Notes for Lecture 8

Today we discuss the complexity of "counting problems" and of their approximate versions.

1 Counting Classes

Definition 1 *R is an* **NP**-relation*, if there is a polynomial time algorithm A such that $(x, y) \in R \Leftrightarrow A(x, y) = 1$ and there is a polynomial p such that $(x, y) \in R \Rightarrow |y| \leq p(|x|)$.*

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

Definition 2 *#***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.

Definition 3 *We say there is a* parsimonious reduction *from $\#A$ to $\#B$ (written $\#A \leq_{par} \#B$) if there is a polynomial time transformation f such that for all x, $|\{y, (x, y) \in A\}| = |\{z : (f(x), z) \in B\}|$.*

Often this definition is a little too restrictive and we use the following definition instead.

Definition 4 *$\#A \leq \#B$ if there is a polynomial time algorithm for $\#A$ given an oracle that solves $\#B$.*

#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.*

PROOF: Let $\#R$ be in #**P**and A and p be as in the definition. Given x we want to construct a circuit C such that $|\{z : C(z)\}| = |\{y : |y| \leq p(|x|), A(x, y) = 1\}|$. We then construct $\hat{C}_n$ that on input x, y simulates $A(x, y)$. From earlier arguments we know that this can be done with a circuit with size about the square of the running time of A. Thus $\hat{C}_n$ will have size polynomial in the running time of A and so polynomial in x. Then let $C(y) = \hat{C}(x, y)$. $\square$

Theorem 2 *#3SATis #***P***-complete.*

PROOF: We show that there is a parsimonious reduction from #CIRCUITSAT to #3-SAT. That is, given a circuit C we construct a Boolean formula ϕ such that the number of satisfying assignments for ϕ is equal to the number of inputs for which C outputs 1. Suppose C has inputs $x_1, \ldots, x_n$ and gates $1, \ldots, m$ and ϕ has inputs $x_1, \ldots, x_n, g_1, \ldots, g_m$, where the g_i represent the output of gate i. Now each gate has two input variables and one output variable. Thus a gate can be complete described by mimicking the output for each of the 4 possible inputs. Thus each gate can be simulated using at most 4 clauses. In this way we have reduced C to a formula ϕ with $n + m$ variables and $4m$ clauses. So there is a parsimonious reduction from #CIRCUITSAT to #3SAT. $\square$

2 Complexity of counting problems

We will prove the following theorem:

Theorem 3 *For every counting problem #A in #**P**, there is an algorithm C that on input x, compute 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 $\frac{1}{\epsilon}$, using an oracle for **NP**.

The theorem says that #**P** can be approximate in $\mathbf{BPP}^{\mathbf{NP}}$. We have a remark here that approximating #3SAT is **NP**-hard. Therefore, to compute the value we need at least the power of **NP**, and this theorem states that the power of **NP** and randomization is sufficient.

Another remark concerns the following result.

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

This implies that #3SAT is Σ_k-hard for every k, i.e., #3SAT lies outside **PH**, unless the hierarchy collapses. Recall that **BPP** lies inside Σ_2, and hence approximating #3SAT can be done in Σ_3. Therefore, approximating #3SAT cannot be equivalent to computing #3SAT exactly, unless the polynomial hierarchy collapses.

We first make some observations so that we can reduce the proof to an easier one.

- It is enough to prove the theorem for #3SAT.

  If we have an approximation algorithm for #3SAT, we can extend it to any #A in #**P** using the parsimonious reduction from #A to #3SAT.

- It is enough to give a polynomial time $O(1)$-approximation for #3SAT.

  Suppose we have an algorithm C and a constant c such that

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

  Given φ, we can construct $\varphi^k = \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_k$. If φ has t satisfying assignments, φ^k has t^k satisfying assignments. Then, giving φ^k to the algorithm we get

  $$\frac{1}{c}t^k \leq C(\varphi^k) \leq ct^k$$
  $$\left(\frac{1}{c}\right)^{1/k} t \leq C(\varphi^k)^{1/k} \leq c^{1/k}t.$$

  If c is a constant and $k = O(\frac{1}{\epsilon})$, $c^{1/k} = 1+\epsilon$.

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

  This can be done by iteratively asking the oracle the questions of the form: "Are there k assignments satisfying this formula?" Notice that these are **NP** questions, because the algorithm can guess these k assignments and check them.

3 An approximate comparison procedure

Consider the following approximate comparison procedure `a-comp` defined as:

$$\texttt{a-comp}(\varphi, k) = \begin{cases} \text{YES} & \text{if } \#3\text{SAT}(\varphi) \geq 2^{k+1} \\ \text{NO} & \text{if } \#3\text{SAT}(\varphi) < 2^{k} \\ \text{whatever} & \text{otherwise} \end{cases}$$

Given `a-comp`, we can construct an algorithm that 2-approximates #3SAT as follows:

Input: φ

compute:
  $\texttt{a-comp}(\varphi, 1)$
  $\texttt{a-comp}(\varphi, 2)$
  $\texttt{a-comp}(\varphi, 3)$
  $\vdots$
  $\texttt{a-comp}(\varphi, n)$

if `a-comp` outputs NO from the first time **then**
  // The value is either 0 or 1.
  // The answer can be checked by one more query to the **NP** oracle.
  Query to the oracle and output an exact value.
else
  Suppose that it outputs YES for $t = 1, \ldots, i-1$ and NO for $t = i$
  Output 2^i

We need to show that this algorithm approximates #3SAT within a factor of 2. If `a-comp` answers NO from the first time, the algorithm outputs the right answer because it checks for the answer explicitly. Now suppose `a-comp` says YES for all $t = 1, 2, \ldots, i-1$ and says NO for $t = i$. Since $\texttt{a-comp}(\varphi, i-1)$ outputs YES, $\#3\text{SAT}(\varphi) \geq 2^{i-1}$, and also since $\texttt{a-comp}(\varphi, 2^i)$ outputs NO, $\#3\text{SAT}(\varphi) < 2^{i+1}$. The algorithm outputs $a = 2^i$. Hence,

$$\frac{1}{2}a \geq \#3\text{SAT}(\varphi) < 2 \cdot a$$

and the algorithm outputs the correct answer with in a factor of 2.

Thus, to establish the theorem, it is enough to give a $\mathbf{BPP}^{\mathbf{NP}}$ implementation of the `a-comp`.

4 Constructing `a-comp`

The procedure and its analysis is similar to the Valiant-Vazirani reduction: for a given formula ϕ we pick a hash function h from a pairwise independent family, and look at the number of assignments x that satisfy h and such that $h(x) = \mathbf{0}$.

In the Valiant-Vazirani reduction, we proved that if S is a set of size approximately equal to the size of the range of $h()$, then, with constant probability, exactly one element of S is mapped by $h()$ into $\mathbf{0}$. Now we use a different result, a simplified version of the Leftover Hash Lemma proved by Impagliazzo, Levin, and Luby in 1989, that says that if S is sufficiently larger than the range of $h()$ then the number of elements of S mapped into $\mathbf{0}$ is concentrated around its expectation.

Lemma 5 (The Leftover Hash Lemma) *Let H be a family of pairwise independent hash functions $h : \{0,1\}^n \to \{0,1\}^m$. Let $S \subset 0, 1^n$, $|S| \geq \frac{4 \cdot 2^m}{\epsilon^2}$. Then,*

$$\mathbf{Pr}_{h \in H}\left[\left| |\{a \in S : h(a) = 0\}| - \frac{|S|}{2^m} \right| \geq \epsilon \frac{|S|}{2^m}\right] \leq \frac{1}{4}.$$

From this, `a-comp` can be constructed as follows.

input: φ, k

if $k \leq 5$ **then** check exactly whether $\#3\mathrm{SAT}(\varphi) \geq 2^k$.
if $k \geq 6$,
pick h from a set of pairwise independent hash functions $h : \{0,1\}^n \to \{0,1\}^m$,
where $m = k - 5$
answer YES iff there are more then 48 assignments a to φ such that
a satisfies φ and $h(a) = 0$.

Notice that the test at the last step can be done with one access to an oracle to $\mathbf{NP}$. We will show that the algorithm is in $\mathbf{BPP}^{\mathbf{NP}}$. Let $S \subseteq \{0,1\}^n$ be the set of satisfying assignments for φ. There are 2 cases.

- If $|S| \geq 2^{k+1}$, by the Leftover Hash Lemma we have:

$$\mathbf{Pr}_{h \in H}\left[\left| |\{a \in S : h(a) = 0\}| - \frac{|S|}{2^m} \right| \leq \frac{1}{4} \cdot \frac{|S|}{2^m}\right] \leq \frac{3}{4},$$

(set $\epsilon = \frac{1}{2}$, and $|S| \geq \frac{4 \cdot 2^m}{\epsilon^2} = 64 \cdot 2^m$, because $|S| \geq 2^{k+1} = 2^{m+6}$)

$$\mathbf{Pr}_{h \in H}\left[|\{a \in S : h(a) = 0\}| \leq \frac{3}{4} \cdot \frac{|S|}{2^m}\right] \geq \frac{3}{4},$$
$$\mathbf{Pr}_{h \in H}\left[|\{a \in S : h(a) = 0\}| \geq 48\right] \geq \frac{3}{4},$$

which is the success probability of the algorithm.

- If $|S| < 2^k$:

Let S' be a superset of S of size 2^k. We have

$$\begin{aligned}
\mathbf{Pr}_{h\in H}[\text{answer YES}] &= \mathbf{Pr}_{h\in H}[|\{a \in S : h(s) = 0\}| \geq 48] \\
&\leq \mathbf{Pr}_{h\in H}[\left|\left\{a \in S' : h(s) = 0\right\}\right| \geq 48] \\
&\leq \mathbf{Pr}_{h\in H}\left[\left|\left|\left\{a \in S' : h(s) = 0\right\}\right| - \tfrac{|S'|}{2^m}\right| \geq \tfrac{|S'|}{2\cdot 2^m}\right] \\
&\leq \frac{1}{4}
\end{aligned}$$

(by L.H.L. with $\epsilon = 1/2, |S'| = 32 \cdot 2^m$.)

Therefore, the algorithm will give the correct answer with probability at least $3/4$, which can then be amplified to, say, $1 - 1/4n$ (so that all n invocations of `a-comp` are likely to be correct).

5 The proof of the Leftover Hash Lemma

We finish the lecture by proving the Leftover Hash Lemma.

Proof: We will use Chebyshev's Inequality to bound the failure probability. Let $S = \{a_1, \ldots, a_k\}$, and pick a random $h \in H$. We define random variables $X_1, \ldots, X_k$ as

$$X_i = \begin{cases} 1 & \text{if } h(a_i) = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, $|\{a \in S : h(a) = 0\}| = \sum_i X_i$.

We now calculate the expectations. For each i, $\mathbf{Pr}[X_i = 1] = \frac{1}{2^m}$ and $\mathbf{E}[X_i] = \frac{1}{2^m}$. Hence,

$$\mathbf{E}[\sum_i X_i] = \frac{|S|}{2^m}.$$

Also we calculate the variance

$$\begin{aligned}
\mathbf{Var}[X_i] &= \mathbf{E}[X_i^2] - \mathbf{E}[X_i]^2 \\
&\leq \mathbf{E}[X_i^2] \\
&= \mathbf{E}[X_i] = \frac{1}{2^m}.
\end{aligned}$$

Because $X_1, \ldots, X_k$ are pairwise independent,

$$\mathbf{Var}\left[\sum_i X_i\right] = \sum_i \mathbf{Var}[X_i] \leq \frac{|S|}{2^m}.$$

Using Chebyshev's Inequality, we get

$$\begin{aligned}
\mathbf{Pr}\left[\left||\{a \in S : h(a) = 0\}| - \frac{|S|}{2^m}\right| \geq \epsilon\frac{|S|}{2^m}\right] &= \mathbf{Pr}\left[\left|\sum_i X_i - \mathbf{E}[\sum_i X_i]\right| \geq \epsilon\mathbf{E}[\sum_i X_i]\right] \\
&\leq \frac{\mathbf{Var}[\sum_i X_i]}{\epsilon^2\mathbf{E}[\sum_i X_i]^2} \leq \frac{\frac{|S|}{2^m}}{\epsilon^2\frac{|S|^2}{(2^m)^2}} \\
&= \frac{2^m}{\epsilon^2\,|S|} \leq \frac{1}{4}.
\end{aligned}$$

□