CS 221: Computational Complexity **Prof. Salil Vadhan**

Lecture Notes 20

April 12, 2010 **Scribe: Jonathan Pines**

1 Agenda.

- **PCP**s.
- Approximation Algorithms.
- **PCP**s = Inapproximability.

2 History.

First, some history on the exciting series of results in the early 1990's on probabilistic proof systems. See later sections for some definitions.

- 1990: **IP** = **PSPACE**. In addition to being interesting in its own right, it was the most dramatic example of a natural (and model-independent) nonrelativizing result, giving hope that the methods could resolve some other open problems in complexity theory. The following results followed in this vein and are also nonrelativizing.
- 1991: **MIP** = **NEXP**. **MIP**, or Multiprover Interactive Proofs, is the set of languages verifiable by a single verifier with several provers who cannot communicate with one another. This class was motivated by cryptography, such as Zero Knowledge Proofs without complexity assumptions. Before this result, it was known that **MIP** = **PCP**$(\text{poly}(n), O(1))$.
- 1992: **PCP** Theorem: **NP** = **PCP**$(\log n, O(1))$. This is basically a scaled down version of the 1991 result, and is one of the most (if not the most) celebrated results in computational complexity theory. This will be our focus for the next few lectures.

3 Probabilistically Checkable Proofs (PCPs).

3.1 Definitions.

We begin by defining **PCP** verifiers and **PCP** languages.

Definition 1 *An $(r(n), q(n))$* **PCP** ***verifier*** *for a language L with* ***completeness*** *$c(n)$ and* ***soundness*** *$s(n)$ is an PPT oracle algorithm V with oracle Π such that the following conditions hold, where $n = |x|$, $r(n)$ is the number of coin tosses, and $q(n)$ is the number of oracle queries to Π.*
completeness: $x \in L \Rightarrow \exists \Pi$ s.t. $Pr[V^{\Pi}(x) = 1] \geq c(n)$.
soundness: $x \notin L \Rightarrow \forall \Pi, Pr[V^{\Pi}(x) = 1] \leq s(n)$.

We also require that V's oracle queries be nonadaptive, i.e. the set of queries depends only on x and the coin tosses r, but not on the answers to previous oracle queries. Note that an adaptive algorithm can be made nonadaptive with an exponential blowup in queries by trying all query sequences.

Definition 2 $\mathbf{PCP}_{c,s}(r,q) = \{L : L$ *has an* $(O(r(n)), q(n))$ **PCP** *verifier with completeness* c *and soundness* $s\}$. *When not specified,* c *and* s *default to* 1 *and* $1/2$ *respectively. Note that Arora–Barak has* $O(q(n))$ *in the definition, but we will want to give exact constants for* $q(n)$.

3.2 PCP Theorem.

Now we can state the main result for **PCP**s:

Theorem 3 *(***PCP** *Theorem)* $\mathbf{NP} = \mathbf{PCP}(\log n, O(1))$.

That is, **NP** statements have proofs that can be probabilistically verified by reading only a constant number of bits. A stronger version of the theorem shows a value of 3 for the $O(1)$.

Note that the proof oracle is very different from a standard **NP** witness. The intuitive idea is that e.g. for a SAT witness, it's possible that every partial assignment violates no clauses, yet the full assignment does not satisfy the formula. For a **PCP**, we need that when $x \notin L$, then any proof oracle has many local errors.

One application of this result is that one can encode standard proofs of mathematical theorems using a verifier that checks only a constant number of bits. In particular, applying the PCP Theorem to the **NP** language

$$L = \{(\phi, 1^n) : \phi \text{ a statement in Zermelo-Frenkel (ZF) Set Theory w/a proof of length} \leq n\},$$

one can encode the proof as a probabilistically checkable one using a constant number of bits.

Now, we proceed with the easy direction of the proof.

Proof:

($\supseteq$) Given a language $L \in \mathbf{PCP}(\log n, O(1))$, we use as an **NP** witness the set of answers to all possible queries made by the **PCP** verifier V to the oracle Π. There are at most $2^{r(n)}$ possible coin tosses and at most $q(n)$ queries for each, for a total of at most $2^{r(n)} \cdot q(n) \leq poly(n)$ queries. Then the **NP** verifier can check the queries for consistency (i.e. the same oracle query for different coin tosses gets the same answer), enumerate all $2^{r(n)} \leq poly(n)$ coin tosses, and deterministically compute the proportion of coin tosses on which V accepts. Accept iff the proportion is at least $c(n)$.

($\subseteq$) This direction is much more involved and would take at least a couple of weeks (a few years ago, it would have taken a month or two). Next time, we will see a weaker version of the **PCP** theorem that gives the flavor of the methods. ■

One point to take away from the above proof is that the length of the proof oracle is effectively bounded by $2^{r(n)} \cdot q(n)$, since the verifier can only read so many locations over all its coin tosses and queries. This one of the main reasons we are interested in PCPs with logarithmic randomness. Logarithmic randomness also ensures that we can completely verify the proof in deterministic polynomial time, by enumerating over all sequences of coin tosses.

Next we'll go over approximation algorithms and then see that they are closely related to **PCP**s.

4 Approximation Algorithms (for NP-complete problems).

4.1 NP optimization problems.

First we look at a class of problems, and next we will discuss algorithms providing approximate solutions thereto.

Recall (from PS0) that an **optimization problem** specifies an objective function for each input x, and then compute the optimal objective value over all assignments y. Thus a maximization problem is $\max_y \mathrm{Obj}_x(y)$, and similarly a minimization problem is $\min_y \mathrm{Obj}_x(y)$. For convenience, we also allow specification of a set S_x of **feasible solutions**, and compute the max or min over $y \in S_x$. This does not change the class of problems, as we could always specify an arbitrarily small objective value for values of y that do not match the constraints of S_x. We require that both the objective function and feasibility be computable in poly-time (in $n = |x|$) for a given y.

4.2 Examples of optimization problems.

- MAX-3SAT: $\max_{y \in \{0,1\}^n} \sum_i \phi_i(y)$, where ϕ_i are the clauses in the formula. That is, the maximum number of clauses satisfiable over all assignments. This problem has no constraints.
- MAX-E3SAT: the same, but with clauses of size exactly 3. This is slightly different with respect to approximation algorithms.
- MAX-qCSP: the same, but the clauses ϕ_i can be arbitrary functions of arity q, i.e. depending on at most q variables. This problem comes up frequently in AI applications. Note that MAX-3SAT is a special case of MAX-3CSP
- MIN-VC or minimal vertex cover, where the input is a graph $G = (V, E)$ and the objective is the smallest vertex cover size: $\min_{S \subseteq V} |S|$ such that $\forall (u, v) \in E, (u \in S) \vee (v \in S)$.
- MAX-IS or maximal independent set, i.e. the largest set such that no two vertices share an edge.
- MIN-TSP: input is a distance matrix D of cities, problem is to minimize total distance traveled over all permutations of cities.

Many **NP**-complete problems have natural optimization versions. Thus, it is natural to consider approximation algorithms computing near-optimal solutions for such problems.

Definition 4 *A* ρ***-approximation algorithm*** *to a minimization problem, for* $\rho \geq 1$*, is an algorithm that on input* x *outputs a solution* y *such that* $\mathrm{Obj}_x(y) \leq \rho \cdot \mathrm{Opt}_x$*, where* Opt_x *is the optimal (minimal) value. For a maximization problem, we require that* $\mathrm{Obj}_x(y) \geq \rho \cdot \mathrm{Opt}_x$ *where* $\rho \leq 1$*.*

Note that Arora–Barak always uses $\rho \geq 1$ so that for a maximization problem one refers to a $1/\rho$-approximation. In context only one makes sense, so one can always switch between the two without confusion.

4.3 Examples of approximation algorithms.

We consider some simple approximation algorithms.

- 7/8-approximation for MAX-E3SAT. Simply output a random assignment. Then the expected number of assignments is $\mathrm{E}_y[\sum_i \phi_i(y)] = \sum_i \Pr[\phi_i(y) = 1] = 7/8 \cdot m \geq 7/8 \cdot \mathrm{Opt}_x$ since only one of eight assignments violates a given clause. This algorithm can be derandomized via a greedy algorithm to always satisfy 7/8 of the clauses.

- 2-approximation for MIN-VC. Greedily find a **maximal** matching M, i.e. a matching such that no other matching is a strict superset thereof, by adding edges until every edge shares a vertex with an already chosen edge, which can be done in one pass. Then the set S of all matched vertices is a vertex cover, since if any edge has no endpoint in S then that edge can be added to the matching. Also, $|S| = 2 \cdot |M| \leq 2 \cdot \mathrm{Opt}_x$, since every edge in the maximal matching must have at least one endpoint in a minimal vertex cover.

Note that all **NP**-complete problems are equivalent for exact solutions, in the sense that an algorithm for one gives an algorithm for another. On the other hand, we'll see that some **NP**-complete optimization problems have no constant approximation, while others have approximations arbitrarily close to 1.

There are many useful approximation algorithms that are less trivial than those described above. We list a couple examples without describing the algorithms in full.

- 0.878... (irrational)-approximation for MAX-CUT, using semidefinite programming.

- $(1+\epsilon)$-approximation for MIN-EUCLIDEAN-TSP, i.e. TSP embedded in Euclidean space, using a divide and conquer approach with dynamic programming.

- $2^{o(n)}$-approximation for shortest vector in an n-dimensional lattice, using the well-known LLL algorithm. This approximation seems weak, but it turns out to be extremely powerful and can be used for factoring polynomials over the integers or breaking certain cryptosystems. There is evidence that no constant-approximation algorithm exists for this problem.

5 PCPs = Inapproximability

5.1 Main result.

At last, we have the machinery to compare **PCP**s with approximation algorithms and get the following result.

Theorem 5 $\mathbf{NP} = \mathbf{PCP}_{c,s}(\log n, q) \Leftrightarrow \textsc{Gap}_{c,s}\textsc{Max}\text{-}q\textsc{CSP}$ *is* **NP***-hard under polynomial-time mapping reductions.*

We need to define $\textsc{Gap}_{c,s}\textsc{Max}\text{-}q\textsc{CSP}$. This will be a promise problem, which is the only reason why we say that it is **NP**-hard rather than **NP**-complete. Alternatively, we could say that it is **prNP**-complete. (**prNP**-hardness is the same **NP**-hardness, since every promise problem in **prNP** reduces to a language in **NP**, namely the set of instances for which there exists a witness that makes the polynomial-time verifier for the problem accept.)

Similar to previous definitions of gap promise problems we have seen (in the context of approximate counting), for an arbitrary maximization problem Π and $\rho \geq 1$, we define

$$\begin{aligned}(\text{GAP}_\rho \Pi)_Y &= \{(x,t) : \text{Opt}_x \geq t\} \\ (\text{GAP}_\rho \Pi)_N &= \{(x,t) : \text{Opt}_x < \rho t\}\end{aligned}$$

It can be shown that having a polynomial-time algorithm that decides $\text{GAP}_\rho \Pi$ is equivalent to having a polynomial-time algorithm that computes a ρ-approximation to the value of Opt_x.

For MAX-qCSP (and analogously MAX-SAT), we also define a version where the thresholds are fixed relative to the number of clauses:

$$\begin{aligned}(\text{GAP}_{c,s}\text{MAX-}q\text{CSP})_Y &= \{\phi : \text{Opt}_\phi \geq c \cdot m\} \\ (\text{GAP}_{c,s}\text{MAX-}q\text{CSP})_N &= \{\phi : \text{Opt}_\phi \leq s \cdot m\},\end{aligned}$$

where m is the number of clauses in ϕ. (A minor technicality is that we have switched from strict inequality to non-strict inequality in the NO instances. This is to allow us to state Theorem 5 cleanly, matching the fact that soundness is defined with non-strict inequality.)

From the definition, we see that $\text{GAP}_{c,s}$MAX-qCSP reduces to GAP_ρMAX-qCSP for any $\rho < c/s$, since one can specify $t = c \cdot m$.

Taking for example the case $c = 1, s = 1/2$, we interpret Theorem 5 as stating that hardness of distinguishing satisfiable from at most 50% satisfiable is equivalent to having a **PCP** characterization for **NP**. Now, the proof.

Proof:

($\Rightarrow$) Given a language $L \in \mathbf{NP}$ with a **PCP** verifier V having completeness c, soundness s, $O(\log n)$ tosses and $q(n)$ queries, we map an instance $x \in L$ to an instance $\phi_x = \{\phi_1, ..., \phi_m\}$ of MAX-qCSP as follows:

- Variables of ϕ_x: bits Π_i of a possible proof oracle Π
- Clauses of ϕ_x: for each sequence r of verifier coin tosses, we have a clause ϕ_r that represents the verifier's acceptance predicate. Specifically, if on coin tosses r, $V(x;r)$ would query the proof oracle at positions $i_1, \ldots, i_q$, then the clause $\phi_r(\Pi_{i_1}, \ldots, \Pi_{i_q})$ accepts an assignment to $\Pi_{i_1}, \ldots, \Pi_{i_q}$ if and only if $V(x;r)$ would accept those responses to its oracle queries.

Since at most $O(\log n)$ coins are tossed, the total number of coin tosses and thus clauses is at most $2^{O(\log n)} \leq poly(n)$. Also, since V reads at most q bits of proof for each toss r, the arity of each clause is at most q. Finally, the total number of variables used is at most $2^{O(\log n)} \cdot q \leq \text{poly}(n)$. It remains to check the completeness and soundness properties, i.e. to verify that there is a gap in number of satisifiable clauses for yes vs. no instances of the original problem.

Completeness: $x \in L \Rightarrow \exists \Pi : \Pr[V^\Pi(x;r) = 1] \geq c$. Then by construction, the same assignment Π to the formula ϕ_x will satisfy at least $c \cdot m$ clauses, namely, the clauses corresponding to coin tosses resulting in V's acceptance.

Soundness: $x \notin L \Rightarrow \forall \Pi, \Pi$ the probability of V^Π accepting is at most s, i.e. any Π satisfies at most $s \cdot m$ clauses of ϕ_x.

($\Leftarrow$) Suppose we have a reduction f from an arbitrary language $L \in \mathbf{NP}$ to $\text{GAP}_{c,s}$MAX-qCSP. Then for a **PCP** proof that $x \in L$, we let the **PCP** proof oracle provide assignments to variables

of $\phi = f(x)$. By nature of the reduction, we know:
$x \in L \Rightarrow \exists$ assignment satisfying at least a c fraction of the clauses of ϕ.
$x \notin L \Rightarrow \forall$ assignment satisfies at most an s fraction of the clauses of ϕ.

Thus if the **PCP** verifier simply flips coins to pick a random clause, asks the proof oracle for the variable assignments in that clause, and checks the clause's value, then exacty the desired gap in acceptance probabilities will hold. Note that this requires $O(\log n)$ coin tosses to pick one of poly clauses, and the arity q of the clause is the number of variable assignment needed and thus the number of oracle bits read. ■

Using this connection, we can restate the **PCP** Theorem as follows:

Theorem 6 (PCP Theorem, restated) *There is a constant q such that* $\text{GAP}_{1,1/2}$MAX-qCSP *is* **NP***-hard under polynomial-time mapping reductions.*

Corollary 7 *There is a constant q such that* MAX-qCSP *has no .51-approximation algorithm.*

5.2 Further Remarks

This gives us a starting point for proving other hardness results for approximation algorithms. Note, however, that standard reductions to do necessarily preserve the approximation factor (as you have seen in the context of counting). Thus, while all **NP**-complete optimization problems are equivalent as far as finding exact solutions, when considering their approximability, they can behave very differently (as we will see).

There is a large and ongoing body of work devoted to figuring out the best approximation factors for different optimization problems. One result is that $\text{GAP}_{1,7/8-\epsilon}$-MAX-3SAT is **NP**-hard.