Sarah Lieberman
Final Project Writeup (first draft)
Error Correcting Codes Lab (18.413)
Prof. Spielman

1 Abstract

The paper "Turbo Equalization Using Non-Systematic and Recursive Systematic Convolutional Codes", by Vladimir D Trajkovic and Pradrag B. Rapajic, has somewhat unusual results. In this paper, the authors state that although systematic codes are known to have lower bit error rates (BERs) than non-systematic ones, in turbo equalization they perform equally well. This seems to be a non-intuitive result, and so we decided to reproduce their results, in order to test this phenomenon for ourselves.

Turbo equalization, which will be explained in more detail later in the paper, is the form of turbo encoding that is used when working with intersymbol interference (ISI) channels. ISI channels are different from ideal channels such as binary erasure, Gaussian, or binary symmetric crossover channels, in that the value that the channel outputs depends not only on the most recent input to the channel, but also on the values of past inputs. This is a realistic scenario in cases where electrical signals take time to dissipate, or in cases of magnetic storage of information, where there might remain some noise from whatever bits were previously stored in a given spot. The first is an example of a finite impulse response (FIR) ISI channel, while the latter is an example of an infinite response. The difference will be described in the next section.

2 The Channel

An ISI channel is characterized by its length L, and by a polynomial H of degree L. The coefficients h0 to hL-1 from that polynomial are multiplied by the message bits to get the channel output symbol. For example, if the channel had length two, and the inputs so far were x1, x2, and x3, then the third output symbol, y3, would be y3 = x3*h0 + x2*h1 + x1*h2. In a channel of infinite length, all of the previous input symbols are taken into account. If L is finite however, only the last L symbols affect the next symbol, and the rest can be safely ignored.

In most cases where an ISI channel appears, the channel has additive Gaussian white noise (AGWN) as well. The combination of ISI and AGWN produces a channel that simulates some actual physical channels, which is why it is used in error correcting codes research. The paper we are using as the basis for our project follows this convention, so we will use AGWN in our channel. In this part, a random Gaussian number is produced using the Java Random class, and this number is then added to the current channel value.

Suppose the vector x is the message going into the channel, with x(k) as its kth symbol. Similarly, let y be the set of outputs from the channel. We will let n(k) be the value of the Gaussian noise added to the kth symbol. Then, this gives us the equation:

$$Y(k) = (\sum x(k-i)*h(i)) + n(k) \qquad (1)$$

(where i goes from 1 to L).

We will use the same three channels that were used in the paper we are basing our experiment on [1].* They have the following polynomials:

H1 = [0.5, 0.71, 0.5]

H2 = [0.407, 0.815, 0.407]

H3= [0.227, 0.46, 0.688, 0.46, 0.227]

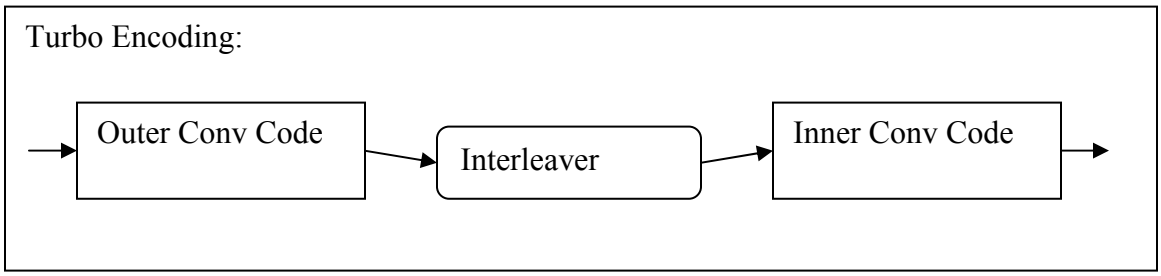
As can be seen, the first two are channels of length three, while the third has length five. Also, all three are symmetrical.

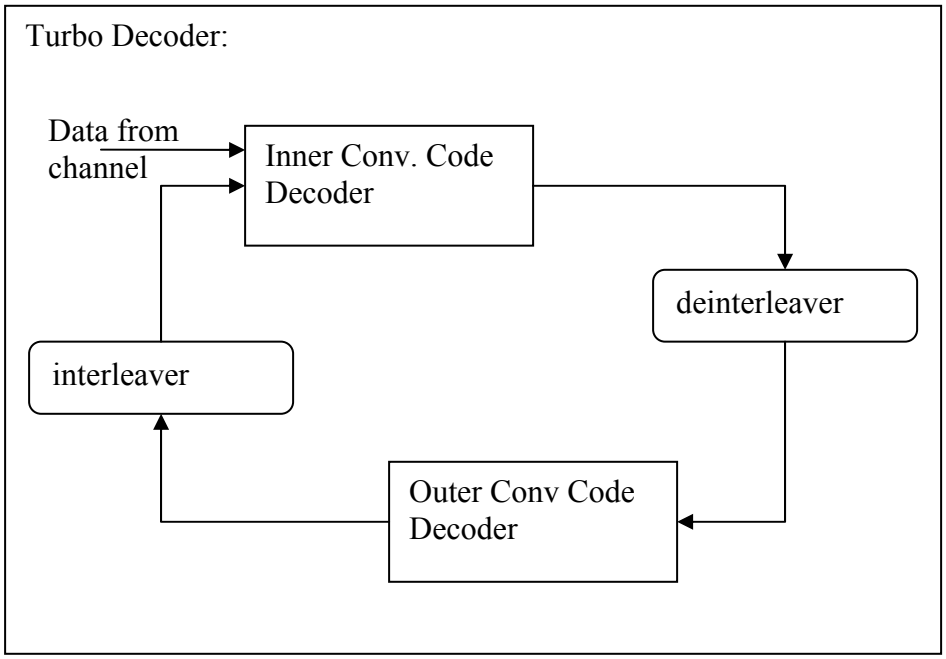
3. Turbo Equalization

When working with channels that do not have ISI, the turbo code has met with significant success. Turbo encoding is a process in which a code word is passed through an outer code, a random interleaver, and then an inner code. The outer and inner codes are often convolutional codes. In the decoding process, soft data is passed between the convolutional decoders for the outer and inner code for a certain number of iterations,

* The H3 given here is actually different from the H3 in the paper. However, the paper says that the ISI channels were obtained from the Proakis textbook. [4] In that textbook, H3 has the values given above. Therefore, we think the H3 in the paper has a typo.

eventually converging on some final value for each of the bits of the original message. The following figures show the very basics of how turbo encoding and decoding work.



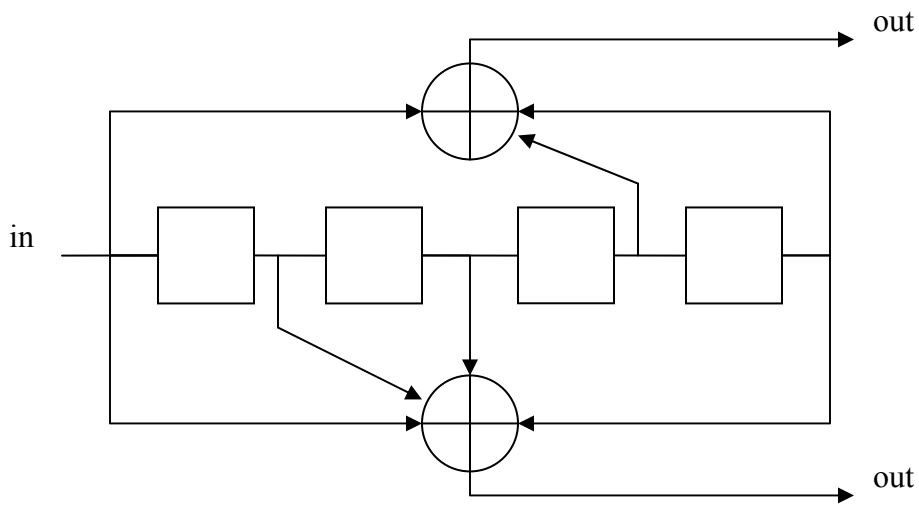


Turbo equalization is similar to turbo decoding, but slightly altered in order to be more effective in the presence of ISI. In turbo equalization, the ISI part of the channel is treated as the inner code. Equalization refers to the process of getting from the real numbers that come off the channel to probabilities for the likelihood that each given bit was a 1, which is the form of information that the outer code's decoder will need as input. Originally, people attempted to completely separate the equalization and decoding processes, but in doing so some information that would have otherwise helped the decoding process was lost in the equalization process. So instead, here we repeatedly

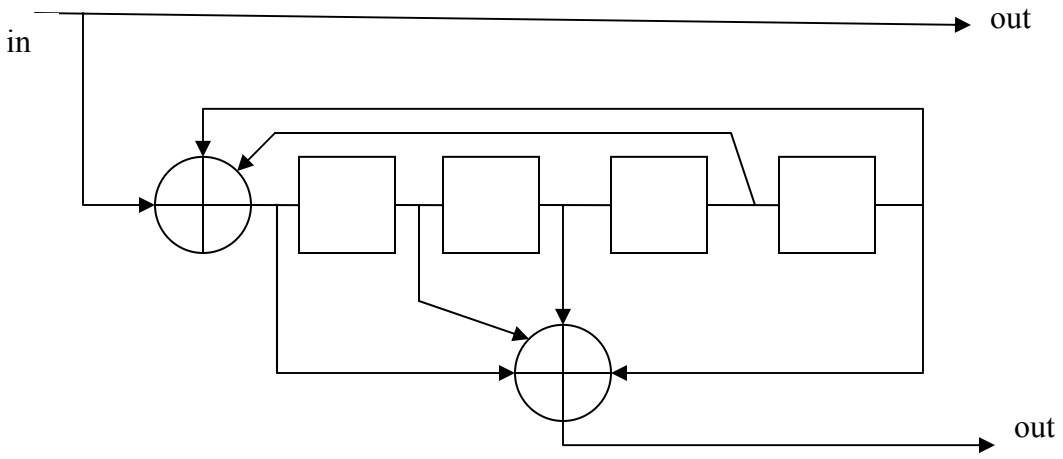
pass data between the decoder for our convolutional code and the equalizer for the ISI channel. [2]

4. The NSC and RSC Convolutional codes

The only difference between the two Turbo Equalizers that we are simulating is the convolutional codes that are used as the outer code for the turbo equalizer. The two convolutional codes are related, but with the major difference that one is a non-systematic (NSC) covolutional code, whereas the other is a systematic code (RSC). Systematic means that the original message is repeated in the output of the encoder. On the other hand, to get the original message using the output of an NSC, one has to first decode. In both of these codes, 2 output bits are produced for every input bit. Also, the basic structures of both codes are very similar, as can be seen in the following diagrams. The squares are delays, and the circles are addition (modulo 2) nodes.
The NSC, which we got from [1] is:



The RSC is similar to the NSC, but of course not exactly the same:



A convolutional code can be represented as a state machine, where each state represents a possible configuration of the delay values. The transitions that are possible in this machine can later be used to perform the decoding. In this case, there are 16 possible states, because there are 4 delays. One can imagine a trellis, in which the transitions from one state to the next are all represented, and any particular path on the trellis represents a particular possible sequence of states. Each path, then, would correspond to a specific and unique sequence of ones and zeroes. To decode an n-bit code word, we need to figure out what sequence of states the code went through. We will have a known start state, in which all of the delay values are zeros. If we have probabilities for which state we were in at step k, and we know all of the possible transitions, then we can get probabilities for what state we were in at step k+1. For example, the NSC has a transition from state 0011 to state 1001, in which the output bit values are 00. The other way to transition into 1001 is from 0010, with an output value of 11. We have values for pr(state k=0011) and pr(state k=0010). Also, we have as input to the decoder pr(b=1) for every bit b in the encoded message. The output bits that would correspond to the kth transition are b[2k] and b[2k+1]. From all these probabilities, we can use the following formula to calculate the probability that the k+1th state was 1001:

pr(state k+1=1001) = pr(state k=0011)*pr(b[2k]=0)*pr(b[2k+1]=0) + (2)
pr(state k=0010)*pr(b[2k]=1)*pr(b[2k+1]=1)

We can make similar calculations to get the probabilities for every other possible value of state k+1. Once we do that, we can likewise calculate what state k+2 was, and just keep making forward calculations until we reach the final state. In a very similar manner, probabilities can be calculated for each state at each transition by using the k+1 probabilities to calculate probabilities for k. This will be done in an almost identical way, except that the transitions in the trellis will reverse directions.

Now, we can use these probabilities to calculate the probabilities that the decoder will output. In turbo equalization, there are two different outputs that the convolutional decoder needs to compute. First, it needs to compute values to send back to the equalizer. These values will be pr(b=1) for each bit b[i] in the encoded message. Second, the decoder will at the end have to compute final guesses for what each bit a[k] in the original message was. To get this probability, we will calculate the probability of each possible transition from state k to state k+1. The probabilities corresponding to transitions where a[k]=1 will be summed up to be the numerator, and the probabilities for all of the transitions will be summed up to make a denominator. The probability for the transition from state i to state j, where the output bits are = b1 and b2, is found usuing the following formula:

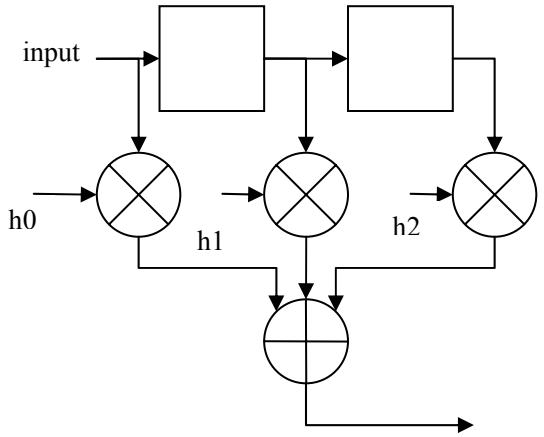
pr(state i to state j) = fwd pr(kth state was i) *pr(b[2k]=b1)* (3)
pr(b[2k+1]=b2)*bkwd pr(k+1th state was j)

The transition probabilities for the b calculations are the same. The only difference is that instead of having the numerator being a sum of all transitions with a certain value for a, we will sum up all transitions with a particular value of b[i]. After these probabilities for b are computed, they are interleaved and then sent to the equalizer. For an alternative description of the decoding process, see [2].

For the RSC, the calculations are slightly different. The overall process will stay the same. The only difference is that in order for the forward and backwards probabilities to be considered extrinsic, they cannon take the systematic bit into account. Therefore, equation 2 would omit one of the pr(b[i]=x) values (the one for the systematic bit b). Other than this, the process should be identical to that of the NSC convolutional decoder.

5. Equalization

The equalization algorithm looks very similar to the convolutional decoder's algorithm, because in turbo equalization the ISI channel is essentially treated as a convolutional code. The ISI effects can be represented by a circuit diagram that looks like this:



The nodes in the diagram represent multiplication and addition nodes. As can be seen here, if the ISI channel has length n, then there are n-1 delays in the circuit. As with the convolutional codes, a certain combination of values in the delays can represent a certain state in a state diagram. The primary difference between this code and the ones in the previous section is that the output values are no longer bits, but rather real numbers. However, each transition does have a particular output symbol associated with it, so this is not as much of a complication as one might initially expect. As before, we move forward on the trellis, computing probabilities for each state. We will use essentially the same equation as equation (2). The only difference is that each transition has only one output symbol associated with it, and the probability for that symbol's value is obtained differently. We have two inputs in the equalizer: the channel, and the decoder. From the decoder, we get probabilities for each b[i] that b[i]=1. From the channel, we get the real numbers that are a sum of the ISI outputs plus the AWGN. We are assuming that the equalizer knows what the ISI coefficients are. We can use the Gaussian distribution function to calculate pr(b[k]=x|state k=y). We will use the function g(n) to represent the probability of having the random Gaussian number =n. We then combine the two b probabilities, and produce the equalizer's equivalent of equation (2), which looks like this (for state k=i, input b[k]=1, output = x,

unchosen output=z, and channel value for output =y):

pr(state k+1=j)= pr(state k=i)* (4)

pr(b[k]=1)*g(y-x)/(pr(b[k]=1)*g(y-x) + pr(b[k]=0)*g(z-x))

The equalizer then calculates for each b[i] the probability that b[i]=1. This is done just as it is done in the convolutional decoder. These values are then de-interleaved and sent to the decoder. For an alternative description of the equalization process, see [2].

6. Conclusion

Unfortunately, we were unable to produce satisfactory results with our simulation code. We tested each component individually and they all seemed to be performing reasonably well, but when we combined all the components we were only able to produce results with bit error rates very close to ½, which implies that our turbo equalizer would have been equally well off just randomly guessing values. The most recent versions of our project code can be found at http://web.mit.edu/sarahl/Public, or at http://web.mit.edu/thinker/courses/18.413/finalproject/final/.

We tried to get information from Trajkovic and Rapajic with regards to their experiments and result data, but we have yet to receive the data that they said they would send us. Apparently, the end of the term is every bit as hectic in Australia as it is here.

The experimental results that Trajkovic and Rapajic reached are surprising, but I have not yet gathered enough information to determine whether my findings correspond to theirs. Perhaps, the reason why no significant difference was noticed between the two convolutional codes was because the equalization process ended up being the limiting factor. I would imagine that if it was significantly worse than the decoders, then the two different turbo codes might end up with similar results.

Bibliography

[1] Trajkovic, V. D.; Rapajic, P. B.; "Turbo Equalization Using Non-Systematic and Recursive Systematic Convolutional Codes", 57th IEEE Semiannual VTC 2003-Spring, vol. 3, April 2003, pp.2125-2129.

[2] Koetter, R.; Singer, A.C.; Tuchler, M.; "Turbo Equalization", in Signal Processing Magazine, IEEE, vol 21, issue 1, Jan 2004, pp67-80.

[3] 18.413 Lecture notes, lectures 14 and 15
http://www-math.mit.edu/~spielman/ECC/handouts.html

[4] Proakis, J.G., *Digital Communications*, New York, McGraw-Hill, 1995, p116.