6.962: Graduate Seminar in Communications

Week 5 Report

Presenter: Stark Draper

Date: 11 October 2000

Paper: "Irregular Repeat-Accumulate Codes"
H. Jin, A. Khandekar, and R. McEliece
2nd International Symposium on Turbo Codes & Related Topics
pp. 1–9, Sept, 2000

Introduction

In this report on the class discussion on Irregular Repeat-Accumulate (IRA) codes, we focus on two things. In Section 1 we focus on the topic of density evolution. We work a simple example for Repeat-Accumulate (RA) codes. This introduces us to the idea of a "noise threshold" parameter of the channel, above which iterative decoding does not converge to the correct answer. It also indicates the need for a method that can increase code complexity without increasing code rate, which led to IRA codes. In Section 2 we focus on the Sum-Product algorithm and the question of cycles. We first review some of the intuition we developed in class for the Sum-Product algorithm. This leads us to understand why the 'no-cycle' condition is important for the correct functioning of the Sum-Product algorithm. Combined with some of the observations of iterative-decoding behavior from Section 1, we argue that cycles may not cause problems, as long as their length is greater than the number of iterations the decoding algorithm needs to converge. Finally, this leads us to a discussion of the Richardson-Urbanke (RU) condition.

1 Density Evolution

Density evolution is an important tool for determine the convergence of iterative decoding algorithms. In this section we will work an example of density evolution for a simple RA code on a BEC. This parallels the example for an IRA code we did in class. We will also present some plots that show how the densities evolve as a function of the number of iterations of the decoding algorithm. This will help to illustrate the idea of a 'noise threshold', also discussed in class.

Given a certain code, our objective is to determine for what sets of channels the probability of decoding error converges to zero as the number of iterations of the decoding algorithm increases. There are generally two ways of doing this analysis, via fixed-point analysis and via numerical iterations, we review both in this section. We present an example of density evolution for a RA code on the BEC because of its simplicity. Similar equations can be derived for IRA codes and have been in [1].

In Figure 1 we present a diagram of a RA code. The top nodes are information nodes. Generally these are hidden in RA codes, and have equal degree, q ($q = 3$ in the diagram). The center row of nodes are the check nodes. The bottom row of nodes are the parity nodes, which are visible.

At iteration L the probability of an erasure (i.e. no knowledge) on an edge from an information node to a check node is $x_0^{(L)}$. The probability of erasure on an edge from a check to a information node is $x_3^{(L)}$. The probability of erasure on an edge from a check node to a parity node is $x_1^{(L)}$. The probability of erasure on an edge from a parity to a check node is $x_2^{(L)}$. Because the BEC never makes a mistake, the probability of erasure along an edge is equal to the probability of error along that edge.

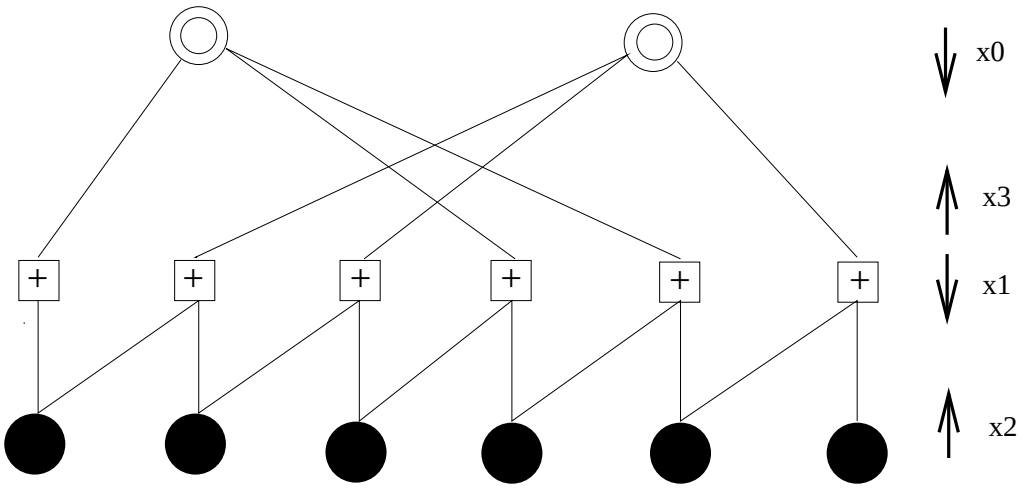

Figure 1: A RA codes

We can then show the following relations (similar to the example we worked through in class on slides #28–31) where the probability of erasure on the channel is p.

$$x_3^{(L)} = 1 - (1 - x_2^{(L)})^2, \quad (1)$$

$$x_0^{(L)} = (x_3^{(L)})^{q-1}, \quad (2)$$

$$x_1^{(L)} = 1 - (1 - x_2^{(L)})(1 - x_0^{(L)}), \quad (3)$$

$$x_2^{(L+1)} = x_1^{(L)} p. \quad (4)$$

This leads to the recursion

$$x_2^{(L+1)} = p(1 - (1 - x_2^{(L)})(1 - (1 - (1 - x_2^{(L)})^2)^{q-1})). \quad (5)$$

To guarantee that realized decoding performance is similar to what density evolution predicts, we must use a code that has a very large block size. In that case the observed distribution of channel erasures is very close to the ensemble behavior, and decoding algorithm performance will be close to that predicted by density evolution.

One approach to showing convergence is to solve for a fixed point of this algorithm, i.e. for a point where $x_2^{(L+1)} = x_2^{(L)} = x_2$. Then we would like to show that there are no fixed points for $x_2 > 0$. Since x_2 is a probability, $x_2 \geq 0$, and thus x_2 must converge to 0.

If equations (1)–(4) cannot easily be compacted into a compound statement (as in (5)), or if we want to learn about how decoding performance changes with the number of iterations (for example to learn how many iterations we should run our decoder for in practice), we follow a second tact. The second approach is to initialize the recursion with $x_2^{(0)} = p$, the probability of channel erasure, and $x_0^{(0)} = x_1^{(1)} = x_3^{(3)} = 1$, and then to iterate equations (1)–(4). We do this in the plots below for various probabilities of erasure. Since in the end we are interested in the information bits, we plot $x_0^{(L)}$ versus iterations L.

In Figure 2 we plot the probability of an information bit error (or erasure) versus number of iterations for various channel probabilities of erasure. As the probability of channel erasure goes up, the probability of error converges to zero more slowly (or not at all). The rate of the code is controlled by q, the degree of the information node sequences. This is because for a RA code each check node has only one edge joining it to an information node, and for each check node there is one parity node. In this figure $q = 3$ so this is a $R = 1/3$ code.

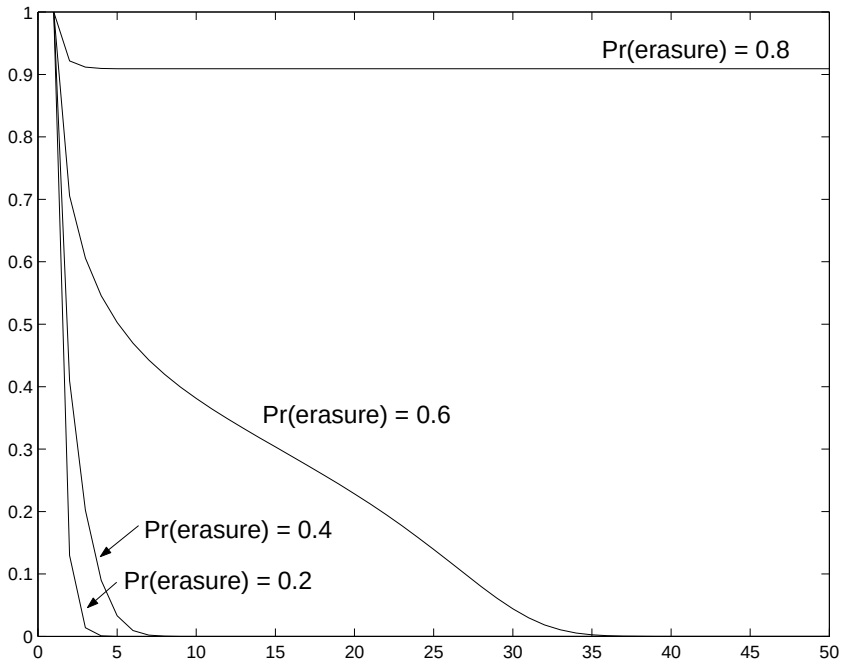

Figure 2: Pr(Info bit error) vs. Number Iterations, $q = 3$

In Figure 3 we plot the same graph for a rate $R = 1/4$ code, i.e. $q = 4$. As we would expect, the code converges more quickly, and for a wider range of channel erasure values.

Finally, in Figure 4 we plot the behavior of the $q = 3$ code near the 'noise threshold' where the decoder does not converge to zero probability of error. The capacity of a BEC is $C = 1 - p$. Our code is a rate $1/3$ code. Capacity results therefore predict that we should be able to correct up to an erasure probability of $p = 2/3$. Here we get close, but not all the way, somewhere between $p = 0.61$ and $p = 0.62$.

In Figure 3 we increased q to decrease probability of error, but that also decreased the rate of the code. This is one reason that IRA codes were introduced: to allow increased code complexity without decreasing the rate of the code. IRA codes allow the check nodes to have left (information-bit-side) degrees greater than one, and allow information nodes of varying degrees. These modifications increase code performance, but still allow simple density evolution analysis. If we go back to slides #43-46 from class we see an analogous presentation of thresholding results.

2 Sum-Product Algorithm and Cycles

In class we discussed an intuition behind the Sum-Product algorithm. We now reiterate that intuition and connect it to the RU condition. The discussion below parallels that of slides #10–12 of the presentation. We only consider using the Sum-Product algorithm to compute marginal densities.

At a check node we have (e.g.) three incoming edges. Each corresponds to a random variable. The form of the joint probability density $p(u, v, w)$ is controlled by the constraint node; i.e. only certain combinations of u, v and w are allowed. If u and v are the incoming variables, then we are interested in the marginal for w. We require the node constraints to be such that given knowledge of u and v and the node constraints, we can uniquely determine w. (Think of a check-node example.) If this is true, given $p(u)$, $p(v)$, and the constraints we can determine the full joint density $p(u, v, w)$. Then we can calculate the marginal for w via the "not-sum", i.e. the sum over all values of u and v. For the binary alphabet case and mod-2 sum check nodes, we showed such a calculation in slide #12.

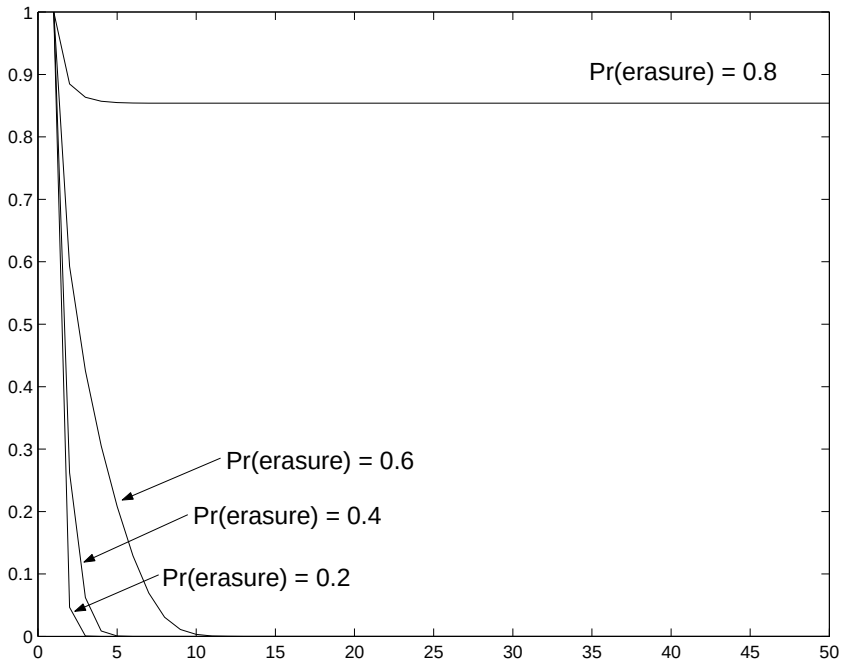

Figure 3: Pr(Info bit error) vs. Number Iterations, $q = 4$

At a variable node we have (e.g.) two incoming edges. Each represents a function, $f_A(x)$ and $f_B(x)$. The output of the node is the product $f_A(x)f_B(x)$. We should think of $f_A(x)$ and $f_B(x)$ as densities conditioned on x. $f_A(x)$ is the conditional density of the observed information from one part of the graph, and $f_B(x)$ is the conditional density of the observed information from the other part of the graph. Think of these two densities as $f(y_A|x)$ and $f(y_B|x)$. Conditioned on x the information observed in the part A of the graph is independent from that observed in part B of the graph. This is because of 1) the memoryless channel assumption and 2) because we assume the code structure has no cycles. Because of the conditional independence, the density at the output of the node is $f(y_A|x)f(y_B|x) = f(y_A, y_B|x)$, the probabilistic fusion of the data from the two parts of the graph.

The discussion in the preceding paragraph indicates that the no cycle assumption is vitally important. If there were cycles we could no longer assume y_A and y_B are conditionally independent. However, most all code graphs have cycles, so we could be in trouble. Let us now consider evolving our densities for L iterations, as discussed in Section 1. Say that by the Lth iteration the probability of error has converged to zero. As long as there are no cycles in our graph with length less than L, our density evolution calculations will be correct, and the decoder will have converged to the correct answer. This observations tells us that it is not the existence of cycles that is most disturbing, but the existence of short cycles, and led us to the Richardson-Urbanke (RU) condition [2]:

> For any fixed L, the probability that the depth-L neighborhood of a randomly selected edge contains a cycle goes to zero as the block size $\to \infty$.

This condition implies that L-fold density evolution gives the limiting value of the ensemble bit error probability after L iterations. The limiting value will depend on a 'noise parameter' of the channel. If the noise is above the *ensemble noise threshold*, the limiting bit error will not go to zero. We observed such a behavior with the RA codes in Figure 4.

A few observations come out of this. First, a generalization of the density evolution set-up in the last section is a good way to analyze decoding performance. Indeed predicted performance results have very

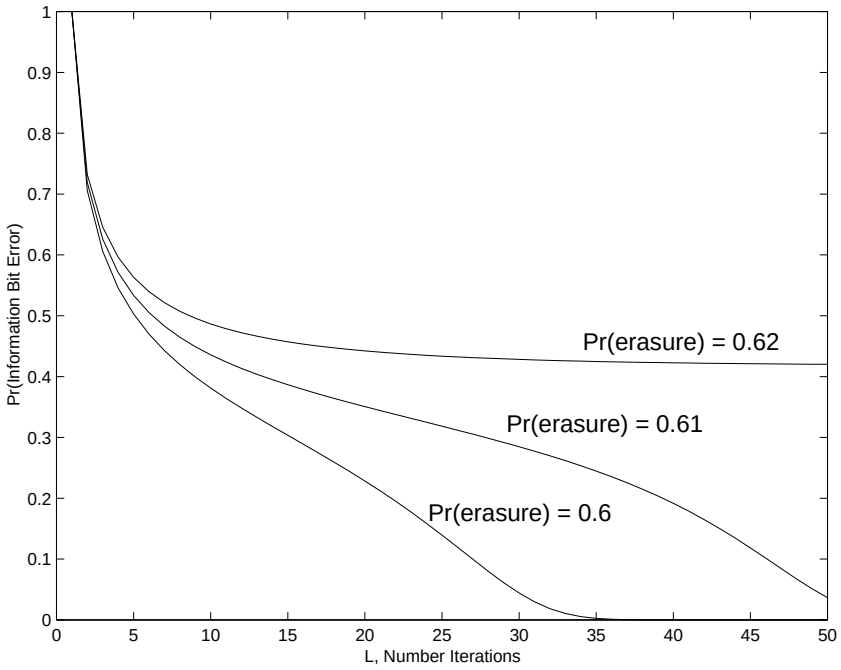

Figure 4: Threshold Behavior, $q = 3$

closely matched the simulated performance of many codes. Second, this condition grounds the no-cycle assumption made in the Sum-Product algorithm above (IRA codes have been shown to satisfy the RU condition). Third, if the ensemble bit error probability goes to zero (over the ensemble of codes), then (similar to the coding theorem) a good code does exist. Finally, David Forney observed that one of the outstanding theoretical problems is to show (or to disprove) that the ensemble noise threshold converges to capacity for some sequence of code ensembles.

References

[1] A. Khandekar H. Jin and R. J. McEliece. Irregular repeat-accumulate codes. In *Proceedings of the 2nd International Symposium on Turbo Codes & Related Topics*, pages 1–8, 2000.

[2] T. J. Richardson and R. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Trans. Info. Theory*, submitted 1998.