

Problem Set 3

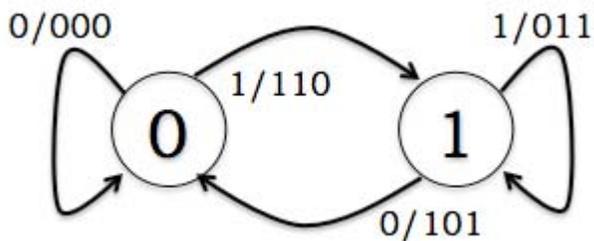
Your answers will be graded by actual human beings (at least that's what we believe!), so don't limit your answers to machine-gradable responses. Some of the questions specifically ask for explanations; regardless, it's **always a good idea to provide a short explanation for your answer**.

Before doing this PSet, please read Chapters 7 and 8 of the readings. Also attempt the practice problems on this material.

Due dates are as mentioned above. Checkoff interviews for PS2 and PS3 will be held together and will happen between October 4 and 8.

Problem 1.

Consider a convolutional code with three generators, and with state transition diagram shown below:



A. Deduce the equations defining the three parity bits.

(points: 1.5)

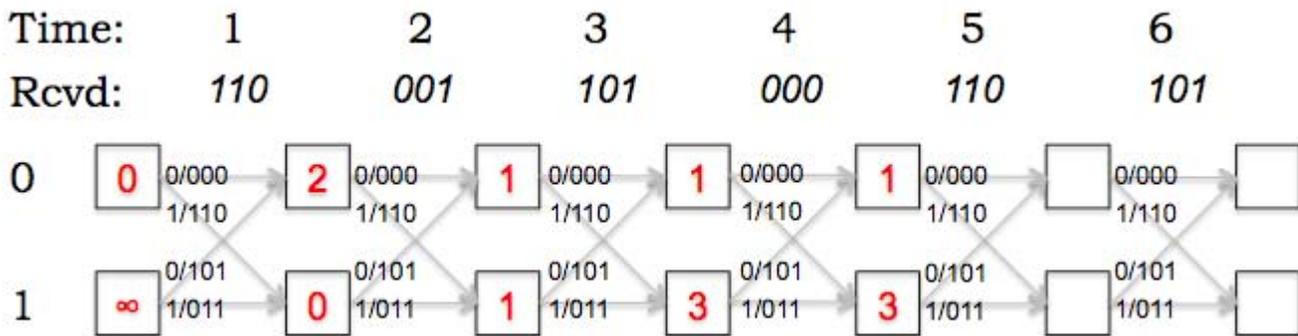
B. What are the constraint length and code rate for this code?

(points: 0.5)

C. What sequence of bits would be transmitted for the message "1011"? Assume the transmitter starts in the state labeled "0".

(points: 0.5)

The Viterbi algorithm is used to decode a sequence of received parity bits, as shown in the following trellis diagram:



D. Determine the path metrics for the last two columns of the trellis.

(points: 0.5)

E. What is the most likely state of the transmitter after time 5?

(points: 0.5)

F. What is the most likely message sequence, as decoded using the trellis above? And how many bit errors have been detected for this most likely message?

(points: 0.5)

G. What is the free distance of this code?

(points: 0.5)

Problem 2.

Let $\text{conv_encode}(x)$ be the resulting bit-stream after encoding bit-string x with a convolutional code, C . Similarly, let $\text{conv_decode}(y)$ be the result of decoding y to produce the maximum-likelihood estimate of the encoded message. Suppose we send a message M using code C over some channel. Let $P = \text{conv_encode}(M)$ and let R be the result of sending P over the channel and digitizing the received samples at the receiver (i.e., R is another bit-stream). Suppose we use Viterbi decoding on R , knowing C , and find that the maximum-likelihood estimate of M is \hat{M} . During the decoding, we find that the minimum path metric among all the states in the final stage of the trellis is D . Then, D is the Hamming distance between two quantities. What are the two quantities? Consider both the cases when the decoder corrects all the errors and when it does not. **Explain your answer.** (This question requires some careful thinking.)

(points: 2)

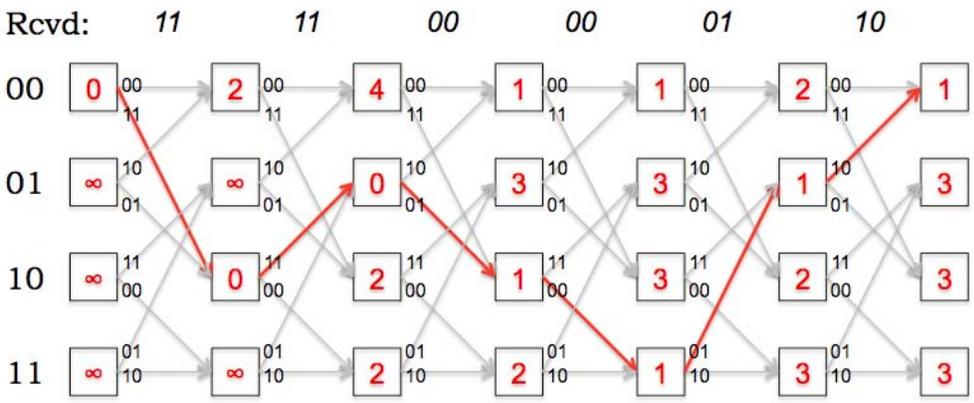
Problem 3.

A technique called **puncturing** can be used to obtain a convolutional code with a rate higher than $1/r$, when r generators are used. The idea is to produce the r parity streams as before and then carefully omit sending some of the parity bits that are produced on one or more of the parity streams. One can express which bits are sent and which are not on each stream as a vector of 1's and 0's: for example, on a given stream, if the normal convolutional encoder produces 4 bits on a given parity stream, the vector (1 1 0 1) says that the first, second, and fourth of these bits are sent, but not the third, in the punctured code.

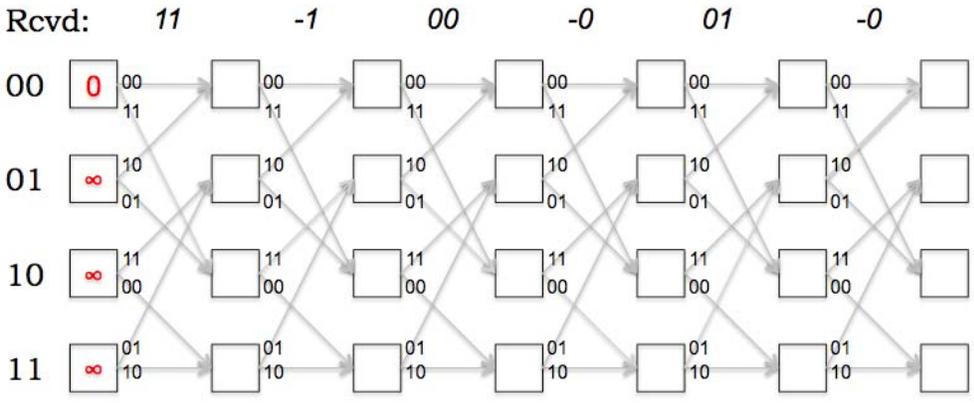
- A. Suppose we start with a rate-1/2 convolutional code without puncturing. The encoder then uses the vector (1 0) on the first parity stream and the vector (1 1) on the second one: that is, it sends the first parity bit but not the second on the first parity stream and every parity bit produced on the second one. What is the rate of the resulting convolutional code?

(points: 0.5)

- B. When using a punctured code, missing parity bits don't participate in the calculation of branch metrics. Consider the following trellis from a transmission using $K = 3, r = 1/2$ convolutional code that experiences a single bit error:



Here's the trellis for the same transmission using the punctured code from part (A) that experiences the same error:



Fill in the path metrics, remembering that with the punctured code from part (A) the branch metrics are calculated as follows:

$$\begin{aligned}
 \text{BM}(-0, 00) &= 0 & \text{BM}(-0, 01) &= 1 & \text{BM}(-0, 10) &= 0 & \text{BM}(-0, 11) &= 1 \\
 \text{BM}(-1, 00) &= 1 & \text{BM}(-1, 01) &= 0 & \text{BM}(-1, 10) &= 1 & \text{BM}(-1, 11) &= 0
 \end{aligned}$$

where the missing incoming parity bit has been marked with a "-". Use a text representation of the matrix (a 2-D array of numbers) when typing in the answer.

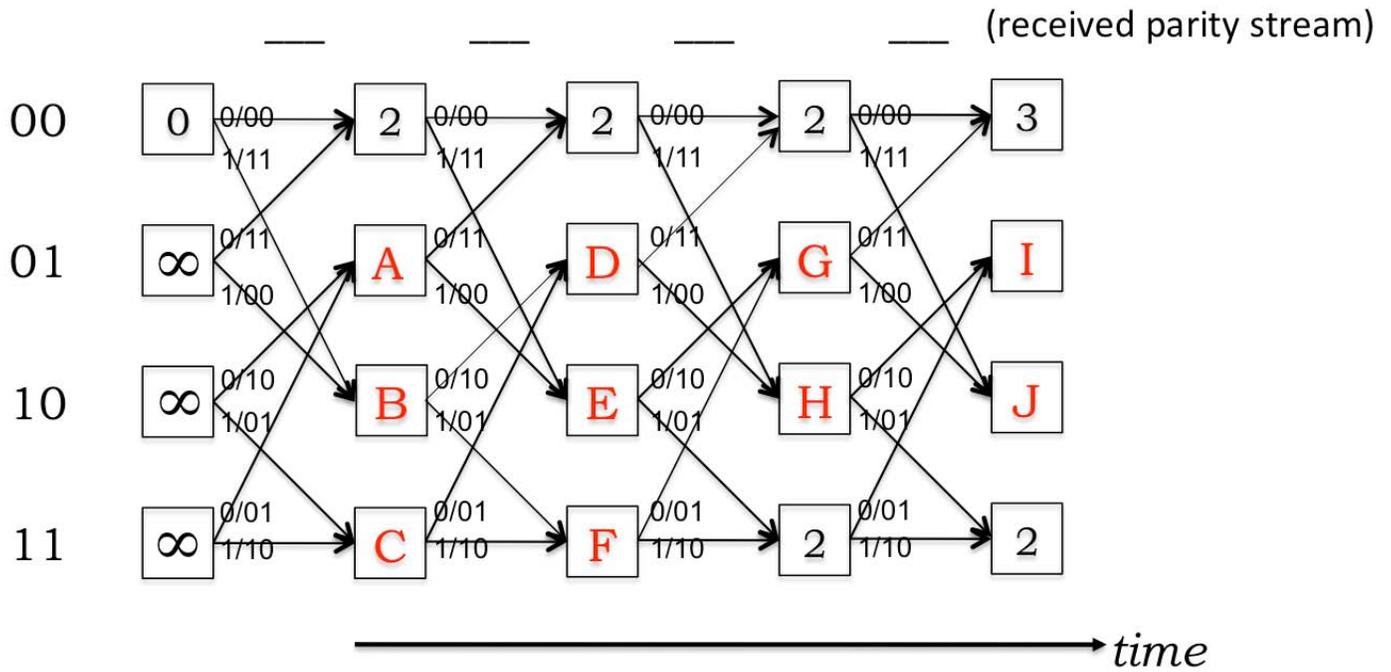
(points: 1)

C. Did using the punctured code result in a different decoding of the message?

(points: 0.5)

Problem 4.

Ben enters the 6.02 lecture hall just as the professor is erasing the convolutional coding with Viterbi decoding example he has worked out, and finds the trellis shown below with several of the numbers and bit strings erased (yeah, the professor is strangely obsessive about how he erases items!).



Your job is to help Ben piece together the missing information by answering the questions below. Provide a brief explanation.

- a. The constraint length of this convolutional code is ____.

(points: .25)

- b. The rate of this convolutional code is ____.

(points: .25)

- c. Assuming hard-decision Viterbi decoding, write the missing path metric numbers inside the boxes in the picture above. (Work carefully and check your work to avoid a cascade of errors!)

(points: 1)

- d. What is the **received parity stream**, which the professor has erased? Write your answer in order from left to right in the space below.

(points: 1)

- e. What is the maximum-likelihood message returned by the Viterbi decoder, if we stopped the decoding process at the last stage in the picture shown? Explain your answer below.

(points: 1)

Problem 5.

Dona Ferentes is debugging a Viterbi decoder for her client, The TD Company, which is building a wireless network to send gifts from mobile phones. She picks a rate-1/2 code with constraint length 4, no puncturing. Parity stream p_0 has the generator $g_0 = 1110$. Parity stream p_1 has the generator $g_1 = 1xyz$, but she needs your help determining x,y,z , as well as some other things about the code. In these questions, each state is labeled with the most-recent bit on the left and the least-recent bit on the right.

These questions are about the state transitions and generators.

- a. From state 010, the possible next states are ___ and ___.

(points: 0.25)

- b. From state 010, the possible predecessor states are ___ and ___.

(points: 0.25)

- c. Given the following facts, find g_1 , the generator for parity stream p_1 . g_1 has the form $1xyz$, with the standard convention that the left-most bit of the generator multiplies the most-recent input bit.

Starting at state 011, receiving a 0 produces $p_1 = 0$.

Starting at state 110, receiving a 0 produces $p_1 = 1$.

Starting at state 111, receiving a 1 produces $p_1 = 1$.

(points: 1)

- d. Dona has just completed the forward pass through the trellis and has figured out the path metrics for all the end states. Suppose the state with smallest path metric is 110. The traceback from this state looks as follows:

000 ← 100 ← 010 ← 001 ← 100 ← 110

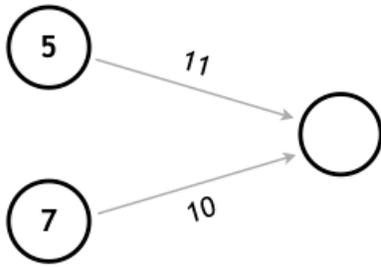
What is the most likely transmitted message? Explain your answer, and if there is not enough information to produce a unique answer, say why.

(points: 0.5)

During the decoding process, Dona observes the voltage pair (0.9, 0.2) volts for the parity bits p_0, p_1 , where the sender transmits 1.0 volts for a "1" and 0.0 volts for a "0". The threshold voltage at the decoder is 0.5 volts. In the portion of the trellis shown below, each edge shows the expected parity bits p_0, p_1 . The number in each circle is the path metric of that state.

- e. With **hard-decision decoding**, give the branch metric near each edge and the path metric inside the circle.

received voltages: .9, .2



(points: 0.5)

- f. Timmy Dan (founder of TD Corp.) suggests that Dona use soft-decision decoding using the squared Euclidean distance metric. Give the branch metric near each edge and the path metric inside the circle.

(points: 0.5)

Python Task 1: Viterbi decoder for convolutional codes using hard-decision decoding

Useful download links:

PS3_tests.py -- test jigs for this assignment

PS3_viterbi.py -- template file for this task

As explained in lecture and Chapter 7, a convolutional encoder is characterized by two parameters: a constraint length κ and a set of r generator functions $\{G_0, G_1, \dots, G_r\}$. The encoder processes the message one bit at a time, generating a set of r parity bits $\{p_0, p_1, \dots\}$ by applying the generator functions to the current message bit, $x[n]$, and $\kappa-1$ of the previous message bits, $x[n-1], x[n-2], \dots, x[n-(\kappa-1)]$. The r parity bits are then transmitted and the encoder moves on to the next message bit.

The operation of the encoder may be described as a state machine, as explained in Chapter 7. The generator functions corresponding to the parity equations can be described compactly by simply listing its κ coefficients as a κ -bit binary sequence, or even more compactly (for a human reader) if we construe the κ -bit sequence as an integer, e.g., $G_0: 7, G_1: 5$, etc. Here, "7" stands for the generator "111" and "5" for the generator "101", which in turn work out to

$x[n] + x[n-1] + x[n-2]$, and $x[n] + x[n-2]$, respectively. We will use this integer representation in this task.

As explained in Chapter 8, the Viterbi decoder works by determining a **path metric** $PM[s, n]$ for each state s and bit time n . Consider all possible encoder state sequences that cause the encoder to be in state s at time n . In hard-decision decoding, the most-likely state sequence is the one that produced the parity bit sequence nearest in Hamming distance to the sequence of received parity bits. Each increment in Hamming distance corresponds to a bit error. In this task, $PM[s, n]$ records this smallest Hamming distance for each state at the specified time.

The Viterbi algorithm computes $PM[\dots, n]$ incrementally. Initially

$$PM[s, 0] = 0 \text{ if } s == \text{starting_state} \text{ else } \infty$$

The decoder uses the first set of r parity bits to compute $PM[\dots, 1]$ from $PM[\dots, 0]$. Then, it uses the next set of r parity bits to compute $PM[\dots, 2]$ from $PM[\dots, 1]$. It continues in this fashion until it has processed all the received parity bits.

The `viterbiDecoder` class contains a method `decode`, which does the following (inspect its source code for details):

1. Initialize $PM[\dots, 0]$ as described above.
2. Use the Viterbi algorithm to compute $PM[\dots, n]$ from $PM[\dots, n-1]$ and the next r parity bits; repeat until all received parity bits have been consumed. Call the last time point N .
3. Identify the most-likely ending state of the encoder by finding the state s that has the minimum value of $PM[s, N]$.
4. Trace back along the most likely path from start state to s to build a list of corresponding message bits, and return that list.

In this task you will write the code for three methods of the `ViterbiDecoder` class, which are crucial in the steps described above. Once these methods are implemented, one can make an instance of the class, supplying κ and the parity generator functions, and then use the instance to decode messages transmitted by the matching encoder.

The decoder will operate on a sequence of received voltage samples; the choice of which sample to digitize to determine the message bit has already been made, so there is one voltage sample for each bit. The transmitter has sent a 0 Volt sample for a "0" and a 1 Volt sample for a "1", **but those nominal voltages have been corrupted by additive Gaussian noise zero mean and non-zero variance**. Assume that 0s and 1s appear with equal probability in the transmitted message.

`PS3_viterbi.py` is the template file for this task.

You need to write the following functions:

```
number = branch_metric(self, expected, received)
```

expected is an r -element list of the expected parity bits (or you can also think of them

as voltages given how we send bits down the channel). *received* is an r -element list of actual sampled voltages for the incoming parity bits (floats in the range $[0,1]$). In this task, we will do hard-decision decoding; that is, we will digitize the received voltages using a threshold of 0.5 volts to get bits, and then compute the Hamming distance between the expected sequence and the received sequences. That Hamming Distance will be our branch metric for hard-decision decoding.

You may use `PS3_tests.hamming(seq1, seq2)`, which computes the Hamming distance between two binary sequences.

```
viterbi_step(self, n, received_voltages)
```

update `self.PM[... , n]` using the batch of r parity bits and `self.PM[... , n-1]` computed on the previous iteration. In addition to making an entry for `self.PM[s, n]` for each state s , keep track of the most-likely predecessor for each state in the `self.Predecessor` array. You'll find the following instance variables and functions useful (**please read the code we have provided to understand how to use them -- learning to read someone else's code is a useful life skill :-)**)

```
self.predecessor_states
self.expected_parity
self.branch_metric()
```

```
s = most_likely_state(self, n)
```

Identify the most-likely state of the encoder at time n by finding the state s that has the minimum value of `PM[s, n]`. If there are several states with the same minimum value, choose one arbitrarily. Return the state s .

```
message = traceback(self, s, n)
```

Starting at state s at time n , use the `self.Predecessor` array to trace back through the trellis along the most-likely path, determining the corresponding message bit at each time step. Note that you're tracing the path *backwards* through the trellis, so that you'll be collecting the message bits in reverse order -- don't forget to put them in the right order before returning the final decoded message at the result of this procedure. You can determine the message bit at each state along the most likely path from the state itself -- think about how the current message bit is incorporated into the state when the transmitter makes a transition in the state transition diagram. Be sure that your code will work for different values of `self.K`. Also, note that we are storing these states as integers (e.g. state '101' is stored as 5); you may find the `>>` (right shift) operator, or the `bin` function useful in extracting specific bits from integers.

The testing code at the end of the template tests your implementation on a few messages and reports whether the decoding was successful or not. **Note that passing these tests does not guarantee that your code is correct, and your code may be graded on different test cases; you should make fairly certain that your code will work for arbitrary tests (i.e., not just the ones in the file) before submitting.**

File to upload for Task 1:

(points: 8)

Here s the debugging printout generated when Alyssa P. Hacker ran her correctly written code for the (3, (7,6)) convolutional code:

Final PM table :

0	2	3	2	2	3	2
inf	inf	1	2	3	2	4
inf	0	3	2	3	3	4
inf	inf	1	2	2	4	4

Final Predecssor table :

0	0	0	1	0	0	1
0	2	2	3	3	3	2
0	0	0	1	1	0	0
0	3	2	3	2	2	2

Each column shows the new column added to the PM and Predecessor matrices after each call to `viterbi_step`. The last column indicates that the most-likely final state is 0 and that there were two bit errors on the most-likely path leading to that state.

- A. Consider the most-likely path back through the trellis using the Predecessor matrix. At which processing steps did the decoder detect a transmission error? Call the transition from column 1 of the debugging printout to column 2 Step 1, from Column 2 to 3 Step 2 and so on.

(points: 0.5)

Ben Bitdiddle ran Task #1 and found the following path metrics at the end for the four states:

```
[ 6200.  
 6197.  
 6199.  
 6199.]
```

- B. What is the most likely ending state? And what can you say about the number of errors the decoder detected? Were they all corrected?

(points: 0.5)

Here s the debugging output from running decoder tests with a 500000-bit message a $K = 3$ ' code with generators 7 (111) and 6 (110).

```
**** ViterbiDecoder: K=3, rate=1/2 ****
BER: without coding = 0.006200, with coding = 0.000218
Trellis state at end of decoding:
[ 6200.
  6197.
  6199.
  6199.]
```

Please answer the following questions based on the above output for the $K = 3$ code:

- C. What was the most-likely final state of the transmitter? And what were the most-likely **last two bits** of the message?

(points: 0.5)

- D. The test message used was 500,000 bits long. Since we re using a rate $1/2$ convolutional code, there were 1,000,000 parity bits transmitted over the channel. The BER "without coding" was calculated by comparing the transmitted bit stream that went into the noisy channel with the bit stream that came out of the noisy channel. How many transmission errors occurred?

(points: 0.5)

- E. After passing the received parity bits through the decoder and calculating the most-likely transmitted message, the BER "with coding" was calculated by comparing the original message with the decoded message. In this example, the density of bit errors was high enough that the decoder was unable to find and correct all the transmission errors. How many uncorrected errors remained after the decoder did the best it could at correcting errors?

(points: 0.5)

Here's the debugging output from running decoder tests with a 500000-bit message a $K = 4$ code with generators 0xD (1101) and 0xE (1110).

```
**** ViterbiDecoder: K=4, rate=1/2 ****
BER: without coding = 0.006200, with coding = 0.000022
Trellis state at end of decoding:
[ 6203.
  6204.
  6205.
  6200.
  6203.
  6202.
  6203.
  6202.]
```

F. What was the most-likely final state of the transmitter? And what were the most-likely final *three* bits of the message?

(points: 0.5)

G. How many uncorrected errors remained after the decoder did the best it could at correcting errors?

(points: 0.5)

H. Consider a given message bit. How many transmitted (coded) bits are influenced by that message bit in both the $K = 3$ and $K = 4$ cases?

(points: 0.5)

Python Task 2: Soft-decision Viterbi decoding

Useful download link:

PS3_softviterbi.py -- template file for this task

Let's change the branch metric to use soft decision decoding, as described in lecture and Chapter 8. The soft metric we'll use is the **square of the Euclidean distance** between the received vector of dimension r , the number of parity bits produced per message bit, of voltage samples and the expected r -bit vector of parities. Just treat them like a pair of r -dimensional vectors and compute the squared distance.

PS3_softviterbi.py is the template file for this task.

Complete the `branch_metric` method for the `SoftViterbiDecoder` class. Note that other than changing the branch metric calculation, the hard-decision and soft-decision decoders are identical. The code we have provided runs a simple test of your implementation. It also tests whether the branch metrics are as expected.

File to upload for Task 2:

(points: 2)

MIT OpenCourseWare
<http://ocw.mit.edu>

6.02 Introduction to EECS II: Digital Communication Systems
Fall 2012

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.