will it be for the bsc.

It will be a hamming distance computation that is all, so you will have two bits received for the for this particular stage Sij will be two bits. You simply take the hamming distance between those two things and every other part of the algorithm remains exactly the same, nothing else you have to change except that the branch metrics instead of being Euclidean distance simply become hamming distance. That gives you hard ML decoding so that is the.

Student: Sir, how much it is going to incur for us if we do not end with exactly all zeroes.

I will make a comment on that do not worry. So hard m L decoding, you have to simply replace Euclidean distance, Euclidean distance in definition of what definition of branch metric with hamming distance that is it. So, that is a definitely doable, it can be efficient also similar in efficiency 2 power mu is the complexity.

So, the next interesting thing to worry about is some kind of an implementation detail if you want really much lower complexity. So, in fact I was saying k equals thousand is not too bad because thousand times 2 power mu is not a big number, but still thousand is a fairly large number. If you want to reduce your, reduce your what to say we all say footprints has to be very small. You cannot really may be do even thousand, but if your original length was thousand and you terminated only after thousand, you have to go all the way to thousand before you can finish up your decoding right so it seems like a kind of a anyway, it is more expensive than what you would have thought beforehand.

So, what people usually do is, they go sufficiently far into the Trellis so they would not really go all the way to thousand. They would go if your memory is mu the rough number is suppose to be four times mu or five times mu. So you go some fifty or hundred into the Trellis, how much every your secured complexity can take. So, you go as far as that and then you kind of stop there say out of all the paths, you have you simply pick the minimum path and then you go back, but you do not make decisions immediately.

So, if your memory was let us say six so you go all the way up to hundred. Then you back track and do not make decisions on the immediate fifty. You would go up go from 0 to 50 and make decisions there. Then you will allow for this fifty lag and it is called windowing. You can do windowed versions for efficient implementations. You do not have to really go all the way to thousand and then come back so you go partially and then you pick among the paths, you have the best one and then come back.

But do not make decisions towards the end of the path because you know the end of the path is not very you are not terminated, you are not very sure. So, you leave some gap there and then you decide after that and that gap suppose to be 4 mu or something is good enough 4 mu or 5 mu is supposed to be good enough based on experiments people have done, this several times and found that 4 mu 5 mu is good enough.

So, you do that and that simplifies your complexity a lot, in fact most implementations will do that only windowed versions are implemented. That is hardly any loss, but if you do not terminate you will get a huge hit in terms of you will do soft m L. That is not a problem, but then your decoder performance will be very bad because minimum distance will be very less.

So, that is the next thing we are going to talk about. One thing to worry about is implementation details. I am not going to spend too much time about on this except that windowing is an important idea in implementation. Another important idea is making sure that you do not overflow. So you have to take care of overflows because particularly if your s n r is low then this branch metrics can blow up. So what people do is, since you are always comparing paths of similar lengths they would subtract the minimum of these branch metrics and keep only the difference. That is good enough.

So, those are ideas that you can use if minimum is difficult to compute. You simply subtract a large enough number after several times. Those are all tricks that people use to keep the overflow problem done, so that is the implementation detail. More important is performance, so how is performance affected? How do you measure performance? It turns out the union bound gives you a very nice of way of thinking about how this will work. We will talk about that in the next lecture. Time is up, so we talk about performance and how minimum distance plays a role in the performance, and all that we talk about in the next lecture.