(67902) Topics in Theory and Complexity Dec 20, 2006

Lecture 8

Lecturer: Irit Dinur *Scribe: Ram Boukobza*

1 PCP Verifiers

In the previous lecture we defined the class NP as the class of languages that have a polynomial time verifier. We then added randomness to the verifier, and allowed it to access only a limited number of bits in the given proof. This led to the definition of an (r, q)-verifier, which we define again here.

Definition 1 ((r, q)-verifier) *An algorithm V is an (r, q)-verifier for a language L if on input strings x and y:*

step 1 V reads input x and draws r random bits $\rho \in \{0,1\}^r$.

step 2 V decides (after a polynomial-time computation in $|x|$) on q indexes: $i_1, \ldots, i_q$ and on a predicate $\varphi : \{0,1\}^q \to \{0,1\}$.

step 3 V outputs $\varphi(y[i_1], \ldots, y[i_q])$.

and the following holds,

1. *(Completeness) If $x \in L$ then $\exists y$ s.t. $Pr_\rho[Ver^y(x, \rho) = 1] = 1$.*
2. *(Soundness) If $x \notin L$ then $\forall y$ $Pr_\rho[Ver^y(x, \rho) = 1] \leq 1/2$.*

Notes:

1. Ver^y is the notation for the fact that that the verifier can access any index in y in one step (see also previous lecture).
2. This is an equivalent definition to the one given in the previous lecture.
3. The above definition is called a *non-adaptive verifier*. An *adaptive verifier* accesses y one bit at a time and may adapt its computation according to the read bit. In our definition V has a fixed predicate after step 2, and simply outputs that predicate in step 3.
4. The length of the proof y is at most $2^r q$ since there are 2^r possible random strings of length r and q possible accesses for each string. Therefore if r is $c \log n (n = |x|)$ for some constant c, the proof y is of length at most $n^c q$, i.e. polynomial in n.

Definition 2 *We define the class $PCP[r, q]$ to be the collection of all languages L for which there is an (r, q)-verifier.*

We use the notation $PCP[O(\log n), O(1)]$. A language $L \in PCP[O(\log n), O(1)]$ if there are constants c and q s.t. L has a $(clogn, q)$-verifer.

Theorem 3 (The PCP Theorem) $NP = PCP_{1,\frac{1}{2}}[O(\log n), O(1)]$

The fact that $NP \supseteq PCP[O(\log n), O(1)]$ is not hard to see. Given an $(O(\log n), O(1))$-verifer V for a language L, the NP Verifer V' runs V on all the possible $O(\log n)$ random strings. V' accepts if V accepted on all runs and rejects otherwise. V' is still polynomial because there are a total of n^c runs for some constant c. Note that the PCP witness (proof) is also the NP witness and that its length is polynomial in the input length, as required (note 4 above).

The other direction $NP \subseteq PCP[O(\log n), O(1)]$ is much more difficult. It states that given an NP verifier for a language L, there exists a PCP verifier which uses only a constant number of bits in the proof. More concretely, in the Hamiltonian Cycle problem, the NP witness for the NP Verifier is a Hamiltonian cycle. If we try to modify the NP Verifier to read only a constant number of vertexes in a potential witness (a path in the graph), then obviously the NP verifier may randomly choose the "wrong" ones and if the graph is only "almost" Hamiltonian the probability of detecting this is very very small (certainly not a constant).

An interesting question is what is the smallest constant number of bits for which the theorem is correct? The answer is the following theorem (given without proof), which states that for a soundness of 1/2 only 3 bits are required.

Theorem 4 $NP = PCP_{1,\frac{1}{2}}[O(\log n), 3]$

Another interesting question is the length of the PCP witness, which we would like to be as short as possible. As noted above, an upper bound on the length of the witness is $n^c q$. There is a theorem which shows that c is $1 + o(1)$ (although not necessarily with q being 3).

2 Optimization Problems

Informally, a combinatorial optimization problem is a problem of picking the "best" solution from a finite set, where "best" refers to a maximum or minimum of some function over the set of possible solutions to the problem. Some well-known examples follow:

- Max-Clique - Given a graph find the largest clique in the graph.
- Max-Knapsack - Given n objects with weights $\{a_1, \ldots, a_n\}$ and corresponding prices $\{c_1, \ldots, c_n\}$ find a set $I \subseteq [n]$ s.t. $\sum_{i \in I} a_i \leq 1$ and that maximizes $\sum_{i \in I} c_i$.
- Max-Setcover - Given universe U and subsets $S_1, \ldots, S_n \subseteq U$ find the smallest $I \subseteq [n]$ s.t. $\bigcup_{i \in I} S_i \supset U$
- Max-SAT - Given a CNF formula find an assignment that satisfies the maximal number of clauses.
- Max-CSP. Max-Constraint Satisfaction Problem. This problem will be useful later, so we give a full definition below.

Definition 5 (Constraint) *Let Σ, the alphabet, be a finite set, and let $V = \{v_1, \ldots, v_n\}$ be a set of variables. A q-ary constraint is a tuple $(i_1, \ldots, i_q, \varphi)$ such that $\forall j \in \{1, \ldots, q\}$ $i_j \in [n]$ and $\varphi : \Sigma^q \to \{0, 1\}$ is a Boolean predicate. A constraint $(i_1, \ldots, i_q, \varphi)$ is satisfied by an assignment $a : V \to \Sigma$ if and only if $\varphi(a(v_{i_1}), \ldots, a(v_{i_q})) = 1$.*

Definition 6 *Let Σ be a finite alphabet, $V = \{v_1, \ldots, v_n\}$ a set of variables, $C = \{c_1, \ldots, c_m\}$ a set of q-ary constraints and $a : V \to \Sigma$ an assignment. Define:*

$$\text{sat}_a(C) = Prob_{i \in [m]}[a \text{ satisfies } c_i]$$

which is the fraction of satisfied constraints under a.

$$\text{sat}(C) = max_{a:V \to \Sigma} \text{sat}_a(C)$$

which is the maximum fraction of constraints satisfied under any assignment.

Definition 7 (Max-CSP) *Given as input an alphabet Σ, a set of variables $V = \{v_1, \ldots, v_n\}$, and a set of q-ary constraints $C = \{c_1, \ldots, c_m\}$, find an assignment $a : V \to \Sigma$ that satisfies the maximum number of constraints (i.e. that achieves $\text{sat}_a(C) = \text{sat}(C)$).*

A couple of well-known examples of Max-CSP are:

- Max-3SAT

  Max-3SAT is the same as Max-SAT except the clauses all have exactly 3 literals.

  To see this as a Max-CSP problem let V be the existing variables, $\Sigma = \{0, 1\}$, $q = 3$ and the set of triary constraints is the set of clauses. Each clause can be viewed as a constraint on 3 variables. For example, the clause $x_i \vee \overline{x_j} \vee x_k$ is a constraint (i, j, k, φ) and φ is the predicate using the normal logic rules which is satisfied by all assignments to x_i, x_j, x_k except $(0, 1, 0)$ (Note that all constraints are satisfied by 7 of the 8 possible assignments). Maximizing the number of satisfied constraints is then the same as maximizing the number of satisfied clauses.

- Max-3COL

  A graph $G = (V, E)$ is 3-colorable if each $v \in V$ can be assigned one of 3 colors s.t. for each $e = (u, v) \in E, color(u) \neq color(v)$.

  Max-3COL - Given a graph G, find a 3-coloring of the vertexes that maximizes the number of properly colored edges (not monochromatic).

  To see this as a Max-CSP problem let V be the set of vertexes in G, $\Sigma = \{1, 2, 3\}$ (the 3 colors), $q = 2$ and the set of binary constraints forces the edges of the graph to be non-monochromatic. So, (i, j, φ) is a constraint if (v_i, v_j) is an edge in the graph and $\varphi : (1, 2, 3)^2 \to \{0, 1\}$ s.t. $\varphi(a, b) = 1$ iff $a \neq b$.

As a consequence of the above two examples we have that Max-CSP is NP-hard, since Max-3COL (which is MAX-CSP with $|\Sigma| = 3$ and $q = 2$) and Max-3SAT($|\Sigma| = 2$, $q = 3$) are NP-hard.

3 Hardness of Approximation

One approach to dealing with NP-hardness is approximation. We will see later that this is closely linked to the PCP theorem.

Definition 8 (Approximation Algorithm) *Let $\alpha \geq 1$. An algorithm A is an α-approximation for an optimization problem O if for every instance x of O with optimum $OPT(x)$:*

$$\frac{1}{\alpha} OPT(x) \leq A(x) \leq OPT(x)$$

where O is a maximization problem, and

$$OPT(x) \leq A(x) \leq \alpha OPT(x)$$

where O is a minimization problem.

Note that α may be a function of the input length ($\alpha = \alpha(|x|)$).

We mention some approximation results for some well known problems:

- Min-Setcover has an $O(lnn)$-approximation (the greedy algorithm).
- Max-3SAT has an $\frac{8}{7}$-approximation.
- Max-Knapsack has a $(1 + \epsilon)$-approximation for any $\epsilon > 0$.
- Max-Clique has an $\frac{n}{(\log n)^2}$-approximation.

Historically, as various approximation results for NP-hard problems were obtained, it was not clear why some problems had quite good approximations while others had bad ones, in other words why was it apparently "difficult" to approximate some problems, but "easier" to approximate others.

An illustrative example is the Max-Independent-Set and Min-Vertex-Cover problems. Max-Independent-Set is the problem of finding the maximum size subset of vertexes in a graph such that each edge has *at most one* vertex in the set. Min-Vertex-Cover is the problem of finding a minimum size subset of vertices in a graph such that each edge has *at least one* vertex in the set. Observe that a set $S \subset V(G)$ which is an optimal solution for one of the problems easily gives an optimal for the second, namely $V(G) - S$. But, Min-Vertex-Cover has a 2-approximation while Max-Independent-Set only has an $\frac{n}{poly(\log n)}$-approximation.

The PCP theorem helped to make some sense of it all - by providing a tool to prove hardness of approximation for the above problems. How would one prove hardness of approximation? As with the case of NP, one must formulate a decision version of the problem. In this case, it is the gap version. Rather than providing a general definition, let us focus on the example of max-CSP.

Definition 9 (The gap-CSP Problem) *gap-$CSP_s(q, \Sigma)$ is the following problem: Given Σ, V, C as above and a soundness parameter $0 < s < 1$ decide between the following 2 possibilities:*

- $\text{sat}(C) = 1$
- $\text{sat}(C) \leq s$

Definition 10 *An algorithm A is said to* solve *a gap-CSP_s problem C if the following holds:*

- *If* $\text{sat}(C) = 1$ *then* $A(C) = 1$,
- *If* $\text{sat}(C) \leq s$ *then* $A(C) = 0$, *and*
- *If* $s < \text{sat}(C) < 1$ *then A is allowed any output.*

Definition 11 (NP-hardness for gap-CSP) *gap-CSP is NP-hard if for any language L in NP there exists a polynomial-time reduction mapping input x for L to input C for gap-CSP and satisfying:*

- *If* $x \in L$ *then* $\text{sat}(C) = 1$,
- *If* $x \notin L$ *then* $\text{sat}(C) \leq s$

Note that this reduction maps any x to gap-CSP with a "gap" - no x's are mapped into this gap. This will be critical in proving hardness of approximation.
Additionaly, although the above definitions refer specifically to gap-CSP, it is possible to consider similar definitions for clique, SAT (gap-Clique or gap-SAT) or other NP-hard problems.

Claim 12 *If $gap - CSP_s(q, \Sigma)$ is NP-hard then there is no $\frac{1}{s}$-approximation for $Max - CSP(q, \Sigma)$ unless P=NP*

Proof If $gap - CSP_s(q, \Sigma)$ is NP-hard then there is a polynomial reduction mapping any SAT formula Φ (since SAT $\in$ NP) to the constraint system C_Φ for the $gap - CSP_s$ problem and this mapping satisies the following:

- $\Phi \in SAT$ then $\text{sat}(C_\Phi) = 1$
- $\Phi \notin SAT$ then $\text{sat}(C_\Phi) < s$

Now, if $\text{sat}(C_\Phi) = 1$ then $OPT(C_\Phi)$ (for $Max - CSP(C_\Phi)$) is an assignment that satisfies all of the constraints, and a $\frac{1}{s}$-approximation algorithm, will find an assignment satisfying at least sm constraints (where m is the number of constraints), i.e. at least an s-fraction of the constraints. On the other hand if $\text{sat}(C_\Phi) < s$ then $OPT(C_\Phi)$ is an assignment that satisfies less than sm constraints, i.e. less than an s-fraction of the constraints.

So an algorithm to decide whether $\Phi \in SAT$, runs the polynomial reduction to gap-CSP and then runs the approximation algorithm to obtain an assignment and decides as follows:

If the given assignment satisifies less than an s-fraction of the constraints decide $\Phi \notin SAT$ (because if $\Phi \in SAT$ at least an s-fraction of the constraints would be satisfied), and otherwise decide $\Phi \in SAT$ (because if $\Phi \notin SAT$ less than an s-fraction of the constraints would be satisfied). Thus we can decide SAT and P=NP. ■

We note that this result would work with any gap-problem.

In conclusion, to prove hardness of approximation for a problem A, we need to find a polynomial gap-creating reduction from some NP-hard problem, and the size of the gap (i.e. the s parameter) determines the hardness of approximation.

4 The PCP theorem and gap-CSP NP-hardness

In this section we prove a theorem which will be useful in proving the PCP-theorem.

Theorem 13 (PCP theorem is equivalent to gap-CSP NP-hardness) *The following claims are equivalent:*

1. $NP = PCP_{1,\frac{1}{2}}[O(\log n), O(1)]$.
2. *There exist constants* q, $0 < s < 1$, *and* $|\Sigma|$ *such that* $gap - CSP_s(q, \Sigma)$ *is NP-hard.*

Proof (2) $\Longrightarrow$ (1)

The proof is a bit technical, so we first describe the idea, which is quite simple.

Given a language L $\in$ NP, (2) states that it is possible to map and instance x for the language L to an instance C for the gap-CSP problem such that the mapping has a gap s. We build a PCP Verfier for L (with soundness s) as follows: given an instance x map it to C and then for a potential assignment y, select one of the constraints of C randomly, accept if that constraint is satisfied by y, and reject otherwise.

If $x \in L$, the gap reduction guarantees there is an assignment y satisfying all constraints, so for that y the PCP-verifier will always accept. If, on the other hand, $x \notin L$, the gap reduction guarantees at most an s-fraction of the constraints are satisfied, so for any y the PCP verifier accepts with probability at most s.

Finally, to achieve a soundness of 1/2, the above algorithm is repeated a sufficient number of times and the PCP-verifier accepts iff the algorithm accepted on all runs.

Now more formally,

From (2) there is a polynomial reduction from any $L \in$ NP, and specifically SAT, to $gap-CSP_s(q, \Sigma)$ for some q, $0 < s < 1$, and Σ which maps an instance Φ to an instance C_Φ and satisfies:

- $\Phi \in SAT$ then $\text{sat}(C_\Phi) = 1$
- $\Phi \notin SAT$ then $\text{sat}(C_\Phi) < s$

We note that in order to prove (1), it is sufficient to find a PCP Verifier for SAT. This is because an instance $x \in L$ for any $L \in$ NP can be reduced (polynomially) to $\Phi \in$ SAT, such that $x \in L$ iff $\Phi \in$ SAT. So a PCP Verifier for L will first apply the reduction to SAT and then use the PCP Verifier for SAT.

Because the first reduction is polynomial in $|x|$, Φ is polynomial in $|x|$ as well, and so the PCP Verifier will use only $O(\log|x|)$ random bits as required.

The PCP Verifier V for SAT is defined as follows:

step 1 On input Φ and proof y, V reads Φ and computes the reduction to produce C_Φ.

Let m be the number of constraints in C_Φ. Let Σ be the alphabet of C_Φ, and denote $l = \lceil \log|\Sigma| \rceil$ as the length of a letter of Σ encoded in binary.

step 2 V draws $\lceil \log m \rceil$ random bits ($\rho \in \{0,1\}^{\lceil \log m \rceil}$) and uses them to choose some $i \in \{1,\ldots,m\}$.

step 3 Let $C_i = (i_1,\ldots,i_q,\varphi)$ be the i'th constraint. And let $y[j]$ be the letter (of Σ) that is binary coded using the bits $j(l+1)\ldots j(l+l)$.

V outputs according to the i'th constraint. That is, V reads the bits corresponding to $y[i_1],\ldots,y[i_q]$ and then outputs $\varphi(y[i_1],\ldots,y[i_q])$.

First we verify that V is polynomial and uses $O(\log(|\Phi|))$ random bits and $O(1)$ accesses to the proof. V runs in polynomial time in $|\Phi|$ because the gap-reduction is polynomial, and so is computing the predicate of the constraint in step 3. (Note that access to y is assumed to cost one-step as in the $(r,q)-$verifier definition).

Because the gap-reduction is polynomial in $|\Phi|$, $m \leq poly(|\Phi|)$, and hence $\lceil logm \rceil$ random bits are $O(\log(|\Phi|))$ random bits. The number of accesses to the proof y is lq which is constant as well.

Now, we verify the completeness and soundness conditions of V.

Because the proof y is an encoding of some assignment a to the variables of C_Φ, it holds that:

If $\Phi \in SAT$ then $\exists y$ s.t. $Prob_\rho[Ver^y(\Phi,\rho)=1]=1$ (y is the encoding of the assignment a which satisfies all of the constraints). Also,

$\Phi \notin SAT$ then $\forall y\ Prob_\rho[Ver^y(\Phi,\rho)=1] \leq s$ (since for any assignment a at most an s-fraction of the constraints are satisfied).

Thus V is a $(O(\log(|\Phi|),O(1))$-verifier with soundness s. To achieve soundness $1/2$, run V $\lceil \log_s \frac{1}{2} \rceil$ times, and output accept if V outputs accept on all runs, reject otherwise. It is not hard to see that this algorithm is correct in terms of completeness and soundess$(1/2)$ and also maintains the $(O(\log(|\Phi|),O(1))$ requirements since they are both multiplied by the constant $\lceil \log_s \frac{1}{2} \rceil$.

(1) $\Longrightarrow$ (2)

NP $\subseteq PCP[O(\log n),O(1)]$ so specifically 3-COL has an $(O(\log(n),O(1))$-verifier Ver.

We show a polynomial reduction from 3-COL to gap-CSP$_s(q,\Sigma)$ (for some specific parameters s,q and Σ). This will show that any language $L \in$ NP can be reduced to $gap-CSP_s(q,\Sigma)$ because 3-COL is NP-hard.

Ver reads the input graph G, draws $r = O(\log n)$ random bits (string ρ), computes $\{i_1,\ldots,i_q\}$ indices and a predicate $\varphi : \{0,1\}^q \to \{0,1\}$, and outputs $\varphi(y[i_1],\ldots,y[i_q])$.

The constraint system C_G is defined as follows: $\Sigma = \{0,1\}$, create a boolean variable for each possible index in y that Ver may access, and create a constraint $C_\rho = \{i_1,\ldots,i_q,\varphi\}$ for each random string ρ (where $\{i_1,\ldots,i_q\}$ are the indices and φ is the predicate computed by Ver).

The number of constraints is $2^r = poly(n)$ because $r = O(\log n)$, and the number of variables is at most $O(1)poly(n)$, so the reduction is polynomial and obviously deterministic. Now, if G $\in$ 3COL then there exists a proof y such that $Prob_\rho[Ver^y(G,\rho)=1]=1$, so y is a proof such that for all random strings ρ the predicate over the indices accessed in y is satisfied. Now because the reduction of G to C_G simply transforms these predicates to constraints, one for each random string, all the constraints can be satisfied by the assignment matching y so $\text{sat}(C_G)=1$. If, on the other hand $G \notin$ 3COL for any proof y $Prob_\rho[Ver^y(G,\rho)=1] \leq 1/2$, so at most half of the possible predicates are satisfied and hence at most half of the constraints will be satisfied in C_G so $\text{sat}(C_G) \leq 1/2$.

So we have a reduction from 3-COL to $gap-CSP_s(q,\Sigma)$ for $s=1/2$, $q=O(1)$ and $\Sigma=\{0,1\}$. ■

Conlcusion: The following statements are equivalent:

(i) $NP \subseteq PCP[O(\log n), O(1)]$

(ii) There exist $0 < s < 1$, q, and Σ such that $gap - CSP_s(q, \Sigma)$ is NP-hard.

(iii) gap-3SAT is NP-hard (for some soundness parmeter $0 < s' < 1$).

Note: items (ii) and (iii) are not difficult (possibly to be proved in the exercise).