CS 229r Essential Coding Theory, Lecture 23 April 25, 2017

Lecture 23: Codes in complexity, wrapup

Instructor: Madhu Sudan *Scribes: Christina Ilvento*

Administrivia

Projects:

- Writeups due 5/1
- Presentations on 5/2 (MD 323)
- Slides highly recommended
- 15 minutes/person
- Please send slides to Madhu in advance so we can present from one laptop

Today: Codes in Complexity

Two main things:

1. PRG $\leftarrow$ OWP: we'll work towards showing that we can build pseudorandom generators out of one-way functions, we'll do a weaker version in class using list decoding in an interesting way
2. Hardness amplification - we'll use list decoding for this as well

And finally, we'll wrap up with a course review.

Pseudorandom Generation

Pseudorandom generators (PRGs):

$$G : \{0,1\}^n \to \{0,1\}^m$$

Where G satisfies the following conditions:

1. $m > n$
2. G should be efficiently computable (polynomial time)
3. Output of G should "look random" to polynomial time algorithms

"Looking Random": We say that the output of G "looks random" if there does not exist a distinguisher D with non-negligible distinguishing probability for G.

A distinguisher $D : \{0,1\}^m \to \{0,1\}$ has distinguishing probability ϵ for G if

$$|\Pr_{y \sim U_m}[D(y) = 1] - \Pr_{x \sim U_n}[D(G(x)) = 1]| < \epsilon$$

So formally, the output of G is ϵ-pseudorandom to some class of algorithms $\mathcal{C}$ if for all $D \in \mathcal{C}$, D has distinguishing probability $\leq \epsilon$.[1]

How much does G need to extend the input seed? Ideally, we want to start with a short seed (small n) and get a very large output (large m). Why don't we put this in the formal definition? Assume that we have G which extends by one bit and is ϵ-pseudorandom. We can apply the generator again and again to continue extending the outputs. If we repeat the process t times, G applied t times will be $t\epsilon$-pseudorandom and produce $n + t$ bits. So if we can build a PRG which extends by just one bit, we are in good shape!

Folklore Proposition: if $P = NP$, PRG's do not exist. So we don't expect to be able to prove that PRG's exist unconditionally (if we did, we win $1 million :).

Oneway functions and permutations

Theorem: (Blum-Micali, Yao, Goldreich-Levin) If One-way Permutations exist, then pseudorandom generators exist.

A function f is a one-way permutation[2] if

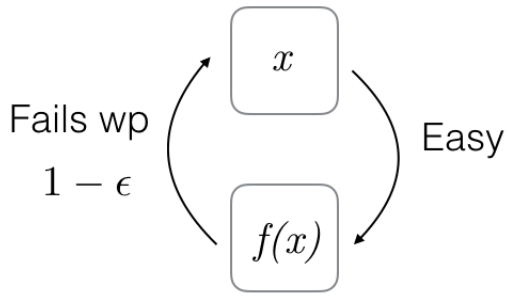
1. f is one-to-one
2. f is easy to compute (eg, polynomial time algorithm which computes $f(x)$ given x)
3. f is hard to invert: $\forall$ polynomial time algorithms A,

$$\Pr_{x \sim U_n}[A(f(x)) = x] \leq \epsilon$$

Another way to think of this is finding a preimage of $f(x)$

$$\Pr_{x \sim U_n}[f(A(f(x))) = f(x)] \leq \epsilon$$

Essentially, we want it to be the case that we very, very rarely get an inversion. However, one-way permutations do not imply that we get any stretching of our input seed, even by a single bit. Suppose we have $f(x) : n \to n$; we can take $f'(f(x), y)$ which just concatenates a random string y to the output of $f(x)$. It's clear to see that f' is one-way if f is. However, it's not clear that f' is "random". For example, given $RSA(x)$, the time it takes to compute the i^{th} bit is exponentially increasing, but the least-significant bits really don't look that random. So what is guaranteed is that we can't invert the whole thing, not that every single bit is difficult to predict.



So our goal for the remainder of this section is to build a one-way function with all bits hard to predict with stretching in a black-box way.

The Key Ingredient: Assume $1/2 - \epsilon/4$ fraction error list-decodable code. Want a function $C : \{0,1\}^n \to \{0,1\}^N$ such that C is efficiently computable and C is efficiently list-decodable from $1/2 - \epsilon/4$ fraction errors.

[1] If we did not insist that D came from the class $\mathcal{C}$, and instead allowed arbitrary functions, then the only things that are random are things which are statistically close to random, and we won't be able to get PRGs which can extend by even one bit with $\epsilon < 1/2$.

[2] A one-way *function* satisfies the same requirements, except the first

That is, for $y \in \{0,1\}^N$ such that there exists x such that $\Delta(C(x), y) \leq 1/2 - \frac{\epsilon}{4}N)$ then $D(y)$ outputs a list that includes x.

We have already seen such codes, and we can get $N \approx n/poly(\epsilon)$.

Constructing the PRG

Now we're going to use one-way permutations and list-decodable codes to build a PRG.[3]

Theorem:

Given $f : \{0,1\}^n \to \{0,1\}^n$ which is a one-way permutation, construct $G : \{0,1\}^n \times [N] \to \{0,1\}^n \times [N] \times \{0,1\}$ such that $G(x,i) = (f(x), i, C(x)_i)$. G is ϵ-pseudorandom for any polynomial time function A.

Proof: The canonical way of proving is to assume that G is not a PRG, and try to get a contradiction to f being one-way.

Assume $\exists D$ that has distinguishing probability $> \epsilon$. WLOG, assume

$$\Pr[D(f(x), i, C(x)_i) = 1] > \Pr[D(random) = 1] + \epsilon$$

Which we can rewrite as

$$\Pr_{x,i,b}[D(f(x), i, C(x)_i) = 1] > \Pr_{x,i,b}[D(f(x), i, b) = 1] + \epsilon$$

First we'll show how, given a distinguisher, we can produce a poly time predictor $P(f(x), i)$ tries to predict $C(x)_i$ and gets it right with probability $1/2 + \epsilon/2$, that is

$$\Pr_{x,i}[P(f(x), i) = C(x)_i] \geq \frac{1}{2} + \epsilon/2$$

$P(f(x), i)$ wants to know $C(x)_i$.

$$P(f(x), i) = \begin{cases} 1 \text{ if } D(f(x), i, 1) = 1 \text{ and } D(f(x), i, 0) = 0 \\ 0 \text{ if } D(f(x), i, 1) = 0 \text{ and } D(f(x), i, 0) = 1 \\ \text{else random bit} \end{cases}$$

Claim: P predicts $C(x)_i$ with probability $\frac{1}{2} + \epsilon/2$

Proof: Exercise

Now that we have such a predictor P, then given $f(x)$, for $i = 1, ..., n$: let $y_i = P(f(x), i)$. The bits of y_i will look like $C(x)$.

When we compare the two matrices with rows y_i and $C(x)$, we know that there will be $\epsilon/4$ such rows which are very close together, eg $\Delta(y, C(x))$ is small. So we can split the probability of inversion into two probabilities

$$\Pr_x[\Pr_i[P(f(x), i) = y_i] \geq \frac{1}{2} + \epsilon/4] \geq \epsilon/4$$

by simple Markov.

So when we list-decode y, we get $\{x^{(1)}, ...x^{(L)}\}$, so if $f(x^{(i)}) = f(x)$, output $x^{(i)}$

So we can invert f with probability $\epsilon/4$, which contradicts our assumption that f is a one-way permutation.

[3] This was one of the results that made CT interesting to computer scientists (Goldreich-Levin)

Hardness Amplification

At the intersection of Complexity, Cryptography and Coding Theory.

Hardness Amplification Looks at the following question:

If $\exists$ function that is easy to compute and (worst-case) hard to invert, does there exist a function that is easy to compute and (average case) hard to invert?

Another way to look at it:

If f^{-1} is computable in NP, but not computable in P in worst-case, does there exist some g such that g^{-1} is computable in NP but g^{-1} but not computable in P on average.

This is an open question. We don't know theorems of this form, and we think that it's going to take a lot more work. Coding theory can change the definitions of the big class (NP) and the small class (P)

Levin and Sudan/Trevisan/Vadhan looks at this. If f is computable in EXPTIME then we get g which is computable in EXPTIME. If f is not computable in polynomial time ciruits (P/poly), in worst case, then g is not computable $(\frac{1}{2} + \epsilon)$ in $P/poly$. Eg you get it wrong with probability $\frac{1}{2} + \epsilon$.

These results depend on efficient local decoding algorithms. (To get close to 1/2, need local list decoding)

In these results, we're applying CT in a modular way, we don't look at the structure of the codes or the functions at all.

Course Review and Future Directions

Main theme: we want to understand error correcting codes; structures which supposedly help us correct from errors that occur during information transmission.

We wanted to understand what kinds of good error correcting codes exist, and to that end, we studied:

1. Limits of codes: best codes we can get and reasons why we run into limits. If you have a q-ary channel, then the fraction of errors we can deal with is $\leq 1 - 1/q$. Prooofs of impossibility - why do codes of a certain type not exist. (We'll see one more such proof in the projects) So we have our targets set
2. Constructions of codes:
   (a) Algebra - algebraic codes give remarkable packing; work great over large alphabets, but they don't work well over small alphabets. Ex: AG codes, Reed-solomon, Hadamard, BCH, ...
   (b) Graph-theoretic - Good performace, but most impressive is that they come with very efficient, often linear time algorithms
   (c) Information-theoretic mechanisms (Polar codes): good for random errors only, but achieve great performance

(d) Composition - Tensor product, concatenation, alon-luby transformation; all of these operations move you between codes with extra properties, particularly useful when you want to build special features into code, not so much when you want to squeeze the best possible peformance (for example, good for locality)

3. Features of codes: in contrast to the Electrical Engineering versions of this course, we cared about features: (a) Asymptotics (b) Algorithmics

   We didn't even mention Golay codes, because they are finite codes, so asymptotically they aren't interesting. (Sources/references if people are interested in this type of stuff?)

4. Modern focuses in coding theory

   - In the last 20 years, we see in the top CS conferences 2-3 papers on coding theory per year; we saw this in polar codes and interactive coding, and local decoding/list decoding and applications in complexity, and we'll see more of this in the projects
   - Probabilistically Checkable Proofs (PCPs): owe their existence to ECC; want to verify claims/analyses based on the ECC not on redoing the analysis (we didn't talk about these)
   - Reed-Muller Codes achieve capacity on BEC(p); major recent result

     $$f \in \mathbb{F}_2[x...x_m]$$

     with $deg(f) \leq d$. Major open question, can we prove the same thing for other channels?