Lecture 21: April 1

Lecturer: Mohammad R. Salavatipour *Scribe: Xiaozhen Niu*

21.1 PCP Theorem

To be consistent with the PCP definition later on, we give a slightly modified definition for class NP which is clearly equivalent to the original one:

Definition 21.1 *A language L is $\in NP$ if and only if there is a deterministic polynomial time verifier (i.e. algorithm) V that takes an input x and a proof y with $|y| = |x|^c$ for a constant $c > 0$ and it satisfies the following:*

- *Completeness: if $x \in L \Rightarrow \exists\ y$ such that $V(x, y) = 1$.*
- *Soundness: if $x \notin L \Rightarrow \forall y$, $V(x, y) = 0$.*

Now we define probabilistic verifiers that are restricted to look at (query about) only a few bits of the alleged proof y and make their decision based on those few bits instead of reading the whole proof.

Definition 21.2 *A $(r(n), b(n))$-restricted verifier is a randomized verifier that uses at most $r(n)$ random bits. It runs in probabilistic polynomial time and reads/queries at most $b(n)$ bits of the proof.*

Finally we provide the main definition of a language $\in PCP$. This definition is for $PCP_{1,\frac{1}{2}}(r(n), b(n))$.

Definition 21.3 *A language L is $\in PCP(r(n), b(n))$ if and only if there is a $(r(n), b(n))$-restricted verifier V such that given an input x with length $|x| = n$ and a proof π, it satisfies the following:*

- *Completeness: if $x \in L \Rightarrow \exists$ a proof π such that $Pr[V(x, \pi) = 1] = 1$.*
- *Soundness: if $x \notin L \Rightarrow \forall \pi$, $Pr[V(x, \pi) = 1] \leq \frac{1}{2}$.*

The probabilities in completeness and soundness given in definition above are 1 and $\frac{1}{2}$, respectively. A more general definition of PCP languages is by allowing these probabilities be some other constant values:

Definition 21.4 *For $0 \leq s < c \leq 1$, a language L is $\in PCP_{c,s}(r(n), b(n))$ if and only if there is a $(r(n), b(n))$-restricted verifier V such that given an input x with length $|x| = n$ and a proof π, it satisfies the following:*

- *Completeness: if $x \in L \Rightarrow \exists$ a proof π such that $Pr[V(x, \pi) = 1] \geq C$.*
- *Soundness: if $x \notin L \Rightarrow \forall \pi$, $Pr[V(x, \pi) = 1] \leq S$.*

In the more general definition of PCP language, we need to have the following restrictions on parameters:

- c and s are constants and $0 \leq S < C \leq 1$. This is to make sure the verifier can give a correct answer with higher probability than a wrong answer.
- $r(n)$ and $b(n)$ are at most polynomial, this is to make sure the verifier runs in polynomial.
- Proofs are at most $2^{r(n)}$ bits long. The reason is that the verifier V uses at most $r(n)$ random bits, so it can access at most $2^{r(n)}$ positions, and it must be able access to any position of the proof.

Now we give the first lemma about PCP:

Lemma 21.5 $PCP_{c,s}(O(\log n, n^{O(1)}) \subseteq NP$

Proof: Let L be a language in $PCP_{c,s}(O(\log n, n^{O(1)})$ with a verifier V. We construct a non-deterministic polytime Turing machine M for L. Starting with an input x, M guesses a proof π and simulates V on all $2^{O(\lg n)} = n^{O(1)}$ possible random bits. M accepts the proof π if at least a fraction c of all these runs accept, rejects otherwise. Thus:

- if $x \in L \Rightarrow V(x, \pi)$ accepts with $Prob \geq C \Rightarrow$ at least a fraction c of random bits cause the verifier V accept $\Rightarrow M$ accepts.
- if $x \notin L \Rightarrow$ the verifier accepts with $Prob \leq s < c \Rightarrow$ for only a fraction $< c$ of random bits the verifier V accepts $\Rightarrow M$ rejects.

Since there are $O(n^{O(1)})$ random bits and each simulation takes polytime, the running time of M is polytime. Therefore we get $PCP_{C,S}(O(\log n, n^{O(1)}) \subseteq NP$, and finished the proof. ■

A trivial observation about PCP is that if we do not read any random bits then it can be written as $PCP_{c,s}(0, n^{O(1)})$, this is just the same definition as NP. Therefore we get $NP \subseteq PCP_{c,s}(0, n^{O(1)})$. Combined with the lemma we just proved, we conclude that $PCP_{c,s}(O(\log n, n^{O(1)}) = NP$.

The remarkable PCP theorem, which is the least obvious (and probably the most difficult) result in computer science, proved by Arora and Safra [arora1] and Arora, Lund, Motwani, Sudan, and Szegedy [arora2] states:

Theorem 21.6 (PCP Theorem) $NP = PCP_{1,\frac{1}{2}}(O(\log n, O(1))$

Basically, this miraculas theorem says that for every problem in NP there is a verifier that queries only a constant number of bits of the proof (regardless of the length of the proof) and with sufficiently high probability gives a correct answer whether the proof is correct or not.

21.2 Hardness of MAX-3SAT

Starting from the PCP theorem, we show that approximating MAX-3SAT within some constant factor is NP-hard.

Theorem 21.7 *For some absolute constant $\epsilon > 0$, there is a gap-introducing reduction from SAT to MAX-3SAT such that it transforms a boolean formula ϕ for SAT to a boolean formula ψ with m clauses for MAX-3SAT such that:*

- *if ϕ is satisfiable, then $OPT(\psi) = m$.*
- *if ϕ is a NO-instance, then $OPT(\psi) \leq (1-\epsilon)m$.*

Corollary 21.8 *Approximating MAX-3SAT with a factor better then $(1-\epsilon)$ is NP-hard for some constant $\epsilon > 0$.*

Proof of Theorem 21.7: Since SAT is a NP problem, by PCP theorem, we know that it has a $PCP_{1,\frac{1}{2}}(O(\log n, n^{O(1)}$ verifier V. Let us assume that it is $PCP_{1,\frac{1}{2}}(d\log n, k)$ where d and k are some constants.

Let $r_1, \ldots, r_{n^d}$ be all the possible random bits (of length $d\log n$) that can be given as seed to verifier V. We will construct a formula f_i for every possible random bit r_i. Thus we will have formulas $f_1, \ldots, f_{n^d}$.

For any particular choice of random bits, the verifier can be considered to evaluate a boolean binary decision tree of height at most k. This tree contains at most 2^k variables and therefore, the total number of variables that we will have over all boolean formulas $f_1, \ldots, f_{n^d}$ will be $2^k n^d$. These variables correspond to the set of all possible positions that the verifier may read. We call this set of possible positions B.

This decision tree can be encoded as a boolean formula with at most 2^k variables and 2^k clauses each of length k. Think of every leaf is a variable and every path from root to leaf forms a clause.

Figure 1 shows an example. Here suppose $k = 2$ and we have a fixed random bit string. Based on the first random bit the position we read from the proof is x_j, if it returns 0 we get the second random bit and based on that we read position x_k, else if x_j was 1 we read position x_l. So we can use four variables $\overline{x_j}, \overline{x_k}, x_j, x_l$ to form a formula encoding this tree: $(\overline{x_j} \wedge \overline{x_k}) \vee (x_j \wedge x_l)$. It is easy to see that the formula is satisfied if and only if the path that the verifier traverses on the tree ends at an "accept" leaf. Any truth assignment to the variables i.e. any proof, will give a unique path for each decision tree. If for a fixed random bit string and a proof (truth assignment) the path ends in an "accept" it means that the verifier accepts the proof, otherwise it rejects the proof.

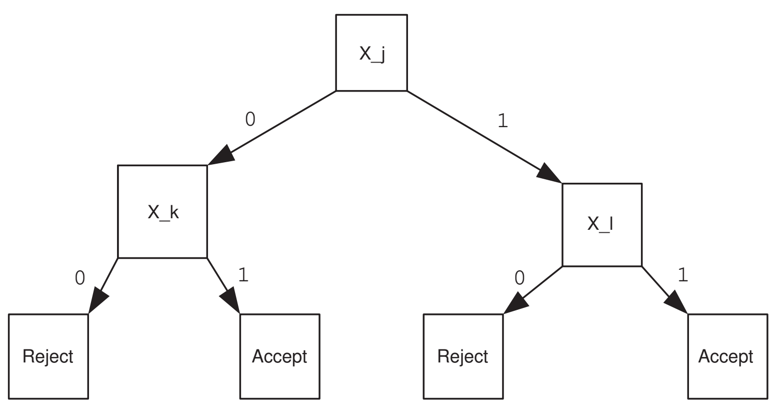

Figure 21.1: Example of decision tree

Alternately we can say that acceptance/rejection of V is a function of ϕ, r and the k bits V reads. Given ϕ, for any r_i we can consider the restriction of this function to the k bits of the proof. This is our function f_i and it can be computed in polynomial time.

If ϕ is a YES-instance, $\Rightarrow$ there is a truth assignment that works/accpets with probability of 1 (i.e., for any random bits it will accept) $\Rightarrow$ the corresponding truth assignment will give a path from root to an "accept" leaf in every decision tree (corresponding to a random bit string); so it satisfies all formulas $f_1, \ldots, f_{n^d}$.

If ϕ is a NO-instance $\Rightarrow$ for any proof (truth assignment) V accepts with probability $\leq \frac{1}{2}$ (this is from the PCP definition) $\Rightarrow$ for at least half of the decision trees, the truth assignment will give a root to leaf path that ends in "reject", i.e. the formula is not satisfied. Therefore, among all n^d formulas, at least $\frac{n^d}{2}$ of them are not satisfied.

Now on we can transform all formulas $f_1, \ldots, f_{n^d}$ into 3-CNF formulas $f'_1, \ldots, f'_{n^d}$ such that that f'_i is satisfiable if and only if f_i is. During the transformation (it is polynomial time) there will be some new disjoint variables created for each formula, however the number of these new variables is polynomial-bounded. In addition, the size of f'_i is at most k times of the size of f_i. This transformation is basically the Karp reduction used to prove NP-completeness of 3SAT from SAT and can be found in any standard text book (e.g. see the text book) It has the property that if f_i is satisfied then all the 3-clauses of f'_i are satisfied and if f_i is not satisfied then at least one 3-clause of f'_i is not satisfied.

Let $F = \bigcup_{i=1}^{n^d} f'_i$, the size of F (total number of clauses) is at most $k2^k \times n^d$. So the number of clauses that are not satisfiable in F (if ϕ is a No instance) is at least $\frac{n^d}{2}$; the ratio of the number of clauses that are not satisfied over the total number of clauses is $\frac{n^d/2}{k2^k \times n^d} = \frac{1}{k \times 2^{k+1}}$ then at most a $(1 - \frac{1}{k \times 2^{k+1}})$ fraction of F is satisfied. ■

This theorem shows how to derive a gap-introducing reduction from SAT to Max-3SAT. Interestingly, we can also derive the PCP theorem assuming the existence of such a reduction. The basic idea was explained in the previous lecture. There we showed how to derive a version of PCP theorem, assuming a gap-introducing reduction from SAT to Max-3SAT. These two results show that PCP theorem is equivalent to saying that Max-3SAT is not approximable within some constant factor > 1 unless P=NP.

References

[1] S. Arora and S. Safra, *Probabilistic checking of proofs: a new characterization of NP*, J. ACM, 45(3):501-555, 1998. Earlier version in Proc. of IEEE FOCS 1992, pp 2-12.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, *Proof verification and intractability of approximation problems*, J. ACM, 45(1):70-122, 1998. Earlier version in Proc. IEEE FOCS 1992, pp 13-22.