Lecture Notes for Math 696
Coding Theory
Iterative Decoding

Michael E. O'Sullivan
mosulliv@math.sdsu.edu
wwww-rohan.sdsu.edu/ ~ mosulliv

April 13, 2004

These notes are my effort to understand several articles concerning decoding of low-density parity-check codes. The paper by Aji and McEliece called "The Generalized Distributive Law" [6] gives a clear description of an algorithm that is useful in many areas of applied mathematics including error correction: They cite artificial intelligence, belief propagation algorithms, fast Fourier transforms and several decoding algorithms. The GDL yields a computationally efficient algorithm for computing certain complicated formulas. The algorithm is best understood by looking at a graph that expresses the relationship between the variables and the functions that occur in the formula. The algorithm involves "message passing" between the vertices of the graph. I will actually use the graphical representation from a different article, "Factor Graphs and the Sum-Product Algorithm" by Kschischang, Frey and Loeliger. The algorithm is only exact when the underlying graph is a tree, but numerous experiments using these message passing algorithms for decoding have been done, and the performance for non-trees is remarkably good.

Theoretical understanding of the algorithm has improved greatly in the last three years, but my impression is that a definitive explanation of the performance is still lacking.

1 Probability Theory: The basics

Definition 1.1. Let X be a finite set. A *probability distribution* on X is a map $P : X \to [0, 1]$ such that $\sum_{x \in X} P(x) = 1$. We say the distribution is *uniform* if $P(x) = P(x')$ for all $x, x' \in X$. We allow for the possibility that $P(x) = 0$ for some $x \in X$. We say that $x \in X$ is *possible* if $P(x) > 0$.

For any subset A of X we will write $P(A)$ for $\sum_{x \in A} P(x)$.

There are several ways in which we will abuse notation for the sake of simplicity. For example, in the definition we defined P to be a function whose domain is X, yet we also treat it as a function on the power set 2^X. We will be mainly interested in distributions on a Cartesian product and in that context we use the symbol P in several ways.

Let P be a distribution on a Cartesian product $X \times Y$. We say that P is a *joint distribution on X and Y*. We will also use P for distributions induced on X and on Y as follows. Define

$$P(x) = P(\{x\} \times Y) = \sum_{y \in Y} P(x, y) \tag{1}$$

It is clear that $\sum_{x \in X} P(x) = 1$, so P gives a distribution on X. Similarly, one defines a distribution on Y.

Example 1.2. A deck of cards is physical model of $R \times S$ with the set of ranks $R = \{A, 2, 3, 4, \ldots, 10, J, Q, K\}$ and suits $S = \{\heartsuit, \diamondsuit, \spadesuit, \clubsuit\}$. The uniform distribution on the deck, where each card has probability $1/52$ models the likelihood of drawing a particular card.

$$\begin{aligned} P(A, \heartsuit) &= 1/52 \\ P(A) &= 1/4 \\ P(\heartsuit) &= 1/13 \end{aligned}$$

Example 1.3. Consider a stacked deck of cards, with an extra royal flush $\{10, J, Q, K, A\}$ of spades and three extra royal flushes of hearts. That's 20 extra cards. The likelihood of drawing a particular card from the stacked deck is modeled by the distribution on $R \times S$ given by

$$P(r, s) = \begin{cases} 1/72 & \text{if } r = 2, 3, \ldots, 9 \\ 1/72 & \text{if } s = \diamondsuit, \clubsuit \\ 1/36 & \text{if } s = \spadesuit \text{ and } r = 10, J, Q, K, A \\ 1/18 & \text{if } s = \heartsuit \text{ and } r = 10, J, Q, K, A \end{cases}$$

We have

$$P(r) = \begin{cases} 1/18 & \text{if } r = 2, 3, \ldots, 9 \\ 1/9 & \text{if } r = 10, J, Q, K, A \end{cases}$$

$$P(s) = \begin{cases} 13/72 & \text{if } s = \clubsuit, \diamondsuit \\ 1/4 & \text{if } s = \spadesuit \\ 7/18 & \text{if } s = \heartsuit \end{cases}$$

Definition 1.4. Given the distribution P on $X \times Y$ we can define a distribution on Y for each possible $x \in X$ by

$$P_x(y) = \frac{P(x, y)}{P(x)} \tag{2}$$

The distribution P_x is called the conditional distribution on Y given x. The notation is read "the probability of y given x." It is commonly written $P(y|x)$. Similarly one has $P_y(x)$ a.k.a. $P(x|y)$.

It is clear that P_x is a distribution since $\sum_{y \in Y} P(x,y) = P(x)$.

Example 1.5. For the normal deck of cards, $P_r(s) = 1/4$ and $P_s(r) = 1/13$ for all r, s.

Example 1.6. For the stacked deck, the probability of different suits given the rank K is

$$P_K(s) = \begin{cases} 1/8 & \text{if } s = \diamondsuit, \clubsuit \\ 1/4 & \text{if } s = \spadesuit \\ 1/2 & \text{if } s = \heartsuit \end{cases}$$

and the probability of different ranks given the suit $\spadesuit$ is

$$P_{\spadesuit}(r) = \begin{cases} 1/18 & \text{if } r = 2, 3, \ldots, 9 \\ 1/9 & \text{if } r = 10, J, Q, K, A \end{cases}$$

Given a distribution P on $X \times Y$ we have defined a unique distribution on X, a unique distribution on Y, unique conditional distributions P_x on Y for each $x \in X$, and unique conditional distributions P_y on X for each $y \in Y$. Is this reversible in any way? Given a distribution on X and a distribution on Y is there a unique distribution on $X \times Y$? The answer is no, as the next example shows.

Example 1.7. Take two decks and remove all the red royals $\{10, \ldots, A\}$ and all the black non-royals. The collection of cards that remains has

$$P(r,s) = \begin{cases} 1/26 & \text{if } r = 2, 3, \ldots, 9 \text{ and } s = \heartsuit, \diamondsuit \\ 1/26 & \text{if } r = 10, J, Q, K, A \text{ and } s = \spadesuit, \clubsuit \\ 0 & \text{otherwise} \end{cases}$$

Yet $P(r) = 1/13$ for all r and $P(s) = 1/4$ for all s, which is also true for the standard deck.

On the other hand, given a distribution P on X and a set of conditionals P_x for each possible $x \in X$, there is a unique distribution Q on $X \times Y$ given by

$$Q(x,y) = P(x)P_x(y)$$

such that the distribution induced on X by (1) is P and the conditionals induced on Y by (2) are P_x. Of course the distribution Q on $X \times Y$ also determines a distribution on Y and conditional distributions on X for each $y \in Y$. Now that we have established this point we will use the same letter P for the distribution on $X \times Y$ determined by P on X and the P_x on Y.

Bayes' theorem

Bayes' theorem, a fundamental result in probability theory, tells how to compute $P_y(x)$.

Theorem 1.8. *Let P be a distribution on $X \times Y$ and let P_x be conditional distributions on Y for each possible x. For each possible $y \in Y$ the conditional distribution on X is given by*

$$P_y(x) = \frac{P(x)P_x(y)}{\sum_{x'} P(x')P_{x'}(y)} \tag{3}$$

The sum is over all possible $x' \in X$.

PROOF: For x and y both of nonzero probability, we have from (2),

$$\begin{aligned} P_y(x) &= \frac{P(x,y)}{P(y)} \\ &= \frac{P(x,y)}{\sum_{x' \in X} P(x',y)} \\ &= \frac{P(x)P_x(y)}{\sum_{x'} P(x')P_{x'}(y)} \end{aligned}$$

Of course the latter sum is only over the possible x'. □

The proof is so simple it hardly seems that this result should be called a theorem. The reason for the prestige is more practical than theoretical.

Example 1.9. Suppose you have two urns, H and T. Urn H has 5 red balls and 3 black, while urn T has 4 red balls and 5 black. Someone flips a coin and then draws from urn H or urn T, depending on whether heads or tails shows. Suppose that at the end of the process someone shows you a black ball, what is the probability that the coin showed H?

Here is our probabilistic model of this situation. We have a uniform distribution on $X = \{H, T\}$ and for each element of X we have a distribution on $Y = \{\text{red}, \text{black}\}$. This gives the joint distribution on $X \times Y$

$$\begin{aligned} P(H, \text{red}) &= 5/16 \quad & P(H, \text{black}) &= 3/16 \\ P(T, \text{red}) &= 2/9 \quad & P(T, \text{black}) &= 5/18 \end{aligned}$$

The question asks for $P_{\text{black}}(H)$. To compute this we first compute the denominator in (3), $P(\text{black}) = 13/16 + 5/18 = 67/144$. Then

$$\begin{aligned} P_{\text{black}}(H) &= P(H, \text{black})/P(\text{black}) \\ &= \frac{3/16}{67/1440} \\ &= 27/67 \end{aligned}$$

The decoding problem, at its most basic level, is essentially an application of Bayes' theorem. Let's suppose we are using a binary $[n, k, d]$ code over $\mathbb{F}_2$. Let X be the code (2^k elements) and let Y be $\mathbb{F}_2^n$. We posit some distribution on codewords (usually the uniform distribution). We also posit conditional distributions, $P_x(y)$, the probability that the vector y is received given that x is sent. When the vector y is received the goal

of decoding is to find the vector x that is most likely to have been sent. In other words to find the vector x that maximizes $P_y(x)$. This is called *maximum likelihood decoding.*

Now refer to Bayes' theorem. For fixed y, the denominator of $P_y(x)$ is the same for all x, so for the purpose of maximizing we may ignore it. We need to do 2^k multiplications $P(x)P_x(y)$ and comparisons between these values. For k large (say 100-1000), this is computational impossible, so we must find other means. Assuming that all codewords are equally likely eliminates the necessity of doing the multiplications, as the following proposition shows. But we still have comparisons of 2^k values.

Proposition 1.10. *Let P be a distribution on $X \times Y$. Suppose that X has the uniform distribution, that is $P(x)$ is independent of x. For a fixed $y \in Y$ the x that maximizes $P_y(x)$ is the same x that maximizes $P_x(y)$.*

PROOF: For a fixed y the denominator in (3) is constant. If $P(x)$ is also constant as x varies, then $P_y(x)$ is proportional to $P_x(y)$. □

2 Decoding

In this section we'll look at maximum likelihood decoding in four different situations, from very simple to fairly complex, The first two concern decoding to a single bit, the latter two look at the general problem of decoding a linear code.

The binary symmetric (hard-decision) channel

Let us start with the simplest nontrivial channel, the binary symmetric channel. A source sends a 0 or a 1 to a target. The probability of corruption of the bit, α, is the same for both 0 and 1. We presume that $\alpha < 1/2$.

We want to decide on a decoding strategy for the target. At first glance the question seems silly; if the target receives a 1 it should decode to 1 and if it receives a 0 to 0. But this decoding strategy presumes that 0 and 1 are sent with equal probability. Suppose on the contrary that the source probabilities are ϱ for 1 and $1-\varrho$ for 0 and that $\varrho < 1/2$. Suppose also that the target knows the value of ϱ and of α. Would it ever make sense for the target to decode a 1 to a 0?

Consider the table of joint probabilities.

If the target receives a 1 it is more likely that a 1 was sent provided $\varrho(1-\alpha) > (1-\varrho)\alpha$. This is equivalent to $\varrho/(1-\varrho) > \alpha/(1-\alpha)$. Since the function $x/(1-x)$ is strictly increasing, this is equivalent to $\varrho > \alpha$. On the other hand, if $\varrho < \alpha$ it makes sense to decode a 1 to a 0.

		Bit received	
		0	1
Bit Sent	0	$(1-\varrho)(1-\alpha)$	$(1-\varrho)\alpha$
	1	$\alpha\varrho$	$\varrho(1-\alpha)$

In error correction coding it is usually assumed that 0 and 1 are equiprobable at the source. We will adopt this assumption from now on.

The binary soft decision channel

In practice a bit is represented electronically in some fashion, and the target receives some corrupted version of the signal. The hard-decision channel interprets the received signal as a 0 or 1, while a soft-decision channel allows for a spectrum of values. Let us take a relatively simple model where a bit is represented as a voltage, $2V$ for 0 and $-2V$ for 1. Let us also suppose that the target measures the received signal as an integral value between $-3V$ and $3V$. Based on the electronics or experiments, the designer has a probabilistic model for $p_b(a)$ the probability that $b \in \{0,1\}$ is sent (as $\pm 2V$) and $a \in \{-3,-2,-1,0,1,2,3\}$ is received. If we presume that 0 and 1 are equally likely to have been sent, $p(0) = p(1)$, then to decode a received value of a is to compare

$$p_a(0) = \frac{p(0)p_0(a)}{p(0)p_0(a)+p(1)p_1(a)} \text{ and } p_a(1) = \frac{p(1)p_1(a)}{p(0)p_0(a)+p(1)p_1(a)}$$

to see which is the largest. Since the denominators are the same and $p(0) = p(1)$ this is equivalent to finding the larger of $p_0(a)$ and $p_1(a)$. We call this a *soft decision channel* because the electronics has the capability of measuring a variety of values each associated with a different likelihood that 0 or 1 was sent. This contrasts with The channel in the previous subsection where a definitive decision—also called a *hard decision*—is made by the electronics as to whether 0 or 1 was sent.

In general the "alphabet" of possible received values, A, could be quite large. For example, we could assume that the error is due to a Gaussian distribution with variance σ: The received alphabet is $A = \mathbb{R}$ and a probability density function is used. The probability that $a \in \mathbb{R}$ is received when 0—represented by $-2V$—is sent is $\int_{-\infty}^{a} p_0(a)\,da$ where

$$p_0(a) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(a-2)^2}{2\sigma^2}}$$

The probability that $a \in \mathbb{R}$ is received when 1—represented by $2V$—is sent is $\int_{-\infty}^{a} p_0(a)\,da$ where

$$p_1(a) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(a+2)^2}{2\sigma^2}}$$

Hard decision decoding

Consider now the situation with error-correction coding. Let $C \subseteq \mathbb{F}_2^n$ be a linear code. We assume that for any component $i \in \{1,\ldots,n\}$ some codeword is nonzero in that

component. This is reasonable, for it would make no sense to have all codewords be 0 in a particular position. Under this assumption it is easy to show that, for any component, exactly half of the codewords are nonzero in that component (see exercise). We assume that the source produces code vectors with equal probability; $P(c) = P(c')$ for all $c, c' \in C$.

We also assume that the corruption of any given bit happens independently of the corruption of the others. This may not be realistic, but it makes analysis much easier. More formally, we assume a *memoryless binary symmetric channel with crossover probability* $\alpha < 1/2$: the probability of a bit being received erroneously is given by the binary symmetric channel,

$$p_b(a) = \begin{cases} 1-\alpha & \text{if } a = b \\ \alpha & \text{else} \end{cases}$$

and the probability of receiving $w \in \mathbb{F}_2^n$ when $c \in \mathbb{F}_2^n$ is sent depends only on the individual components

$$P_c(w) = \prod_{i=1}^{n} p_{c_i}(w_i) \tag{4}$$

Therefore the probability depends only on $t = \mathrm{wt}(w - c)$,

$$P_c(w) = \alpha^t (1-\alpha)^{n-t} \tag{5}$$

It is an exercise to check that P_c is indeed a distribution on $\mathbb{F}_2^n$.

The decoding problem is: Given that the target receives w, find the c maximizing $P_w(c)$. Since we assumed codewords are equally likely, Proposition 1.10 shows that this is equivalent to finding the c that maximizes $P_c(w)$. Since $\alpha < 1/2$, $P_c(w)$ is maximal for t as small as possible. Thus decoding reduces to finding the closest codeword c to the received vector w, using the Hamming distance.

Proposition 2.1. *Using the memoryless binary symmetric channel, and assuming all codewords are equally likely, maximum likelihood decoding is equivalent to minimum distance decoding.*

Soft decision decoding of a linear code

We now integrate the soft decision channel for each bit with a linear code. Let $C \subseteq \mathbb{F}_2^n$. We assume a uniform distribution on C. We also assume, as in the previous section, that the channel is memoryless— the corruption of any given bit occurs independently of the corruption of the others—but this time the target electronics allow for measurement of each corrupted bit as one of a range of values. Let A be the set of possible measurements. Formally, we assume a *memoryless soft decision channel*: the conditional probabilities $p_0(a)$ and $p_1(a)$, for each $a \in A$, are independent of the component i for $1 \leq i \leq n$, and the probability that the vector v is sent and $w \in A^n$ is received is

$$p_v(w) = \prod_{i=1}^{n} p_{v_i}(w_i)$$

If the target receives $w \in A^n$ the decoding goal is to find the $c \in C$ that maximizes $P_w(c)$. Since w is fixed and the codewords are equally likely, Proposition 1.10 says this is equivalent to maximizing

$$P_c(w) = \prod_{i=1}^{n} p_{c_i}(w_i)$$

Now focus on each factor. Recall that exactly half of the codewords are nonzero in each component. Since the codewords are equally likely, $p(c_i) = 1/2$ whether $c_i = 0$ or 1. We multiply the previous equality by the constant $2^{-n} \prod_{i=1}^{n} 1/p(w_i)$. Or objective is then to find c maximizing

$$\begin{aligned} \frac{1}{2^n \prod_{i=1}^{n} p(w_i)} P_c(w) &= \prod_{i=1}^{n} p_{c_i}(w_i) \left(\frac{p(c_i)}{p(w_i)} \right) \\ &= \prod_{i=1}^{n} \frac{p(c_i, w_i)}{p(w_i)} \\ &= \prod_{i=1}^{n} p_{w_i}(c_i) \end{aligned}$$

At the end we have a fairly simple expression. It is also intuitively reasonable. For each i, we have a probability distribution p_{w_i} on $\mathbb{F}_2$ based on the fact that we received w_i. The previous formula says that we should compute $\prod_{i=1}^{n} p_{w_i}(c_i)$ for each codeword c and find the codeword maximizing that expression.

This computation is infeasible when $k = \dim C$ is large since there are 2^k codewords. In the next section we introduce the generalized distributive law and show how it can dramatically simplify the computation. It only applies to a very small family of codes, which are not strong error correctors. But the concept in the generalized distributive law leads to an efficient algorithm which is broadly applicable and very effective.

Exercises 2.2.

1. Let V be a subspace of $\mathbb{F}_2^n$. Suppose that for some i with $1 \leq i \leq n$, there exists a vector $v \in V$ with $v_i \neq 0$. Show that exactly half of the vectors in V are 1 in the ith coordinate.

   Generalize to arbitrary finite fields.

2. For $i \in \{1, \ldots, n\}$, let p_i be a distribution on $\mathbb{F}_2$. Define a function on $\mathbb{F}_2^n$ by $P(w) = \prod_{i=1}^{n} p_i(w_i)$. Show that this is a distribution (even when the p_i are distinct).

3 The Generalized Distributive Law and Efficient Soft Decision Decoding

The distributive law, $ab + ac = a(b + c)$ can be interpreted in terms of computational complexity. The left hand side requires 2 multiplications and 1 addition, while the right hand side requires just 1 multiplication and 1 addition. The difference becomes more dramatic with several terms:

$$\sum_{i=1}^{r} ab_i = a \sum_{i=1}^{r} b_i$$

involves r multiplications and $r-1$ additions versus 1 multiplication and $r-1$ additions, and

$$\sum_{i=1}^{r}\sum_{j=1}^{s}\sum_{k=1}^{t}\sum_{l=1}^{v} a_i b_j c_k d_l = \left(\sum_{i=1}^{r} a_i\right)\left(\sum_{j=1}^{s} b_j\right)\left(\sum_{k=1}^{t} c_k\right)\left(\sum_{l=1}^{v} d_l\right)$$

involves $rstv$ multiplications and $rstv - 1$ additions versus 3 multiplications and $r + s + t + v - 4$ additions.

You probably assumed that a, b, c were real numbers in the preceeding paragraph, but the distributive law is assumed in a variety of algebraic objects. The more general formulas above can be proved by induction from the basic $ab + ac = a(b + c)$. The minimal requirements we might impose are the following:

- R is a set with two operations $+$ and $\times$,
- $+$ and $\times$ are associative and commutative,
- 0 is the identity for $+$ and 1 is the identity for $\times$, and
- $\times$ distributes over $+$.

Such an object is called a *semiring* since it lacks only the existence of an additive inverse to be a ring. The paper of Aji and McEliece gives several examples, but we will just consider three which are of interest in decoding. One is the real numbers with the usual operations of addition and multiplication. Another is the nonnegative reals with maximization (in the role of $+$) and multiplication. We have a distributive law

$$\max(ab, ac) = a \max(b, c)$$

In this context the formulas above become

$$\max_{i=1}^{r} ab_i = a \max_{i=1}^{r} b_i$$

involves r multiplications and $r-1$ comparisons versus additions versus 1 multiplication and $r-1$ comparisons, and

$$\max_{i=1}^{r}\max_{j=1}^{s}\max_{k=1}^{t}\max_{l=1}^{v} a_i b_j c_k d_l = \left(\max_{i=1}^{r} a_i\right)\left(\max_{j=1}^{s} b_j\right)\left(\max_{k=1}^{t} c_k\right)\left(\max_{l=1}^{v} d_l\right)$$

involves $rstv$ multiplications and $rstv - 1$ comparisons versus 3 multiplications and $r + s + t + v - 4$ comparisons.

The final example is the set of reals with minimization (in the role of $+$) and addition (in the role of $*$).

$$\min(a + b, a + c) = a + \min(b + c)$$

Here is an application of the GDL.

Proposition 3.1. *For $i = 1, \ldots, n$, let p_i be distributions on $\mathbb{F} = \mathbb{F}_2$. For $a \in \mathbb{F}$, let*

$$\nu(a) = \sum_{\substack{v \in \mathbb{F}_2^n \\ \mathrm{wt}(v) = a \mod 2}} \prod_{j=1}^{n} p_j(v_j)$$

Then $\nu(0) + \nu(1) = 1$, and $\nu(0) - \nu(1) = \prod_{i=1}^{n}(p_i(0) - p_i(1))$.

Proof:

$$\begin{aligned}
\nu(0) + \nu(1) &= \sum_{\substack{v \in \mathbb{F}^n \\ \mathrm{wt}(v) = 0 \mod 2}} \prod_{j=1}^{n} p_j(v_j) \qquad + \sum_{\substack{v \in \mathbb{F}^n \\ \mathrm{wt}(v) = 1 \mod 2}} \prod_{j=1}^{n} p_j(v_j) \\
&= \sum_{v \in \mathbb{F}^n} \prod_{j=1}^{n} p_j(v_j)
\end{aligned}$$

Using the GDL

$$\begin{aligned}
&= \prod_{j=1}^{n} (p_j(0) + p_j(1)) \\
&= 1
\end{aligned}$$

On the other hand,

$$\begin{aligned}
\nu(0) + \nu(1) &= \sum_{\substack{v \in \mathbb{F}^n \\ \mathrm{wt}(v) = 0 \mod 2}} \prod_{j=1}^{n} p_j(v_j) \qquad - \sum_{\substack{v \in \mathbb{F}^n \\ \mathrm{wt}(v) = 1 \mod 2}} \prod_{j=1}^{n} p_j(v_j) \\
&= \sum_{v \in \mathbb{F}^n} \prod_{j=1}^{n} ((-1)^{v_j} p_j(v_j))
\end{aligned}$$

Using the GDL

$$= \prod_{j=1}^{n} (p_j(0) - p_j(1))$$

□

Here is the general formulation of the GDL.

Theorem 3.2. *Let I be a finite set. For each $i \in I$ let D_i be a finite set and F_i a function with domain D_i taking values in a semiring R. Let $\prod_{i\in I} D_i$ be the Cartesian product of the sets D_i. Then*

$$\sum_{v\in\prod_{i\in I} D_i} \left(\prod_{i\in I} F_i(v_i)\right) = \prod_{i\in I}\left(\sum_{x\in D_i} F_i(x)\right)$$

PROOF: Induction on the number of elements in I. □

An example of efficient soft decision decoding

Here is an example from Wiburg's thesis [5]. Let $\mathbb{F}_2$ be the field of two elements and consider the code C with check matrix

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

We want an efficient algorithm for maximum likelihood decoding of C. We will index the rows of H with $V = \{1, 2, 3, 4, 5, 6, 7\}$ and the columns with $\Lambda = \{A, B, C\}$. Consider the graph G whose vertex set is $V \cup \Lambda$ and whose edge set is $E = \{(i, \alpha) : H_{i,\alpha} \neq 0\}$. The graph is called bipartite because edges can exist between a vertex in V and a vertex in Λ but not between two vertices in V or two vertices in Λ.

In the section on soft decision decoding we reduced the problem of maximum likelihood decoding to the maximization of $Q(c)$ for $c \in C$ where

$$Q(c) = \prod_{i=1}^{n} p_i(c_i)$$

Here p_i is a probability distribution on $\{0, 1\}$ that is based on the target having received some particular signal w_i for the ith bit. (We won't write w_i since it is relevant only to determine the probabilities p_{w_i}.) Let us assume the codeword maximizing $Q(c)$ is unique and let us call it $\bar{c}$. For each $k \in V$ let

$$q_k(0) = \max_{\substack{c\in C \\ c_k=0}} Q(c) \tag{7}$$

$$q_k(1) = \max_{\substack{c\in C \\ c_k=1}} Q(c) \tag{8}$$

Then

$$\max_{c\in C} Q(c) = \max_{c_k\in\{0,1\}} (\max_{\substack{c\in C \\ c_k=0}} Q(c), \max_{\substack{c\in C \\ c_k=1}} Q(c)) \tag{9}$$

$$= \max_{c_k\in\{0,1\}} q_k(c_k) \tag{10}$$

$$\tag{11}$$

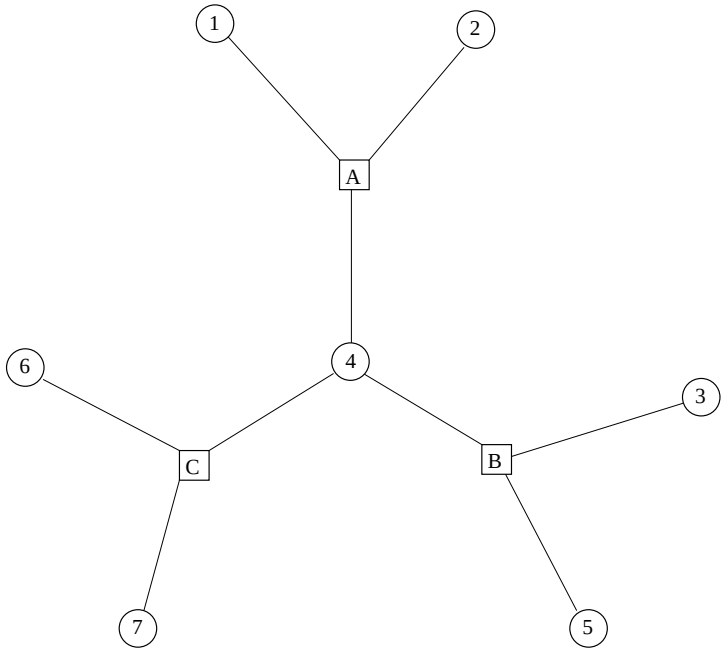


Figure 1: The bipartite graph for the check matrix (6)

We can discover the value of $\bar{c}_k$ from q_k: if $q_k(0) > q_k(1)$ then $\bar{c}_k = 0$ and vice versa. Thus maximum likelihood decoding can be achieved by computing for each k the distributions q_k.

We will apply the generalized distributive law to compute the functions q_k. First we need to extend the function Q to all of $\mathbb{F}^7$ so that Q is a function of 7 independent variables. Let

$$\chi(a,b,c) = \begin{cases} 1 & a+b+c=0 \\ 0 & \text{otherwise} \end{cases}$$

For any vector $v \in \mathbb{F}^7$ which is not in C, either $\chi(v_1, v_2, v_4)$, $\chi(v_3, v_4, v_5)$, or $\chi(v_4, v_6, v_7)$ is 0. On the other hand if $v \in C$ then each of these functions is 1. Thus for $v \in \mathbb{F}^7$ we define

$$Q(v) = p_1(v_1)p_2(v_2)p_3(v_3)p_4(v_4)p_5(v_5)p_6(v_6)p_7(v_7)\chi(v_1,v_2,v_4)\chi(v_3,v_4,v_5)\chi(v_4,v_6,v_7)$$

We now seek the $v \in \mathbb{F}^7$ maximizing $Q(v)$.

Let us compute q_4 from (7) and thereby find $\bar{c}_4$. We apply the GDL and get for $a \in \mathbb{F}_2$,

$$q_4(a) = p_4(a)\nu_A(v_4)\nu_B(v_4)\nu_C(v_4)$$

where

$$\nu_A(v_4) = \max_{v_1} \max_{v_2} p(v_1)p(v_2)\chi(v_1, v_2, v_4)$$
$$\nu_B(v_4) = \max_{v_3} \max_{v_5} p(v_3)p(v_5)\chi(v_3, v_4, v_5)$$
$$\nu_C(v_4) = \max_{v_6} \max_{v_7} p(v_6)p(v_7)\chi(v_4, v_6, v_7)$$

Computing $\nu_A(0)$ and $\nu_A(1)$ requires 4 multiplications and 2 comparisons since

$$\nu_A(0) = \max(p_1(0)p_2(0), p_1(1)p_2(1))$$
$$\nu_A(1) = \max(p_1(0)p_2(1), p_1(0)p_2(1))$$

The same is true for ν_B and ν_C, so the three together cost 12 multiplications and 6 comparisons.

Now

$$q_4(0) = p_4(0)\nu_A(0)\nu_B(0)\nu_C(0)$$

requires 3 multiplications as does $q_4(1)$, and finally we need to compare $q_4(0)$ and $q_4(1)$, so this adds 6 multiplications and 1 comparison. Thus altogether 18 multiplications and 7 comparisons.

This compares favorably with computing $Q(c)$ directly for all 16 codewords (6 multiplications each) and then comparing the results (15 comparisons). One might argue that we have only computed $\bar{c}_4$ and that computing the other $\bar{c}_k$ will multiply the complexity by 7, but this is not the case, as we now show.

Let us compute q_7. For $a \in \mathbb{F}_2$,

$$q_7(a) = p_7(a) \left(\max_{v_4, v_6} \mu_C(v_4)p_6(v_6)\chi(v_4, v_6, v_7) \right)$$

where

$$\mu_C(v_4) = p_4(v_4)\nu_A(v_4)\nu_B(v_4)$$

Since we have already computed ν_A and ν_B, computing μ_C requires 2 multiplications for each value of v_4, thus 4 multiplications. Computing the term in parenthesis is analogous to the computation of ν_A so it requires 4 multiplications and 2 comparisons. Finally, computing $q_7(0)$ and $q_7(1)$ and choosing the larger requires 2 multiplications and 1 comparison.

Computing $\bar{c}_k$ for the other values of k is similar, but notice that μ_C is computed once and used for both $k = 6$ and 7.

The best way to analyze this is to relate the computational burden for each of the functions we introduced to the graph. For each edge, we have to compute a function like ν_A. For each inner edge, we have to compute a function like μ_C. At each outer vertex the final step in the computation of q_k involves 2 multiplications and 1 comparison, while at $k = 4$ we have 6 multiplications and 1 comparison. This is tabulated in the figure.

The results are not that impressive when compared with the brute force method. One needs a larger graph to see gains in efficiency.

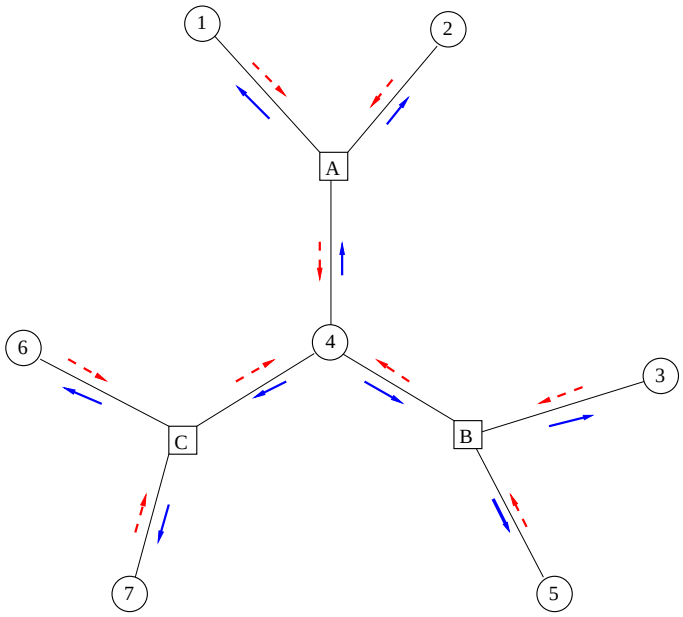


Figure 2: The bipartite graph labeled with message functions

	in	out	total
inner edge	(4,2)	(4,0)	(24,6)
outer edge	0	(4,2)	(24,12)
inner vertex	(6,1)	x	(6,1)
outer vertices	(2,1)	x	(12,6)
TOTAL			(66,25)

Table 1: Computational complexity (multiplications, comparisons) as it relates to the graph.

4 Belief Propagation Algorithms

Suppose we define a code as the null space of some $n \times m$ binary matrix H. It will be convenient to describe belief propagation algorithms using the bipartite graph of H. The bipartite graph of H has $n+m$ vertices, one for each row and one for each column. Let $\{r_1, \ldots, r_n\}$ be the row vertices and $\{c_1, \ldots, c_m\}$ the column vertices. There is an edge (r_i, c_j) if and only if H_{ij} is nonzero.

Given any bipartite graph G, after enumerating the two sets of vertices, there is a unique binary matrix whose graph is G. We can construct a code as the left null-space of this matrix. Different enumerations give different matrices, but they differ only by permutation, so the resulting codes are equivalent.

We will describe the decoding algorithm in terms of a bipartite graph with vertex sets V and Λ with the understanding that the code is defined as the left nullspace of a $\#V \times \#\Lambda$ check matrix corresponding to some enumeration of V and Λ.

Notation

We will write the binary field as $\mathbb{F}$ in this section.

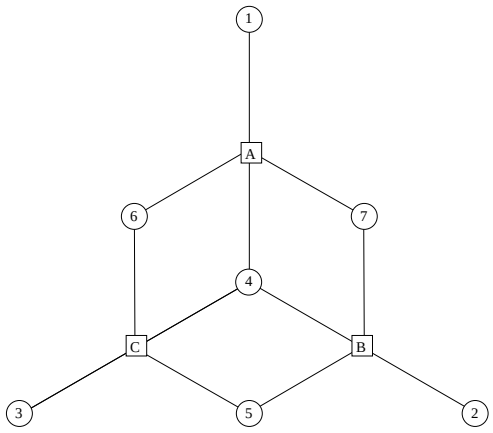
Let V be a finite set of n elements. We will write the n dimensional vector space with basis elements indexed by V as $\mathbb{F}^V$. For any subset A of V we can project $\mathbb{F}^V$ onto $\mathbb{F}^A$ and embed $\mathbb{F}^A$ into $\mathbb{F}^V$ in the natural way. For $v \in \mathbb{F}^V$ and $A \subseteq V$ we will write v_A for the projection of v onto $\mathbb{F}^A$. For a singleton $\{i\} \subseteq V$ we simply write v_i instead of $v_{\{i\}}$. We define χ to act on vectors in any space $\mathbb{F}^A$ for $A \subseteq V$ by

$$\chi(v_A) = \begin{cases} 1 & \text{if } \sum_{i \in A} v_i = 0 \\ 0 & \text{otherwise} \end{cases}$$

Let V and Λ be the vertex sets of a bipartite graph, G. For each $\alpha \in \Lambda$, there is an associated subset of V, $\{i \in V : i \operatorname{adj} \alpha\}$. We will identify α with this subset, so that we may project a $v \in \mathbb{F}^V$ onto the space $\mathbb{F}^\alpha$ and call the image v_α. Thus we may also think of Λ as a collection of subsets of V. The code $C \subseteq \mathbb{F}^V$ defined by the graph is $\{v \in \mathbb{F}^V : \chi(v_\alpha) = 1 \quad \forall \alpha \in \Lambda\}$. In other words, $\prod_{\alpha \in \Lambda} \chi(v_\alpha)$ is the indicator function for C.

In the example of the previous section, $V = \{1, 2, 3, 4, 5, 6, 7\}$ and $\Lambda = \{A, B, C\}$ where $A = \{1, 2, 4\}$, $B = \{3, 4, 5\}$, $C = \{4, 6, 7\}$. In the graph below, $V = \{1, 2, 3, 4, 5, 6, 7\}$ and $\Lambda = \{A, B, C\}$ with $A = \{1, 4, 6, 7\}$, $B = \{2, 4, 5, 7\}$ and $C = \{3, 4, 5, 6\}$. With the expected enumeration of rows and columns, this gives the Hamming code with check matrix

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \tag{12}$$



The local max-product algorithm

We return now to the decoding problem. For each $i \in V$ we have distributions p_i on $\{0,1\}$. We want to find the vector $\bar{c} \in C$ maximizing

$$Q(c) = \prod_{i \in V} p_i(c_i)$$

We extend Q to all of $\mathbb{F}^V$ by multiplying by the indicator function for C,

$$Q(v) = \prod_{i \in V} p_i(v_i) \prod_{\alpha \in \Lambda} \chi(v_\alpha)$$

Notice $Q(v) = 0$ for $v \notin C$.

Input For each vertex $i \in V$, the distribution p_i on $\mathbb{F}^{\{i\}}$.

Data Structures

For each edge (i, α), two functions on $\mathbb{F}^{\{i\}}$: $\mu_{i\alpha}$ and $\nu_{\alpha i}$.

For each i a function on $\mathbb{F}^{\{i\}}$: q_i.

Initialization For each edge (i, α) and for each $a \in \mathbb{F}^{\{i\}}$:

$$\mu_{i\alpha}(a) = p_i(a)$$

Algorithm For $r = 1$ to R.

For each edge (i, α) and for each $a \in \mathbb{F}^{\{i\}}$:

$$\nu_{\alpha i}(a) = \max_{\substack{v \in \mathbb{F}^\alpha \\ v_i = a}} \chi(v) \prod_{\substack{j \in \alpha \\ j \neq i}} \mu_{j\alpha}(v_j)$$

$$\mu_{i\alpha}(a) = p_i(a) \prod_{\substack{\beta \ni i \\ \beta \neq \alpha}} \nu_{\beta i}(a)$$

For each $i \in V$ and each $a \in \mathbb{F}$,

$$q_i(a) = p_i(a) \prod_{\beta \ni i} \nu_{\beta i}(a)$$

Output The hard decision based on q_i: Vector $w \in \mathbb{F}^V$ such that

$$w_i = \begin{cases} 1 & \text{if } q_i(1) > q_i(0) \\ 0 & \text{else} \end{cases}$$

The local sum-product algorithm

The preceeding algorithm is designed to compute the most likely codeword, that is the $c \in C$ maximizing $Q(c) = \prod_i^n p_i(c_i)$. We could also have computed for each i the most likely value for the ith bit. We do this by computing

$$q_k(0) = \sum_{\substack{c \in C \\ c_k = 0}} Q(c) \tag{13}$$

$$q_k(1) = \sum_{\substack{c \in C \\ c_k = 1}} Q(c) \tag{14}$$

$$\tag{15}$$

and then comparing the two. The computational problem is exactly analogous to the max product algorithm but with summation replacing maximization.

Input For each vertex $i \in V$, the distribution p_i on $\mathbb{F}^{\{i\}}$.

Data Structures

For each edge (i, α), two functions on $\mathbb{F}^{\{i\}}$: $\mu_{i\alpha}$ and $\nu_{\alpha i}$.

For each i a function on $\mathbb{F}^{\{i\}}$: q_i.

Initialization For each edge (i, α) and for each $a \in \mathbb{F}^{\{i\}}$:

$$\mu_{i\alpha}(a) = p_i(a)$$

Algorithm For $r = 1$ to R.

For each edge (i, α) and for each $a \in \mathbb{F}^{\{i\}}$:

$$\nu_{\alpha i}(a) = \sum_{\substack{v \in \mathbb{F}^{\alpha} \\ v_i = a}} \chi(v) \prod_{\substack{j \in \alpha \\ j \neq i}} \mu_{j\alpha}(v_j)$$

$$\mu_{i\alpha}(a) = p_i(a) \prod_{\substack{\beta \ni i \\ \beta \neq \alpha}} \nu_{\beta i}(a)$$

For each $i \in V$ and each $a \in \mathbb{F}$,

$$q_i(a) = p_i(a) \prod_{\beta \ni i} \nu_{\beta i}(a)$$

Output The hard decision based on q_i: Vector $w \in \mathbb{F}^V$ such that

$$w_i = \begin{cases} 1 & \text{if } q_i(1) > q_i(0) \\ 0 & \text{else} \end{cases}$$

5 Local Algorithms Applied to a Tree

Definition 5.1. Let G be a graph with vertex set V. A *path of length* n in G is a sequence of $n+1$ vertices, $v_0, v_2, \ldots, v_{n-1}, v_n$ such that $\{v_i, v_{i+1}\}$ is an edge of G for each $i = 0, \ldots, n-1$. The *origin* of the path is v_0 and the *terminus* is v_n. The path is called *simple* if no edge is traversed twice.

Definition 5.2. Let T be a finite tree with vertex set V. For each $t \in V$ let

$$d_T(t) = \max\{\text{ length } P : P \text{ is a simple path with origin } t\}$$

Let the *diameter* of T be

$$D(T) = \max_{t \in V} d(t)$$

Definition 5.3. Let T be a tree and let $\{t, u\}$ be an edge of T. Then T without the edge $\{t, u\}$ is the disjoint union of two trees. Let $T_{t,u}$ be the tree containing t and let T_{ut} be the tree containing u.

Lemma 5.4. *Let T be a finite tree and let $\{t, u\}$ be an edge of T. Then $d_{T_{t,u}}(t) < d_T(u)$.*

PROOF: Let $n = d_{T_{t,u}}(t)$ and let $P = t, x_1, \ldots, x_n$ be a simple path of length n with origin t. Then $Q = u, t, x_1, \ldots, x_n$ is a simple path in T of length $n+1$ with origin u. Therefore, $d_T(u) \geq n+1$ □

The following proof applies to both the sum-product and the max-product algorithms.

Theorem 5.5. *Let V and Λ be the vertex sets of a bipartite graph G. Suppose that G is a tree and that the leaves of G are all elements of V. The local sum-product will converge after $D(G)/2$ iterations.*

PROOF: We proceed by induction using as induction hypothesis the following two statements.

***** For any edge (i, α) with $d_{G_{i\alpha}}(i) \leq 2r$, $\mu_{i\alpha}$ is constant after r iteration.
For any edge (i, α) with $d_{G_{\alpha i}}(\alpha) \leq 2r-1$, $\nu_{\alpha i}$ is constant after r iterations.

It is clear that $d_{G_{i\alpha}}(i) \leq D(G)$ and, since $\alpha \in \Lambda$ is not a leaf, $d_{G_{\alpha i}}(\alpha) \leq D(G) - 1$. Therefore, once * is established, for $r = D(G)/2$, we see that $\mu_{i\alpha}$ and $\nu_{\alpha i}$ are both constant after r iterations.

When $r = 0$, $d_{G_{\alpha i}}(\alpha) \leq -1$ is never true and $d_{G_{i\alpha}}(i) \leq 0$ only when i is a leaf and α is the lone vertex adjacent to i. In the algorithm,

$$\begin{aligned}\mu_{i\alpha}(a) &= p_i(a) \prod_{\substack{\beta \ni i \\ \beta \neq \alpha}} \nu_{\beta i}(a) \\ &= p_i(a)\end{aligned}$$

since the product is over the empty set.

Assume the induction hypothesis is true for $r-1$. Let (i, α) be an edge such that $d_{G_{\alpha i}}(\alpha) \leq 2r-1$. From the algorithm we see that $\nu_{\alpha i}$ is dependent only on the unchanging

function χ and the $(r-1)$th computation of $\mu_{j\alpha}$, for j adjacent to α, and $j \neq i$. Each such j is a vertex of $G_{\alpha i}$, and by the lemma,

$$d_{G_{j\alpha}}(j) \leq d_{G_{\alpha i}}(\alpha) - 1 \leq 2r - 2$$

Therefore the induction hypothesis applies and $\mu_{j\alpha}$ is constant after $r-1$ iterations. consequently, $\nu_{\alpha i}$ is constant after r iterations.

Let(i, α) be an edge such that $d_{G_{i\alpha}}(i) = 2r$. Let $\beta \neq \alpha$ be adjacent to i. From the algorithm we see that $\mu_{i\alpha}$ is dependent only on the rth computation of $\nu_{\beta i}$ for β adjacent to i and $\beta \neq \alpha$. Each such β is a vertex of $G_{i\alpha}$ and by the lemma

$$d_{G_{\beta i}}(\beta) \leq d_{G_{i\alpha}}(i) - 1 \leq 2r - 1$$

The previous paragraph shows that $\nu_{\beta i}$ is constant after r iterations. Thus $\mu_{i\alpha}$ is constant after r iterations. □

The induction hypothesis can be made more precise.

Corollary 5.6. *For any edge (i, α) with $d_{G_{i\alpha}}(i) \leq 2r$, let $G_{i\alpha}$ have vertex sets $U \subseteq V$ and $\Gamma \subseteq \Lambda$. Then after r iterations*

$$\mu_{i\alpha}(a) = \sum_{\substack{v \in \mathbb{F}^U \\ v_i = a}} \prod_{j \in U} p_j(v_j) \prod_{\beta \in \Gamma} \chi(v_\beta)$$

For any edge (i, α) with $d_{G_{\alpha i}}(\alpha) \leq 2r - 1$, let G' be the union of $G_{\alpha i}$ with the vertex i and the edge (i, α). Let G' have vertex sets $U \subseteq V$ and $\Gamma \subseteq \Lambda$. Then

$$\nu_{\alpha i}(a) = \sum_{\substack{v \in \mathbb{F}^U \\ v_i = a}} \prod_{\substack{j \in U \\ j \neq i}} p_j(v_j) \prod_{\beta \in \Gamma} \chi(v_\beta)$$

PROOF: By induction. The notation is a bit out of hand, so I am still working on this. Perhaps the best way to prove it is to work in a more general setup as in [6]. □

Notice also that

$$\sum_{\substack{v \in \mathbb{F}^V \\ v_i = a}} Q(v) = \mu_{i\alpha}(a) \nu_{\alpha i(a)}$$

6 Another Version of the Sum-Product Algorithm

We have shown that the sum-product algorithm produces the most likely bit when the graph of the check matrix is a tree (13). Unfortunately the resulting codes tend to have low minimum distance and are not good for error correction. In practice, decoding is very successful using variants of these local algorithms even when the graph is not a tree. If the graph is sparse then the computational complexity is low, and if the graph has no small cycles (say girth 8, or even girth 6 if the 6 cycles are rare) then the performance is excellent.

This is rather mysterious to me, and I would like to understand it better. A recent paper [2] has several references and a very nice approach to identifying the conditions that determine good coding performance.

A separate issue is the computational one. The initial values $p_i(a)$ are between 0 and 1 so for n large the products that we take become quite small. [7] discusses several computational tricks to handle this practical problem. Here is my interpretation, but it is untested!

Let p be a distribution on a two element set, say $\mathbb{F}$. We define

$$\begin{aligned}\delta p &= p(0) - p(1)\\ \varrho p &= p(1)/p(0)\end{aligned}$$

Then it is easy to verify that

$$\delta p = \frac{1-\varrho p}{1+\varrho p} \qquad \varrho p = \frac{1-\delta p}{1+\delta p}$$

Notice also that δp is in the interval $[-1,1]$ while, if we allow infinity as a limit, ϱp is in the interval $[0,\infty]$.

$$\begin{array}{lll}\delta p = 0 & \Longleftrightarrow p(0) = p(1) & \Longleftrightarrow \varrho p = 1\\ \delta p = 1 & \Longleftrightarrow p(0) = 1 & \Longleftrightarrow \varrho p = 0\\ \delta p = -1 & \Longleftrightarrow p(0) = -1 & \Longleftrightarrow \varrho p = \infty\end{array}$$

In the sum-product algorithm,

$$\mu_{i\alpha}(a) = p_i(a) \prod_{\substack{\beta \ni i\\ \beta \neq \alpha}} \nu_{\beta i}(a)$$

So

$$\varrho\mu_{i\alpha} = \varrho p_i(a) \prod_{\substack{\beta \ni i\\ \beta \neq \alpha}} \varrho\nu_{\beta i}$$

From the definition of q_i in the algorithm we also have

$$\varrho q_i = \varrho p_i(a) \prod_{\beta \ni i} \varrho\nu_{\beta i}$$

Then

$$\varrho\mu_{i\alpha} = \varrho q_i / \varrho\nu_{\alpha i}$$

The local sum-product algorithm also sets

$$\nu_{\alpha i}(a) = \sum_{\substack{v\in\mathbb{F}^{\alpha} \\ v_i=a}} \chi(v) \prod_{\substack{j\in\alpha \\ j\neq i}} \mu_{j\alpha}(v_j)$$

Let $\gamma = \alpha \setminus \{i\}$, then

$$\nu_{\alpha i}(a) = \sum_{\substack{v\in\mathbb{F}^{\gamma} \\ \mathrm{wt}(v)=a \mod 2}} \prod_{j\in\gamma} \mu_{j\alpha}(v_j)$$

Applying Proposition 3.1,

$$\delta\nu_{\alpha i} = \prod_{\substack{j\in\alpha \\ j\neq i}} \delta\mu_{j\alpha}$$

Letting Q_α be defined by

$$\delta Q\alpha = \prod_{j\in\alpha} \delta\mu_{j\alpha}$$

We have

$$\delta\nu_{\alpha i} = \frac{\delta Q_\alpha}{\mu_{i\alpha}}$$

This motivates the following algorithm

Input For each vertex $i \in V$, ϱp_i.

Data Structures

For each edge (i, α), two real numbers. $\delta\mu_{i\alpha}$ and $\varrho\nu_{\alpha i}$.

For each i a real number ϱq_i.

For each α a real number δQ_α

Initialization For each edge (i, α)

$$\delta\nu_{\alpha i}^{(0)} = 1$$
$$\varrho q_i^{(0)} = \varrho p_i$$

Algorithm For $r = 1$ to R.

$$\begin{aligned}
\delta\mu_{i\alpha}^{(r)} &= \frac{\varrho\nu_{\alpha i}^{(r-1)} - \varrho q_i^{(r-1)}}{\varrho\nu_{\alpha i}^{(r-1)} + \varrho q_i^{(r-1)}} \\
\delta Q_\alpha^{(r)} &= \prod_{i \,\mathrm{adj}\, \alpha} \delta\mu_{i\alpha}^{(r)} \\
\varrho\nu_{\alpha i}^{(r)} &= \frac{\delta\mu_{i\alpha}^{(r)} - \delta Q_\alpha^{(r)}}{\delta\mu_{i\alpha}^{(r)} + \delta Q_\alpha^{(r)}} \\
\varrho q_i^{(r)} &= \varrho p_i \prod_{\alpha \,\mathrm{adj}\, i} \varrho\nu_{\alpha i}^{(r)}
\end{aligned}$$

Output The hard decision based on ϱq_i: Vector $w \in \mathbb{F}^V$ such that

$$w_i = \begin{cases} 1 & \text{if } \varrho q_i > 1 \\ 0 & \text{else} \end{cases}$$

When ϱq_i goes to 0, ith bit is a 0.
When ϱq_i goes to ∞, ith bit is 1.

Exercises for iterative decoding

.

1) Let C be the binary code of length n consisting of all codewords whose weight is even. In other words, a check matrix for C is a single column vector of all 1's. Write a program for iterative decoding.

2) Consider the Hamming code of length 7 with check matrix.

$$H = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

a) Draw the graph.
b) Let p be the vector whose entry p_i is the probability that the ith bit is 0. Let $p = [.2, .4, .9, .7, .4, .6, .9]$. Do one iteration of the iterative decoding algorithm by hand.
c) Write a program to take as input an arbitrary vector of probabilities $(p_1(0), \ldots, p_7(0))$. Experiment to determine which vectors decode to the all zero vector.

3) Majority logic decoding uses hard decision input and a simple iterative algorithm that changes a given bit if the majority of its neighbors "think" it should change. Implement this algorithm and test it.

4) Here is another simplified algorithm. For each bit, the input is either a 0 or a 1 or an ersasure. The decoder never changes a 0 or 1, but it can change erasures. If a particular check bit is connected to exactly one erasure bit, it uses the data from the other "known" bits to decide what the erasure should be. Implement and test this algorithm. It should be very fast. Experiment with a variety of codes of this type. [14] .

5) Implement the iterative decoding algorithm for arbitrary soft decision input. Experiment with different ways to represent the probability distribution for each bit. For each vertex i you can store $p_i(0)$, $p_i(0) - p_i(1)$, or $p_i(0)/p_i(1)$. Justify the formulas you use. See [7, p.512].

6) Experiment with constructions of LDPC codes from groups. See [8, 9, 10, 11] .

7) Experiment with codes from generalized quadrangles. See See [12].

8) Experiment with codes constructed from geometries. See [13].

9) Experiment with the construction of erasure correcting codes. See [14].

References

[1] R. G. Gallagher, "Low-density parity-check codes," *IRE Trans. Inform. theory* vol IT-8, pp. 21-28, Jan. 1962.

[2] R. Koetter, P. Vontobel, "Graph-covers and iterative decoding of finite length codes," manuscript.

[3] M. Tanner "A recursive approach to low complexity codes" *IEEE Trans. Inform. Theory*, Sep. 1981.

[4] J. Perl, *Probabilistic Reasoning in Intelligent Systems*, Morgan Kauffman, 1988.

[5] Wiberg, "Codes and decoding on graphs," Ph.D. dissertation, no. 440, Linkoping Sweden, 1996.

[6] Aji, McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory*, Mar. 2000.

[7] F. R. Kschischang, B. JK. Frey, H.-A. Loelinger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, Feb. 2001.

Constructions **********

[8] Rosenthal, Vontobel. "Construction of LDPC codes using Ramanujan graphs and ideas from Margulis" see http://www.nd.edu/ ~ rosen/preprints.html, or http://www.isi.ee.ethz.ch/ ~ vontobel/.

[9] Margulis, "Explicit constructions of graphs without short cycles and low density codes. *Combinatorica*, 2(1), pp. 71-78, 1982

[10] Margulis, "Explicit group theoretic constructions of combinatorial schemes and their applications in the constructin of expanders and concentrators." *Problems Inform. Transmission* 24(1):39-46, 1988.

[11] Tanner, Srkdhara, Fuja "A Class of Group-Structured LDPC Codes," http://www.cse.ucsc.edu/ ~ tanner/pubs.html.

[12] Tanner, Vontobel, "Construction of codes based on finite generalized quadrangles for iterative decoding," http://www.isi.ee.ethz.ch/ ~ vontobel/ under porjects.

[13] Kou, Lin, Fossorier, "Low density parity check codes based on finite geometries," IEEE Trans. Inform. Theory, Vol. 47, no. 7, 2001, pp. 2711-2736.

http://www-ee.eng.hawaii.edu/ ~ marc/publications.html (#43).

[14] Luby, Mitzenmacher, Shokrollahi, Spielman, "Efficient erasure correcting codes" *IEEE, Trans. Inform. Theory* Feb. 01, pp 569-584.