18.413: Error-Correcting Codes Lab April 1, 2004

Lecture 14

Lecturer: Daniel A. Spielman

14.1 Related Reading

- Fan, pp. 97–105.

14.2 Convolutional Codes

This lecture, and small project 3, is about convolutional codes. Convolutional codes are classical, and have proved to be among the most useful codes in practice. We will be able to perform precise belief propagation on convolutional codes. Convolutional codes are to Turbo codes what parity check codes are to LDPC codes, with some modifications of course.

Convolutional codes are generated by devices that look like this:

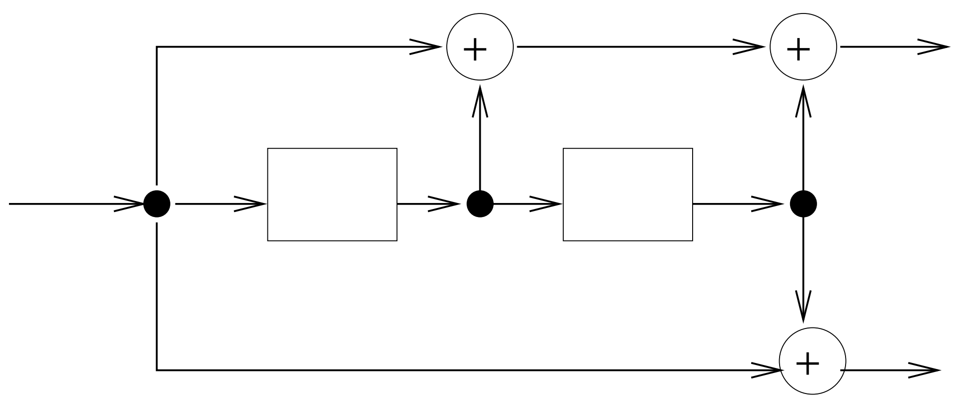

Figure 14.1: A convolutional encoder.

To interpret these diagrams, I should say that a square box is a delay, and a "+" node outputs the parity of its inputs. One may assume that these encoders are initialized with 0's in both of their memories/delays. At each time step, one bit will come in on the arrow from the left, and each delay will output the bit it has stored and take in a new bit. The filled-in circles represent points that forward their input to both of their outputs. We assume that all of these operations occur at once.

For example, consider the second encoder, assume that it starts with 0's in both delays, and that a 1 comes in from the left-hand-side. Then, we can immediately see that a 1 will go out the top output on the right-hand-side. The input 1 will also go down, until it encounters the left-most

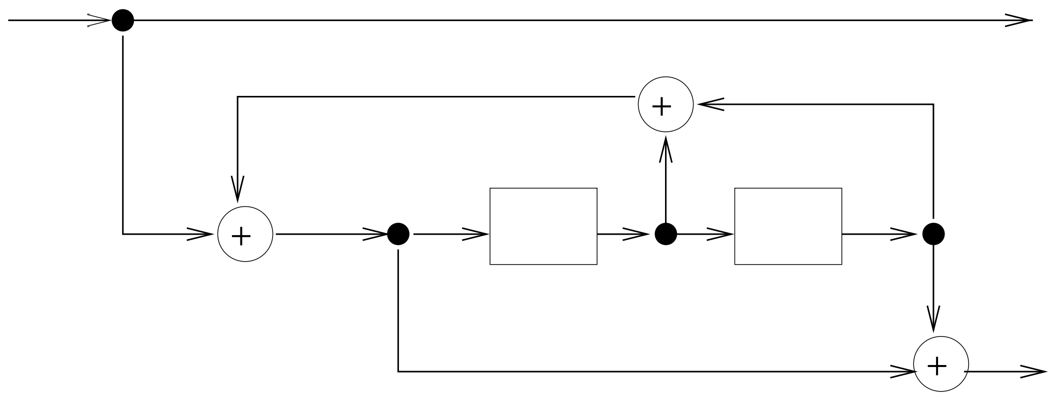

Figure 14.2: A recursive convolutional encoder, and the one we will use in small project 3

parity gate. At this point, it will be xored with another input, which we can see is the parity of the bits stored at the delays. Since both of these are zeros, the xor will be 1, and a 1 will be passed into the left-most delay. This 1 will also pass to the bottom parity gate, where it will be xored with the output of the right delay, which is 0. So, the bottom output on the right will be a 1. One should also note that the right delay will store the output of the left delay, which is a 0.

By going through such arguments for all possible states of the delays and each input, one can figure out what the device will output for each possible input. We see that the device is a finite automaton, which can be depicted as in Figure 3.

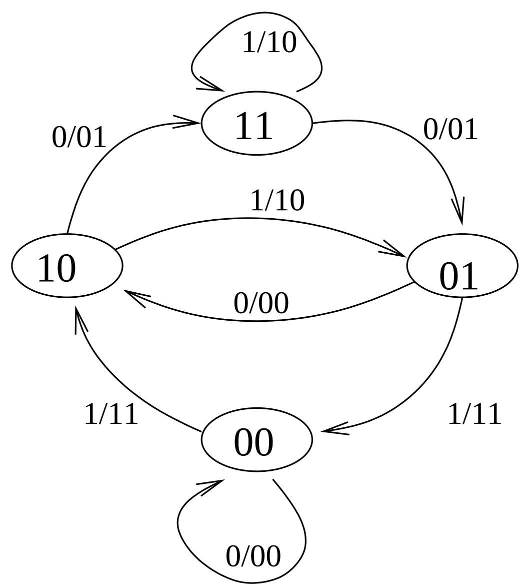

Figure 14.3: The finite automaton for the encoder in Figure 2. Each state is labeled with the Each transition has a label of the form w/v_1v_2, where w represents the input bit and v_1 and v_2 represent the two output bits. In this case, we always have $w = v_1$.

The second encoder has the nice property that its input appears among its outputs. Such encoders are called systematic. However, the second encoder has the complication that it involves feedback. In contrast, all the arrows in the first encoder move forwards. For conventional uses of convolutional codes, it is often preferable to use codes without feedback. However, for the construction of Turbo codes, we will need to use systematic codes with feedback.

14.3 Trellis Representation

On any input, the sequence of state transitions and outputs of a convolutional encoder can be represented as a path through a trellis, which looks like this:

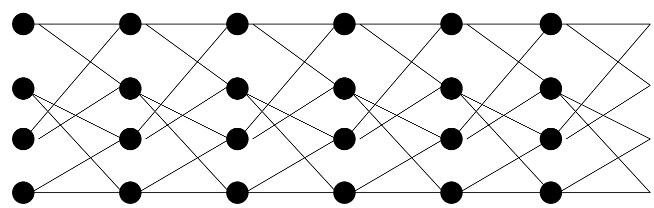

Figure 14.4: A trellis might make a nice tatoo.

To blow this up, each trellis section looks like this:

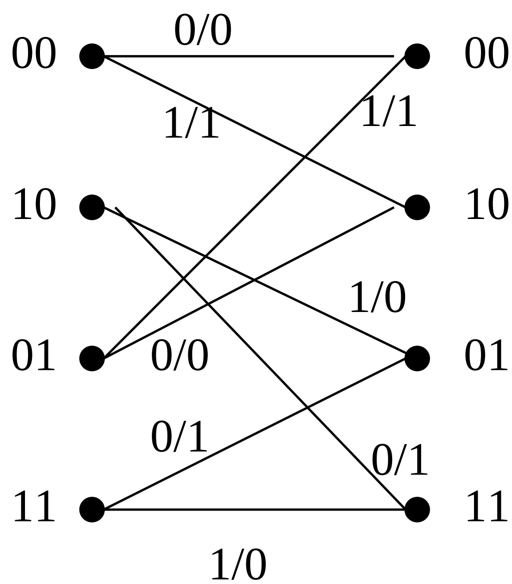

Figure 14.5: A trellis section. The nodes represent the states, and the transisions are labeled by input/other output.

The trellis structure is very close to the normal graph that we will use for decoding convolutional codes.

14.4 Decoding convolutional codes

To describe the decoding algorithm, which is often called the forward-backward or BCJR algorithm, we will set up a normal graph. The internal edges of the graph will contain variables representing the states of the encoder at various times. We will have one external edge for each input and one external edge for each output. Nodes will enforce valid transitions: each node will be attached to one input edge, one output edge, and internal edges representing the previous and next state. The constraint at the node will enforce that the output and next state properly follow from the input and previous state.

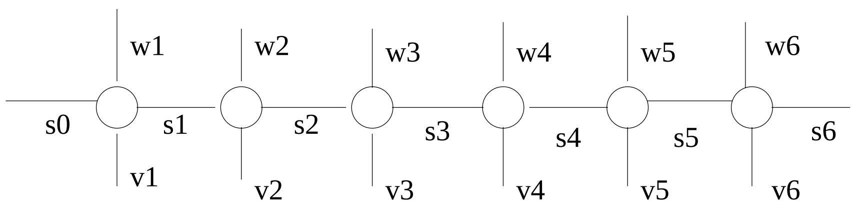

Figure 14.6: The normal graph.

Here, we let the w_is denote the inputs and the v_is denote the outputs.

The channel will provide us with the probabilities that each v_i and w_i was 1 given the symbol received for these bits. Our goal is to determine the probability that each w_i was 1 given *all* of the received symbols. As in the case of the repitition code and the parity-check code, there is a very efficient way to do this if the convolutional encoder does not have too many delays. The reason is that the normal graph is a tree, so the belief propagation algorithm will provide the exact answer here.

The belief propogation algorithm will begin by computing messages in each direction for each internal edge. These messages carry the probabilities of each state given all the received channel outputs to the left or right.

To derive the correct update formula, we recall Lemma 7.0.1 (the three variable lemma):

Lemma 14.4.1. *If X_1, X_2 and X_3 lie in A_1, A_2 and A_3, and $(X_1, X_2) \in \mathcal{C}_{12} \subseteq A_1 \times A_2$ and and $(X_2, X_3) \in \mathcal{C}_{23} \subseteq A_2 \times A_3$, and if (X_1, X_2, X_3) are chosen uniformly subject to this condition, then*

$$P^{ext}[X_1 = a_1 | Y_2 Y_3 = b_2 b_3] = \sum_{a_2 : (a_1, a_2) \in \mathcal{C}_{12}} P[X_2 = a_2 | X_1 = a_1] \cdot P^{ext}[X_2 = a_2 | Y_2 = b_2] \cdot P^{ext}[X_2 = a_2 | Y_3 = b_3].$$

In this case, we will set

$$X_3 = \{s_0, \ldots, s_{j-1}, v_1, \ldots, v_j, w_1, \ldots, w_j\}$$
$$X_2 = \{s_j, x_{j+1}, w_{j+1}\}$$
$$X_1 = \{s_{j+1}\}\,.$$

We then have that

$$\mathrm{P}^{ext}\left[s_{j+1} = a_1 | v_0, \ldots, v_{j+1}, w_0, \ldots, w_{j+1}\right] = \sum_{s_j,\ w_{j+1},\ \text{and the induced } v_{j+1} \text{ that result in } s_{j+1}} (1/2) \cdot \mathrm{P}^{ext}\left[w_{j+1} v_{j+1} | \text{channel output for these}\right] \cdot \mathrm{P}^{ext}\left[X_2 = a_2 | Y_3\right].$$

Here the first term, $\mathrm{P}\left[X_2 = a_2 | X_1 = a_1\right]$ is 1/2 because there are only two choices for s_j, w_{j+1} that lead to s_{j+1}. The second term, $\mathrm{P}^{ext}\left[X_2 = a_2 | Y_2 = b_2\right]$ is simplified because we only have channel outputs for w_{j+1} and v_{j+1}. The third term, $\mathrm{P}^{ext}\left[X_2 = a_2 | Y_3\right]$, can also be simplified because w_{j+1} does not depend upon the previous variables. So, we can reduce it to $(1/2)\mathrm{P}^{ext}\left[s_j | Y_3\right]$. You will now note that this is exactly like what we are computing for X_1, and so we can propagate all these terms forwards.

Propagating backwards will be almost the same, with one crucial difference: the base case. We should have observed that we can be sure that $s_0 = 00$. On the other hand, if we end transmission after time t, then we have $\mathrm{P}\left[s_t =??\right] = 1/4$ for all values. You might wonder how one can go from having equal probability at the last state, and propagate back useful information. The answer lies in the fact that if you know exactly the input and output of the encoder at two consecutive time steps, then you can determine the states at those times.

Our discussion of how to decode is almost finished. As we said, we should produce all forwards and backwards messages for the internal edges. It then remains to produce the outputs for the external edges corresponding to the input bits. We do this in two (artificial) stages. We first get the extrinsic probability that $w_i = 1$. To obtain this, we must compute the probability that $w_i = 1$ and $w_i = 0$ given all the v_js, and all the w_js for $j \neq i$. To do that, we sum over all s_{i-i}, s_i, and v_i that produce 0 or 1 respectively, and in each term of the sum take the product of the respective probabilities. To now compute the posterior probability, we multiply the extrinsic probability by the probability coming from the channel output for w_i.