CS 229r Essential Coding Theory Apr 22, 2020

Lecture 23

Instructor: Madhu Sudan *Scribe: Pratap Singh*

1 Today

1.1 Admin

- Optional, updated problem set 6 now out on Piazza.

1.2 Coding Theory in Complexity/Cryptography

Two major topic areas for today:

- Hardcore predicates for one-way permutations, using list decoding.
- Worst-case to average-case reduction (example: matrix permanent), using a locally decodable code.

2 Hardcore predicates for one-way permutations

2.1 One-way permutations

Definition 1 (One-way permutation)**.** A function $f : \{0,1\}^k \to \{0,1\}^k$ is a *one-way permutation* iff the following hold:

1. f is a permutation: $\exists f^{-1} : \{0,1\}^k \to \{0,1\}^k$ s.t. $\forall x \in \{0,1\}^k, f^{-1}(f(x)) = x$.
2. f is computable in polynomial time in the worst case.
3. f^{-1} is "very hard" to compute on average.

We define "very hard" as follows:

Definition 2 ($\varepsilon(k)$-approximable)**.** A function $g : \{0,1\}^k \to \{0,1\}^k$ is $\varepsilon(k)$-approximable if there exists some polynomial-time computable algorithm A s.t. $\Pr_x [A(x) = g(x)] \geq \varepsilon(k)$.

Definition 3 ("Very hard" function)**.** A function $f : \{0,1\}^k \to \{0,1\}^k$ is said to be "very hard" to compute if for every polynomial $p(k)$, f is not $(1/p(k))$-approximable.

2.2 Hardcore predicates

One-way permutations are relevant in cryptography–we could encrypt a message efficiently using an o.w.p., but an eavesdropper would not be able to recover the message from its encryption. But some o.w.p.s may "leak" information–some part of the message may be present unchanged in the output. Consider the following exercise:

Exercise 4. *Let $f : \{0,1\}^k \to \{0,1\}^k$ be an o.w.p., and let $F : \{0,1\}^{2k} \to \{0,1\}^{2k}$ be defined by $F(x,y) = (f(x), y)$. Prove that F is an o.w.p.*

Sketch of Proof Clearly, if f is a permutation then F is also a permutation, and if f is computable in worst-case polynomial time then F is also computable in worst-case polynomial time. In order to ε-approximate F^{-1} we would need to ε-approximate f^{-1}, which by definition is not possible for any polynomial accuracy ε; therefore F^{-1} is not ε-approximable for any polynomial epsilon. □

The F defined in Exercise 4 is an o.w.p., but the last k bits of the message are visible in the last k bits of the output. This is not much good for encryption. In general, we want a way of discussing what information abut x can be gleaned just by looking at $f(x)$. This gives us the following definition.

Definition 5 (Hardcore predicate)**.** For some function $f : \{0,1\}^k \to \{0,1\}^k$, a *hardcore predicate* is some function b that satisfies the following properties:

- b is a predicate: $b : \{0,1\}^k \to \{0,1\}$.
- b is "hardcore":
  1. b is computable in polynomial time given x.
  2. b is hard to compute given $f(x)$.
  2′. It is "very hard" to compute $b(x)$ given $f(x)$.
  Specifically, for every polynomial p, $b \circ f^{-1}$ is not $\left(\frac{1}{2} + \frac{1}{p(k)}\right)$-approximable.

Note that property 2′ implies property 2 in Definition 5. Since b is a predicate, one of the constant 0 or constant 1 functions would be a 1/2-approximator; hence we require that we cannot do any better than a 1/2-approximation.

Exercise 6. *Suppose b is a hardcore predicate for some permutation f. If f is easy (poly-time computable), show that f is an o.w.p. (This should be immediate from definitions.)*

Sketch of Proof Immediate from definitions. If b is hardcore for f, then since b is computable in poly-time (property 1 from Definition 5) but $b \circ f^{-1}$ is not poly-time approximable (property 2 from Definition 5), it cannot be the case that f^{-1} is poly-time approximable. If f^{-1} were poly-time approximable, we could approximate $b \circ f^{-1}$ in the obvious way. Therefore, since f is easy but f^{-1} is hard, f is an o.w.p. □

2.3 Motivation: PRGs from hardcore predicates and o.w.p.s

We shall see that it is possible to create pseudorandom generators from hardcore predicates and o.w.p.s. For this class, we will use a much stronger cryptographic definition of PRG than last class: a PRG that can "fool every polynomial-time function to ε".

Definition 7 (Cryptographic PRG)**.** *A function $G : \{0,1\}^n \to \{0,1\}^m$ is a* pseudorandom generator *iff $m > n$ and $\forall$ polynomial-time functions A, $\forall$ polynomials P, $\forall x$,*

$$A(G(x)) \approx_{\frac{1}{P(n)}} A(\text{Unif}(\{0,1\}^m))$$

where $\text{Unif}(\{0,1\}^m)$ *is the uniform distribution over* $\{0,1\}^m$.
If $\frac{1}{P(n)} = \varepsilon(n)$, we say that G is an ε-PRG.

We can generate a PRG from an o.w.p. and a hardcore predicate, as follows.

Theorem 8 (PRG from o.w.p. and hardcore predicate)**.** *Suppose $f : \{0,1\}^k \to \{0,1\}^k$ is an o.w.p. and b is hardcore for f. Then*

$$G(x) = (b(x), b(f(x)), b(f^2(x)), \ldots, b(f^{m-1}(x))$$

is a PRG (for $m > k$).

Proof. We omit many of the details, but the general structure is as follows. Let $G_i(x) = (b(x), G_{i-1}(f(x)))$, $G_0(x) = x$. First, show that if b is hardcore for f, G_1 is an ε-PRG. Second, show that if $G_1(x)$ is an ε-PRG, then G_m is an $m \cdot \varepsilon$-PRG. □

Exercise 9. *Fill in the details of the proof of theorem 8.*

Sketch of Proof We first show that $G_1(x) = (b(x), f(x))$ is a PRG. For contraposition, assume G_1 is not a PRG and there exists some algorithm B that distinguishes it from the uniform distribution. In this case we could break property $2'$ of Definition 5 as follows. Suppose we want to compute $b \circ f^{-1}$ on some input y. We compute $A(1, y)$, which will tell us if $(1, y)$ is a uniform random string or the output of G_1. If it is the output of G_1, we know that $b \circ f^{-1}(y) = 1$; otherwise, $b \circ f^{-1}(y)$ must be 0 and we can output 0. Since A is poly-time computable and gives us advantage ε, we can thus poly-time approximate $b \circ f^{-1}$ with advantage ε.

We can use a similar argument to show that if G_1 is a PRG, the m-bit construction of G is not a PRG. Suppose we can distinguish G from the uniform distribution using algorithm A. Now, consider $h(x) = (U_1, b(f(x)), b(f^2(x)), \ldots, b(f^{m-1}(x))$, where U_1 is a single uniform bit. If we can distinguish $h(x)$ from $G(x)$ then we are clearly able to break the PRG property of G_1, since we would then be able to compute $b(x)$ given $f(x)$ (as described above). We can use the algorithm A to distinguish $h(x)$ from $G(x)$ by applying A on m samples from h. Note also that this argument can extend to t uniform bits at the start of the message, thus giving us the required inductive argument. Each step reduces the advantage of the PRG by ε, so that if G_1 is an ε-PRG, then G is an $(m \cdot \varepsilon)$-PRG. □

2.4 A hardcore predicate for (many) o.w.p.s

We would like to be able to generate hardcore predicates for every o.w.p. At present, specific examples are known, such as $f =$ RSA and $b =$ most significant bit. But these constructions are not known to generalize nicely; for example, $f =$ RSA and $b =$ least significant bit. The ideal result would be if there was one b that is hardcore for every o.w.p., but this is unlikely to be the case. Today we will get close by using a list-decodable code.

Theorem 10 (Hardcore predicate for many o.w.p.s)**.** *For all k, there exists some $m > k$ and some function $b : \{0,1\}^m \to \{0,1\}$ such that for every o.w.p. $f : \{0,1\}^k \to \{0,1\}^k$, there exists some o.w.p. $F : \{0,1\}^m \to \{0,1\}^m$ that is a padding of f (in the sense of exercise 4) and such that b is hardcore for F.*

Proof. We first give the m and b, then prove that the theorem holds. We use some code $C : \{0,1\}^k \to \{0,1\}^n$ which is efficiently encodable and list-decodable from $(\frac{1}{2} - \varepsilon)$ fraction of errors. Let $m = k + \log n$, $i \in [n]$. Then let $F(x, i) \triangleq (f(x), i)$, and let $b(x, i) = C(x)_i$.

We give a contrapositive proof of the theorem: suppose that b is not hardcore for F, then show that f is not an o.w.p. In particular, assume that there exists some poly-time computable A s.t. $A(f(x), i) = C(x)_i$ with probability at least $\frac{1}{2} + \alpha$ over x and i.

By Markov's inequality, we have that

$$\Pr_x \left[\Pr_i \left[A(f(x), i) = C(x)_i \right] \geq \frac{1}{2} + \frac{\alpha}{2} \right] \geq \frac{\alpha}{2}.$$

(We apply Markov's inequality by thinking of $\Pr_i \left[A(f(x), i) = C(x)_i \right]$ as a random variable over $[0, 1]$, then bounding the tail.) Let a particular x be "good" if it satisfies the inner condition that $\Pr_i \left[A(f(x), i) = C(x)_i \right] \geq \frac{1}{2} + \frac{\alpha}{2}$. We claim that the following algorithm will approximate the inverse of f.

- Input: $y = f(x)$ for some x.
- for i in $[n]$ let $w_i = A(y, i)$.
- List-decode w as $\{x_1, x_2, \ldots, x_L\}$.

- If $f(x_i) = y$, output x_i.

Now, if x is "good" then $\Delta(w, C(x)) \leq \frac{1}{2} - \frac{\alpha}{2}$. In this case, list-decoding is successful and the inverter algorithm will output x. Since x is good with probability at least $\alpha/2$, if $\alpha/2 \geq \varepsilon$, then we can invert f with probability at least ε. Thus, if b is 2ε-approximable, then f is ε-approximable. This gives us the contraposition we need. We also note that since we can list-decode C in polynomial time (by assumption), our construction yields a polynomial-time algorithm. □

3 Worst-case to average-case reduction

In general it is (relatively) easy to reason about worst-case complexity of functions, and we have lots of results about worst-case complexity. But often we are more interested in the complexity of an "average" case, or the kind of case we are likely to encounter in applications. For example, SAT is a well-known NP-complete problem, for which the best known deterministic algorithms are exponential-time, but in practice there are many widely-used SAT solvers which can efficiently solve SAT for many instances.

In cryptography, average-case complexity is particularly important. If complexity measures the security of a particular encryption scheme, then it is not enough that there exists some pathological key which causes high complexity. We want a randomly chosen key to give high complexity, such that all (or most) keys will be secure.

We haven't been able to do worst-case to average-case reduction for NP as a whole; but note that the worst-case to average-case reduction is trivial for problems in P. We give an example of a non-trivial reduction below.

3.1 The matrix permanent

Definition 11 (Matrix permanent)**.** $\forall M \in \mathbb{F}_q^{n \times n}$, the *permanent* of M, $\mathrm{Perm}(M)$, is defined as:

$$\mathrm{Perm}(M) = \sum_{\pi \in S_n} \prod_{i=1}^{n} M_{i\pi(i)}$$

where S_n is the set of all permutations of $[n]$.

Intuitively, the permanent is the determinant without sign. In the determinant calculation we multiply each permutation product by the sign of the permutation. Unlike the determinant, however, permanent is very hard to compute (over any field except $\mathbb{F}_2$, where $\mathrm{Perm}(M) = \det(M)$. In fact, Perm is $\#P$-complete, meaning that (in particular) #SAT $\leq_p$ Perm.

Notice that Perm is a degree-n polynomial over n^2 variables, the M_{ij}. This is the key to the worst-case to average-case reduction.

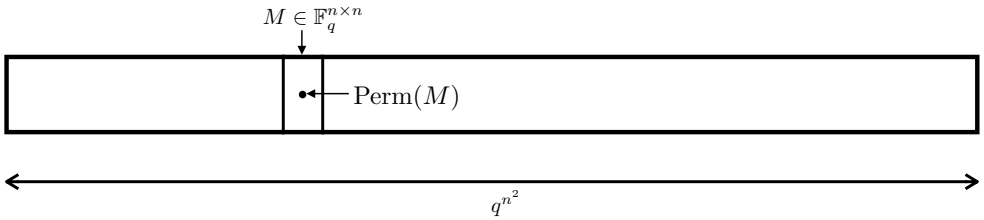
Theorem 12 (Worst-case to average-case for permanent)**.** *If $q > n^2$ and there exists some polynomial time algorithm A s.t.*

$$\Pr_{R \sim \mathbb{F}_q^{n \times n}} [A(R) = \mathrm{Perm}(R)] \geq \frac{7}{8},$$

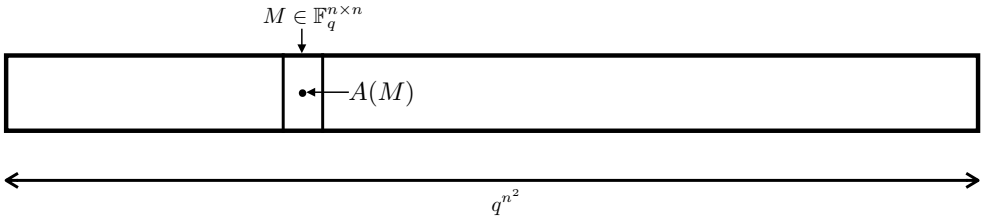
then there exists some polynomial time algorithm B s.t. $\forall M \in \mathbb{F}_q^{n \times n}, B(M) = \mathrm{Perm}(M)$ w.p. $1 - o(1)$.

(Note the contraposition: we show that if the random case is "easy", then the worst case is "easy". Equivalently, if the worst case is "hard", then the random case is "hard".)

Proof. The idea of the proof is to consider calculating the matrix permanent as a local decoding problem. Suppose we list the permanents of every matrix in $\mathbb{F}_q^{n \times n}$ as a single vector v_P of length q^{n^2}, as follows:



If we could access arbitrary elements of v_P, we would be done. Of course, we can't access arbitrary elements of this vector. The best we can do is construct arbitrary elements of the corresponding vector v_A built using A instead of Perm:



Now, since Perm is jut a degree-n polynomial in n^2 variables over $\mathbb{F}_q$, v_P is a codeword in the Reed-Muller code $\mathrm{RM}(q, n, n^2)$. So in order to construct v_P, we could just decode v_A using the Reed-Muller list-decoding algorithm we have seen before. However, this would not be efficient since the codewords have size exponential in n. Instead, we can use the local decoding algorithm for RM codes.

We can specify the algorithm B:

- Input: $M \in \mathbb{F}_q^{n \times n}$
- Choose R uniformly at random in $\mathbb{F}_q^{n \times n}$.
- Choose $\{\alpha_1, \alpha_2, \ldots, \alpha_{10n}\}$ distinct from $\mathbb{F}_q \setminus \{0\}$.
- For each $i \in [10n]$, let $\beta_i = A(M + \alpha_i R)$.
- Reed-Solomon decode $\{(\alpha_i, \beta_i)\}_i$ to polynomial $P(\alpha)$.
- Output $P(0)$.

This algorithm effectively follows the pattern of the RM local decoding algorithm we have seen before. We now prove it correct.

Let $g(t) \triangleq \mathrm{Perm}(M + tR)$; then we want to output $g(0)$. Note that by definition of A, $\Pr_M[\beta_i = g(\alpha_i)] \geq 7/8$ for every i. This then implies that the probability of getting more than a $1/4$ fraction of incorrect β_i is at most $1/4$. But this inner condition is just the condition for the RS decoding algorithm to work. So with probability at least $3/4$, we will output $\mathrm{Perm}(M)$. □

The lower bound of $7/8$ on the probability that A succeeds can be improved to $1/2 + \varepsilon$ by using a more careful decoding and reduction. Further, we can improve to $1/poly(n)$ by using a concept called "local list decoding" which is not discussed in this course.

The above is an of a general class of theorem due to Sudan, Trevisan, and Vadhan [1]. An example of this is (roughly) that if there exists some $f \in TIME(2^{O(n)})$ but $f \notin P_{/poly}$, then there exists some $F \in TIME(2^{O(n)})$ that is "very hard" relative to $P_{/poly}$. The proof of this result relies again on locally list-decodable codes.

References

[1] Madhu Sudan, Luca Trevisan, and Salil Vadhan. "Pseudorandom Generators without the XOR Lemma." In: *J. Computer and System Sciences* 62.2 (March 2001). ISSN: 0022-0000. DOI: 10.1006/jcss.2000.1730. URL: https://doi.org/10.1006/jcss.2000.1730.