Lecture Notes for Math 696
Coding Theory
Iterative Decoding

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

March 18, 2002

These notes are my effort to understand a paper by Aji and McEliece called "The Generalized Destributive Law" [5]. The article gives a clear description of an algorithm that is useful in many areas of applied mathematics: They cite artificial intelligence, belief propogation 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 efficiency of the algorithm is described by "message passing" between the vertices of the graph. I will describe how this works for decoding of binary low-density parity check codes. The algorithm only works for a small class of codes, where the underlying graph is a tree, and therefore seems to be useless in practice. But, the spirit of the algorithm, passing messages between the vertices of the graph, is easily adapted to any low-density parity-check code. I will show one such adaptation. Numerous experiments using these message passing algorithms have been done, and the performance is remarkably good. My understanding is that there is no good theoretical explanation for the performance.

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 probablity $1/52$ 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 $P(x|y)$. Similarly one has $P_y(x)$.

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

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

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

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 distrbution 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 $X \times Y$ 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 $X \times Y$ that determined by P on X and the P_x on Y. 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, A and B. 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), $P(\text{black}) = 1/6 + 1/5 = 11/30$. Then

$$\begin{aligned} P_{\text{black}}(H) &= P(H, \text{black}) \\ &= \frac{1/6}{11/30} \\ &= 5/66 \end{aligned}$$

In a practical application the flip of the coin might represent the choice of two destinations, say Mexico or Arizona, for a fugitive. Drawing from the urn might represent a choice of one of two possible freeways—say 5 So. (then 94 E to Arizona or 5 So on to Mexico) or 8 E. (then to Calexico and the border, or on to Arizona). Knowing that the fugitive was seen on a certain route, what is the likelihood of each destination?

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 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 an unacceptable computational burden, so we must find other means.

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 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 p for 1 and $1-p$ for 0 and that $p < 1/2$. Suppose also that the target knows the value of p and of α. Would it ever make sense for the target to decode a 1 to a 0?

Consider the table of joint probabilities.

	0	1
0	$(1-p)(1-\alpha)$	$(1-p)\alpha$
1	αp	$p(1-\alpha)$

If the target receives a 1 it is more likely that a 1 was sent provided $p(1-\alpha) > (1-p)\alpha$. This is equivalent to $p/(1-p) > \alpha/(1-\alpha)$. Since $p, \alpha \in [0, 1/2]$ and the function $x/(1-x)$ is strictly increasing, this is equivalent to $p > \alpha$. On the other hand, if $p < \alpha$ it makes sense to decode a 1 to a 0.

In error correction coding it is usually assumed that $p(0) = p(1)$ 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. 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 $p_i(j)$ the

probability that $i \in \{0,1\}$ is sent (as $\pm 2V$) and $j \in \{-3,-2,-,0,1,2,3\}$ 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 j is to compare

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

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 $p_0(j)$ and $p_1(j)$. We call this the *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.

Hard decision decoding

Consider now the situation with coding. Let $C \subset \mathbb{F}_2^n$ 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. 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 code vectors are sent through a *memoryless binary symmetric channel with crossover probability* $\alpha < 1/2$. By this we mean that the probability that $w \in \mathbb{F}_2^n$ is received when $v \in \mathbb{F}_2^n$ is sent is

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

where for any i,

$$p_{v_i}(w_i) = \begin{cases} 1-\alpha & \text{if } v_i = w_i \\ \alpha & \text{else} \end{cases} \tag{5}$$

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

$$p_v(w) = \alpha^t (1-\alpha)^{n-t} \tag{6}$$

The decoding problem is: Given that the target receives w, find the c maximizing $p_w(c)$. Our discussion in the section on Bayes theorem shows that since the codewords are equally likely, 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 $C \subset \mathbb{F}_2^n$. We assume a uniform distribution on C. We also assume a *memoryless soft decision channel*: each bit is treated independently in transmission and 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. The conditional probabilities $p_0(a)$ and $p_1(a)$, for each $a \in A$, are known to the target. They are also the same for each component of the code. Thus 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 $p_w(c)$. From Bayes' theorem,

$$p_w(c) = \frac{p(c)p_c(w)}{p(w)}$$

Since $p(w)$ is constant for fixed received vector w and since the codewords are assumed to be equally likely, 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 intuitive. For each i, we have a probability distribution p_{w_i} on $\mathbb{F}_2$ based on the fact that we received w_i. Since w is fixed for the decoding problem let's simplify the notation and call it p_i instead of p_{w_i}. The previous formula says that we should compute $\prod_{i=1}^{n} p_i(c_i)$ for each codeword c and find the codeword maximizing that expression.

Since there are 2^k codewords for $k = \dim C$ this is not feasible. In the next section we introduce the generalized distributive law and show how it can dramatically simplify the computation. Unfortunately it only applies to a very small family of codes.

Exercises 2.2. 1) Let F be a field and let $V \subset F^n$. Supoose that for $i \in \{1, \ldots, n\}$ there is a vector $v \in V$ such that $v_i \neq 0$. Show that $w_i \neq 0$ for half of the vectors $w \in V$.

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 two which are of interest in decoding. One is the real numbers with the usual operations of addition and multiplication. The other 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.

Figure 1: The bipartite graph for the check matrix (7)

An example of efficient soft decision decoding

Here is an example from Wiburg's thesis [4]. Let $\mathbb{F}$ be the field of two elements and consider the code C with check matrix

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

We want an efficient algorithm for maximum likelihod 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 recieved some particular signal w_i for the ith bit. Let us assume the codeword maximizing $Q(c)$

is unique and let us call it $\bar{c}$. For each $k \in V$ let

$$q_k(0) = \max_{\substack{c \in C \\ c_k = 0}} Q(c) \tag{8}$$

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

Then

$$\max_{c \in C} Q(c) = \max_{c_k \in \{0,1\}} (\max_{\substack{c \in C \\ c_k = 0}} Q(c), \max_{\substack{c \in C \\ c_k = 1}} Q(c)) \tag{10}$$

$$= \max_{c_k \in \{0,1\}} q_k(c_k) \tag{11}$$

$$\tag{12}$$

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 (8) and thereby find $\bar{c}_4$. We apply the GDL and get

$$\max_{v \in \mathbb{F}^7} Q(v) = \max_{v_4} p_4(c_4)\mu_A(v_4)\mu_B(v_4)\mu_C(v_4)$$

where

$$\mu_A(v_4) = \max_{v_1} \max_{v_2} p(v_1)p(v_2)\chi(v_1, v_2, v_4)$$

$$\mu_B(v_4) = \max_{v_3} \max_{v_5} p(v_3)p(v_5)\chi(v_3, v_4, v_5)$$

$$\mu_C(v_4) = \max_{v_6} \max_{v_7} p(v_6)p(v_7)\chi(v_4, v_6, v_7)$$

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

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

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

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

Now

$$q_4(0) = p_4(0)\mu_A(0)\mu_B(0)\mu_C(0)$$

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.

$$\max_{v\in\mathbb{F}^7} Q(v) = \max_{v_7} p_7(v_7) \left(\max_{v_6} p_6(v_6) \max_{v_4} \chi(v_4, v_6, v_7)\nu_C(v_4) \right)$$

where

$$\nu_C(v_4) = p_4(v_4)\mu_A(v_4)\mu_B(v_4)$$

Since we have already computed μ_A and μ_B, computing ν_C requires 2 multiplications for each value of v_4, thus 4 multiplications. Computing the term in parenthesis is analogous to the computation of μ_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 ν_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 μ_A. For each inner edge, we have to compute a function like ν_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.

	in	out	total
inner edge	(4,2)	(4,0)	(18,7)
outer edge	0	(4,2)	(12,0)
inner vertex	(6,1)	x	(24,12)
outer vertices	(2,1)	x	(12,6)
TOTAL			(66,25)

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

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

4 The GDL for Decoding Codes Defined by a Tree

Suppose we define a code as the null space of some check matrix H. It will be convenient to index each column of H by a subset of $\{1, \ldots, n\}$, namely the set of indices for where the column is nonzero. Before we describe the decoding algorithm it will be useful to introduce a number of such notational conveniences.

Notation

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 $A \subset V$ we will write v_A for the projection of v onto $\mathbb{F}^A$. For a singleton $\{i\} \subset V$ we simply write v_i instead of $v_{\{i\}}$.

We define χ to act on vectors in any space $\mathbb{F}^A$ for $A \subset V$ by

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

For any subset A of V let $H^A \in \mathbb{F}^V$ be such that

$$(H^A)_i = \begin{cases} 1 & \text{if } i \in A \\ 0 & \text{otherwise} \end{cases}$$

Let $\Lambda \subset 2^V$ be a collection of subsets of V. The vectors H^α for $\alpha \in \Lambda$ determine a code by

$$C = \{c \in \mathbb{F}^V : c \cdot H^\alpha = \text{ for all } \alpha \in \Lambda\}$$

Let G be the graph with vertex set $V \cup \Lambda$ and edge set

$$E = \{\{i, \alpha\} : i \in V, \alpha \in \Lambda, \text{ and } i \in \alpha\}$$

The graph G is bipartite since edges can only exist between a vertex from V and a vertex from Λ. In the example of the previous section, $V = \{1, 2, 3, 4, 5, 6, 7\}$ and $\Lambda = \{A, B, C\}$ where $A = \{1, 2, 4\}$, $B = \{3, 4, 5\}$, $C = \{4, 6, 7\}$.

Back to decoding

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

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

We will make the (huge) assumption that Λ is such that the graph G is a tree. We will see that the GDL can be applied to efficiently compute $\bar{c}$, component by component.

Let A be an arbitrary element of Λ and let $1 \in A$, (1 is arbitrary, I just want to avoid another letter!). Then $\{1, A\}$ is an edge of G. Since G is a tree, removing this edge from G leaves two disconnected trees. Let $I \subset V$ be the set of vertices in V connected to 1 and let $J = V \setminus I$ be the vertices connected to A. Every $\alpha \in \Lambda$ different from A is either contained in I or contained in J, for if it met both there would be a circuit in G. Thus we have

$$\begin{aligned} Q(v) &= \prod_{i \in I} p(v_i) \prod_{j \in J} p(v_j) \chi(v_\alpha) \left(\prod_{\alpha \subset I} \chi(v_\alpha) \right) \left(\prod_{\beta \subset J} \chi(v_\beta) \right) \\ &= \left[\left(\prod_{i \in I} p(v_i) \right) \left(\prod_{\alpha \subset I} \chi(v_\alpha) \right) \right] \left[\chi(v_A) \left(\prod_{j \in J} p(v_i) p(v_j) \right) \left(\prod_{\beta \subset J} \chi(v_\beta) \right) \right] \end{aligned}$$

Let $I' = I \setminus \{1\}$ and define

$$\begin{aligned} \nu_{1A}(v_1) &= p_i(v_i) \max_{v_{I'} \in \mathbb{F}^{I'}} p_{I'}(v_{I'}) \left(\prod_{\alpha \subset I} \chi(v_\alpha) \right) \\ \mu_{A1}(v_1) &= \max_{v_V \in \mathbb{F}^J} \chi(v_A) p(v_J) \left(\prod_{\beta \subset J} \chi(v_\beta) \right) \end{aligned}$$

Then

$$\max_{v \in \mathbb{F}^V} Q(v) = \max_{v_1} \nu_{1A}(v_1) \mu_{A1}(v_1)$$

For $a \in \{0, 1\}$ define,

$$q_1(a) = \nu_{1A}(a) \mu_{A1}(a)$$

Then q_1 determines $\bar{c}_1$: if $q_1(0) > q_1(1)$ then $\bar{c}_1 = 0$ and vice versa.

For $i \in V$ the set of vertices adjacent to i is $\{\alpha \in \Lambda : \{i, \alpha\}\}$ is an edge of G. It is often written as adj (i) or $N(i)$ and it is often called the neighborhood of i. Because of the way we defined our edge set, α is adjacent to i if and only if $i \in \alpha$. Similarly for fixed α, the vertices in V adjacent to α are the i such that $i \in \alpha$.

Proposition 4.1. *Let* 1 *be an arbitrary element of* V *and let* A *be an arbitrary element of* Λ. *Let* $B = A \setminus \{i\}$.

$$\begin{aligned} \mu_{A1}(v_1) &= \max_{v_B} \chi(v_1, v_B) \prod_{i \in B} \nu_{iA}(v_i) \\ \nu_{1A}(v_1) &= p_1(v_1) \prod_{\substack{\alpha \ni 1 \\ \alpha \neq A}} \mu_{\alpha 1}(v_1) \end{aligned}$$

Computational Complexity

We assume that the graph G is a tree, that the length of the code is $n = |V|$, that the degree of any vertex in V is at most r, and that the degree of any vertex in Λ is at most d. We need to compute for each edge $\{1, A\}$ the two functions $\mu A, 1$ and $\nu 1, A$.

From the proposition, computation of $\nu i\alpha(v_i)$ requires less than r multiplications, since there are at most r sets α containing 1. There are n vertices $i \in V$ and at most r vertices in Λ containing i so the complexity of computing all the $\nu_{i\alpha}$ is $O(r^2 n)$.

Computation of $\mu_{A1}(v_1)$ requires choosing all vectors in $\mathbb{F}^B$ and taking the product of $|B|$ terms for each. Since $|B| < d$ this is $d2^d$. We also have to compare the results, which is $O(2^d)$. There are $O(rn)$ pairings of an element of V with one of Λ, so we get $O(d2^d rn)$. This clearly dominates the computational complexity.

5 Decoding of Codes on an Arbitrary Graph

In the last two sections we computed the codeword maximizing $Q(c) = \prod_i^n p_i(c_i)$. We could also have computed for each k the most likely value for that 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 the same as discussed in the previous sections, but with summation replacing comparison.

When the underlying graph is not a tree, the GDL gives little if any benefit. The approach that people have taken is to take the computational organiztion of the algorithm, slightly change the input, and experiment. The experiments have proved to be very successful, but the theoretical understanding of the algorithm is weak. Here is one way to do it, which is a corruption of the algorithm for computing the most likely bit for each vertex $i \in V$..

Initialization : For each vertex $i \in V$, the distribution p_i on $\{0, 1\}$.

Algorithm : For each $1 \in V$,

$$p_1(v_1) := p_1(v_1) \prod_{\substack{\alpha \ni 1 \\ \alpha \neq A}} \mu_{\alpha 1}(v_1)$$

where we let $B = \alpha \setminus \{1\}$ and define

$$\mu_{\alpha 1}(v_1) = \sum_{v_B} \chi(v_1, v_B) \prod_{i \in B} p_i(v_i)$$

The computational complexity is $O(d2^d rn)$ where r is an upper bound on the degree of the vertices in V, d is an upper bound on the vertices in Λ and $n = |V|$ is the length of the code.

References

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

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

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

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

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

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