Notes on Complexity Theory: Fall 2005 Last updated: December, 2005

Lecture 12

Jonathan Katz

1 Basics of PCP

Work on *interactive* proof systems has motivated a look at the efficiency of non-interactive proof systems (i.e., the classes $\mathcal{NP}$ and the randomized variant **MA**). One specific question of interest turns out to be the following: how many bits of the proof (sent by the prover) need to be read by the verifier? (Note that in the "default" non-interactive proof for a language $L \in \mathcal{NP}$, the prover sends the witness w that $x \in L$ and the verifier must read the *entire* witness in order to be convinced. The question is whether we can get away with having the verifier read fewer bits. Turning as usual to the analogy with mathematical proofs, this would be analogous to being able to verify the proof of a mathematical theorem by reading only a couple of letters in the proof!) Amazingly, we will see that for $L \in \mathcal{NP}$ it is possible to have the verifier read only a *constant* number of bits.

Abstracting the above ideas, we define the class PCP of *probabilistically-checkable proofs*:

Definition 1 Let r, q be arbitrary functions. We say $L \in \mathsf{PCP}(r(\cdot), q(\cdot))$ if there exists a randomized polynomial-time verifier A such that:

- $A^\pi(x)$ uses (at most) $r(|x|)$ random coins and reads (at most) $q(|x|)$ bits of π.[1] (The running time of A is also measured as a function of $|x|$ — note that $|\pi|$ may be exponential in $|x|$ (cf. footnote 1).)
- If $x \in L$ then there exists a π such that $\Pr[A^\pi(x) = 1] = 1$.
- If $x \notin L$ then for all proofs π we have $\Pr[A^\pi(x) = 1] \leq 1/2$.

If R, Q are classes of functions, then $\mathsf{PCP}(R(\cdot), Q(\cdot)) \overset{\text{def}}{=} \cup_{r \in R, q \in Q} \mathsf{PCP}(r(\cdot), q(\cdot))$. For convenience, we also sometimes let $\mathsf{PCP} \overset{\text{def}}{=} \mathsf{PCP}(\mathsf{poly}, \mathsf{poly})$. ♢

Some remarks are in order:

- The soundness error can, as usual, be reduced by repetition. For exponentially-small error, this does not affect the class PCP; however, such repetition affects the parameters r, q. (E.g., if $L \in \mathsf{PCP}(r, q)$ by the above definition then $L \in \mathsf{PCP}^*(|x| \cdot r, |x| \cdot q)$ if by PCP^* we mean that we require soundness error $2^{-|x|}$.)
- One can also relax the completeness condition (as long as there is an inverse polynomial gap between the acceptance probabilities when $x \in L$ and when $x \notin L$). As with the case of soundness, this does not affect the class PCP (via a similar argument as in the case of **MA**) but does affect the parameters.

[1]Formally, A has a special oracle tape on which it can write any index i and then obtain the i^{th} bit of π in the next step. Since A runs in polynomial time (in $|x|$), the length of i is polynomial and so it is only meaningful for $|\pi|$ to be at most exponential in $|x|$.

- One can view a probabilistically-checkable proof as a form of interactive proof where the (cheating) prover is restricted to committing to its answers *in advance*. Since the power of the cheating prover is restricted but the abilities of an honest prover are unaffected, $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly})$ is at least as powerful as $\mathcal{IP}$. In particular, since $\mathsf{PSPACE} \subseteq \mathcal{IP}$ it follows that $\mathsf{PSPACE} \subseteq \mathsf{PCP}(\mathsf{poly}, \mathsf{poly})$.
- The above definition allows A to query bits of π *adaptively* (i.e., it may request to read the i^{th} bit, and then based on this value determine with bit j it wants to read next). We will only consider non-adaptive verifiers. For future reference, however, we note that any adaptive verifier making a constant number of queries can be converted into a non-adaptive verifier which still makes only a constant (but larger) number of queries.
- The above definition allows the proof π to be different for every $x \in L$. Without loss of generality, however, we may assume a *single* proof that works for all $x \in L$. The reason is that we can simply concatenate all the individual proofs together, and the verifier can still access the relevant proof by specifying a poly-size index.

1.1 Toward Understanding the Power of PCP

An easy observation is that $\mathsf{PCP}(0, \mathsf{poly}) = \mathcal{NP}$. In fact, we have the following stronger result:

Lemma 1 $\mathsf{PCP}(\log, \mathsf{poly}) = \mathcal{NP}$.

Proof Containment of $\mathcal{NP}$ in $\mathsf{PCP}(\log, \mathsf{poly})$ is obvious. For the reverse containment, let $L \in \mathsf{PCP}(\log, \mathsf{poly})$ and let A be the verifier for L. For given $x \in L$, we will show how to construct a witness for x; the $\mathcal{NP}$-machine deciding L will follow naturally. Note that we cannot simply use a "good" proof π_x (which is guaranteed to exist since $x \in L$) because π_x may be exponentially long. However, we *can* use a "compressed" version of π_x. In particular, imagine running A for all possible settings of its $O(\log n)$ random coins (here, $n = |x|$). This results in a set S of only *polynomially-many* indices at which A potentially reads π_x (for each setting of its random coins, A reads poly-many indices; there are only $2^{O(\log n)} = \mathsf{poly}(n)$ settings of the random coins). These queries/answers $\{(i, \pi_i)\}_{i \in S}$ will be our $\mathcal{NP}$ witness. Our $\mathcal{NP}$ algorithm for L is simple: on input a witness w of the above form, simulate the computation of A (in the natural way) for all possible settings of its random coins. Accept only if A accepts in all these executions. (If A tries to read an index which is not present in w, we count this as a rejection by A.) It is not hard to see that this indeed gives a $\mathcal{NP}$ machine deciding L. ■

At the other extreme, if we allow no queries to π we obtain $\mathsf{PCP}(\mathsf{poly}, 0) = \mathrm{co}\mathcal{RP}$ (at least if we require perfect completeness, as we do in our definition). This, along with the previous result, shows that we only "gain" something from our definition of PCP if we consider the power of randomness and proof queries in tandem. Doing so yields the following deep (and important) result:

Theorem 2 (The PCP Theorem) $\mathcal{NP} = \mathsf{PCP}(\log, O(1))$.

We remark that the number of queries can be taken to be a fixed constant which is the same for all $\mathcal{NP}$-languages L (and not, e.g., a constant that depends on the language but not the input length). To see that this follows from the theorem, note that the theorem implies $SAT \in \mathsf{PCP}(c \log n, t)$ for some constants c, t (where n is the length of the input formula). Now for any $L \in \mathcal{NP}$ we can construct the PCP system in which the verifier first applies a Karp reduction to the input to obtain a 3CNF formula ϕ, and then runs the PCP system for SAT on "input" ϕ. If the Karp reduction maps n-bit inputs to n^k-bit formulae (for some constant k), this proves that $L \in \mathsf{PCP}(ck \log n, t)$.

The above characterization is tight under the assumption that $\mathcal{P} \neq \mathcal{NP}$, since $\mathcal{P} \neq \mathcal{NP}$ implies $\mathcal{NP} \not\subseteq \mathsf{PCP}(o(\mathsf{log}), o(\mathsf{log}))$ (Arora and Safra [2, Sec. 1.2.2] observe that this follows from results of Feige, et al. [3]).[2] Also, although not explicit in the theorem, the PCP theorem also shows how to efficiently convert any witness w for a given x (with respect to a given $\mathcal{NP}$ relation R) into a proof π_x for which the corresponding PCP verifier always accepts.

For completeness, we state the following result (which we will not explore any further):

Theorem 3 $\mathsf{NEXP} = \mathsf{PCP}(\mathsf{poly}, \mathsf{poly})$.

2 PCP and Inapproximability

Assuming $P \neq \mathcal{NP}$, we know that we cannot hope to exactly solve $\mathcal{NP}$-complete problems in polynomial time. However, we can in general hope to perhaps find an *approximate* solution to the given problem. Unfortunately, the PCP theorem limits the extent to which we can approximate a number of different $\mathcal{NP}$-complete problems.

As an example, we will show now that there exists some constant α such that it is infeasible (in polynomial time) to approximate to within a multiplicative factor of α the maximum number of satisfiable clauses in a 3SAT formula. We begin with some definitions; in what follows *we restrict our attention to 3CNF formulae ϕ.*

Definition 2 For a formula ϕ and an assignment $\vec{b}$ to the variables in ϕ, let $\mathsf{sat}_{\vec{b}}(\phi)$ denote the fraction of clauses satisfied by the given assignment. Let $\mathsf{max\text{-}sat}(\phi) = \max_{\vec{b}}\{\mathsf{sat}_{\vec{b}}(\phi)\}$. ♢

Note that $\mathsf{max\text{-}sat}(\phi) = 1$ iff ϕ is satisfiable. Also, for any formula ϕ we have $\mathsf{max\text{-}sat}(\phi) \geq 7/8$ (since a random assignment satisfies any given clause in ϕ with probability at least 7/8).

Definition 3 A value k is an *α-approximation for ϕ* if $k \leq \mathsf{max\text{-}sat}(\phi) \leq \alpha \cdot k$. Polynomial-time algorithm A is an $\alpha(\cdot)$-approximation algorithm for 3SAT if $A(\phi)$ always outputs an $\alpha(|\phi|)$-approximation for ϕ. ♢

A 1-approximation algorithm for 3SAT would obviously imply that we could solve 3SAT in polynomial time. What can we say about the existence of a β-approximation algorithm for some constant $\beta > 1$? Note that by what we have said above, it is trivial to find an 8/7-approximation in polynomial time by always outputting the answer "7/8." Can we do better?

Toward showing that there is a limit to how well we can do (assuming $\mathcal{P} \neq \mathcal{NP}$), we introduce the notion of an *amplifying reduction.*

Definition 4 A *c-amplifying reduction* of 3SAT is a poly-time function f mapping the set of 3CNF formula to itself and such that:

- If ϕ is satisfiable, then $f(\phi)$ is satisfiable. I.e., if $\mathsf{max\text{-}sat}(\phi) = 1$ then $\mathsf{max\text{-}sat}(f(\phi)) = 1$.
- If ϕ is not satisfiable, then every assignment to the variables in $f(\phi)$ satisfies at most a c-fraction of the clauses in $f(\phi)$. I.e., if $\mathsf{max\text{-}sat}(\phi) < 1$ then $\mathsf{max\text{-}sat}(f(\phi)) < c$.

We will say that 3SAT has an amplifying reduction if it has a c-amplifying reduction for some $c < 1$. In particular, an amplifying reduction is a Karp reduction. ♢

It is fairly immediate to show that an amplifying reduction for 3SAT implies a hardness-of-approximation result for $\mathsf{max\text{-}sat}$:

[2]It is possible, however, that $\mathcal{NP} \subseteq \mathsf{PCP}(o(\mathsf{log}), \mathsf{log})$.

Lemma 4 *Assume $\mathcal{P} \neq \mathcal{NP}$ and that 3SAT has a c-amplifying reduction. Then there is no c^{-1}-approximation algorithm for 3SAT.*

Proof Assume to the contrary that there does exist a c^{-1}-approximation algorithm A for 3SAT. We can then deterministically solve SAT in polynomial time as follows: on input formula ϕ, run $A(f(\phi))$ to obtain output k. If $k \geq c$, output 1; otherwise, output 0. To see correctness of this algorithm, note that when ϕ is satisfiable then $\textsf{max-sat}(f(\phi)) = 1$ and so the output k of A must satisfy

$$k \geq \textsf{max-sat}(f(\phi))/c^{-1} = c.$$

On the other hand, when ϕ is not satisfiable then $\textsf{max-sat}(f(\phi)) < c$ and so the output k of A must satisfy $k < c$. The claim follows. ■

To establish the connection between the PCP theorem and inapproximability, we show that the PCP theorem implies the existence of an amplifying reduction for 3SAT. In fact, the implication goes in both directions, thus showing that one way to prove the PCP theorem is to construct an amplifying reduction for 3SAT.

Lemma 5 $\mathcal{NP} \subseteq \textsf{PCP}(\textsf{log}, O(1))$ *if and only if 3SAT has an amplifying reduction.*

Proof One direction is easy. If 3SAT has an amplifying reduction f, then we can construct the following PCP system for 3SAT: On input ϕ, the verifier computes $f(\phi)$. The proof will contain a satisfying assignment for $f(\phi)$ (i.e., position i of the proof contains the assignment to x_i). To check the proof, the verifier chooses a random clause in $f(\phi)$, queries for the assignments to the 3 variables of that clause, and then determines whether that clause is satisfied for those settings of the variables. It accepts if and only if that is the case.

If ϕ is satisfiable then $f(\phi)$ is satisfiable and so a valid proof (consisting of a satisfying assignment for $f(\phi)$) exists. On the other hand, if ϕ is not satisfiable then at most a c-fraction of the clauses in $f(\phi)$ are satisfiable (for *any* assignment to the variables), and so the verifier accepts with probability at most c regardless of the proof. Since c is a constant, repeating the above procedure a constant number of times (and accepting only if each procedure would lead to acceptance) will give the desired soundness error 1/2 using a constant number of queries. Also, the number of random bits needed to select a random clause is logarithmic in $|\phi|$ since $|f(\phi)|$ is polynomial in $|\phi|$.

The other direction is the more interesting one. If $\mathcal{NP} \subseteq \textsf{PCP}(\textsf{log}, O(1))$ then, in particular, $SAT \in \textsf{PCP}(\textsf{log}, O(1))$. Let A be a verifier for such a proof system for SAT, using $c \log n$ random coins (on input ϕ with $|\phi| = n$) and making t queries. We now describe our amplifying reduction f: on input a SAT instance ϕ (with $|\phi| = n$) do:

- For each setting r of the random coins for A, do the following:
  - Determine the t indices $q_1, \ldots q_t$ that $A(\phi; r)$ would query in the proof when using random coins r (recall that without loss of generality these indices are chosen non-adaptively).
  - Run $A(\phi; r)$ on all possible settings for these bits of the proof to determine when A accepts in this case. In this way, one may define a CNF formula $\hat{\phi}_r$ on the variables $x_{q_1}, \ldots, x_{q_t}$ such that $\hat{\phi}_r$ evaluates to true exactly when $A(\phi; r)$ would accept. (We stress that variables of the type x_{q_i} are the same for the different settings of r.) The important point is that the number of clauses in $\hat{\phi}_r$ is constant since t is constant. Using auxiliary variables (different for each r), we may convert $\hat{\phi}_r$ to an equivalent 3CNF formula ϕ_r. The number of clauses in ϕ_r is constant as well.

- Set the output $f(\phi)$ to be $\bigwedge_{r \in \{0,1\}^{c \log n}} \phi_r$.

Note that the above can be implemented in polynomial time and, in particular, both the number of clauses an the number of variables in $f(\phi)$ are polynomial.[3]

We claim that f, as given above, is an amplifying reduction. It is not hard to see that if ϕ is satisfiable then $f(\phi)$ is (this follows from perfect completeness of the PCP system). On the other hand, assume ϕ is not satisfiable. Then for *any* setting of the variables in $f(\phi)$, at least half of the $\{\phi_r\}$ are not satisfied (this follows from soundness of the PCP system). In each unsatisfied ϕ_r there is at least one unsatisfied clause. Let $t' = O(1)$ denote the maximum number of clauses in any of the $\{\phi_r\}$. It follows that for any setting of the variables, the fraction of unsatisfied clauses in $f(\phi)$ is at least

$$\beta \stackrel{\text{def}}{=} \frac{2^{|r|-1}}{2^{|r|} \cdot t'} = \frac{1}{2t'},$$

and so the fraction of satisfied clauses is at most $1 - \beta$. Since β is a constant greater than 0, this means that f is a c-amplifying reduction for any $c \in (1 - \beta, 1)$. ■

2.1 Inapproximability of Clique

An alternate way of viewing approximation problems is to view them as *promise* problems. Taking the case of clique for example, we may define the promise problem $CLIQUE_{\alpha,\beta}$ as consisting of "yes" instances which are graphs whose largest clique contains at least $\alpha(n)$ vertices (here, n is the total number of vertices in the graph), and "no" instances which are graphs whose largest clique contains at most $\beta(n)$ vertices. We have the following nice theorem, which implies that it is $\mathcal{NP}$-hard to approximate the size of the largest clique of a graph to within a factor better than 2:

Theorem 6 *There exists a function $\alpha = \Theta(n)$ such that* $\text{CLIQUE}_{\alpha,\alpha/2}$ *is $\mathcal{NP}$-hard.*

Proof Given an $\mathcal{NP}$-language L, we show a transformation T that takes an input x and outputs a graph G such that the following hold: if $x \in L$, then $T(x)$ is a "yes" instance of $CLIQUE_{\alpha,\alpha/2}$ while if $x \notin L$ then $T(x)$ is a "no" instance for $CLIQUE_{\alpha,\alpha/2}$. We stress that α is fixed, and so is the same for *all* $\mathcal{NP}$-languages L.

By the PCP theorem, there exists a constant t such that for any $L \in \mathcal{NP}$ there is a poly-time verifier A which on input x makes t queries using $O(\log |x|)$ coin tosses. Let $r_1, \ldots, r_m$ denote the sequence of possible coin tosses of A (note that m is polynomial in $|x|$), and let $q_1^i, \ldots, q_t^i$ denote the queries made on random coin tosses r_i. Let $a_1^i, \ldots, a_t^i$ be a sequence of possible answers. Define a graph G_x as follows:

Vertices For each set of random coins r_i and each possible set of answers $\{a_j^i\}_{j=1}^t$, the tuple

$$(r_i, (q_1^i, a_1^i), \ldots, (q_t^i, a_t^i))$$

is a vertex if and only if A would accept x when using random tape r_i and receiving these answers to its queries.

Since r_i and x uniquely determine the queries, there are at most $m \cdot 2^t$ vertices in G_x.

[3]Even though the *indices* of the variables (i.e., the q_i) are polynomial length, at most $2^{c \log n} \cdot t$ indices are ever potentially queried by A.

Edges Two vertices v and u have an edge between them if and only if they are consistent. (Two vertices are *not* consistent if they contain different answers to the same query.) Note that if u, v contain the same random tape r_i then they are inconsistent and so do not share an edge.

Finally, add isolated vertices (if necessary) to obtain a graph with exactly $m \cdot 2^t$ vertices.

Define $\alpha(n) \stackrel{\text{def}}{=} n/2^t$, so that $\alpha(m \cdot 2^t) = m$. We show that G_x satisfies our desiderata:
– When $x \in L$, there exists a proof π for which A accepts for every setting of its random tape (this follows due to perfect completeness). It is not hard to see that this implies the existence of a clique in G_x of size at least m.

– When $x \notin L$, the existence of a clique with more than $m/2$ nodes would imply the existence of a proof that would cause A to accept with probability greater than 1/2, a contradiction to the soundness of the PCP system. ■

Bibliographic Notes

This lecture is based on [4, Lect. 12]. The PCP theorem as stated here was proved in [1, 2], but these papers represent the culmination of a long sequence of work in this area. The application of the PCP theorem to inapproximability (and, in particular, the connection to the clique problem shown in these notes) is due to [3].

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof Verification and Intractability of Approximation Problems. *J. ACM* 45(3): 501–555, 1998.

[2] S. Arora and S. Safra. Probabilistic Checking of Proofs: A New Characterization of *NP*. *J. ACM* 45(1): 70–122, 1998.

[3] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive Proofs and the Hardness of Approximating Cliques. *J. ACM* 43(2): 268–292, 1996.

[4] O. Goldreich. Introduction to Complexity Theory (July 31, 1999).