CS 493: Algorithms for Massive Data Sets	**Tornado Codes**
DATE Tuesday, 3/6/2002	**Scribe: Keith Vallerio**

1 Berlekamp-Welch (Continued...)

1.1 Notation

Last time, we encoded a message using polynomials based on the following notation:

- K symbols: b_0 ... b_{k-1}
- Transmission: t_0 ... t_{n-1}
- Reception: f_0 ... f_{n-1}
- V(x) polynomical of degre K-1+r
- W(x) polynomical of degre r
- V(i) = f_i W(i)

1.2 Basic Idea

We use V(x) and W(x) along with an error locator polynomial E(x) to determine the original message. After determining V(x) and W(x), it is possible to find P(x) since it is the ratio between the two. V(x) has degree K-1+r and K+r unknowns. W(x) has degree r and r unknowns since the leading coefficient of E(x) is 1. From the last class, we saw that these polynomials do exist.

- V(x) = E(x) P(x)
- W(x) = E(x)

Theorem 1.1 *P(x) can be determined from V(x) of degree K-1+r and W(x) of degree K+r as long as we receive at least n bits such that:* $n \leq K + 2r$

Proof. Given polynomials $V_1(x)$, $W_1(x)$, $V_2(x)$, and $W_2(x)$:

- $V_1(i) = f_i\ W_1(i)$
- $V_2(i) = f_i\ W_2(i)$
- $f_i\ V_1(i)\ W_2(i) = f_i\ V_2(i)\ W_1(i)$

The f_i terms cancel when $f_i \neq 0$ and $V_1(i) = V_2(i) = 0$ when $f_i = 0$. Thus, $V_1(x)W_2(x) = V_2(x)W_1(x)$. Since the number of unknowns is K + 2r, if these polynomials agree at n points with n $\geq$ K + 2r, then the they are identical. Thus,

$$\frac{V_1(x)}{W_1(x)} = \frac{V_2(x)}{W_2(x)} = P(x)$$

■

1.3 Alternative To Using a Prime

We saw that you can take a message of length K and add additional information to get length n. Approximately half of the data can be corrupted and you can still recover the original message. However, this was based on using a large value of P > n such that P is a prime number. Why? It turns out that instead of using mod P, you can work with any field. Thus, you can use something more suitable such as GF(2^r). This would be a set of polynomials with coefficients mod 2 mod some prime polynomial of degree r.

Example: GF(2^8) Using bit vectors of size 8 (GF(2^8))

- Given the polynomial: π(x) $= x^8 + x^6 + x^5 + x + 1$
- Represent the polynomial: $x^7 + x^4 + x \rightarrow 10010010$
- If you add another polynomial:

$$\begin{array}{rr} & 10010010 \\ = & 10100010 \\ \hline & 00110000 \end{array}$$

Note, the addition didn't really use the first polynomial. However, it is used when you perform multiplication.

- First note that: $x^8 = (-x^6 - x^5 - x - 1) = x^6 + x^5 + x + 1$
- Using this:

$$\begin{aligned} (x^4 + x)(x^4 + x^2) &= x^8 + x^6 + x^5 + x^3 \\ &= (x^6 + x^5 + x + 1) + x^6 + x^5 + x^3 \\ &= x^3 + x + 1 \end{aligned}$$

1.4 Complexity

This algorithm results in some number of linear equations to solve in order to decode the message. One way to solve these is to use Gaussian elimination. However, the complexity of this is rather high: O(n^3). If a systematic code is used (i.e., $P_i = b_i$), the message can be decoded in O(u v) time. In this case, u is the length of the original message and v is the length of the redundant message. Thus, this method is better than the first. A much better method exists which solves the equations based on performing an FFT. The time complexity of this method is O(nlg^2n).

2 Tornado Codes

This section is based on the following papers. The first paper gives a more detailed proof and the second paper has a simpler description.

- M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical Loss-Resilient Codes," *STOC* 1997
- M. Luby, M. Mitzenmacher, and A. Shokrollahi, "Analysis of Random Processes via And-Or Tree Evaluation," *SODA* 98

2.1 Basics

2.1.1 General Idea

- Start with an n bit message.
- Blow up message into c n bits (c > 1)
- Can decode as long as we receive n bits (Actually, n + ϵ bits)

2.1.2 Why Do This?

- Multicast
- Packet loss/corruption
- Good for large n information and low decode time per bit

2.1.3 Metrics

Measure the code effectiveness based on coding time per bit. The reception effeciency is ideally 1, but this algorithm settles for 1+ϵ (Note: Reed-Solomon reception efficiency was 1.)

- Code effectiveness: $\frac{time\ length}{encode\ length}$
- Reception efficiency $= \frac{\#\ bits\ needed\ to\ decode}{message\ length}$

2.2 Construction

- Transmit c n bits. (Let c = 2 for discussion)
- Reception efficiency = 1 + ϵ
- Time overhead $\approx \ln\left(\frac{1}{\epsilon}\right)$
- (Previous best time efficiency used to be $\left(\frac{1}{\epsilon}\right) \ln\left(\frac{1}{\epsilon}\right)$)

The construction of this code is done in a series of layers. The following discussion assumes that the message granularity is packets instead of bits.

1. Take original n bits.
2. Add $\frac{n}{2}$ parity bits. Each bit is an xor of some entries from the original message of size n.
3. Repeat 2 by adding half as many parity bits as the previous time and performing the xor only on the bits which were added in the previous layer. Repeat this step until you reach a small size which still contains multiple packets. These remaining packets are encoded using a really good (robust) coding scheme. This part can be expensive since the size is small.

In the end n packets becomes 2n packets. These packets are then permuted and transmitted. The encoding time is approximately the average degree of nodes in the graph times n. The discussion which follows assumes that the bit loss rate is distributed across all layers!

2.3 Decoding

Decoding is performed layer by layer in reverse. It is assumed that the receiver obtains n(1+ϵ) bits. Thus, for each layer, the fraction of drops is $\frac{1}{2}(1-\epsilon)$. The basic idea of this scheme is that you start with a layer of size $\frac{n}{2}$ for which you know all the bits. You also have a larger layer of size n bits for which you know $\frac{1}{2}(1+\epsilon)$ bits. Finally, you have a connectivity graph between these two layers which tells you how the xor order was determined. Based on this information, you reconstruct the larger layer. This process is iterated until the final message is decoded.

2.3.1 Crucial Property

If you have a graph with edges such that the end points have an unknown on one side, make sure you have at least 1 edge with only one unknown. You can get the original message as long as you can start determining one bit and then use that to find another edge which has only one unknown.

2.4 How Do You Get Such a Graph?

2.4.1 Constructing a Regular Graph

1. Give each vertex on the left side (size n) 3 outgoing edges.

2. Give each vertex on the right side (size $\frac{n}{2}$) 6 outgoing edges. Thus each size has 3n edges coming out of it.

3. Create a random graph by using a random permutation π (i.e., map π(i) $\mapsto$ j) to link the edges from the left side to the edges on the right. Note that π is completely random.

2.4.2 Evaluating the Graph

Look at the graph by looking at a vertex and it's 5 neighbors. You get something which isn't a tree, but is kind of similar in structure. How is a layer attached to the next?

In a particular layer, a bit is unknown with probabily X with each probabilty X being independent.

- P(All known) $= (1 - X)^5$
- P(You can't decode a given previous layer) $= (1 - (1 - X)^5)^2$

If initially α fraction of bits unknown,

- P(Bit unknown) $= \alpha$
- P(Next bit unknown) = y $= \alpha(1 - (1 - X)^5)^2$
- Note: X < y (where x ϵ (0,α])

Thus, $\alpha(1 - (1 - X)^5)^2 <$ x. It is provable that if you have this ration you can decode the code. With $\alpha < .43$, the decoding scheme will work.

2.4.3 Using an Irregular Graph

Actually, irregular graphs are better. We will see this in detail in the next class, but here is the basic idea.

- Left degree = d
- Right degree = 2d
- If you have approximately $\frac{1}{2}$ unknowns, P(Right vertex has 1 unknown) $\approx \frac{1}{2^{2d-1}}$
- E(Degree 1 vertices that a given vertex on the left side is next to) $= \frac{d}{2^{2d-1}}$. This is a small number which can be improved by using irregular graphs.