Computational Complexity: Lecture 20.

Recap:
- #P – counting witnesses.
- #SAT, Perm are #P-complete
- $PH \subseteq P^{\#P}$. – VV++ & some modular magic.

Agenda: – Approximate counting

Qn: Exactly computing #satisfying assignments is hard.
But can we approximate the number of SAT assignments?

φ either $\xrightarrow{\geq 2 \text{ ass.}}$ / $\xrightarrow{\text{no sat-ass.}}$

$$\varphi(x^{(1)}) \wedge \dots \wedge \varphi(x^{(k)}) = \varphi^k \to \geq 2^k \quad \text{or} \quad \to 0$$

Computing #SAT approximately seems at least as hard as SAT itself.

Thm: For any ε, δ, there is a BPP^{NP} algorithm A^{SAT} which, on input φ, satisfies
$$\Pr\left[A^{SAT}(\varphi) \in \#SAT(\varphi) \cdot (1 \pm \varepsilon)\right] \geq 1 - \delta.$$
with running time $\text{poly}(|\varphi|, \frac{1}{\varepsilon}, \log \frac{1}{\delta})$.

What can we do with an NP oracle?

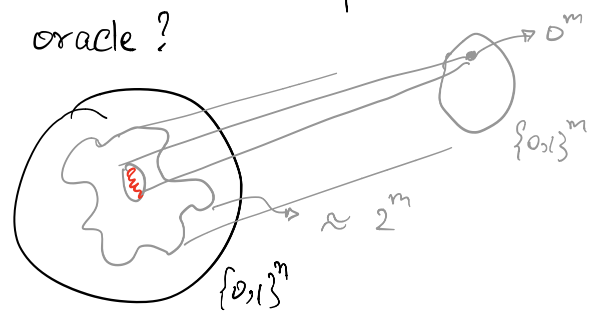
- Checking if φ is SAT/TAUT
- Checking if φ has exactly 1 SAT assignment.
- Checking if φ has ≥ 42 SAT assignments.

$\therefore$ If $\#SAT(\varphi)$ is "small" then we can compute this exactly using an NP oracle.

A simpler promise problem:

$$\text{approx-count}(\varphi, k) = \begin{cases} \text{Yes} & \text{if } \#SAT(\varphi) \geq 2^{k+1} \\ \text{No} & \text{if } \#SAT(\varphi) \leq 2^k. \end{cases}$$

How can we hope to solve this even with an NP oracle?



$|h^{-1}(0^m) \cap S|$ should roughly tell us the size of S.

$S = \{a : \varphi(a) = 1\}$.

Lemma: (Leftover Hash Lemma) Let $\mathcal{H} = \{h : \{0,1\}^n \to \{0,1\}^m\}_h$ be a family of pairwise independent hash functions and let $\varepsilon > 0$. For any $S \subseteq \{0,1\}^n$ s.t. $|S| \geq 4 \cdot 2^m / \varepsilon^2$, we have.

$$\Pr_{h \in \mathcal{H}}\left[\left|\{x \in S : h(x) = 0^m\}\right| \in \frac{|S|}{2^m} \cdot (1 \pm \varepsilon) \right] \geq 3/4.$$

& $\Pr[h(x) = a] = 1/2^m$.

$\forall x \neq y, a, b$ $\quad \Pr[h(x) = a \;\&\; h(y) = b] = \Pr[h(x) = a] \cdot \Pr[h(y) = b]$

Let us just assume this for now and proceed.

BPP^{NP} algo for approx-count (φ, k):

- $\triangleright$ If $k \leq 5$ we can check if $\varphi \geq 2^{k+1}$ sat. assignments using an NP oracle.
- $\triangleright$ If $k \geq 6$. Set $m = k-5$ and pick a random $h: \{0,1\}^n \to \{0,1\}^m$ from a p.i.h.f.
  Return yes if the formula
  "$\varphi(x) = 1 \wedge h(x) = 0$" has ≥ 48 sat. assignmnt
  & "No" otherwise.

Correctness:

Let $S = \{a : \varphi(a) = 1\}$.

"Yes" case: $|S| \geq 2^{k+1} = 2^{m+6} = 4 \cdot 2^m / \varepsilon^2$ for $\varepsilon = 1/4$.

LHL says $\Pr\left[|\{a \in S : h(a) = 0\}| \geq \underbrace{(1-\varepsilon) \cdot \frac{|S|}{2^m}}_{\frac{3}{4} \cdot 64 = 48} \right] \geq \frac{3}{4}$

"No" case: $|S| \leq 2^k$. $S \subseteq S'$ $|S'| = 2^k = 2^{m+5}$ $\varepsilon = 1/2$

$$\Pr\left[|\{a \in S' : h(0)\}| > \underbrace{(1+\varepsilon) \cdot \frac{|S'|}{2^m}}_{\shortparallel \atop 48} \right] \leq 1/4$$

$\square$

Remark: We can push the error down by repeating + majority.

$B^{NP}(\varphi)$:

Run approx-count(φ, i) for $i = 0, 1, \ldots, n$.

If k is the last point where approx-count(φ, k) = Yes, return 2^k.

$$\frac{\#SAT(\varphi)}{4} \le 2^k \le \#SAT(\varphi) \le 2^{k+2}$$

"Yes" at k. "No" at $k+1$

$\therefore$ This alg. gets $\#SAT(\varphi)$ right within a factor of 4.

Cor: There is a BPP^{NP} algo to compute #SAT within a factor of 4.

Can we get within a $(1 \pm \varepsilon)$ factor?

$A^{SAT}(\varphi)$:

- Consider $\psi = \varphi^t$ (t to be chosen shortly)
- $N = B^{SAT}(\psi)$
- Return $N^{1/t}$

M

Say $M = \#SAT(\varphi) \Rightarrow M^t = \#SAT(\psi)$

$$\left(\frac{1}{4}\right)^{1/t} M^{\varepsilon} \le \left(B^{SAT}(\psi)\right)^{1/t} \le M^{\varepsilon}$$

$$\left(\frac{1}{4}\right)^t \approx 1 - \varepsilon \qquad t = O(1/\varepsilon)$$

Again the prob. of error can be pushed to any δ by repeating etc.

Essentially finishes this $\longrightarrow$ Thm: For any ε, δ, there is a BPP^{NP} algorithm A^{SAT} which, on input φ, satisfies

$$\Pr\left[A^{SAT}(\varphi) \in \#SAT(\varphi)\cdot(1 \pm \varepsilon)\right] \geq 1-\delta.$$

with running time $poly(|\varphi|, \frac{1}{\varepsilon}, \log\frac{1}{\delta})$.

modulo the LHL.

Proof of the Leftover Hash Lemma:

Lemma: (Leftover Hash Lemma) Let $\mathcal{H} = \{h : \{0,1\}^n \to \{0,1\}^m\}_h$ be a family of pairwise independent hash functions and let $\varepsilon > 0$. For any $S \subseteq \{0,1\}^n$ s.t. $|S| \geq 4 \cdot 2^m/\varepsilon^2$, we have.

$$\Pr_{h \in \mathcal{H}}\left[\,|\{x \in S : h(x) = 0^m\}| \in \frac{|S|}{2^m}\cdot(1 \pm \varepsilon)\right] \geq 3/4.$$

Say $S = \{a_1, .., a_r\}$. $\quad X_i = \mathbb{1}(h(a_i) = 0^m)$

$\mathbb{E}[X] = |S|/2^m = r/2^m = \mu \quad \sum X_i = X$

Interested in $\Pr[|X - \mu| \geq \varepsilon\mu]$

If X_i's were indep. then Chernoff would have worked..

But X_i's need not be indep... but they are pairwise indep.

Obs: For any $i \neq j$ $\quad \Pr[X_i = a, X_j = b] = \Pr[X_i = a]\Pr[X_j = b]$

Since X_i's are pairwise indep,

$$\mathrm{Var}(X) = \mathbb{E}[(X-\mu)^2]$$

pairwise indep. $\rightarrow$
$$= \sum \mathrm{Var}(X_i)$$
$$= \sum_{i=1}^{r} \left(\mathbb{E}[X_i] - \mathbb{E}[X_i]^2 \right)$$
$$= \sum_{i=1}^{r} \left(\frac{1}{2^m} - \frac{1}{2^{2m}} \right) \leq \frac{r}{2^m}.$$

$$\Pr[|X-\mu| \geq \varepsilon\mu] = \Pr[(X-\mu)^2 \geq \varepsilon^2\mu^2] \leq \frac{\mathbb{E}[(X-\mu)^2]}{\varepsilon^2\mu^2}$$
$$\leq \frac{\cancel{r/2^m}}{\varepsilon^2 \cdot (r/2^m)^2} = \frac{2^m}{\varepsilon^2 \cdot |S|} \leq 1/4.$$

if $|S|$ is large.

Chebyshev's Ineq: $\Pr[|X-\mu| \geq \varepsilon\mu] \leq \frac{\mathrm{Var}(X)}{\varepsilon^2\mu^2}$.

So what have we learnt? $\square$.

- Perm & #SAT are #P-complete.
- $\#P = FP \Rightarrow P = NP$.

  But we can do a lot more with a #P oracle than with an NP-oracle.

  $P^{NP} \subseteq \Sigma_2$. $\qquad P^{\#P} \supseteq PH$.
- But if we only wish to approximately compute #SAT (or Perm) we can do that in BPP^{NP}.

(Kuldeep Meel & Moshe Vardi)

▷ **Fact:** Perm of a <u>non-negative</u> matrix can actually be approximately computed in P ! [Jerrum - Sinclair - Vigoda].

Some properties of the Permanent.

▷ Downward self-reducibility:

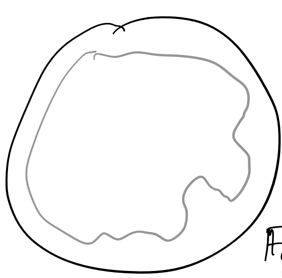
If you know how to solve $n \times n$ permanents, you can solve $n+1 \times n+1$ permanents.

$$\begin{bmatrix} x_{11} & \cdots & x_{1\,n+1} \\ \vdots & & \\ x_{n+1\,1} & & x_{n+1\;n+1} \end{bmatrix} = x_{11} \cdot \Big[\quad \Big] + x_{12} \cdot \Big[\quad \Big] + \cdots + x_{1n+1} \Big[\quad \Big]$$

▷ Random self reducibility.

$$\Big[\qquad \Big]_{n \times n}$$

Fix a prime p $(\approx n^2)$.
Consider the task of computing $\text{Perm}(X) \bmod p$.



$\mathbb{F}_p^{n \times n}$.

Suppose we have an algorithm A that computes Perm mod p on $\geq 1 - \frac{1}{3(n+1)}$ fraction of inputs.

Claim: Given A, we can compute Perm mod p everywhere with prob $\geq 2/3$.

Pf: Say we are given a matrix $X_{n \times n}$.
Pick a matrix Y uniformly at random.
Since $p \approx n^2$, let $\alpha_0, \alpha_1, \alpha_2 \ldots \alpha_n$ be distinct & non-zero residues mod p.

$$\mathrm{Perm}(X+tY) = f_0 + f_1 t + f_2 t^2 + \cdots + f_n t^n.$$

$$\begin{bmatrix} x_{11}+t y_{11} & & \\ \vdots & & \\ & & x_{nn}+t y_{nn} \end{bmatrix} \qquad \mathrm{Perm}(X)$$

Let $\beta_i = \mathrm{Perm}(X + \alpha_i Y)$ for $i = 0, 1, \ldots, n$

$$\begin{bmatrix} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^n \\ & & & & \\ 1 & \alpha_n & \alpha_n^2 & \cdots & \alpha_n^n \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_n \end{bmatrix}$$

Matrix is invertible.

$\mathrm{Det} = \prod (\alpha_i - \alpha_j)$

$$\Rightarrow \begin{bmatrix} f_0 \\ \vdots \\ f_n \end{bmatrix} = \boxed{V^{-1}} \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_n \end{bmatrix}$$

$$\Rightarrow f_0 = \sum_{i=0}^{n} (V^{-1})_{0,i} \cdot \beta_i$$

What are the chances that $A(X+\alpha_i Y) = \mathrm{Perm}(X+\alpha_i Y)$ for all $i = 0, \ldots, n$?

Obs: For any fixed matrix X, if Y is random and $\alpha_i \neq 0$ then $X + \alpha_i Y$ is also random!

$$\Pr\left[\exists i : A(X+\alpha_i Y) \neq \mathrm{Perm}(X+\alpha_i Y)\right] \leq \frac{1}{3(n+1)} (n+1) \leq 1/3 \qquad \square$$

Turns out, even if A computes Perm on $1/n$ fraction of inputs, that's enough to get the same conclusion above!

"low-degree polynomials are error-correcting codes".

Ref: "Permanent is hard even on a good day" by Yuval Filmus.