18.413: Error-Correcting Codes Lab March 18, 2004

Lecture 13

Lecturer: Daniel A. Spielman

13.1 Related Reading

- "Efficient Erasure Correcting Codes", from Stellar web page. I'm not suggesting that you read this paper right now. But, this is where the material comes from.

13.2 The Binary Erasure Channel

The binary erasure channel can be understood to output three symbols: "0", "1", and "?". The symbols 0 and 1 indicated that the input was a 0 or 1 respectively, and the symbol ? indicates that the input was equally likely to have been a 0 or 1. Thus, upon receiving these symbols, the demodulator tells us that the probability that the input symbol was a 1 is 0 or 1 if a 0 or 1 was received, or 1/2 if a ? was received.

I claim that if one runs the belief propagation decoding algorithm for a LDPC code after transmitting over the erasure channel, then these three probabilities are the only quantities that will be transmitted over edges. To see this, let's first consider the formula for parity nodes:

$$p_1^{ext} = \frac{1 - \prod_{i=2}^{k}(1 - 2p_i^{int})}{2}.$$

We see that if $p_i^{int} = 1/2$ for some i, then the product will equal zero, and we will get $p_1^{ext} = 1/2$. On the other hand, if $p_i^{int} = \pm 1$ for all i, then each term in the product will be ± 1, the product will be ± 1, and we will get $p_1^{ext} \in \{0, 1\}$.

Similarly, for the equality nodes the update formula is:

$$p_1^{ext} = \frac{\prod_{i=2}^{k} p_i^{int}}{\prod_{i=2}^{k} p_i^{int} + \prod_{i=2}^{k}(1 - p_i^{int})}.$$

Here, we see that if $p_i^{int} \in \{0, 1/2\}$ for all i, and $p_i^{int} = 0$ for some i, then we will get $p_1^{ext} = 0/(0 + 1) = 0$. Similarly, if $p_i^{int} \in \{1, 1/2\}$ for all i and $p_i^{int} = 1$ for some i, then we get $p_1^{ext} = 1$. Finally, if $p_i^{int} = 1/2$ for all i, then we get

$$p_i^{ext} = \frac{2^{-(k-1)}}{2^{-(k-1)} + 2^{-(k-1)}} = 1/2.$$

One might wonder if we can possibly have both a 0 and a 1 among the values of p_i^{int}. If we did, it would clearly be bad as we would then get $p_1^{ext} = 0/(0 + 0) = \textit{undefined}$. However, if there are

no numerical errors, then this should be impossible: a 0 is an indication that the bit is definitely 0 and a 1 is an indication that the bit is definitely 1. By the correctness of the belief propagation algorithm on trees, it should be impossible for us to reach both conclusions simultaneously. (While the graph is not a tree, the output of the algorithm corresponds to the output for some tree).

13.3 Analysis of LDPC on BEC

The nice thing about using Low-Density Parity-Check codes on the BEC is that it is possible to analyze their performance. We'll begin with heuristic approach to the analysis. Let p_0 denote the probability of erasure in the channel. We are going to track the proportion of messages being sent at each iteration that correspond to probability 1/2. We let a_t denote the proportion of 1/2 messages leaving equality nodes during the tth iteration, and b_t denote the proportion of 1/2 messages leaving parity nodes during the tth iteration. As we expect that p_0 of the message bits will be erased, and in the first stage these are the only inputs to the equality nodes, we expect to have $a_1 = p_0$.

The computation of the expected value of b_1 will be more interesting. Note that a parity node will output 1/2 along an edge if any of its incoming messages on other edges were 1/2. As the graph is random, let's assume that the probability that any of these incoming messages is 1/2 is a_1. Then, the probability that a particular edge does not see a 1/2 is $1 - a_0$. As there are 5 incomming messages, the probability that none of them are 1/2 is roughly $(1 - a_1)^5$. Similary the probabiility that some one of the incomming messages is a 1/2, and thus the outgoing message is a 1/2 is $1 - (1 - a_1)^5$. Thus, we obtain

$$b_1 = 1 - (1 - a_1)^5.$$

There was nothing special about this being the first stage. So, in general we obtain

$$b_i = 1 - (1 - a_i)^5.$$

We now perform the analogous computation at the equality nodes. The value output by an equality node along an edge is a function of three inputs: the incomming messages on the other two edges and the value received by the channel. The output value will be a 1/2 only if all three of these are 1/2. The probability the input from the channel is 1/2 is p_0, and, assuming the graph is random, we expect the probability that each of the incomming messages is 1/2 to be b_i. So, we get

$$a_{i+1} = p_0 b_i^2.$$

13.3.1 Discussion

The above computation is heuristically reasonable. However, the above justification does not really hold up. In particular, the repeated assertion "because the graph is random" needs some mathematics behind it.

One might also wonder why we bother to do the computation in terms of the messages on edges rather than by fractions of nodes. The main reason is that you get the wrong answer this way. I would explain why, but I fear that the exercise of debunking a bad analysis would only serve to teach a bad analysis. If there is demand, I'll explore why this would be wrong later.

13.4 Making the analysis rigorous on trees

The equations derived in the previous section can be made rigorous in the case where the graph is a tree. While the graph is not a tree, we have reason to believe that the algorithm should behave similarly on infinite trees and graphs, so it is not so bad to go through trees. The computation that we get for a tree will be identical to that derived above, but the probabilities will have just slightly different meanings.

Assuming that the trees have external edges at the leaves input to parity nodes, we understand a_1 to be the probability that a 1/2 is transmitted up an external edge, b_1 to be the probability that a 1/2 is transmitted up an edge above a tree containing just one parity node, a_i to be the probability for a tree containing i levels of equality nodes, etc.

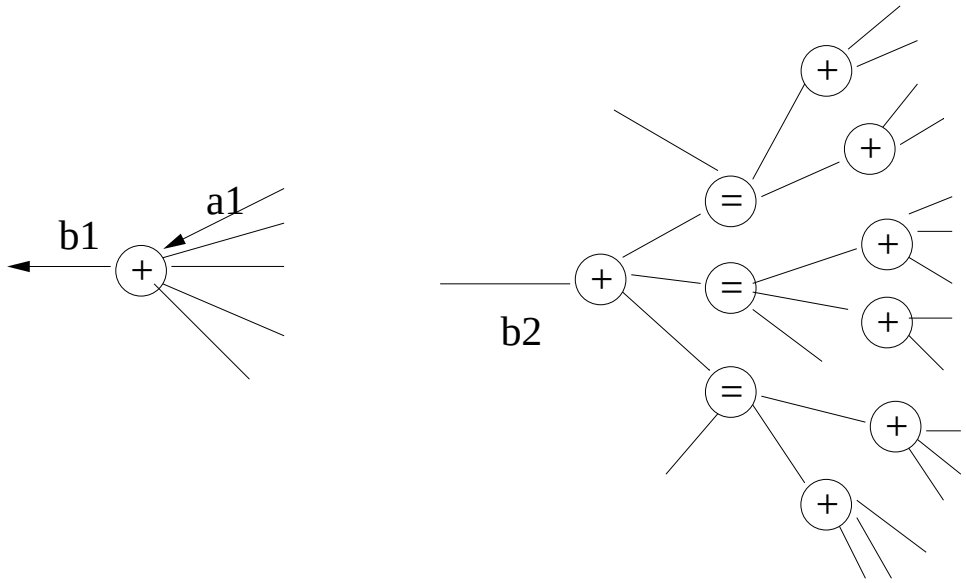

Figure 13.1: The probabilities on the tree

In this case the analysis is exact as all of the probabilities considered really are independent.

13.5 Using the polynomials

Once we have these polynomial relations, we believe that we can correct all the erasures if b_t goes to 0 as t grows large. To check if it does, we can combine the two polynomial equations, and observe that

$$a_{t+1} = p_0 \left(1 - (1 - a_t)^5\right)^2 .$$

We can then do some derivations like:

```
>> p0 = .4;
>> a(1) = p0;
```

```
>> for i = 1:20, a(i+1)= p0 * (1-(1-a(i))^5)^2; end
>> a

a =

  Columns 1 through 4

   0.40000000000000   0.34021064704000   0.30622652487968   0.28175176205271

  Columns 5 through 8

   0.26169574373974   0.24375339697360   0.22659100186113   0.20925153584554

  Columns 9 through 12

   0.19090062329355   0.17069720280653   0.14774283981889   0.12116376063821

  Columns 13 through 16

   0.09053460661436   0.05709275330846   0.02594402938913   0.00606751131232

  Columns 17 through 20

   0.00035931960004   0.00000128925140   0.00000000001662   0.00000000000000

  Column 21

                  0

>> p0 = .44;
>> a(1) = p0;
>> for i = 1:20, a(i+1)= p0 * (1-(1-a(i))^5)^2; end
>> a

a =

  Columns 1 through 4

   0.44000000000000   0.39287014786402   0.37040180678629   0.35724986631803

  Columns 5 through 8

   0.34875815840488   0.34295307142171   0.33883631866955   0.33584307908024

  Columns 9 through 12
```

```
 0.33362801453247   0.33196773069905   0.33071148036034   0.32975420756774

Columns 13 through 16

 0.32902085739940   0.32845676540375   0.32802151644683   0.32768487882640

Columns 17 through 20

 0.32742403093216   0.32722162169403   0.32706438559285   0.32694213647138

Column 21

 0.32684702611889
```

So, we can tell that the limit of p_0 that we can handle seems to be between .4 and .44.

Of course, we would like to determine the threshold. There are two ways to go about doing this: algebraic and visual. I think it helps to work visually. We consider the function

$$f(x) = p_0 \left(1 - (1-x)^5\right)^2,$$

and plot it. Our iterated application of $f(x)$ can be understood by reading off the point $(p_0, f(p_0))$ on the plot, then taking $(f(p_0), f(f(p_0)))$, etc.

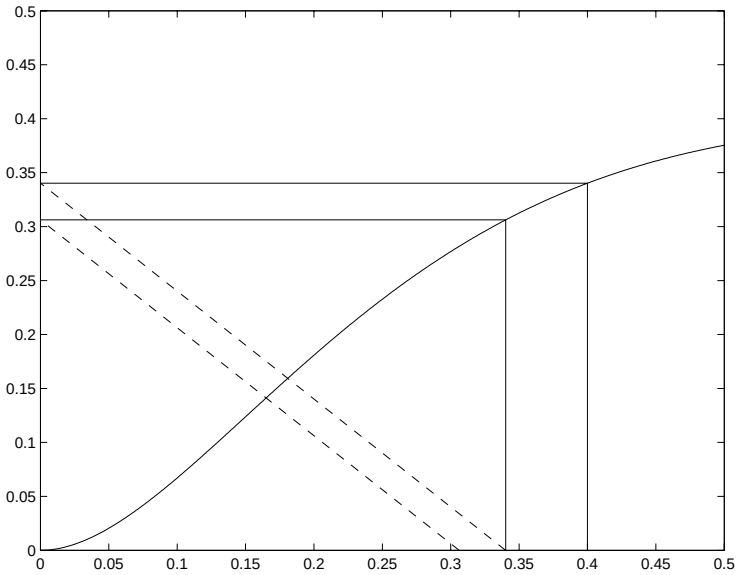

Figure 13.2: Generated by code http://math.mit.edu/∼spielman/ECC/lect13b.m

A better way to visualize this is to place a mirror on the line $x = y$ in the plot. We can then view the plot as bouncing around off this mirror and the curve. It is easy to show that this is equivalent

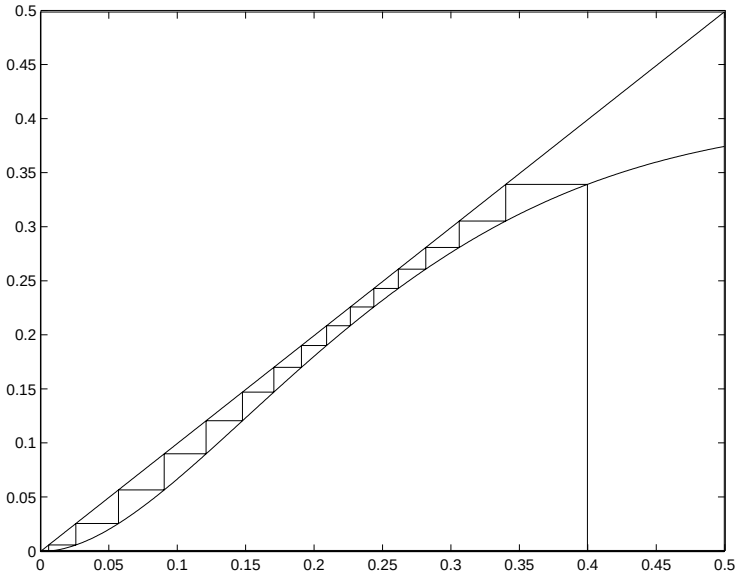

Figure 13.3: Generated by code http://math.mit.edu/∼spielman/ECC/lect13c.m

to the previous process. We then generate a much nicer figure: In particular, we can tell from this figure that if the curve never crosses the mirror, then the decoding process should converge.

We can still improve this further by dropping the mirror, plotting both polynomials, and bouncing between them:

From this figure, we can tell that the code will converge as long as the curves don't cross. Even better, given one curve, we can try to optimize our choice of the other curve, and vice versa. We may thereby obtain better and better codes.

13.6 Capacity Estimation, revisited

The big question is how to obtain such an analysis for other channels. One problem is that the distributions of messages being passed around cannot be so easily characterized by one variable. A characterization that we will sometimes try will be the capacity.

In small project 1, we estimated the capacity by computing $i(x; y)$ given that we sent x and received y, and then averaging over values. It turns out that we could have done this more simply. Recall that a symmetric channel can be described as a distribution over binary symmetric channels. That is, we have probabilities $q_1, \ldots, q_k$, where q_i is the probability of choosing channel i, which has a crossover probability of p_i. A channel of crossover probability p_i has capacity

$$1 - H_2(p_i),$$

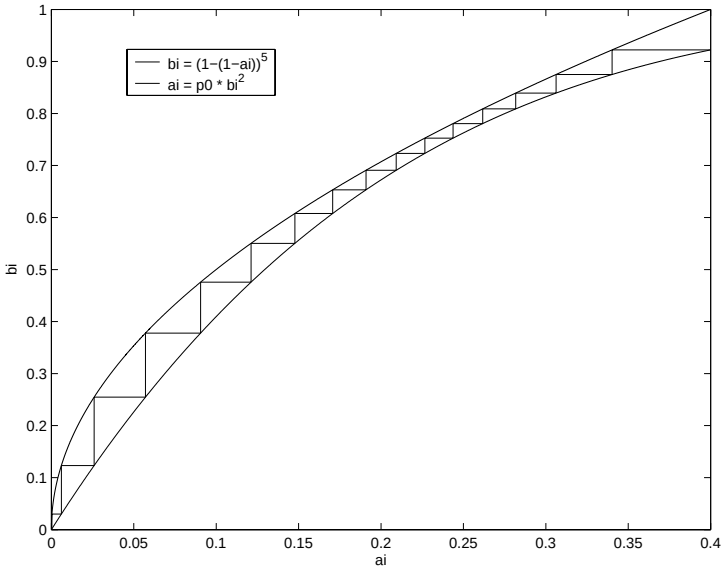

Figure 13.4: Generated by code http://math.mit.edu/∼spielman/ECC/lect13d.m

so the capacity of this symmetric channel is

$$1 - \sum_{i=1}^{k} q_i H_2(p_i).$$

What that means for us is that to estimate the capacity of a channel, we need merely take the probabilities p that pop out from it, and average the resulting values $1 - H_2(p)$ (recall that $H_2(p) = -p\log_2(p) - (1-p)\log_2(1-p)$). We don't actually need to know whether the estimate is leaning towards correct or incorrect because $H_2(p) = H_2(1-p)$.

I would like to revise project 1 by asking for you to treat the messages being sent at some iteration as a meta-channel, and empiracally computing its capacity in this way instead of keeping the statistics that I asked for.