6.440 Essential Coding Theory Apr 29, 2013

Lecture 21

Lecturer: Madhu Sudan *Scribe: Sam Wong*

1 Overview

In this lecture, we will

- Wrap up Alon-Edmonds-Luby's graph theoretical codes: construction and analysis of distance
- Introduce the interplay between codes and computational complexity: open problems and pseudorandomness

2 Alon-Edmonds-Luby

The construction of the code depends on three main ingredients:

- A binary code $C_0 = [d, R_0 d, \delta_0 d]_2$, where R_0 and δ_0 are our target rate and distance respectively.
- An ϵ-uniform degree d bipartite graph $G = (L \cup R, E)$ with bipartition $L \cup R$ and $|L| = |R| = n$. Recall that such a graph G is ϵ-uniform if for any $X \subseteq L$ and $Y \subseteq R$, we have

  $$\left| \frac{|E(X,Y)|}{dn} - \frac{|X|}{n} \cdot \frac{|Y|}{n} \right| \leq \epsilon,$$

  where $E(X,Y)$ denotes the set of edges between X and Y.

  Intuitively, this says that roughly $|Y|/n$ fraction of the $d|X|$ edges incident to X should go to Y, which is a property that we would expect from random graphs.
- An additive code $C_{big} = \{n, k, D\}_{\mathbb{F}_2^{R_0 d}}$ (think $k \approx n$).

2.1 Construction

Our composed code will be of the form

$$C = \{n, R_0 k, ?\}_{\mathbb{F}_2^d},$$

with the distance to be determined.

For each codeword $c \in C_{big}$, define a codeword $c' \in C$, as follows:

1. Label the n left vertices of G with the n symbols of the codeword c. Thus each left vertex $i \in L$ is associated with some $w_i \in \mathbb{F}_2^{R_0 d}$.

2. Then, label the d incident edges to each left vertex i by encoding w_i into a d-bit codeword using C_0. More concretely, we label each of these d edges using exactly one bit of the encoded w_i according to some fixed ordering of R. We can think of this as the left vertices sending a d-bit message via its incident edges.

3. Each right vertex collects the d bits σ_i sent by its d neighbours which in turn form a d-bit vector $\sigma_1\sigma_2...\sigma_d$ according to some fixed ordering of L. Now the n vectors for each of the right vertices R constitute the codeword c' in $C \subseteq (\mathbb{F}_2^d)^n$.

It is easy to verify that the rate of the composed code C is R_0.

2.2 Distance

We give a bound on the distance of C. Consider a nonzero codeword c of C_{big} which has distance D. This implies that there must be at least D left vertices with nonzero w_i's. Now each of these D nonzero w_i's will in turn generate a d-bit nonzero vector. Together they label at least $D \cdot \delta_0 d$ edges with '1'. Let $Y \subseteq R$ be the set of the right endpoints of these edges. Now $|Y|/n$ will be a bound on the distance since the d-bit vectors on the vertices of Y are nonzero.

We bound $|Y|/n$ using the fact that G is ϵ-uniform, which says that

$$\#\text{nonzero edges} \leq |E(X,Y)| \leq |X||Y|\frac{d}{n} + \epsilon dn.$$

On the other hand, we have

$$\#\text{nonzero edges} \geq D \cdot \delta_0 d.$$

Combining the two inequalities gives

$$\text{distance of } C \geq \frac{|Y|}{n} \geq \delta_0 - \frac{\epsilon n}{D}.$$

Finally, we remark that an ϵ-uniform degree d bipartite graph can be easily constructed by random graphs. There are also nontrivial explicit constructions which are some form of extractors. One of the nice properties of our code is that there are very efficient decoding algorithms, given by Guruswami and Indyk in 2001.

3 Codes and Computational Complexity

There are two natural complexity questions which arise from the study of coding theory, namely:

- How efficient can encoding and decoding be?
- Can we estimate the distance of a given code?

On the other hand, there are also applications from coding theory to computational complexity. Two examples:

- Errors model unknown; codes overcome error/unknown [Cryptography].
- Good codes are extremal combinatorial structures [Pseudorandomness].

For the rest of the section, we survey some of the prominent open problems in this area before giving a very brief introduction to pseudorandomness.

3.1 Complexity of constructing/decoding/evaluating codes

Evaluating codes Given the generator matrix $G \in \mathbb{F}_q^{k \times n}$ of a linear code, the problem of computing its distance has been shown to be NP-complete by Vardy in 1996. In fact, it is still hard to approximate the distance within a factor of $2^{\log^{1-\epsilon} n}$.

Intuitively, if we know that this problem is 2-inapproximable, we can extend this to 4-inapproximablility as follows. Suppose we are to differentiate whether a code C is $[n, k, d]$ or $[n, k, d/2]$. By considering its tensor $C \oplus C$, our problem is then reduced to differentiating whether $C \oplus C$ is $[n^2, k^2, d^2]$ or $[n^2, k^2, d^2/4]$. This shows that it should be hard to approximate within a factor of 4.

A slight more modest goal would be to decide, for a (small) constant ϵ, if

$$\delta(G) \geq \frac{1}{2} - \epsilon \text{ or } \delta(G) \leq \frac{1}{2} - \sqrt{\epsilon}.$$

Nevertheless, even this problem appears to be difficult and the best result we currently have is for something like $\frac{1}{2} - \epsilon$ vs. $\epsilon^{1/10}$.

Recovering Reed-Solomon Let G be the generator matrix of a Reed-Solomon code. Thus G has entries of the form α^{ij}. Suppose we try to mask G by multiplying it by some invertible matrix R and permuting its columns, i.e. we compute

$$RA\pi,$$

where π is a permutation matrix. A natural question is, given $RA\pi$, can we recover the underlying Reed-Solomon matrix A? The answer is conjectured to be no and the intuition is that the loss of information in this transformation that makes it hard to recover A.

Decoding Given the generator matrix $G \in \mathbb{F}_q^{k \times n}$ of a linear code and some $y \in \mathbb{F}_2^n$, we wish to find $x \in \mathbb{F}_2^k$ which minimises $\delta(xG, y)$. Again, this problem is NP-hard and still hard to approximate within a factor of $2^{\log^{1-\epsilon} n}$.

However, this problem may seem unreasonable as a code of distance d is only designed to correct $d/2$ (adversarial) errors. This observation suggests a less restrictive problem. We are still given G and y but now we are promised that $\triangle(G) \geq d$ and $\triangle(y, G) < d/2$. Can we find x for which $\triangle(xG, y) < d/2$?

Preprocessing In some real-life applications, we typically know G well ahead of time before knowing y. Thus one may hope to preprocess G (with unbounded but finite time) in such a way that minimising $\delta(xG, y)$ becomes easy. More informally, the goal is to compute in finite time a polynomial size circuit $C : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^k$ such that for $y \in \mathbb{F}_2^n$,

$$C(y) \in \arg\min_x \delta(xG, y).$$

Unfortunately, even this relaxed problem is hard (Bruck-Naor).

3.2 Pseudorandomness

We first introduce some basics of computational complexity. Functions and relations are two prevailing notions which model computation. A function f computes the output $f(x)$ given an input x. Given a relation R and x, the goal is to find some y such that $(x, y) \in R$.

If there is one and only one answer to our problem, we model this by functions (e.g. is a 3-SAT instance satisfiable?). But when there is more than one possible answer (e.g. find a shortest path), relations are a more appropriate concept.

The class P consists of all boolean functions computable in polynomial time. BPP is the class of problems which can be solved in bounded-error polynomial time, i.e. given a function f, there is some function A which takes on input x and y such that for all x,

$$\Pr_y[A(x, y) = f(x)] \geq \frac{2}{3}.$$

Furthermore, A must be computable in time polynomial in $|x|$.

As an example, we consider the problem of finding a good assignment for a 10-SAT formula, which consists of m clauses of 10 literals. Observe that a random assignment satisfies each clause with probability $1 - 2^{-10}$ and hence $1 - 2^{-10}$ clauses in expectation.

A big open problem is whether P is equal to BPP. The notion of pseudorandomness was proposed partly towards resolving this problem. Informally, pseudorandomness is about producing a pseudorandom string y sufficient for our randomised algorithms from a seed of true randomness z.

Pseudorandom generator More formally, P ϵ-fools (A, f) if for all x,

$$\left| \Pr_{y \to \{0,1\}^n}[A(x,y) = f(x)] - \Pr_{z \to \{0,1\}^l}[A(x, P(z)) = f(x)] \right| \leq \epsilon.$$

In future lectures, we will explore various ways to boost randomness:

- Limited independence
- Small biased spaces
- Almost limited independence