CS 710: Complexity Theory 4/27/2010

Lecture 27: The PCP Theorem

Instructor: Dieter van Melkebeek Scribe: Dmitri Svetlov, Tom Watson and Nathan Collins

In the last few lectures, we have developed a generalized notion of proofs and proof complexity, considering particularly those proofs that are *efficiently veriable*, and therefore correspond to the set of non-deterministic computations –therefore, there are short proofs of membership in the language L exactly if $L \in$ NP.

Later we relaxed this notion by allowing randomness in proof verification. Under this definition, we can develop *interactive proof systems* in which a computationally unlimited, randomized *prover* interacts with and tries to convince an efficient, *i.e.*, polytime, randomized *verifier* to accept the validity of a purported proof provided by the prover. We quantified the notions of *completeness* and *soundness* for such systems by requiring that for a prover-verifier protocol with completeness c and soundness s, the verifier must accept the members of the language with a probability no less than c and reject, regardless of the choice of prover, non-members of the language with a probability at least s. The set of all languages with short proofs using these systems, IP, was shown to be exactly PSPACE.

Then we introduced the concept of *probabilistically checkable proof systems*, in which the verifier is a polytime probabilistic oracle Turing machine that uses a certain number of random bits and queries a certain number of bits of a provided proof, and whose prover must be a non-adaptive oracle. We showed that the class PCP($r^{O(1)}, q^{O(1)}$), whose verifiers use only polynomially many random bits and query only polynomially many bits of the proof, is exactly equal to NEXP and also to MIP, the class of interactive proof systems with multiple provers whom the verifier can query.

In this lecture, we will state and prove (in part) the PCP Theorem, which analogously characterizes the class NP as the set of languages with PCP systems whose verifiers use only logarithmically many random bits and query only a constant number of bits of the proof.

1 The Power of the Honest Prover

Although thus far we have allowed the provers of interactive and probabilistically checkable proof systems to be computationally unbounded, for a variety of reasons we would like to know how much work an honest (as opposed to malicious) prover would need to do to convince a verifier of the membership of the given input in the language of interest, and we would like this work to not be too great. Specifically, the computation required shouldn't be more difficult than simply solving the original problem (membership in the language) from scratch; ideally then, the honest prover should be able to answer queries in polynomial time, given access to an oracle for the language of interest.

Let us review several examples of this that we've already seen.

- *Our private-coin IP system for graph non-isomorphism.* Recall that, given the two graphs as input, the verifier becomes convinced by selecting one of the graphs at random, permuting the vertices, and asking the prover which graph is isomorphic to the new one. To determine

this, the prover essentially needs just two queries to an oracle for graph non-isomorphism –alternatively, a power equivalent to P^{GNI}.

- *Our sumcheck protocol for* #P *functions.* Recall that the prover only ever needs to finds the coefficients of a univariant polynomial by summing the values of a multivariant polynomial over all binary settings of all but one variable. The prover can do this by plugging in a value for that free variable and using a #P oracle to find the value of the polynomial on that input. (This in turn is due to the closure of #P under uniform exponential summations.) Since the desired polynomial is of low degree, the prover can reconstruct it from its values on relatively few inputs –so again, this prover needs just the power of $\mathrm{P}^{\#\mathrm{P}}$.

- *Our prover for the* PSPACE *protocol.* Recall that the proof of the inclusion IP $\subseteq$ PSPACE involved simulating the IP system and finding which strategy caused the verifier to accept with the highest probability. The prover can follow this strategy and require only polynomially much space to compute its responses to the verifier. Hence the prover is P^{PSPACE}.

However, there are examples for which it is open whether an honest prover requires more computational power than the amount needed to solve the problem at hand.

- This is true of our public-coin IP system for graph non-isomorphism.

- From some earlier results we can obtain IP systems for coNP, but we do not know whether an honest prover for these protocols can compute its responses with an oracle for coNP.

We discuss below two implications of the powers of honest provers.

1.1 Collapse of Complexity Classes

Recall these earlier results.

Theorem 1. *If* NP $\subseteq$ P/ poly*, then* PH $= \Sigma_2^p$.

Theorem 2. *If* PSPACE $\subseteq$ P/ poly*, then* PSPACE $= \Sigma_2^p$.

Theorem 3. *If* EXP $\subseteq$ P/ poly*, then* EXP $= \Sigma_2^p$.

We proved Theorem 1 in an earlier lecture and left Theorems 2 and 3 as exercises. We can now prove stronger versions of the latter two theorems.

Theorem 4. *If* PSPACE $\subseteq$ P/ poly*, then* PSPACE $=$ MA.

Proof. Recall we showed earlier that PSPACE $\subseteq$ MA, so this result is indeed stronger. Let L be a language in PSPACE. There is an IP system for L in which the prover's responses are computable in polyspace. But if PSPACE $\subseteq$ P/ poly, then there is a polysize circuit that implements the honest prover's strategy, *i.e.*, given a transcript of a message history, it computes each bit of the next message sent by the honest prover. So we have Merlin determine this circuit and send it to Arthur, who can then carry out the protocol himself –since the circuit is polysize, it is within his computational bounds. If $x \in L$, then there is a circuit that convinces Arthur to accept x with probability 1, as argued earlier. But if $x \notin L$, then no prover and, specifically, no polysize circuit exists that could convince Arthur to accept x with a probability greater than the soundness parameter of the IP system for L. So $L \in$ MA. □

Since the power of IP systems extends only as far as PSPACE, we can simply use an analogous argument to strengthen Theorem 3. However, we can do this in a very similar way using the fact that the power of MIP systems extends up to NEXP.

Theorem 5. *If* $\mathrm{EXP} \subseteq \mathrm{P/poly}$*, then* $\mathrm{EXP} = \mathrm{MA}$.

Proof. Let L be a language in EXP. Since, as we proved earlier, MIP = NEXP, $L \in \mathrm{MIP}$. Together this shows that the strategies of the honest provers in this protocol can be realized in exponential time. Now assuming that $\mathrm{EXP} \subseteq \mathrm{P/poly}$, there exist polysize circuits that implement these strategies. Thus Merlin needs only to determine these circuits and send them to Arthur, who can then carry out this protocol himself and use the circuits to compute the prover's responses. So $L \in \mathrm{MA}$. □

One might naturally wonder whether analogous results hold for NP and NEXP. For both classes this question is in fact open; we do not know whether honest provers in protocols for NP and NEXP can compute their responses efficiently given access to oracles for the language to be decided. (As it often so happens with these classes, problems arise with the complementation of the relevant language.) However, it is believed that such a result does hold, as argued below.

Conjecture 1. *If* $\mathrm{NP} \subseteq \mathrm{P/poly}$*, then* $\mathrm{PH} = \mathrm{MA}$.

We argue this conjecture as follows. Given a language $L \in \mathrm{coNP}$, suppose we knew that an honest prover for the IP system described earlier could compute its responses efficiently, given access to an oracle for NP. Then, using an argument similar to the proof of Theorem 4, together with the fact that $\mathrm{coNP} \subseteq \mathrm{MA}$ implies that $\mathrm{PH} = \mathrm{MA}$, we could establish Theorem 1. Indeed, the honest prover need only use an oracle for some language in PH, since we know that $\mathrm{NP} \subseteq \mathrm{P/poly}$ implies that $\mathrm{PH} \subseteq \mathrm{P/poly}$. But the best honest provers we know for languages in coNP require the power of counting.

1.2 Program Verification and Instance Checking

The power of the honest prover also has implications for the project of program verification –the attempt to determine whether a given algorithm works as intended for all inputs. In general, this problem is undecidable, but we can aim for the more modest goal of *instance checking*, which attempts to determine whether a specific program P works correctly for a specific input x by examining its behavior on x and some related inputs. We would like to avoid dealing with inputs totally unrelated to x and also to not require as many resources as the trivial verification procedure that decides the problem from scratch and then explicitly checks the program's correctness on the given input. Further, we assume oracle access to P so that we incur no costs for running it as a black-box procedure. Our model is then formally defined as follows.

Definition 1. *An* instance checker C *with completeness* c *and soundness* $s < c$ *for a language* L *is a probabilistic polytime randomized oracle Turing machine such that for an oracle* O *and input* x*, if* $O = L$ *(i.e. if the program is correct), then* $\Pr[C^O(x) \textit{accepts}] \geq c$*, and if* $O(x) \neq L(x)$ *(i.e. if the program is incorrect on the given input), then* $\Pr[C^O(x) \textit{accepts}] \leq s$.

For instance checkers we typically set the completeness and soundness parameters to be $c = 1, s = 1/2$.

As we demonstrate below, there is a strong connection between these instance checkers and PCP systems for identical languages.

Theorem 6. *A language L has an instance checker if and only if both L and $\overline{L}$ have a PCP system with completeness c and soundness s such that the honest prover has a power equivalent to P^L, i.e. its responses are computable in polytime, given access to an oracle for L.*

Proof. We first argue the forward direction. Assume that L has an instance checker with completeness c and soundness s. Note that therefore $\overline{L}$ has an instance checker with the same parameters: it is identical to the checker for L except that it flips the answers to its queries. So by symmetry we need only show that L has a PCP system with the same parameters and an honest prover with the power of P^L. To do this, we simply run our instance checker on the input x and use the proof to answer the checker's queries. We first need to find what the prover says about the membership of x in L, since the soundness of the instance checker holds only for oracles O such that $O(x) \neq L(x)$. So we have the verifier ask this question of the prover, and if the prover answers that $x \notin L$, we reject immediately. Otherwise, we have the verifier run the instance checker for L (recall that since the instance checker is by definition a polytime machine, this is within the verifier's computational bounds). For the cases where $x \in L$, completeness follows from the completeness of the instance checker, and for the other cases, a cheating proof must falsely claim that $x \in L$ to convince the verifier to accept. Soundness will then follow from the soundness of the instance checker.

Now we argue the reverse direction. Assume that L and $\overline{L}$ have PCP systems as stated above. Then since $\mathrm{PCP}(\mathrm{poly}(n), \mathrm{poly}(n)) \subseteq \mathrm{MIP}[2]$, as proved earlier, both languages have MIP systems with parameters at least as good and honest provers with the power of P^L or $\mathrm{P}^{\overline{L}}$, as appropriate. So we design an instance checker for L as follows. On input x, we first ask our oracle whether $x \in L$. If the oracle says it is, we run the MIP for L; otherwise, we run the one for $\overline{L}$. In simulating the chosen verifier, we respond to queries by running the appropriate honest prover's strategy using our oracle. We accept the proof if and only if our chosen verifier accepts. To demonstrate that we do preserve the necessary degree of completeness and soundness, first suppose that our oracle agrees with L everywhere. Then the MIP system we choose will be such that x is in the language it decides, and since all responses to proof queries agree with the honest prover's strategy, we accept with probability at least c and so preserve completeness. Suppose instead that our oracle disagrees with L on x. Then the MIP system we choose will be the wrong one; x is not in the language it decides. In this case, it is irrelevant how we respond to proof queries; we will accept only with a probability at most s, so we also preserve soundness. □

From this theorem it immediately follows that there exist instance checkers for several languages we've been considering.

Corollary 1. *There exist instance checkers with perfect completeness for graph non-isomorphism, every* PSPACE*-complete language, every* #P*-complete function (provided we extend our definition of an instance checker to function problems in the natural way), and every* EXP*-complete language.*

Proof. We noted early in Section 1 that the private-coin IP system for graph non-isomorphism has an honest prover whose strategy can be realized in polytime with access to an oracle for graph non-isomorphism. Further, this protocol has perfect completeness. Also, there is trivially an IP system with perfect completeness and soundness 0 for graph isomorphism, where the honest prover's strategy is simply to find an isomorphism (if one exists) between two given graphs. We noted in an earlier lecture that this can be solved in polytime with access to an oracle for (the decision version of) graph isomorphism. Our first result now follows from the proof of Theorem 6, bypassing the

PCP systems in the statement of the theorem and instead directly applying the IP systems given above.

The second result holds similarly for every PSPACE-complete language using the IP systems for PSPACE developed earlier and the fact that all polyspace computations can be carried out in polytime with access to an oracle for some PSPACE-complete language.

The third result holds using an argument similar to the proof of Theorem 6, where we adapt our definition of an instance checker to function problems as follows. First we have the instance checker ask its oracle for the value of the function on the given input x. It then parsimoniously reduces x to a #SAT formula and runs the sumcheck protocol. The strategy of the honest prover can be implemented in polytime, given access to an oracle for #SAT and hence, given access to an oracle for L by PSPACE-completeness.

The final result follows from the proof of Theorem 5, where we noted that every EXP-complete language has an MIP system whose honest provers' responses are computable in polytime with access to an oracle for an EXP-complete language (and whose protocol also has perfect completeness). Since EXP is closed under complementation, the conclusion follows from this fact and the proof of Theorem 6. □

2 The PCP Theorem

We now turn to the PCP Theorem itself. From the time hierarchy theorem for nondeterministic computation, we know that there exist languages in NEXP without short, efficiently verifiable proofs. On the other hand, some recent results imply that *all* languages in NEXP have efficiently verifiable proofs where we allow the verifier to randomly spot-check just a few locations of the proof. Can we scale this result down to an analogous one for NP, the realm of classical proofs? In other words, can we replace classical proofs with ones where it is only necessary to spot-check a handful of random locations? This analog is captured by the PCP Theorem, one of the greatest and most influential results in the theory of computation.

Theorem 7 (The PCP Theorem). $\text{NP} = \text{PCP}(O(\log n), O(1))$.

The inclusion $\text{PCP}(O(\log n), O(1)) \subseteq \text{NP}$ follows from our earlier result that $\text{PCP}(r(n), q(n)) \subseteq \text{NTIME}(2^{O(r(n)+q(n))} n^{O(1)})$. However, the other inclusion is significantly harder. We do not prove it in this course, since it would require more time than what remains, but we will illustrate some of the flavor of the proof by proving later the following weaker result, whose demonstration requires some of the same techniques.

Theorem 8. $\text{NP} \subseteq \text{PCP}(O(\text{poly}\, n), O(1))$.

The proof we will design for the PCP system that will prove this result will be an exponentially long encoding of a classical proof; nonetheless, it will demonstrate one of the astonishing implications of the PCP Theorem –namely that we only need to query a constant number of bits (specifically, three) of a proof in order to verify its correctness.

2.1 Implications for the Hardness of Approximation

One of the major practical applications of the PCP Theorem, and the one we will consider, concerns the hardness of approximation. It has been shown that many important combinatorial optimization

problems are NP-hard to solve exactly, and the PCP Theorem goes further to imply that is NP-hard to even approximate them to within certain factors. We even have tight results for some problems, *i.e.*, approximation algorithms and hardness results whose approximation factors essentially match, allowing us to characterize the approximability of these problems in terms of the PCP Theorem.

Basically, the PCP Theorem has implications for the hardness of approximation because the prover is attempting to solve an optimization problem –that of getting the verifier to accept the provided proof –and this theorem introduces a non-negligible gap in this probability between those cases where the input is in the language of interest and those where it is not.

We now illustrate the relationship between the PCP Theorem and hardness of approximation results by presenting one such result and proving that it is equivalent to the PCP Theorem.

Theorem 9. *There exists a constant $\alpha < 1$ and a $<_m^p$-reduction f from 3-SAT to 3-SAT such that if a 3-CNF $x \notin 3-SAT$, then no more than a fraction α of the clauses of $f(x)$ can be simultaneously satisfied, and if $x \in 3-SAT$, than so is $f(x)$.*

Theorem 10. *The* PCP *Theorem is equivalent to Theorem 9.*

Proof. ($\Downarrow$): Assume the PCP Theorem, where the relevant PCP has a completeness of 1 and a soundness of 1/2. Then there exist constants q (the $O(1)$ number of queries made to the proof) and c (the constant in the $O(\log n)$ number of random bits used), and a polytime verifier V, such that for every 3-CNF x

- If $x \in$ 3-SAT then there exists a proof Π such that

$$\Pr_{\rho \in \{0,1\}^{c \log n}}[V^{\Pi}(x)] = 1. \tag{1}$$

- If $x \notin$ 3-SAT then for all proofs Π

$$\Pr_{\rho \in \{0,1\}^{c \log n}}[V^{\Pi}(x)] \leq 1/2. \tag{2}$$

Now let ρ denote a random string in $\{0,1\}^{c \log n}$ and V_ρ denote V supplied with a sequence ρ of random bits. The important point to note is that after ρ is fixed, the value of $V_\rho(x)$ depends only upon the (up to) q bits of the supplied proof Π that $V_\rho(x)$ happens to query. In general, the kth bit queried depends upon the values of the previous $k-1$ bits queried (thus the first bit queried is independent of Π). Essentially then, the sequence of queries can be thought of as a decision tree of at most q levels, where each node corresponds to a particular sequence of query results thus far obtained and each branch corresponds to the particular answer to the most recent query.

We can then use this tree (see Figure 1) to construct a CNF equivalent to $V_\rho(x)$. To see how, suppose we take the disjunction of all rejecting path signatures, where by "path signature" we mean the conjunction of variables and variable negations equivalent to the values of queries of specific bits of Π made by V along that path down the tree (alternatively, along that sequence of queries). Any one of these conjunctions is true exactly if $V_\rho^{\Pi}(x)$ rejects after that path of queries, so the disjunction of them is true exactly if $V_\rho^{\Pi}(x)$ rejects anywhere. Negating this DNF thus gives a CNF which is false exactly if $V_\rho^{\Pi}(x)$ rejects, and is therefore equivalent to $V_\rho^{\Pi}(x)$. (See Figure 2.) Since the query decision tree has at most 2^q traversing paths, each of length at most q, the CNF we have constructed has at most 2^q clauses, each with at most q literals.

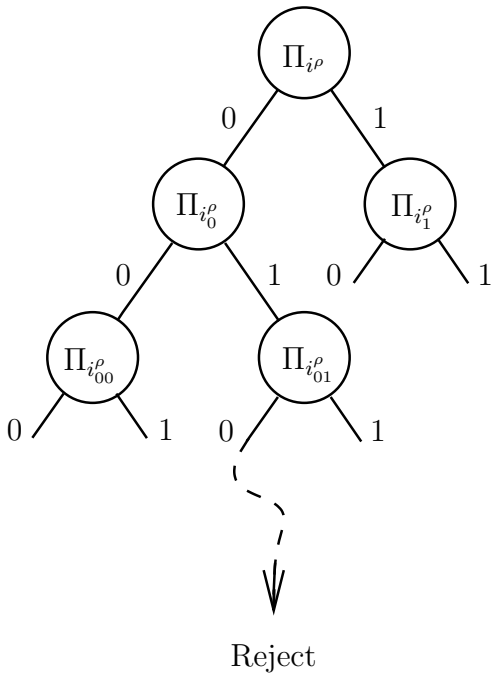

Figure 1: The bits in Π queried by V_ρ^Π depend only on ρ and previously queried bits of Π. The nodes are labeled by the queried proof bit, and the outgoing edges are labeled by the node's value.

Each ($\leq q$)-clause in the CNF above can be expanded into at most q clauses to produce a 3-CNF we will denote by C_ρ. Then C_ρ has at most $q2^q$ clauses, and we define $f(x) = \bigwedge_\rho C_\rho$, where the top two levels of ANDs are contracted to form a single CNF. Then we can show that $f(x)$ has at most $n^c q2^q$ 3-clauses and either

- $x \in$ 3-SAT: Then since the PCP system has perfect completeness (Equation (1)), $f(x) \in$ 3-SAT as well, and thus there exists a Π that makes $V_\rho^\Pi(x)$ accept and C_ρ satisfied for all ρ.
- $x \notin$ 3-SAT: Then since the PCP system has soundness 1/2 (Equation (2)), $f(x) \notin$ 3-SAT, and thus for all Π the computation $V_\rho^\Pi(x)$ rejects and C_ρ is unsatisfiable for at least one half of all ρ. If C_ρ is unsatisfiable, then at least one of its clauses is unsatisfiable; in this case the proportion of clauses of $f(x)$ that are unsatisfiable is at least $(1/2)(1/q2^q) > 0$, so take $\alpha = 1 - 1/q2^{q+1} < 1$.

($\Uparrow$): This direction is much easier than the other. Assume Theorem 9 and let f be the reduction and α the simultaneously-satisfiable constant stated in the theorem. Given a 3-CNF x let V^Π calculate $f(x)$ and treat Π as a Boolean assignment for the variables of $f(x)$. V can access only $O(\log n)$ proof bits, not enough to check the entire formula, but enough to choose a clause of $f(x)$ at random and check if Π satisfies it, querying at most 3 bits of Π. Now suppose $f(x)$ is satisfiable. Then Π can be a satisfying assignment and we ensure perfect completeness. Suppose instead that $f(x)$ is unsatisfiable. Then with probability α the clause of $f(x)$ chosen by V is not satisfied by Π, giving a soundness of at most α. Our chosen definition of PCP requires a soundness of 1/2, so we repeat this procedure $\lceil (\log 1/2)/(\log \alpha) \rceil$ times (since the error is one-sided). This shows

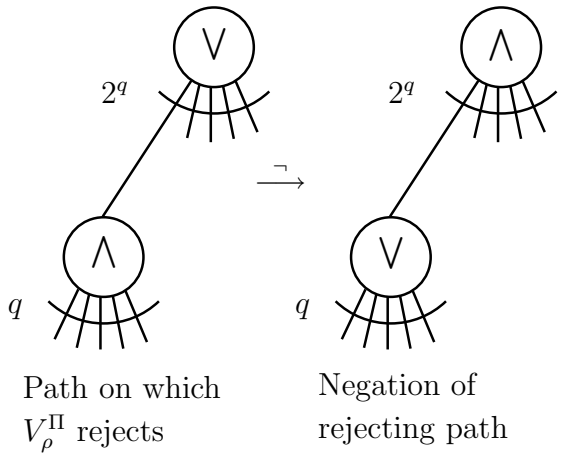

Figure 2: Since the set of paths is prefix-free, V_ρ^Π rejects exactly if the query indices and values of a rejecting path are realized by Π. We form a DNF of rejecting path "signatures" and negate it to get a CNF that is true precisely when no rejecting path is realized.

that 3-SAT $\in$ PCP$(O(\log n), O(1))$, and so by the NP-completeness of 3-SAT we obtain the PCP Theorem. $\square$

3 Next lecture

In the next lecture, we give two additional examples of hardness of approximation results, for MAX-3-SAT and MAX-IND-SET. We will then prove the weaker version of PCP Theorem, presented here as Theorem 8, in which the PCP can use up to polynomially many random bits. To develop the necessary PCP system, we will revisit the Hadamard code and use it in a novel way –to encode classical proofs. As one part of this proof uses discrete harmonic analysis, we introduce just enough of these techniques to make that portion of the argument.