Notes for Lecture 9

Today we see a problem related to approximate counting, and then start discussing average-case complexity. Our first approach to average-case complexity will be problem-specific: we show that a particular #**P**-complete problem (the PERMANENT problem) is as hard to solve on average as it is to solve on worst-case instances.

1 Approximate Sampling

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.[1]

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

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

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 $|p_b - \mathbf{Pr}_{(x_1,\ldots,x_n)\leftarrow S}[x_1 = b]| \leq \epsilon/n$. 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

[1] 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.

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.

The reduction and the results from last lecture 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.

2 The Permanent Problem

The *permanent* is a number assigned to a matrix $M \in R$, for a ring R, according to the formula

$$\operatorname{perm}(M) = \sum_{\pi \in S_n} \prod_{i=1}^{n} M_{i,\pi(i)}$$

where S_n is the group of permutations on n items. The definition of the permanent closely resembles that of the determinant; the only difference is that the determinant has a coefficient $(-1)^{\operatorname{sgn}(\pi)}$ in front of the product. However, this resemblance is deceptive: the determinant can be calculated in polynomial time, whereas Permanent is #**P**-complete.

Claim 1 *Let $R = \mathbb{Z}$ and let M be a 0/1 matrix, which we think of as representing a bipartite graph $G = (U, V, E)$ with $|U| = |V| = n$ as follows: number both edge sets U and V, separately, by $1, 2, \ldots n$, and let $M_{i,j}$ be 1 if edge $(i, j) \in E$, and 0 otherwise. Then* $\operatorname{perm}(M)$ *is the number of perfect matchings in G.*

Proof: Each $\pi \in S_n$ is a potential perfect matching, in that it can be thought of as a bijection between U and V. This corresponds to an actual perfect matching iff all edges $(i, \pi(i))$ are in E, i.e. if $M_{i,\pi(i)} = 1$ for all i, which is equivalent to saying that $\prod_{i=1}^{n} M_{i,\pi(i)} = 1$. Thus we get a summand of 1 in the formula for $\operatorname{perm}(M)$ for each perfect matching of G. □

Suppose we allow the entries of M to be nonnegative integers. Then M represents a bipartite graph where the edges have multiplicity given by the entries of M. It still makes sense to interpret $\operatorname{perm}(M)$ as the number of permanent matchings, because the product of the multiplicities of the edges of a matching is the number of ways of realizing that matching.

Note that in $\mathbb{Z}_2$, $\operatorname{perm}(M) = \det(M)$ and thus is polynomial time. But for sufficiently large moduli (or no modulus at all) it's #**P**-complete.

Theorem 2 *Suppose for fixed n and $\mathcal{F}$, $|\mathcal{F}| > n^{O(1)}$, we can compute* $\operatorname{perm}(M)$ *on a fraction $1 - \frac{1}{2n}$ of the inputs M in polynomial time. Then there is a (worst-case) polynomial time probabilistic algorithm for computing* $\operatorname{perm}(M)$.

Rather than proving this theorem directly, we will just observe that perm(M) is a degree n polynomial in indeterminates $M_{11}, M_{12}, \ldots, M_{1n}, \ldots, M_{nn}$, and prove the following much more general result.

Theorem 3 *Suppose we have oracle access to a function $f : \mathcal{F}^n \to \mathcal{F}$ and f agreees with some degree d polynomial $p : \mathcal{F}^n \to \mathcal{F}$ on a fraction $1 - \frac{1}{3d+3}$ of its inputs where $|\mathcal{F}| \geq d+2$. Then we can compute $f(x)$ with high probability for all $x \in \mathcal{F}^n$ in poly(n, d) time, assuming field operations can be performed in poly(n, d) time.*

Proof: Suppose that we input $x \in \mathcal{F}^n$ and pick $y \in \mathcal{F}^n$ uniformly at random. Consider the line $l(t) = x + ty$. Each point of l is uniformly distributed except $x = l(0)$. Now,

$$\begin{aligned}\Pr[p(l(i)) = f(l(i)) \text{ for } i = 1, 2, \ldots, d+1] &= 1 - Pr[\exists i \in 1, \ldots, d+1 : p(l(i)) \neq f(l(i))] \\ &\geq 1 - \frac{d+1}{3(d+1)} = \frac{2}{3}\end{aligned}$$

Now $q(t) = p(l(t))$ is a degree d polynomial in t, and we know $d+1$ of its values, so we can interpolate the others. Thus we have the following algorithm:

```
input: x,
  pick random y ∈ F^n
  find univariate degree-d polynomial q() such that
    q(i) = p(x + iy) for i = 1, ..., d + 1
  return (q(0))
```

This runs in poly(n, d) time, and with probability $\geq \frac{2}{3}$, it outputs $p(x)$. $\square$

We can strengthen our result to hold even if $p(x)$ only agrees with $f(x)$ on $\frac{3}{4} + \epsilon$ of its inputs. This stronger proof depends on the following theorem, due to Berlekamp and Welch.

Theorem 4 *Let $q : \mathcal{F} \to \mathcal{F}$ be a polynomial of degree d. Say we are given m pairs $(a_1, b_1), \ldots, (a_m, b_m)$ such that for fewer than $\frac{m-d}{2}$ pairs (a_i, b_i), $b_i \neq q(a_i)$. Then q can be reconstructed with poly(n, d) field operations.*

Now, as promised, we apply Berlekamp-Welch to strengthen our main theorem:

Theorem 5 *Suppose we have oracle access to a function $f : \mathcal{F}^n \to \mathcal{F}$ and f agreees with some degree d polynomial $p : \mathcal{F}^n \to \mathcal{F}$ on a fraction $\frac{3}{4} + \epsilon$ of its inputs where $|\mathcal{F}| \geq 1 + d/\epsilon$. Then we can compute $p(x)$ with high probability for all $x \in \mathcal{F}^n$ in poly(n, d) time, assuming field operations can be performed in poly(n, d) time.*

Proof: Consider the following algorithm:

```
input: x,
  pick random y ∈ F^n
  find degree-d univariate polynomial q() such that
    |{t ∈ F − {0} : q(t) ≠ f(x + ty)}| < (|F| − 1 − d)/2
  halt and fail if no such polynomial exists
  return (q(0))
```

If $q(t) = p(l(t))$ and $f(l(t))$ disagree on $< \frac{|\mathcal{F}|-d-1}{2}$ values of t, Berlekamp-Welch will find $q(t) \equiv p(l(t))$ and the algorithm succeeds (and in polynomial time). So, assuming only $\frac{1}{4} - \epsilon$ of our points disagree, we need to show that less than half of our runs fail.

$$\begin{aligned}
\mathbf{E}\left[\#\text{ values of } t \in \mathcal{F} - \{0\} \text{ with } p(l(t)) \neq f(l(t))\right] &= \sum_{t \neq 0} \mathbf{E}\left[\begin{cases} 0 & \text{if } p(l(t)) = f(l(t)) \\ 1 & \textit{otherwise} \end{cases}\right] \\
&\leq (|\mathcal{F}| - 1)\left(\frac{1}{4} - \epsilon\right)
\end{aligned}$$

To clean up the notation let us use $m = |\mathcal{F}| - 1$. The previous expression and Markov's inequality tell us that

$$\Pr\left[|\{t \in \mathcal{F} - \{0\} : p(l(t)) \neq f(l(t))\}| > \frac{m-d}{2}\right] \leq \frac{m(\frac{1}{4} - \epsilon)}{\frac{m-d}{2}} \leq \frac{1}{2} - \frac{\epsilon}{2} \quad \text{if } m \geq \tfrac{d}{\epsilon}$$

Thus, our algorithm succeeds more than half the time, so it can be repeated to yield any degree of accuracy we wish. □