Computational Complexity. Lecture 18

#P and Approximate Counting.

Alexandra Kolla

Today

- Counting classes.
- Reductions and complete problems.
- Complexity of counting.
- Leftover Hash Lemma.

NP relations and counting certificates

- R is an NP relation if there is a poly time algorithm A(.,.) and a polynomial p s.t. $(\mathrm{x}, \mathrm{y}) \in \mathrm{R} \Leftrightarrow A(x, y) = 1$ and $(\mathrm{x}, \mathrm{y}) \in \mathrm{R} \Rightarrow$ |y|≤p(|x|).

- #R is the problem that, given x, asks how many y satisfy $(\mathrm{x}, \mathrm{y}) \in \mathrm{R}$.

Counting classes

- **Definition**. #P is the class of all problems of the form #R, where R is an NP-relation.

- Unlike for decision problems there is no canonical way to define reductions for counting classes. There are two common definitions.

Reductions for counting classes

- **Definition 1**. We say there is a parsimonious reduction from #A to #B (written $\#\mathrm{A} \leq_{par} \#B$) if there is a polynomial time transformation f such that for all x, $|\{y, (x, y) \in A\}| = |\{z: (f(x), z) \in B\}|$

Reductions for counting classes

- Previous definition restrictive, we use the next one instead sometimes:

- **Definition 2**. #A $\leq$ #B if there is a polynomial time algorithm for #A given an oracle that solves #B.

Complete problems

- #CIRCUITSAT is the problem where given a circuit, we want to count the number of inputs that make the circuit output 1.
- **Theorem 1**. #CIRCUITSAT is #P-complete under parsimonious reductions.

Complete problems

- **Theorem 2**. #3SAT is #P-complete under parsimonious reductions.

- If a counting problem #R is #P- complete under parsimonious reductions, then the associated language LR is NP-complete.

- For the oracle definition this is not true. There are problems whose decision version is in P, that are #P-complete (2SAT,counting perfect matchings in bipartite graph).

Complexity of counting problems

- **Theorem 3**. For every counting problem #A in #P, there is an algorithm C that on input x, computes with high probability a value v such that

$$(1-\epsilon)\#A(x) \leq v \leq (1+\epsilon)\#A(x)$$

In time polynomial in |x| and in 1/ ϵ , using an oracle for NP.

Complexity of counting problems

- The theorem says that #P can be approximated in BPP^{NP}.
- Note that approximating #3SAT is NP-hard, thus to compute the value v we need at least the power of NP.
- Theorem says that the power of NP and randomization is sufficient.

Complexity of counting problems

- Another result :
- **Theorem 4(Toda).** For every k, $\Sigma_k \subseteq P^{\#P}$

- Implies that #3SAT is Σ_k-hard for every k, unless the hierarchy collapses.
- Recall that BPP is in Σ_2 hence approximating #3SAT can be done in Σ_3.
- Therefore approximating #3SAT cannot be equivalent to computing it, unless PH collapses.

Proof of Theorem 3

- Some observations that will make the proof easier.
- Enough to prove it for #3SAT. If we have approximation algorithm for #3SAT we can extend it to any #A in #P using the parsimonious reduction from #A to #3SAT.

Proof of Theorem 3

- Enough to give a polynomial time O(1) approximation for #3SAT.
- That is, suppose we have algorithm C and constant c such that

$$\frac{1}{c}\#3SAT(\phi) \leq C(\phi) \leq c\#3SAT(\phi)$$

Then we can construct $\phi^k = \phi_1 \wedge \cdots \wedge \phi_k$, where ϕ_i is a copy of ϕ using fresh variables.

Proof of Theorem 3

- For formula φ that has $O(1)$ sat. assignments, #3SAT(φ) can be found in P^{NP}.

Iteratively, asking the oracle questions of the form: Are there k assignments satisfying the formula? (NP, since algorithm can guess k assignments and check them)

Proof of Theorem 3, simplified

- **Theorem 3'**. There is an algorithm C that on input x, computes with high probability a value v such that, for some constant c=O(1):

$$\frac{1}{\mathrm{c}}\#3\mathrm{SAT}(\varphi) \leq v \leq c\#3SAT(\varphi)$$

In time polynomial in |x|, using an oracle for NP.

- We will show that in the rest of class.

Leftover Hash Lemma

- Like in Valiant-Vazirani, for a given formula ϕ we will pick hash function h and look at the number of assignments x that satisfy ϕ and h(x)=0.

- **Leftover Hash Lemma. (Impagliazzo, Levin, Luby)**

Let H be a family of pairwise independent hash functions $h: \{0,1\}^n \rightarrow \{0,1\}^m$. Let $S \subseteq \{0,1\}^n, |S| \geq 4 \cdot \frac{2^m}{\epsilon^2}$. Then, $\Pr_{h \in H}\left[\left| |\{a \in S: h(a) = 0\}| - \frac{|S|}{2^m}\right| \geq \frac{\epsilon|S|}{2^m}\right] \leq \frac{1}{4}$