6.895 Essential Coding Theory		November 11, 2004
	Lecture 17	
Lecturer: Madhu Sudan		*Scribe: Abhinav Kumar*

1 Topics today

1. Spielman codes + decoding
2. Correcting random error in linear time
3. Expander Codes (Type II)

References : Guruswami, Survey on Error-Correcting codes etc., SIG ACT News 2004

2 Spielman codes

Recall the definition of Spielman codes.

We have a *reducer code* R_k for each k which has a message length of k bits and a parity check of $k/2$ bits. The code is obtained from a low degree bipartite with k left vertices (message) and $k/2$ right vertices. The parity of each vetex on the right is computed as the sum of all its neighbors on the left. Then we take the collection of these parity bits as the total parity check. We will need some expander properties for the graph in order for R_k to be a good code, but these will be specified later.

Now, the Spielman coding procedure can be described thus: C_k : k message bits and $3k$ parity check bits.

$$\begin{array}{rcl} m & \overset{R_k}{\mapsto} & [m, c] \\ c & \overset{C_{k/2}}{\mapsto} & d \\ [c, d] & \overset{R_{2k}}{\mapsto} & d \end{array}$$

Overall, the encoding is $m \mapsto [m, c, d, e]$ where the lengths of m, c, d, e are $k, k/2, 3k/2, k$ respectively.

Assume that $\exists \epsilon > 0$, and a decoding algorithm A for R_k such that if $[m, c]$ is the output of R_k on m, and $[x, y]$ are such that $\delta(x, m) \leq \epsilon$ and $\delta(y, c) \leq \epsilon$ then $A(x, y) = m'$ satisfies $\delta(m, m') \leq \delta(y, c)/4$. That is, the relative error of A is only dependent on the relative error introduced on the parity check. We will show the existence of such an algorithm later in this lecture.

3 Decoding algorithm

The decoding algorithm for the code C_k runs as follows: On input $[m', c', d', e']$,

1. $[c'', d''] \leftarrow A([c', d'], e')$
2. $c''' \leftarrow D_{k/2}([c'', d''])$
3. $m'' \leftarrow A([m', c'''])$
4. Output m''

We hope to correct a constant fraction τ of errors. We can determine τ as below, assuming the existence of R_k, A as given for now.

If less than fraction τ errors happen, then $\Delta([m, c, d, e], [m', c', d', e']) \leq 4\tau k$. So we have

$$\Delta(m, m') \leq 4\tau k, \;\; \Delta([c, d], [c', d']) \leq 4\tau k, \;\; \Delta(e, e') \leq 4\tau k$$

.

Now the inequalities $\Delta([c,d],[c',d']) \le \epsilon(2k)$ and $\Delta(e,e') \le \epsilon k$ hold as long as $4\tau \le \epsilon$. This will imply by the above condition on A that

$$\Delta([c,d],[c'',d'']) \le 2k \cdot \frac{\Delta(e,e')}{4k} \le 2\tau k$$

Now this implies $c = c'''$ in the Step 2 of decoding, by induction, since the fraction of errors is at most $2\tau k/(2k) = \tau$.

Finally, in step 3, we have that $\Delta(m,m') \le 4\tau k \le \epsilon k$, as well as $\Delta(c,c''') = 0$. Therefore the decoding algorithm for R_k gives $\delta(m,m'') \le \delta(c,c''')/4 = 0$. Therefore $m = m''$ as required.

To conclude, we see that we can correct $\tau = \epsilon/4$ fraction of errors.

4 The reducer code R_k

For the reducer R_k, we pick a bipartite graph G_k with k left vertices, $k/2$ right vertices, which is $(c, 2c)$ regular, and a γ, δ expander with $\gamma = 7c/8$. Recall that the parity check consists of the collection of the values assigned to each of the right vertices, which is the parity of all its left neighbors.

For the decoding algorithm, we use the Sipser-Spielman Flip algorithm : while there exists a left vertex with more unsatisfied right neigbours than satisfied ones, flip that vertex.

Suppose that the codeword $[m,c]$ gets mutated to $[m',c']$ where the subset of bits where m and m' differ is the set S of left vertices, and the subset where c and c' differ is T.

Lemma 1 *Let $c > 8$ and $\epsilon \le \delta/(3/2+c)$. Suppose $|S| \le \epsilon k, |T| \le \epsilon(k/2)$. When the Flip algorithm stops, the number of incorrect bits on the left is $\le |T|/2$.*

Proof : Initially, we have

$$\#\text{ unsat right vertices } \le c|S| + T$$

and since the number of unsatisfied right vertices decreases through any iteration of Flip, we see that there can be at most $c|S| + T$ iterations. This ensures that at any stage, if the vertices on the left encode m' and m is the original message, we have

$$\Delta(m,m') \le \Delta(m, m'_{init}) + \#\text{iterations} \le \epsilon k + c|S| + |T| \le \epsilon k(1 + c + 1/2) \le \delta$$

Now, at the end, let S' be the set of incorrect left vertices. The number of unsatisfied neighbors on the right is at least

$$|\Gamma_{unique}(S')| - |T| \ge \frac{3}{4}c|S'| - |T|$$

but also at most $c|S'|/2$ since the algorithm stopped. So we have

$$(\frac{3c}{4} - \frac{c}{2})|S'| \le |T|$$

so that $|S'| \le 4|T|/c < |T|/2$. ■

5 Correcting random errors

The rate vs. distance relationship is not so great for this code, but we can use it to correct random errors as follows.

Take a message of size k, then encode by a Spielman code to get a codeword of size $(1+\epsilon)k$. Break up the codeword into pieces of c bits each (c some large but fixed integer), and then use a good Shannon code of message size c to encode each of these pieces. The Shannon code maps c bits to a length of $\frac{c}{1-H(p)}$ bits, where p is the probability of error on a binary symmetric channel, for instance. For each of the c-sized blocks, we can decode the received message correctly with probability $1 - exp(-c)$. Sp if the Spielman code is designed to tolerate δ fraction of errors, then we can expect that we will decode correctly overall, as long as $1 - exp(-c) < \delta$. the rate of the code is worse off from the Shannon code by a factor of $1/(1 + \epsilon)$, but the gain is that we can tolerate a fractional error in the whole message. Note that if we had just broken up the original message into chunks of c bits, and encoded each of them, then the cumulative probability of decoding correctly would be $(1 - exp(-c))^{(k/c)}$, where k is the length of the original message. If we choose c constant, this becomes really small as n gets large.

Finally, a few words on how to reduce the parity check of the Spielman code to ϵk from $3k$. We just use an appropriate bipartite graph to get a reducer code $R_{k,\epsilon}$ of parity check length $k\epsilon/4$. Then use $C_{\epsilon k/4}$ on the parity check, where C_r is the code described above. This gives a total parity check length of ϵk.

6 ABNNR

Can we build a large distance code over a large alphabet using a smaller alphabet code?

Say we have a alphabet of size $q = 2^a$, whose letters can be represented by a bits (for concreteness, take the alphabet to be $\mathbb{F}_q$. One could try the following idea: take a $(n, k, \epsilon n)$ code over $\{0, 1\}$ and just break up a codeword into n/a codewords over $\mathbb{F}_q$, choosing the first consecutive a bits, the next a bits and so on. This would give us a $(n/a, k/a, \epsilon n/a)_q$ code. The distance bound is also reduced by a factor of a because two codewords may differ in as many as a places which get combined into one codeword over $\mathbb{F}_q$. So far, this process hasn't bought us any mileage even though the field size went up.

One can visualize breaking up the n bit codeword into blocks of size a as follows: consider a bipartite with n vertices on the left, n/a vertices on the right. There's an edge from vertices $1, \ldots, a$ on the left to vertex 1 on the right, vertices $a + 1, \ldots, 2a$ to vertex 2 on the right, and so on. the problem with taking such a restricted graph is that any change of a bit on vertex 1 on the left only shows up as a change in one vertex on the right! So Alon, Bruck, Naor, Naor, and Roth ('92) had the idea that maybe we should consider more general bipartite graphs with expansion properties.

Consider the following definition of expander graph: It's a bipartite graph with left vertices L, right vertices R, with $|L| = |R| = n$, such that $\forall S \subset L, |S| = \delta n$, we have $|\Gamma(s)| \geq (1 - \epsilon)n$. So this expands a set of small size δn on the left to almost the entire right vertex set. We can expect to find $\epsilon = \delta = 1/d$ and d-regular graphs with these properties (more on this later). Now the ABNNR code consists of the following: take a message of length k over $\{0, 1\}$, encode it to n bits using our given code, then for each vertex on the right write down the value of its d left neighbors to get a d-bit string. The n elements of a 2^d-bit alphabet so obtained give the desired codeword. This will be, in fact, an additive code.

ABNNR didn't give a decoding algorithm for this procedure. Recently, Guruswami and Indyk came up with a decoding algorithm. Using this construction, we can construct codes whose rates is $O(\epsilon)$, alphabet size is a function of $(1/\epsilon)$, which corrects $1 - \epsilon$ errors, and has a polytime decoding algorithm (though the dependence on ϵ of the decoding algorithm is probably nasty).