6.893 Approximability of Optimization Problems	October 20, 1999
Lecture 11	
Lecturer: Madhu Sudan	*Scribe: Anna Lysyanskaya*

This lecture is on self-correction of polynomial codes.

1 Inroduction: Relevance to PCP

Suppose we want to prove is that a formula ϕ is satisfiable. The usual proof is the vector $\bar{a} = a_1, ..., a_n$ that is a satisfying assignment. The probabilistically checkable proof is an encoding $T(\bar{a})$ of $\bar{a}$ in some probabilistically checkable error-correcting code. The verifier will not need to read the entire proof to verify that ϕ is satisfiable, which is achieved by adding redundancy into the proof. Thus we are interested in the error-correcting transformation T that takes $\bar{a}$ and outputs its redundant version.

In the PCP setting, the Verifier, given proof π, should perform two checks:

1. *syntactic check:* There exists a string $\bar{a}$ such that $\pi = T(\bar{a})$.
2. *semantic check:* The string $\bar{a}$ is a satisfying assignment for the formula ϕ.

In this lecture, we will postpone the discussion of the semantic check and focus on the syntactic one: is the proof π a valid error correcting transformation of another string, $\bar{a}$?

Let us subdivide the syntactic check into two subtasks that we need to achieve simultaneously:

(a) Verify that there exists a string $\bar{a}$ such that the string π is very close to $T(\bar{a})$.

(b) Recover individual bits of the string $\bar{a}$.

The idea is that not only should we be able to verify that the proof is indeed the encoding of some assignment to the variables of ϕ, but we would also like to be able to efficiently find out what some of the values of these variables are. This will be needed for the semantic check.

2 Self-correction in Hadamard code

Recall the Hadamard code $T : GF(2^m) \rightarrow \{0,1\}^{2^m}$, $T(\bar{a}) = (A(x) = \sum_{i=1}^m a_i x_i)_{x \in \{0,1\}^m}$, in which the codeword $\bar{a}$ is viewed as a linear polynomial in m variables and evaluated for all possible 2^m assignments to these variables.

δ-Self-correction

Suppose we are given oracle access to $A : \{0,1\}^m \rightarrow \{0,1\}$ and a promise that there exists some $\tilde{A}$ and $\bar{a}$ such that $\tilde{A}(x) = \sum a_i x_i$ and for $(1-\delta)$ fraction of the inputs, $\tilde{A}(x) = A(x)$. The goal is to find $\tilde{A}(x_0)$ for some x_0. (For example, suppose $x_0 = 0000100...0$. Then $\tilde{A}(x_0) = \sum a_i x_{0,i} = a_5$.)

It turns out that there is a simple probabilistic algorithm for computing this with just two queries to A, and with the property that the probability of success is taken not over the choice of such an oracle A, but only over the cointosses of the algorithm itself. These ideas were first explored in the work of Blum, Luby, Rubinfeld "Self-testing and Correcting of Programs."

The crucial observation is that $\forall\ x, y \in \{0,1\}^n$, $\tilde{A}(x) + \tilde{A}(y) = \tilde{A}(x+y)$, where the first "+" denotes bit addition, or XOR, and the second "+" denotes vector addition (also XOR).

Owing to this observation, the self-correction algorithm for computing $\tilde{A}(x_0)$ is as follows:

1. Pick $r \in \{0,1\}^m$ at random.
2. Return $(A(x_0 + r) - A(r))$.

Lemma 1 *For all choices of x_0, $Pr_r[A(x_0+r) - A(r) = \tilde{A}(x_0)] \geq 1 - 2\delta$.*

Proof By the observation that we have made before, if $A(x_0+r) = \tilde{A}(x_0+r)$ and $A(r) = \tilde{A}(r)$, then $A(x_0+r) - A(r) = \tilde{A}(x_0)$, and so $A(x_0+r) - A(r) = \tilde{A}(x_0+r) - \tilde{A}(r) = \tilde{A}(x_0)$, and so we get what we want. So the only two bad events are that $A(x_0+r) \neq \tilde{A}(x_0+r)$ and $A(r) \neq \tilde{A}(r)$. Since A is δ-close to $\tilde{A}$, the probability of each of them is $\leq \delta$. Therefore by the union bound, the probability that one of these bad events happens is $\leq 2\delta$. Therefore the probability that neither of the bad events happens is $1 - 2\delta$. ■

Owing to the lemma, for $\delta < 1/4$, we have a probabilistic algorithm that runs in time $O(m)$ and after making 2 queries to the oracle A has the right answer with probability $> 1/2$.

It is left as an exercise to find an analogous algorithm that does the same for fields other than $GF(2^m)$ (for example, $GF(3^m)$) and to give an example of an oracle A which makes the algorithm above fail when $\delta(A, \tilde{A}) = 1/4$ for all possible $\tilde{A}$.

3 Self-correction for multivariate polynomials

In the above error-correcting code, the message was viewed as a linear polynomial in m variables over $GF(2)$, and the encoding corresponded to evaluating this polynomial on all inputs.

Now consider what happens if we view a more general setting. We have a field F, a subset $H \subseteq F$, and a message is defined as a function A that maps vectors from H^m to $\{0,1\}$. That is to say that a message is a string of $|H|^m$ bits. The encoding of A, denoted as $\hat{A}$, is a function that maps vectors from F^m to F, and satisfies the following properties:

1. $\hat{A}(x) = A(x)$ if $x \in H^m$.
2. $\hat{A}(x)$ has degree $|H| - 1$ in each variable. (I.e. total degree is $d \leq m(|H| - 1)$. This condition allows the message A to be an arbitrary function, not necessarily a polynomial.)

How do we self-correct in this more general setting?

(δ, d) multivariate self-correction

Given:

1. Oracle access to $f : F^m \to F$
2. Some $x_0 \in F^m$
3. Promise: there exists some polynomial $\hat{A} : F^m \to F$ such that $\hat{A}$ has degree d and is δ-close to f.

Task: Compute $\hat{A}(x_0)$.

The algorithm we give for this task below has great historical significance. It was proposed by Beaver and Feigenbaum, clarified further by Lipton, paved the way for results such as $IP = PSPACE$, etc.

Let us first present the intuition for this algorithm. We view F^m as an m-dimensional grid (multigrid). There is a δ-sized subset of F^m on which $f(x) \neq \hat{A}(x)$. We are interested in computing $\hat{A}(x_0)$. The idea is to pick a random line in the multigrid, such that x_0 is on that line. This is accomplished by picking a random point y and drawing a line from y to x_0. Since the line is picked at random, we expect that only a δ fraction of the points of the line will have $f(x)$ different from $\hat{A}(x)$. But if we have enough points on the line, and we know that all of them are on some polynomial of degree d, we can determine the right polynomial even if a δ fraction of the points are wrong.

And now more formally. The algorithm proceeds as follows:

1. Pick $y \in F^m$ uniformly at random.

2. Set $r = y - x_0$.

3. Let $L_{x_0,r} = \{x_0 + tr \mid t \in F\}$. What is interesting about the set of points $L_{x_0,r}$?

   - $\hat{A}|_{L_{x_0,r}}$ is now a function of one variable t:

     $$\bar{z} \in L_{x_0,r} \;\Rightarrow\; \exists\, \bar{r} \text{ s.t. } \hat{A}(z_1, z_2, ..., z_m) = \hat{A}((x_0)_1 + tr_1, (x_0)_2 + tr_2, ...) = c_0 + c_1 t_+ ... + c_d t^d$$

     That is to say, we went from a polynomial in m variables of degree d to a polynomial in 1 variable t of degree d.

   - Each point for which $t \neq 0$ is random (independent of x_0) when r is chosen at random. (This is a property only of fields.)

4. Pick $t_1, t_2, ..., t_{d+1} \in F$, distinct, non-zero, uniformly at random. Set $z_i = f(x_0 + t_i r)$.

5. Find a one-variable polynomial p of degree d such that for $1 \leq i \leq d+1$, $p(t_i) = z_i$.

6. Output $p(0)$.

Lemma 2 *Let $\delta < \frac{1}{2(d+1)}$. Then the algorithm above succeeds in producing $\hat{A}(x_0)$ with probability $\geq 1 - (d+1)\delta$.*

Proof Observe that there are $d+1$ possible bad events. Namely, a bad event is that for some $1 \leq i \leq d+1$, $f(x_0 + t_i r) \neq \hat{A}(x_0 + t_i r)$. The probability of such a bad event is δ, since each $x_0 + t_i r$ is a completely random point. Therefore, by union bound, with probability at $\geq 1 - (d+1)\delta$, no bad event happens. ■

For the algorithm above, we need $m \log(|F|)$ bits of randomness, where m is the number of dimensions, and $|F|$ is the size of the field. Exercise: show how to get by with just $O(d)$ bits of randomness, where d is the total degree of the $\hat{A}$, i.e. $d = m(|H|+1)$.

The point is that we can encode m bits of information using $poly(m)$ bits, in such a way that, with high probability, any bit can be recovered by performing $polylog(m)$ queries.

Note that we can relax the requirement on δ, from $\delta < \frac{1}{2(d+1)}$ to $\delta < 1/2$, provided that $|F| >> d$. For example, relaxing δ to a value $< \frac{1}{4}$ can be done by sampling $2(d+1)$ points instead of just $d+1$: the chances are good that there will not be more than $d/2$ errors, and so there will be a unique degree d polynomial that fits more than $3d/4$ of the sampled points. I.e. if the number of bad events is $< \frac{d+1}{2}$, we can recover $\hat{A}(x_0)$ from the remaining $\frac{3d+3}{2}$ good points. Using Markov's inequality, we bound the probability that there are $\geq \frac{d+1}{2}$ bad events by 4δ, as opposed to $(d+1)\delta$ as before.

Self-correction with prover

Suppose that we can get help from interacting with a prover, on top of the oracle access to $f : F^m \mapsto F$, with the same promise as before: f is δ-close to a degree t polynomial $\hat{A}$. The prover wishes to prove statement $\hat{A}(x_0) = b$. Let x_0 and b be given, and our goal is to verify that $\hat{A}(x_0) = b$ for $\hat{A}$ that is a degree d polynomial on m variables that is δ-close to f.

The following requirements capture this goal:

Completeness : if $f = \hat{A}$, then accept with probability 1.

Soundness : if f is δ-close to some polynomial $\hat{A}$ of degree d, and $\hat{A} \neq b$, then reject with probability $\geq 1/2$.

(Note: in terms of soundness, what we will achieve at first will be somewhat weaker.)
Consider the following protocol:

1. Pick a random point y and set $r = y - x_0$. Let $L_{x_0,r}$ denote the set of points that lie on the line determined by x_0 and y. That is to say, $L_{x_0,r} = \{x_0 + tr\}$ for a scalar parameter t.

2. Determine the coefficients of a polynomial $\hat{A}|_{x_0,r} \stackrel{\triangle}{=} \hat{A}(x_0 + tr)$. More specifically:

   (a) Get polynomial $p(t)$ from the Prover. The Prover claims that $p(t) = \hat{A}|_{x_0,r}$.

   (b) Check the Prover's claim: pick a random $t_0 \in F \setminus \{0\}$ and check whether $p(t_0) = f(x_0 + t_0 r)$.

3. Check if $\hat{A}|_{x_0,r} = b$.

Claim 3 *The algorithm above captures our goals.*

Proof It is clear that the algorithm above will give us perfect completeness. The only thing we need to worry about is soundness. Assume that $\hat{A}|_{x_0,r} \neq b$. Then either we successfully reject, or $p(t) \neq \hat{A}|_{x_0,r}(t)$. But if $p(t) \neq \hat{A}|_{x_0,r}$, then, since both are polynomials of degree d, and t_0 is a random point, we bound the probability of this first bad event:

$$\Pr_{t_0}[p(t_0) = \hat{A}|_{x_0,r}(t-0)] \leq d/|F|$$

However, we don't know $\hat{A}|_{x_0,r}$, we only know f which is δ-close to it! Let us rely on $f(x_0 + t_0 r)$ for giving us the right answer; this second bad event's probability can also be bounded:

$$\Pr_{t_0,r}[\hat{A}|_{x_0,r}(t_0) \neq f(x_0 + t_0 r)] \leq \delta$$

because $x_0 + t_0 r$ is a completely random point in F^m. We will accept only if one of the two bad events happens, and so the probability that we accept when we should not is $\leq \delta + d/|F|$. ■

Note that here we may push δ up to any constant $< \frac{1}{2}$. Thus we have arrived at a fairly simple algorithm for self-correcting with just one oracle query and one query to the prover, i.e. with two queries. One reason that we managed to obtain this level of efficiency if because we have a prover that we can interact with. Another reason is that we are taking advantage of the promise that f is δ-close to some degree d polynomial. What if it is not? In the next several lectures, we will see that even when f is not δ-close to any degree d polynomial, this can pretty much be relied upon.