CMPUT 675: Computational Complexity Theory **Winter 2019**

Lecture 14 (Feb 28): Probabilistically Checkable Proofs (PCP)

Lecturer: Zachary Friggstad *Scribe: Haozhou Pang*

14.1 Motivation: Some problems and their approximation algorithms

Many optimization problems are NP-hard, and therefore it is unlikely to efficiently find optimal solutions. However, in some situations, finding a provably good-enough solution (approximation of the optimal) is also acceptable. We say an algorithm is an α-approximation algorithm for an optimization problem iff for every instance of the problem it can find a solution whose cost is a factor of α of the optimum solution cost. Here are some selected problems and their approximation algorithms:

Example: Min Vertex Cover is the problem that given a graph $G = (V, E)$, the goal is to find a vertex cover $C \subseteq V$ of minimum size. The following algorithm gives a 2-approximation for Min Vertex Cover:

Algorithm 1 Min Vertex Cover Algorithm

Input: Graph $G = (V, E)$
Output: a vertex cover C of G.

```
C ← ∅
while some edge (u, v) has u, v ∉ C do
    C ← C ∪ {u, v}
end while
return C
```

Claim 1 *Let C^* be an optimal solution, the returned solution C satisfies $|C| \leq 2|C^*|$.*

Proof. Let M be the edges considered by the algorithm in the loop, we have $|C| = 2|M|$. Also, $|C^*|$ covers M and no two edges in M share an endpoint (M is a matching), so $|C^*| \geq |M|$. Therefore, $|C| = 2|M| \leq 2|C^*|$. ■

Example: Max SAT is the problem that given a CNF formula, the goal is to find a assignment that satisfies as many clauses as possible. Let ϕ be a CNF with m clauses, a simple algorithm would be:

- Set all variables to T
- Set all variables to F

And output the better assignment of these two.

Claim 2 *This satisfies at least $\frac{m}{2}$ clauses.*

Proof. Since each clause is satisfied by at least one of the two assignments, so the better solution satisfies $\geq \frac{m}{2}$ clauses. ■

Example: MAX E3SAT is the problem that given a CNF formula that each clause has exactly 3 literals, the goal is to find a assignment that satisfies as many clauses as possible. One effective algorithm is just simply returning a random assignment. Since for each clause C, $\Pr[C \text{ is satisfied}] = \frac{7}{8}$, so we have $\mathbb{E}[\#\text{ of satisfied clauses}] = \frac{7}{8}m$.

Some NP-hard optimization problems can be approximated very well. We will not discuss the algorithm in the next example, but it is quite standard.

Example: KNAPSACK is the problem that given a set of items, each with a weight and a profit. The goal is to determine the number of each item to include to maximize the total profit but within the capacity limit. For all $\epsilon > 0$, there is a $O(\frac{n^2}{\epsilon})$-time algorithm that produce a $(1-\epsilon)$ approximation via dynamic programming (not covered in this class).

The approximation algorithms discussed above give an upper-bound on the produced solution for the corresponding problems, but what about the lower-bound? For example, it's know that a random assignment yields a $\frac{7}{8}$-approximation algorithm for MAX E3SAT, how hard is it to get a $(\frac{7}{8} + 0.001)$-approximation? In the case of KNAPSACK, we can get within any constant factor of the optimum solution in polynomial time: does MAX E3SAT admit such approximations? To study this kind of question, we will need an alternative characterization of **NP**.

14.2 Probabilistically Checkable Proofs (PCPs)

Our starting point to study how hard it is to even approximation some optimization problems begins with a different notion of what it means to verify a proof using randomness. Previously, we studied interactive proofs where a randomized verifier can interact with a prover over multiple rounds. Here, we consider what happens when we allow the verifier to be randomized but the proof is presented "**NP**-style": the entire proof is presented to the verifier. The limitation is in how the verifier can examine this proof. The number of bits of the proof the verifier is allowed to look at is limited.

Definition 1 *An $(r(n), q(n))$-PCP verifier V for a language L is a probabilistic T.M. that expects 2 inputs:*

- *x: the instance, to be decided $\in L$ or not.*
- *π: the 'proof'*

with the following behaviours

- *V always runs in poly($|x|$) time*
- *V expects π to have length $\leq q(|x|) \cdot 2^{r(|x|)}$*
- *V flips $r(|x|)$ random bits before looking at π*
- *V then queries π at $q(|x|)$ locations non-adaptively [1] with random access.*
- *Finally V accepts or rejects.*

[1] 'Non-adaptively' means the bits queried by V are chosen before any are examined: the next bit to be queried cannot depend on the values of previous queries.

So a verifier is not a traditional Turing Machine as a TM would scan the proof tape sequentially. Ultimately, the point is that determining if the verifier accepts or rejects is only a function of x, the $r(|x|)$ random bits read, and only the $q(|x|)$ bits that are queried.

As usual, we use notation like saying a verifier is a $(O(r(n)), O(q(n)))$-PCP verifier to mean the number of random bits queried is a function in $O(r(n))$ and the number of proof bits queried is a function in $O(q(n))$.

Definition 2 *A language $L \in$ PCP(r(n), q(n)) if and only if there is a $(r(n), q(n))$-PCP verifier V such that given an input x with length $|x| = n$ and a proposed proof π for x, we have:*

- *(Completeness): if $x \in L$, then $\exists\ \pi$ s.t. Pr[V(x,π) = accept] = 1*
- *(Soundness): if $x \notin L$, then $\forall\ \pi$ s.t. Pr[V(x,π) = accept] $\leq \frac{1}{2}$*

Some remarks:

- $\mathbf{P} = \mathbf{PCP}(0, 0)$.
  We don't need any help from a proof to decide if a language is in **P**.
- $\mathbf{NP} = \mathbf{PCP}(0, poly(n))$.
  This holds immediately from the definition because a deterministic verifier V requires zero random bits and queries at most a polynomial number of bits of the proof.
- $\mathbf{PCP}(r(n), q(n)) \subseteq \mathbf{NTIME}(q(n)2^{O(r(n))})$.
  Given an (r(n),q(n))-PCP verifier V for L, instead of performing a PCP verification, a nondeterministic machine could guess the proof π in $q(n)2^{O(r(n))}$ time and verify it by trying all $2^{r(n)}$ possible random bit strings and simulate $V(x, \pi)$ on each to decide if it always accepts or sometimes rejects.
- $\mathbf{PCP}((O(\log n), O(1)) \subseteq \mathbf{NP}$.
  This holds since $\mathbf{PCP}(O(\log n), O(1)) \subseteq \mathbf{NTIME}(2^{O(\log n)}) = \mathbf{NP}$

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

The proof of PCP theorem is difficult and omitted for the moment. We will be discussing it later in the term. For now, we explore its consequences.

The Cook/Levin reduction provides a Karp reduction from any $L \in \mathbf{NP}$ to SAT. We first consider such reductions that introduce a "gap": what if in **no** instances we could only satisfy 90% of the clauses with any truth assignment? The following shows this implies the PCP Theorem.

Lemma 1 *Suppose there exists a Karp reduction f from* SAT *to* E3SAT *s.t. if $\phi \notin$ SAT then any assignment to $f(\phi)$ satisfies at most 0.9-fraction of the clauses of $f(\phi)$. Then $\boldsymbol{PCP}(O(\log n), O(1)) = \boldsymbol{NP}$.*

Proof. Let $L \in \mathbf{NP}$ and let $\overline{\phi}$ be the E3SAT instance obtained by $L \xrightarrow{Cook/Levin}$ SAT $\xrightarrow{f}$ E3SAT, then we can construct an $(O(\log n), O(1))$-PCP verifier V s.t.:

- V expects π to be an assignment to $\overline{\phi}$
- sample a random clause using $(O(\log n)$ random bits

- accept iff the three bits of π corresponding to the variables of $\overline{\phi}$ satisfy the clauses.

Note that if we repeat the process, say, 10 times in parallel $\Rightarrow \Pr[\text{accept}] \leq 0.9^{10} \leq \frac{1}{2}$ if $x \notin L$, otherwise, if $x \in L$ then V always accepts. So $\mathbf{NP} \subseteq \mathbf{PCP}(O(\log n), O(1))$, and therefore $\mathbf{NP} = \mathbf{PCP}(O(\log n), O(1))$ as we have shown $\mathbf{PCP}(O(\log n), O(1)) \subseteq \mathbf{NP}$.

■

Perhaps more striking is that the PCP theorem is in fact equivalent to the existence of such a gap-introducing Karp reduction.

Lemma 2 *The PCP Theorem implies $\exists$ Karp reduction f from* SAT *to* E3SAT *s.t. if $\phi \notin$ SAT then any assignment to $f(\phi)$ satisfies at most a δ-fraction of the clauses of $f(\phi)$ for some constant $\delta < 1$.*

Proof. Let V be a $(c \cdot \log n, q)$-PCP verifier for SAT where c, q are constants. Let ϕ be an instance of SAT, we construct a q-SAT instance $\overline{\phi}$ as follows:

- variables of $\overline{\phi}$: bits of the proof π that V can query
- clauses of $\overline{\phi}$: $\forall y \in \{0,1\}^{c \cdot \log |\phi|}$ (enumerate all random strings), $\forall b_1, ..., b_q \in \{0,1\}$, say V queries $\pi_{i_1}, \pi_{i_2}, ..., \pi_{i_q}$ given y. If $V(x, \pi)$ rejects with random string y and $\pi_{i_j} = b_j$, $\forall 1 \leq j \leq q$, include the clause $\overline{\pi_{i_1} = b_1 \wedge \pi_{i_2} = b_2 \wedge ... \wedge \pi_{i_q} = b_q}$. (e.g. $b = (0,1,0,0)$, include the clause $\pi_{i_1} \vee \overline{\pi_{i_2}} \vee \pi_{i_3} \vee \pi_{i_4}$)

If $\phi \in$ SAT, then the same proof π satisfies $\overline{\phi}$.

If $\phi \notin$ SAT, for each $y \in \{0,1\}^{c \cdot \log |\phi|}$, let C_y be the $\leq 2^y$ clauses generated for this y. We know $\forall \pi$, $\Pr[V(x, \pi)$ = reject$] \geq \frac{1}{2}$, for the same π, the number of unsatisfied clauses are at least $\frac{1}{2} \cdot 2^{c \cdot \log |\phi|}$, since the total number of clauses are at most $2^q \cdot 2^{c \cdot \log |\phi|}$, we have the fraction of unsatisfied clauses is $\geq \frac{1}{2^{q+1}}$, which is constant as q is constant. ■

The proofs of Lemma 1 and Lemma 2 shows that PCP theorem is equivalent to the statement about the existence of such Karp reduction f from SAT to E3SAT. Some remarks about the tightness of some parameters.

- There are, in fact, PCP verifiers for any language in **NP** that only examine 3 bits of the proof.

  Though, this is only with very slightly worse probability at most $1/2 + \epsilon$ of accepting any proof in the no case for any constant $\epsilon > 0$: the running time of the verifier increases as ϵ decreases but is polynomial for fixed $\epsilon > 0$

- However, $\mathbf{PCP}(O(\log n), 2) = \mathbf{P}$.

  This is because the reduction in Lemma 2 produces an instance of q-SAT, which is polynomial-time solvable for $q = 2$.

- $\mathbf{PCP}(o(\log n), O(1)) = \mathbf{P}$.

  Essentially this is because the reduction of Lemma 2 would reduce an instance of SAT to an instance of SAT with smaller size (for large enough n). As we have discussed earlier, we could iterate this reduction until we get an instance of SAT whose size is bounded by a constant, which could then be solved in polynomial time.

References

AB09 S. Arora and B. Barak, Computational Complexity: A Modern Approach *Cambridge University Press*, 2009