MITOCW | 7. Viterbi decoding

PROFESSOR: All right, we're going to continue talking about convolutional code. So I want to give you a quick reminder of how coding works and then talk to you about decoding. Can you hear me OK now? All right? OK.

So we talked in terms of a state diagram, but let me remind you of the shift register picture was. So we had a two-stage shift register. For this particular example, we had xn, a stream of data being fed in here. So since this is a shift register, what sits in here at time n is the previous input.

What sits here is the input from 2 times ago. And you can then feed these off and get your parity check. So take these in particular combinations and make your parity checks.

So you can have one box spitting out a p0 of n. And then you can have another box that takes these same outputs from the shift register and puts out-- let me just show them. Actually, why don't I just put it here?

You generate a bunch of party check bits. And I've shown an example on top where-- this is the same one I used last time. p0 at time n-- I should have had an n there-- is xn plus xn minus 1 plus xn minus 2. And p1, xn, and at xn minus 2. And we skip the xn minus 1.

But you can choose different coefficients there. Different coefficients will give you codes that have different properties. So the choices in the code are how many shift registers do you have, so how much memory. The constraint length here is equal to what? Constraint length in this particular example on the slide?

AUDIENCE: 3.

PROFESSOR: k equals 3? Oh, you can see it. It's the number of message bits that are involved in generating a parity bit at the maximum. It's actually not the number. It's the window over which you're taking message bits to combine to make the parity bits.

All right, so for instance, if you just had p1, you would still say that your constraint length is 3, because you're involving a window of three message bits. It's the span over which you're extending. All right, now, in terms of interpreting this, we've got the possible states of the shift register combination.

So 0, 0; 0, 1' 1, 0' 1, 1, these are the four possible states. So in general, what you have is for a constraint length k, you've got 2 to the k minus 1 states, because one of these is the input. And then the other is stored in memory.

So you've got k minus 1 stored in memory. So that's the number of states that you have. And that's how these circles are labeled here.

And then on each of the arcs, what you have is the message bit that's coming in at that time and the parity bits that are emitted. So for instance, from 0, 0, if you've got 0 here and 0 here, the only places you can go to at the next step are 0, 0, and 0, 1, because you can either-- sorry, 0, 0 and 1, 0, because you can either feet in a 0 or a 1 from here.

If you feed in a 0, then at the next state, you're still in 0. If you feed in a 1, then at the next state, you're 1, 0. So those are the only possibilities from 0, 0.

And if you had a 1 in, you would go from 0, 0 to 1, 0. And what would be your parity check bits? So if you had 1 at the input and you have the parity check expressions that I have up here, you see that what you would be emitting would be a 1 and a 1.

Is that right? If you had a 1 in the input, 0, 0, and you use these two parity bits, you generate a 1 and a 1. So that's how these arcs are labeled.

Now we said, to actually understand the convolutional code well, what you really want to do is translate the state transition diagram to a trellis diagram. This is just showing two stages of the trellis. In general, the trellis would be the state diagram unfolded over the whole time interval of interest.

So it's the same thing. It's just that we've-- let's see. We've chosen to write the states in binary counting order, so 0, 0; 0, 1; 1, 0; 1, 1. It was arranged slightly differently here. Apart from that, it's the same thing.

So we're drawing the state diagram here. We're drawing the same state diagram here except this is representing the state possibilities at time-- at one particular time. And this is the state possibilities of the next particular time. So the state transition arrows are always going from one stage to the next, all right?

So the arrow that we saw here, which takes us from 0, 0 to 1, 0 is going to take us from this box to this box. And what it emits on the way is the 1 and the-- sorry, the 1, 1. What it emits is the 1, 1. So each arc is labeled in the same way. This is just a rearrangement.

Now, the nice thing I mentioned last time, the nice thing about this is, when you have this in binary counting order, then the upper arc of the two that emanate from each box corresponds to an input of 0. The lower arc corresponds to an input of 1. So you can actually dispense with the index or with the bit that's in front of the stroke there.

So you can just make do with labeling by the parity bits. So you'll get used to that. 0 is the upward movement. And 1 is a downward movement.

So if you're thinking at the transmitter-- by the way, I hope I've made these changes well. I had an earlier version of this slide, but I changed it to work for a new set of parity bits, which meant I had to go in and change every one of these transitions. So you might see the odd bug here and there. But hopefully this is correct and consistent with the state transition diagram I showed you.

So what we're saying is now suppose you're starting off in the 0, 0 state. And you get the sequence of message bits. So 0, 1, 1, 1 is your message. And then you bring it back to the 0 state again by appending two 0's. What's the path that you traverse through the trellis?

Well, you're starting off in the 0 state. Every time you have a 0 in the message, you take the upper branch. Whenever you have a 1, you take the lower branch of the two that are available to you. So you can see very quickly how to steer through this trellis for any particular message sequence.

So this is the upper one of the two here, and then the lower one of the two here because it's a 1, then the lower one of the two here because it's a 1, and the lower one of the two here because it's a 1, and then the upper one because it's a 0, and the upper one because it's a 0. So that's your path through the trellis. It's told to you by the message bits.

You should also remember, by the way, this diagram hides a little bit, because I have just a box here for something that's actually a pair of registers. So when I just show the box, let's say, at this point, this actually has the 1 and the 0 sitting in it. So if you were just looking at this box and what was in it, if I just gave you the contents of that box, could you tell me what the input was of the previous time?

If I just told you that the contents of that box are 1 and 0, can you tell me what the input was at the previous time? Yes? It's just what got fed in, right? It's just the one that got fed in.

So this diagram is fine, but we've suppressed a little bit there. There are occasions, especially on homework and quiz problems, where you're given the contents of the shift registers. And you're asked to figure out what happened at the last time step, what message bit came in. So really, don't forget that there is a link between the two. OK, so the steering is straightforward.

Now, what's the code word that's emitted? Well, it's the parity bits that you encounter on the arcs. So on this upper arc here, you've got a 0, 0 that's emitted. So that's what you're going to emit. That's the part of the code word generated by that message bit.

And then on the lower arc, you emit 1, 1. On the lower arc, you emit 0, 1, lower arc, you emit 1, 0, and 0, 1, and then 1, 1. So that's the code word. So the set of all possible code words that you can get with this convolutional code corresponds to the set of all paths you can take through the trellis.

If you're starting at 0, 0, 0, then it's the set of all paths starting at 0, 0, 0. So let's see. Roughly speaking, can you tell me, if I've got l stages-- when I say stages, I mean time, if you want to think of these as happening on a clock. If I've got l stages here and I'm starting off with the 0 state there, for a large l, roughly how many possible paths do I have? Any thoughts?

2 to the l? Yeah. Because you see here, coming out of a box here on each stage, you've got two choices. And you've got those two choices for l stages. So you've got approximately 2 to the l possible paths.

Now, I say approximately, because well, in this case, it's fine. But now if you're a lot to start from other starting states, then you will have to take account of that. But it's of that order. It's exponential. The number of possible paths that you can have, the number of code words is exponential in the length of the trellis, right? OK, so that's a large number of code words.

Our focus, though, is going to be on decoding today. What I did so far was just review what we saw for coding. We're interested in decoding now.

So at the receiver, what you have is a knowledge of what the code is. So you have the trellis. You know what the labels are. You know that things are going to start in the zero state. And then you get your received signal.

Now, what I've shown here is that, actually, your received signal is not necessarily going to be 0's and 1's. It's probably going to be samples of some voltage, where you've got some waveform. You process it. And then you take a sample. And what you've got is a sample of some voltage.

So you're typically looking at real numbers that you then have to decide whether to call as a 0 or a 1. OK, so yeah, maybe this is 0, 0, maybe 0, 1; 0, 1, probably 1, 0; 0, 1; 1, 0, yeah? So if you were forced to choose, if you had a threshold of 0.5, for instance, and this was the range, if nominally these were supposed to be at 0 and 1, then you might actually be willing to call this one way or another.

So if I was to draw this on the real axis thinking of a voltage, so we've got 0 volts that we're expecting, 1 volt or something proportional to 1 volt that we're expecting. These are the two possible values depending on whether a 0 is sent or a 1 is sent. This is because we've coded the bits at the transmitter for physical transmission on a continuous time channel.

And then at the receiving end, we're doing some processing and extracting samples, right? But because of noise, what might happen is that you get samples anywhere around the 0 or anywhere around the 1, depending on the particular transmission instance. It'll vary from one instant to the next. And if the noise is really bad, then of course, what started off as a 0 here with the noise added to it, by the time you sample it, might fall in a region where you call it a 1.

So there is an intermediate step. And very often, you have access to that. And then you've got to figure out how to do your decoding. All right, is this the same slide? Or does it say anything different? OK.

So what are we going to do now? We're going to, of all the paths available to us, we're going to try and find the path along which the emitted parity bits come closest, in some sense, to the sequence of samples here. That's, if you were doing minimum distance, in some sense, that's what you'd want to do. If you believed that errors further away from 0 are less likely than errors close to 0, then you would want to have a reconstructed set of parity bits along whatever path you choose to come close to the values there.

Now, it turns out that it's actually simpler initially to think of first making a decision to call these 0's or 1's and then finding a path through this that comes closest to the 0, 1 sequence that approximates the voltage samples that you've actually got. So we make a distinction between what's called hard decision decoding and soft decision decoding. So in soft decision decoding, which we'll talk about later, you preserve those voltage samples.

And you don't mess with them. But in hard decision decoding, at each stage, you just make a decision, on each sample, decide to call it a 0 or a 1 and proceed from there. OK, so which do you think is likely to get you better performance if you're doing the optimal thing after that?

AUDIENCE: Soft.

PROFESSOR: The soft? Yeah. Because when you make the decision at one stage, you're throwing away some information. You're not taking account of how these samples might relate to each other. You're treating that sample in isolation.

If you know that what you're going to end up with is a code word that corresponds to a path through here, then there is additional information that actually couples the different numbers you're getting across there. And so you have a hope of doing better with soft decision decoding. So postpone the decision until later.

But you pay a cost. Or you could pay a cost for that, because you've got to deal, for instance, with the real numbers and all of that. So hard decision decoding can simplify your processing.

So what you'll say is, I'll just make a choice here. So I'll call this 0, 0; 0, 1; 0, 1, and so on, and then look for a path through the Trellis along which the emitted party bits come closest to what I've approximated that sequence of samples by. So we're talking about having distance again. And minimum timing distance is going to give us the most likely path, given that you've already committed to interpreting the received samples as 0's or 1's.

So what you might imagine is, OK, you've got this received sequence. You've got a tabulation of all the possible paths through the trellis and the parity bits that are emitted along those paths. Each path corresponds to a different message. What you actually have here-- let's see.

We've got 12 bits here, because in addition to the message, I'm appending a 0, 0 to each one, which forces the trellis back down to the 0 state. So what I'm actually doing here is, I actually have a message that's this followed by two 0's.

And so if you're trying to connect these two columns with the trellis that I had on the previous page, that's how you should think about it. But with any particular message, you navigate up and down on the trellis. This particular one, you navigate up, up, down, up.

And that's the sequence that's generated. That's the code word that you would expect if this was the message. What you'll do is you'll search over all possibilities. At least that's one way to do this, in principle, search over all possibilities for the code word here that's closest to what you received.

The trouble is, that's a lot of code words. That's a lot of code words. So this can quickly get out of hand. If you've got long sequences, which is exactly where you want to do convolutional coding, you've got a very long table. So you really want to find an efficient way to do this matching.

I just wrote down the Hamming distance that happens to hold for what is the message that was actually sent, which also will be the message that you will recover at the receiver if you do the optimal thing and you don't get fooled by the errors. So what I've got here as a 2 is the Hamming distance between the code word here and the received message. So if that 2 was the smallest one in that whole stack-- I haven't filled them all out-- then that's the one that you would call.

OK, so a much cleverer way of doing this was invented by the Viterbi, who did his bachelor's degree here, then moved to the West Coast. He was very involved in the JPL program. But he was also a founder of, well, a succession of companies, but most recently, Qualcomm. And he's a big friend of the department. He's on our visiting committee. Or he has served time with the visiting committee.

So this is an algorithm that he developed in the early days. And we're going to talk about it. I think I'll put it all up on the slide. And then let's talk.

All right, there is a lot there. I don't want you to struggle through that. Let's talk about it here.

And when we're done, I think what's up there will make sense. That's for you to refer to from the slides later. And it's my little checklist to know that I've spoken about everything, but don't try and navigate that just yet.

So here is what Viterbi says. He says, we're starting off from some initial state. This is the zero state. At an intermediate state, intermediate time-- sorry, I shouldn't say state. I meant stage or time. At an intermediate stage, we have these four possibilities.

What I'm going to do for a given received sequence-- and let me actually put the received sequence I'm going to use in this example. We've got a received sequence. Let's say it's 0, 0 on the first stage, and then 0, 1 on the second stage, and 0, 1; 1, 0--

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah? Did someone say something? No? I thought I heard-- OK. We'll park the question for now and check in again later.

OK, here is the received sequence. What we're trying to do is find a path through the trellis where the emitted bits come closest to this in Hamming distance. Here is what Viterbi proposes to do. He says, from the starting state, let's find the optimum path to each of these states at any particular time, let's say a time i here. Here is time i.

Let's find the optimum path to these with the associated minimum cost. So let's assume that I have that. So what I'm going to do is, for each of these, I'm going to put in some number. This won't be exactly the notation that we have on this slide, but it's streamlined.

These p's correspond to what we call path metrics. And I should actually have an index i here to tell you that I'm doing this at time i, but I'll just leave that off. p sub 1 is cost along optimal path to state one.

OK, so assume that, magically, you've computed the optimal path to this starting from the initial time. So what that means is maybe you've gone down to a particular stage here. You've gone down further. And then maybe you've come up here. Maybe that's the optimum path.

So what you're going to keep track of is, for each of these times, for each of these states, what's the cost, the optimal cost? Or what's the cost along an optimal path there? OK, now what do I mean by cost? I just mean Hamming distance between what I received and the parity bits emitted along the way up to that point.

So Viterbi is going to keep track of this for every stage as you step along and for every one of these states. Now, let's take this particular one. If I'm transitioning to state one here, let's see. This emits a 0, 0 if I go from-- let's go back to our trellis. I should draw this up, actually. But if I go along the top, I am at 0, 0. What other state comes into the top one?

This comes in. And this emits 1, 1. So what's the cost I incurred if I take the upper path? The cost is just the Hamming distance between the 1, 0 that I received and the 0, 0 that I have here.

So there is a cost of 1. Let me, again, use colors for costs here. What's the cost I incur if I instead come to this point from p2, again cost of 1?

So this is the generic picture. What you're going to do is, you're having this-- you have this at any stage. You compute the branch costs and continues. And now suppose p1 was equal to 3 and p2 was equal to 4, and you wanted to figure out what's the shortest way, what's the minimum-cost way to get from the origin to this point, to p1 at the next time instant? What's the minimum cost? And what's the root?

AUDIENCE: [INAUDIBLE]

PROFESSOR: If you came from here, you've incurred a cost of 3 up to this point. And you're adding an additional cost of 1. You'll end up with a cost of 4 to get to here. If you get to hear from p2, well, you've incurred a cost of, let's say, 4 up to this point. And now you're going to incur an additional cost to bring it to 5.

So your best route to p1 at this time is to come from p1 at this time using this arc. So if you've built it up at a particular stage, then it's actually very straightforward to figure out what you should do at the next stage. So let me now start putting some time indices on this. This would be p1 at time i is equal to 3. p1 at time i plus 1 is equal to 4. This is p2 at time i.

So you can actually forget about this arrow, because there is no way you're going to use that arrow. Whenever you come to this stage at this time, you're going to come via the upper branch. So at every stage, you're going to do this. And it's a very simple calculation. So now we've got slightly more elaborate notation up on the board, but I hope you have the general idea.

This is an instance, by the way, and a way of thinking about such problems that's referred to as dynamic programming. It works for these sorts of rooting problems. We're rooting ourselves along a trellis where the total cost of taking a path is the sum of the costs at every stage.

So the total Hamming distance between the bits you emit along the way and the bits that you've received is made up of the Hamming distance between the branch here and the piece you've received here plus the Hamming distance on the branch here plus the piece you between the branch here and the branch-- sorry, the received segment over there, and so on. So the total Hamming distance is made up of the sum of the Hamming distances along the way. In all such situations where you've got a total cost that's additive over the path and you've got to do an optimization, dynamic programming is something you can think of.

And the idea we've used here is actually one that you might come at naturally. If you found the best way from here to the Student Center, and it happens to go through Lobby 7, what's your best way from here to Lobby 7? Presumably, it's going to be the section of the path that you would take to the Student Center that passes through Lobby 7, because if you had a better way to get to Lobby 7, you would have used it to get to the Student Center via Lobby 7. It's just that idea. So on an optimum path where the costs are additive, it must be the case that the optimum path to an intermediate point is exactly the section of the optimum path to the point that you're looking at, a simple idea.

OK, so let's go back to the more formal way it's written up here on the slide. So we talk about the branch metric. That's just the Hamming distance that we computed here for the branch.

It's the difference between what we received and what would be transmitted if you moved along that arc. So that's the branch metric. It's the piece contributed by the branch.

This is the notation we've used. We've already talked about this, that you could either do a hard decision kind of rule where you've already set these to 1's and 0's. Or you could stick with the original samples.

If you've already converted them to 1's and 0's, there is a natural notion of distance, which is the Hamming distance. And there is a probabilistic reason why you would want to do that. So we're sticking to the Hamming distance setting right now, so hard decision decoding.

And the path metric, this is a more elaborate notation than what I have here. So instead of a subscript to denote the state, this has got the state index here, and the time index here, and pm for path metric instead of just p, but it's the same thing. So for each state and at each stage, so for each of the four states and for each of the stages, you're going to compute this.

And you can the path metric up to time i is the smallest sum of the branch metrics over all the sequences that will get you to that place. And if you assume you have that at any stage, then the computation that takes you to the next stage is an easy one. I think I've said all this. You can come back to it later. So let's actually just step through this.

So we're at some intermediate stage. We're just doing the same thing I had on the board. I'm doing it again in pictures here so you get to think about it one more time. Suppose we've received 0, 0.

We first label each of the arcs here by the Hamming distance between the bits we'd emit along the arc and the bits we've actually received, so Hamming distant 0 here between what we would emit and what we received, Hamming distance 2 here on this arc, Hamming distance 2 on this arc, and so on. So the red numbers here are below the top two just are the costs on the arcs.

Actually, I don't like the last line of that slide. So you may want to strike that. We're not going to really be talking about the most likely branch metric. We're only going to make decisions once we're done with the whole path.

So we assume at some stage, that we have the path metrics up to that point. And then we do the computation that I just talked about. So let's see. In this particular case, what would be the path metric value in this position?

It's the same thing we did already, but just another chance to look at it. What would be the value of the path metric there? 3? Because you can either do 3 plus 1 on that arc.

Or you can do 2 plus 1 on this-- sorry, 3 plus 1 on this arc for a cost of 4 or 2 plus 1 on this arc for a cost of 3. So it should be a 3 there. And this is the arc that you would pick.

And similarly, you can do it for all of them. So once you have one stage, you can fill out the next stage completely and then keep track of the arcs that lead you there. And at some point, you'll-- at each stage, actually, you can prune away things that you're not going to be using.

So you're never going to use that edge. So you don't have to worry about it anymore. You're never going to use this edge.

There are also stages where you might have two different ways of getting to a box and incurring the same cost. And then it doesn't matter which of them you pick. You can pick one or the other. In terms of the overall cost, it's not going to matter.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah, so what you're saying is that, if there isn't a unique way back, then you're not certain. Well, you're never certain here in this business. You're just doing your best guess.

So what you would do when you commit to one particular choice when there are two equally likely costs is you're saying, the probability of error is going to be the same with this choice as it will be with the other choice. And in the end, that's what we have for the metric. It is unsatisfying, perhaps.

Now, there are schemes where you'd keep a list of the possibilities and try and do something with that, because maybe there is some higher level thing that would help you disambiguate between possibilities, but that would complicate the processing. But as far as this goes, you make a choice. And you move on.

So you can imagine actually working through this whole thing. If you knew you were starting from the zero state, you'd start off with a zero cost there. So you're at infinity here, which is going to force all the optimal paths to come from 0. And then you'll continue from there.

So I just wanted to show you a few things that come with this. Actually, I might have shown you everything I want on that. So let's just go back to the soft decision decoding. So how might things differ if you go back to soft decision decoding? So let me find that slide.

The Viterbi algorithm doesn't care how you come at these costs. The Viterbi algorithm is just dynamic programming on this trellis. It finds you the minimum-cost path. It's up to you how you decide what cost to attribute to an edge.

So the question is, are there other costs that you might have come up with? Well, if the received sequence has been translated already to 0's and 1's, then Hamming distance is the natural thing. But if you're keeping particular numbers there, then it turns out that you might want to do things differently.

So suppose at a particular stage, what you got was not-- let's see. Did I put numbers up there? Suppose it wasn't 0 and 1, but it was some particular numbers, let's say, 0.3 and 0.7.

And you had actually translated them to a 0 and a 1 in your hard decision decoding. If you decide not to do that, but to stick with these numbers, then what you have is the task for any particular edge that you're traveling on of finding the distance between the parity bits you would emit on that edge and the samples that you have here.

It turns out that a very widely used cost for soft decision decoding is the sum of squared differences. So what you would have is 1 minus 0.3 squared plus 1 minus 0.7 squared. So it would be the first bit that you emit on this arc minus the first sample that you got squared error plus the second sample, the second bit that you would emit on this arc minus the second sample, whole thing squared.

If there was another arc that was a 1 and 0 arc, then what you would computers 1 minus 0.3 squared plus 0 minus 0.7 squared. So it's just a different way of coming up with the cost. The rest of the Viterbi algorithm is exactly the same. the navigating through the trellis is exactly the same.

It turns out that there is a logic and a reason behind this particular metric for situations where your voltage samples are distributed in the familiar bell-shaped fashion here, what's called a Gaussian distribution. We'll talk about it more next time.

So what we're saying is that if you send a 1, you get a spread of possible values. The probability of your values falling in some particular range here can be computed by the area under this particular curve. It's got an analytical expression.

So this is the most likely spot. But there is certainly probabilities of falling in any particular interval here. Well, what does the Gaussian distribution look like? We'll talk more about it. the essential part of it is e to the minus-- let's see. Let me put some labels here. This is where it's centered. Let me call it mu.

And let's say x is the value along the axis. So we'll have e to the x minus mu all squared divided by some normalizing parameter. Well actually, let's just call it capital N. Think of capital N as a noise variance.

Actually, let me just call it N sub 0 so you don't think it's a counting number. Think of it as a noise variance. So the larger that N is, would you spread out more or less here? Well, just from the fact that I call it a variance, maybe you would guess that, if N is larger, you're going to spread out more.

Well, in this kind of setting, when you take log likelihoods-- you've seen that computation in the chapters-- what ends up appearing in your cost criterion is x minus mu squared. So it's the squared difference from the mean that you want to be looking at. And that's exactly why, in that kind of setting, this is what you end up choosing as your cost metric.

But once you're done computing those metrics, the rest of the Viterbi algorithm is the same. So once you have the convolutional coding in hand, you know how to decode, you can start to do some comparisons of how these different codes perform. There is an extensive discussion in the chapter. Let me just give you some highlights here.

OK, so what are we plotting here? What we're saying is we send a whole bunch of message bits through the channel. And then we decode at the other end. And what we're talking about is-- let's see.

Here is the binary symmetric channel. Here is the error probability on the channel. You can see to well what this is, but it's the-- why is that chopped off? It's the probability of error overall end to end, not of the channel, but after you've done your coding and decoding.

Let's see. Do we recognize any of these codes? Here is the uncoded case where basically you're exposing the stream directly to the error on the binary symmetric channel. We expect higher errors when we have higher probabilities of flipping a bit on the channel.

So this is the uncoded case. What does the Hamming code do? That's a Hamming code probability, 0.74. So the Hamming code performance, you can see here end to end what it looks like. What's the rate that goes with that?

What's the rate of that Hamming code? 4 over 7, right? Because n is the number of bits in the message-- sorry, in the code word. And 4 is the number of bits in the message, so 4 over 7, something over 1/2.

Let's see. Do we know what that code might be? Any codes you know about that take 4 message bits and pad them to 8 code word bits? You've seen at least one such code.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Sorry?

AUDIENCE: Rectangular parity?

PROFESSOR: Rectangular parity, right? If you didn't have that corner parity bit, but you just did the rows and columns, then you'd arrange 4 bits in a 2 by 2 pattern and then have 4 parity bits. So that's a rectangular parity. That's rate 1/2.

What this denotes is a convolutional code. It's actually the code we've been-- sorry, no, this one is the code we've been looking at. So let me explain to you what that notation means when you're reading the chapter. This code is represented as-- the one we've been talking about as represented as this. So what this is is the constraint length. And what this is is just to tell me that the generator bits I used for my party generation correspond to the binary representation of 7 and the binary representation of 5.

So remember that for my first parody bit, I chose xn plus xn minus 1 plus xn minus 2. I picked all three of them. For my second parity bit, I took xn plus xn minus 2. I skipped the middle one. So the notation that's used to denote a convolutional code with these two generators is, just for compactness, the 7, 5 there.

Let's see. Is this redundant, the k, the value of k? Could you have figured out what the constraint length is?

AUDIENCE: Yeah.

PROFESSOR: Yeah. It's already staring at you here what the constraints length. So this is a little redundant. It's just that it's a convenient way to distinguish a convolutional code from a Hamming code.

Now, we have to be a little careful comparing these codes, because the rates are all a little different. Here the rate is 1/2. What's the rate for this convolution for the two convolution codes here, the 3, 7, 6 and the 3, 7, 5?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Sorry, what's--

AUDIENCE: 1/2 [INAUDIBLE].

PROFESSOR: 1/2, right? So the rate is 1 over the number of parity bits you're generating per message. One message bit, r parity bits, therefore, a rate of 1 over r.

So these are rate one-off codes, just like the rectangular case. This is constraint length 4. And you can actually write out what that would be there.

How big is the trellis for the constraint length 4 case? How many stages? This last one down there? 8, right? Constraint length 4, that means k equals 4, 2 to the k minus 1, so 2 cubed states. So there is 8 states that we're talking about.

The Cassini convolution code that I showed you last time had a constraint length of 15. So how many states there on the trellis? 2 to the 14, that's a lot of states.

So that's a lot of computation there happening. And actually, there is no hope of that having been done if it wasn't for the Viterbi algorithm. All right, we'll talk more about comparison between these codes next time.

Sorry, I should do one more thing here. I did talk last time about this notion of free distance. Let's just stare at this a second. We said the free distance was the weight of the smallest non-zero codeword. And it gave you a handle on the performance of the code. It was the Hamming distance for the set of code words you could generate between 0, 0 here and 0, 0 there.

Can you see by inspection here what might be a candidate free distance here? I think what we had last time was 5, right? This is the 1, 1; 0, 1; 1, 1. And we'll pick up 1, 2, 3, 4, 5, a weight of 5. And it turns out there is no other path that's smaller.

So the performance of this particular code is indicated by that number 5. It tells you that you can correct two bits. But actually, it tells you much more than you would typically try and extract from a typical block code where you would say, if there is Hamming distance 5, you only can correct two codes. Here you've got message bits that go on for a long time, thousands of bits.

So what this is telling you is that in a duration that's of the order of five or six message bits, you can, with this scheme, correct up to two bits. You can have bursts of errors that are very frequent and correct them with this Viterbi decoding. So the free distance is an important notion. So when you do the examples in recitation tomorrow, please look out for what the free distance is for your codes and compare with what we have here.