Counting Problems

Today we describe counting problems and the class #**P** that they define, and we show that every counting problem #**P** can be approximately solved in randomized polynomial given access to an **NP** oracle.

1 Counting Classes

[Note: we did not cover most of the material of this section in class.]

Recall that 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 $(x, y) \in R \Rightarrow |y| \leq p(|x|)$.

Definition 1 *If R is an* **NP** *relation, then $\#R$ is the problem that, given x, asks how many y satisfy $(x, y) \in R$.*

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

Observe that an **NP**-relation R naturally defines an **NP** language L_R, where $L_R = \{x : \exists y.(x, y) \in R\}$, and every **NP** language can be defined in this way. Therefore problems in #**P** can always be seen as the problem of counting the number of witnesses for a given instance of an **NP** problem.

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

Definition 2 *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 3 *$\#A \leq \#B$ 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 4 *#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 $C(y) = \hat{C}(x, y)$. □

Theorem 5 *#3SAT is* **#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. □

Notice that if a counting problem $\#R$ is **#P**-complete under parsimonious reductions, then the associated language L_R is **NP**-complete, because $\#3SAT \leq_{par} \#R$ implies $3SAT \leq L_R$. On the other hand, with the less restrictive definition of reducibility, even some counting problems whose decision version is in **P** are **#P**-complete. For example, the problem of counting the number of satisfying assignments for a given 2CNF formula and the problem of counting the number of perfect matchings in a given bipartite graphs are both **#P**-complete.

2 Complexity of counting problems

We will prove the following theorem:

Theorem 6 *For every counting problem $\#A$ in* **#P***, there is a probabilistic 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 $\frac{1}{\epsilon}$, using an oracle for **NP***.*

The theorem says that $\#\mathbf{P}$ can be approximate in $\mathbf{BPP}^{\mathbf{NP}}$. We remark that approximating $\#3SAT$ is $\mathbf{NP}$-hard, and so to compute an approximation we need at least the power of $\mathbf{NP}$. Theorem 6 states that the power of $\mathbf{NP}$ and randomization is sufficient.

Another remark concerns the following result.

Theorem 7 (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 the polynomial hierarchy, unless the hierarchy collapses. Recall that $\mathbf{BPP}$ 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.[1]

We first make some observations so that we can reduce the proof to the task of proving a simpler statement.

- 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 $\#\mathbf{P}$ 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

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

  Given φ, we can construct $\varphi^k = \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_k$ where each ϕ_i is a copy of ϕ constructed using fresh variables. If φ has t satisfying assignments, φ^k has t^k satisfying assignments. Then, giving φ^k to the algorithm we get

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

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

[1] The above discussion was not very rigorous but it can be correctly formalized. In particular: (i) from the fact that $\mathbf{BPP} \subseteq \Sigma_2$ and that approximate counting is doable in $\mathbf{BPP}^{\mathbf{NP}}$ it does not necessarily follow that approximate counting is in Σ_3, although in this case it does because the proof that $\mathbf{BPP} \subseteq \Sigma_2$ relativizes; (ii) we have defined $\mathbf{BPP}$, σ_3, etc., as classes of decision problems, while approximate counting is not a decision problem (it can be shown, however, to be equivalent to a "promise problem," and the inclusion $\mathbf{BPP} \subseteq \Sigma_2$ holds also for promise problems.

- For a formula φ that has $O(1)$ satisfying assignments, $\#3SAT(\varphi)$ 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

Suppose that we had available an approximate comparison procedure `a-comp` with the following properties:

- If $\#3SAT(\varphi) \geq 2^{k+1}$ then $\mathtt{a} - \mathtt{comp}(\varphi, k) =$ YES with high probability;
- If $\#3SAT(\varphi) < 2^{k}$ then $\mathtt{a} - \mathtt{comp}(\varphi, k) =$ NO with high probability.

Given `a-comp`, we can construct an algorithm that 2-approximates $\#3SAT$ as described below:

- Input: φ
- compute:
  - `a-comp`$(\varphi, 0)$
  - `a-comp`$(\varphi, 1)$
  - `a-comp`$(\varphi, 2)$
  - $\vdots$
  - `a-comp`$(\varphi, n+1)$
- if `a-comp` outputs NO from the first time then
  - *// The value is either* 0 *or* 1 *and 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, $\#3SAT(\varphi) \geq 2^{i-1}$, and also since $\texttt{a-comp}(\varphi, 2^i)$ outputs NO, $\#3SAT(\varphi) < 2^{i+1}$. The algorithm outputs $a = 2^i$. Hence,

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

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 8 *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,*

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

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

- input: φ, k
- if $k \leq 5$ then check exactly whether $\#3SAT(\varphi) \geq 2^k$.
- 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 **NP**. We will show that the algorithm is in $\mathbf{BPP^{NP}}$. 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 Lemma 8 we have:

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

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

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

  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

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

  (by Lemma 8 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) by repeating the procedure $O(\log n)$ times and taking the majority answer.

5 The proof of Lemma 8

We finish the lecture by proving Lemma 8.

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, $\mathbb{P}[X_i = 1] = \frac{1}{2^m}$ and $\mathbb{E}[X_i] = \frac{1}{2^m}$. Hence,

$$\mathbb{E}\left[\sum_i X_i\right] = \frac{|S|}{2^m}. \tag{2}$$

Also we calculate the variance

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

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

$$\mathbf{Var}\left[\sum_i X_i\right] = \sum_i \mathbf{Var}[X_i] \le \frac{|S|}{2^m}. \tag{3}$$

Using Chebyshev's Inequality, we get

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

□

6 Approximate Sampling

The content of this section was not covered in class; it's here as bonus material. It's good stuff.

So far we have considered the following question: for an **NP**-relation R, given an input x, what is the size of the set $R_x = \{y : (x,y) \in R\}$? A related question is to be able to sample from the uniform distribution over R_x.

Whenever the relation R is "downward self reducible" (a technical condition that we won't define formally), it is possible to prove that there is a probabilistic algorithm running in time polynomial in $|x|$ and $1/\epsilon$ to approximate within $1+\epsilon$ the value $|R_x|$ if and only if there is a probabilistic algorithm running in time polynomial in $|x|$ and $1/\epsilon$ that samples a distribution ϵ-close to the uniform distribution over R_x.

We show how the above result applies to 3SAT (the general result uses the same proof idea). For a formula ϕ, a variable x and a bit b, let us define by $\phi_{x\leftarrow b}$ the formula obtained by substituting the value b in place of x.[2]

If ϕ is defined over variables $x_1, \ldots, x_n$, it is easy to see that

$$\#\phi = \#\phi_{x\leftarrow 0} + \#\phi_{x\leftarrow 1}$$

Also, if S is the uniform distribution over satisfying assignments for ϕ, we note that

$$\mathbb{P}_{(x_1,\ldots,x_n)\leftarrow S}[x_1 = b] = \frac{\#\phi_{x\leftarrow b}}{\#\phi}$$

Suppose then that we have an efficient sampling algorithm that given ϕ and ϵ generates a distribution ϵ-close to uniform over the satisfying assignments of ϕ.

Let us then ran the sampling algorithm with approximation parameter $\epsilon/2n$ and use it to sample about $\tilde{O}(n^2/\epsilon^2)$ assignments. By computing the fraction of such assignments having $x_1 = 0$ and $x_1 = 1$, we get approximate values p_0, p_1, such that $|p_b - \mathbb{P}_{(x_1,\ldots,x_n)\leftarrow S}[x_1 = b]| \leq \epsilon/n$. Let b be such that $p_b \geq 1/2$, then $\#\phi_{x\leftarrow b}/p_b$ is a good approximation, to within a multiplicative factor $(1+2\epsilon/n)$ to $\#\phi$, and we can recurse to compute $\#\phi_{x\leftarrow b}$ to within a $(1+2\epsilon/n)^{n-1}$ factor.

Conversely, suppose we have an approximate counting procedure. Then we can approximately compute $p_b = \frac{\#\phi_{x\leftarrow b}}{\#\phi}$, generate a value b for x_1 with probability approximately p_b, and then recurse to generate a random assignment for $\#\phi_{x\leftarrow b}$.

The same equivalence holds, clearly, for 2SAT and, among other problems, for the problem of counting the number of perfect matchings in a bipartite graph. It is known that it is **NP**-hard to perform approximate counting for 2SAT and this result, with the above reduction, implies that approximate sampling is also hard for 2SAT. The problem of approximately sampling a perfect matching has a probabilistic polynomial solution, and the reduction implies that approximately counting the number of perfect matchings in a graph can also be done in probabilistic polynomial time.

[2] Specifically, $\phi_{x\leftarrow 1}$ is obtained by removing each occurrence of $\neg x$ from the clauses where it occurs, and removing all the clauses that contain an occurrence of x; the formula $\phi_{x\leftarrow 0}$ is similarly obtained.

The reduction and the results from last section also imply that 3SAT (and any other **NP** relation) has an approximate sampling algorithm that runs in probabilistic polynomial time with an **NP** oracle. With a careful use of the techniques from last week it is indeed possible to get an *exact* sampling algorithm for 3SAT (and any other **NP** relation) running in probabilistic polynomial time with an **NP** oracle. This is essentially best possible, because the approximate sampling requires randomness by its very definition, and generating satisfying assignments for a 3SAT formula requires at least an **NP** oracle.