6.841 Advanced Complexity Theory March 16, 2005

Lecture 13

Lecturer: Madhu Sudan *Scribe: Victor Chen*

1 Overview

Today we study the complexity of counting problems and define the classes #P , $\mathrm{P}^{\#\mathrm{P}}$. In particular, we will examine the permanent of a matrix and its relation to #P-completeness and random instances; this is quite a remarkable problem with a central position in complexity theory. And finally, we will start Toda's landmark theorem, which states that $PH \subset \mathrm{P}^{\#\mathrm{P}}$.

A good reference for today's lecture is Chapter 18 in Papadimitriou's textbook.

2 #P

There are some natural questions that come up with nondeterminism. One is searching; given a problem, find a solution. Another is optimization - find the best solution. There is a third notion, namely counting how many solutions there are to a given problem, which we have yet to explore. To do so, let us first define #P. #P is defined to be the class of counting functions $f : \Sigma^* \to \mathbf{Z}$, such that there exists a polynomial time bounded machine M such that $\forall x$, $f(x) = |\{y | M(x, y) \text{ accepts}\}|$.

Some examples include #SAT, counting the number of satisfying assignments in a formula, and #CLIQUE, counting the number of cliques in a graph. As an exercise, note that the problem of counting the number of cliques in a graph reduces to the problem of counting the number of cliques at least a specified size.

2.1 PERMANENT

One of the most interesting problems in #P is the problem of counting the number of matchings in a graph, which is equivalent to finding the permanent of a matrix. (We will show this equivalence very shortly). Recall that a (perfect) matching in a graph G on n vertices is a subgraph H of degree exactly one at each vertex. Clearly MATCHING, the problem of deciding whether a given graph has a matching, is in NP. It is not obvious how to show this is in coNP. Tutte in the $40s$ proved many fundamental results on matching, and Edmonds in the $60s$ finally showed that MATCHING is in P. Incidentally, it was this work that led Edmonds to define polynomial time to capture our notion of efficiency. The counting version of MATCHING, #MATCHING, is simply counting the number of matchings in a graph. Note that unlike #SAT or #CLIQUE, the decision version of #MATCHING is easy yet the counting version is hard.

Clearly #MATCHING $\in$ #P. Now we show the equivalence between the number of matchings and the permanent of a matrix. Consider a bipartite graph and its associated adjacency matrix $A = \{a_{ij}\}$. It is easy to see that a matching in an n by n bipartite graph is given by a permutation $\pi : [n] \to [n]$. The edges in a matching simply match up each vertex on the left with a unique vertex on the right. Then the number of matchings in a graph is simply

$$\sum_{\pi \in S_n} \prod_{j=1}^{n} a_{j\pi(j)} = \mathrm{perm}(A).$$

In other words, #MATCHING=PERMANENT. Syntactically, this looks like the definition of the determinant of a matrix A. Recall that

$$\det(A) = \sum_{\pi \in S_n} (-1)^{\mathrm{sign}(\pi)} \prod_{j=1}^{n} a_{j\pi(j)},$$

where $\mathrm{sign}(\pi)$ is 1 if the number of swaps needed to sort π into identity is odd and 0 otherwise. Determinant is also a counting function, and we know that it can be computed efficiently. So perhaps the similarity in the algebraic definitions of the perm and the determinant suggest that we may compute the perm by using the determinant somehow. Indeed, one of the earliest approaches in trying to compute $\mathrm{perm}(A)$ involved making an appropriate transformation on A to some other matrix A' with $0, \pm$ entries, such that $\mathrm{perm}(A) = \det(A')$. However, it can be shown that this approach does not work since the permanent can be as large as $n!$ whereas the determinant of a $0, \pm$ matrix can be at most $2n$. Nonetheless, this approach works when the graph associated with the matrix is planar.

2.2 #P-completeness

Valiant showed that PERMANENT is #P-complete. We will not show this in class. Since a self-contained proof is in Papadimitriou's book, we will briefly make some remarks.

To prove completeness, we need a reduction that takes a function f and instance x and outputs A_x such that $f(x)$ can be computed efficiently from $\mathrm{perm}(A_x)$. The key observation is that most reductions we have seen in complexity so far are parsimonious, that is, the reduction preserves the number of solutions. Most importantly, Cook's reduction is parsimonious. In other words, if the reduction takes x and computes ϕ, then the number of satisfying assignment to ϕ is $f(x)k(|\phi|)$, where $k(\cdot)$ is an easily computable function. Note that the function k depends not on ϕ but just on its length. To show that #SAT reduces to PERMANENT, Valiant constructed appropriate gadgets such that if the reduction takes ϕ to A, then $\mathrm{perm}(A) = \#\mathrm{SAT}(\phi) + g(\phi)2^{n^2}$. Note that $\#\mathrm{SAT}(\phi) \leq 2^n$, so we can compute $\mathrm{perm}(A)$ modulo 2^{n^2}. This is sometimes called an affine parsimonious reduction.

2.3 Some Remarks

In complexity, it is much easier to work with languages than functions. We would like to define languages for #P. However, this is not as simple as SAT. Instead, we shall resort to letting $\mathrm{P}^{\#\mathrm{P}}$ capture our notion of languages.

Historical Notes: The complexity of counting led Valiant to define an algebraic version of NP. And the problem of approximate counting has developed into a beautiful area in its own right and sampling techniques.

3 Random Instances

Looking at the algebraic formalization of the permanent, there should be no reason that any particular instance should be easy to compute. In fact, we have the following result due to Lipton.

Theorem 1 *Suppose there exists an expected polynomial time algorithm that computes* $\text{perm}(A)$ *mod* p *for a random prime* $p \in [2^n]$ *and a random* n *by* n *matrix* $A \in Z_p^{n\times n}$*. Then there exists a randomized polynomial time algorithm that computes the permanent.*

Note that the randomness of the algorithm in the antecedent of the theorem is over A and p, whereas the randomness of the second algorithm is over its internal coins. The typical classification of problems into complexity classes depends on worst case analysis, which does not always correspond to how algorithms are used in practice. However, here Lipton's result makes a powerful statement; random instances of the permanent is as hard as worst case instances. We now proceed to give a proof.

Proof Suppose we treat the entries of an n by n matrix M as indeterminates. Then $\text{perm}(M) = \sum_\pi \prod_{i=1}^n X_{i,\pi(i)}$, which is a function of n^2 variables with degree n. Say we wish to compute $\text{perm}(M)$ with permanent $\leq 2^{\sqrt{n}}$, e.g., when M has dimension $\leq 2^{\sqrt{n}}$. Pick a random prime $p \in [2^n]$. With high probability, $p \geq 2^n/n^2$. Pick a random $R \in Z_p^{n\times n}$. Define $g(t) = \text{perm}(M + tR)$, which becomes a function in variable, namely t, with degree at most n. We want to compute $\text{perm}(M) = g(0)$. This can be done if we know all the coefficients of g. Since g has degree at most n, we will compute $g(i)$ for $i = 1, \ldots, n$, which we shall see is easier to compute. And thus, these $n + 1$ points will define g.

Since we are working over Z_p, for $i = 1, \ldots, n$, iR is a random matrix iff R is random. So $M + iR$ is a random matrix as well. So $g(i) = \text{perm}(M + iR)$ is the permanent of a random matrix.

By assumption, a random instance of the permanent is easy to compute. In particular, suppose we are given an algorithm B such that $B(A, p)$ computes $\text{perm}(A)$ mod p in expected n^3 time, for random A and p. Then by our assumption, $B(A)$ must compute $\text{perm}(A)$ mod p in at most n^{10} time for all but n^{-5} fraction of (A, p). So by the Union bound, with probability $1 - n^{-4}$, we have the correct values of $g(1), \ldots, g(n)$. ■

4 Toda's Theorem

In a complete surprise in 1989, Toda showed that $\text{PH} \subset \text{P}^{\#\text{P}}$. We give a quick introduction and save the proof until after spring break. Toda defined operators like BP, $\oplus$, and P that act on complexity classes. These operators essentially capture questions like is there a majority of instances in the language with the following properties. In particular, $L \in \oplus \cdot C$ if there exists M, $L(M) \in C$, such that

$$x \in L \text{ iff } |\{y|M(x,y) \text{ accepts}\}| \text{ is odd.}$$

This allows us to ask for the least significant bit, which is a much richer notion than the most significant bit (since we can approximate the permanent). [1]

Also, we can define $L \in \mathrm{BP} \cdot C$ if there exists M, $L(M) \in C$, such that $x \in L$ implies $\Pr[M(x,y) \text{ accepts}] \geq 2/3$, and $x \not{i}nL$ implies $\Pr[M(x,y) \text{ accepts}] \leq 1/3$.

We will define more operators next time and what they mean next time. In particular, we will show $\mathrm{BP} \oplus \mathrm{P} \subset \mathrm{P}^{\#\mathrm{P}}$ and $\Sigma_i^p \subset \mathrm{BP} \oplus \mathrm{P}$. The latter can be shown by an inductive argument roughly as follows.

$$\begin{aligned}
\Sigma_i^p &\subset \Sigma \cdot \mathrm{BP} \oplus \mathrm{P}, \text{ by induction} \\
&\subset \mathrm{BP} \oplus \mathrm{BP} \oplus \mathrm{P} \text{ base case} \\
&\subset \mathrm{BP}\,\mathrm{BP} \oplus \oplus\, \mathrm{P} \text{ by commuting the operators} \\
&\subset \mathrm{BP} \oplus \mathrm{P} \text{ by collapsing the operators.}
\end{aligned}$$

The magic of these operators will be fully explored next time.

[1] As an exercise, show that approximating #P to a multiplicative factor can be done in PH. Very recently, Jerrum, Sinclair, and Vigola showed in 2001 that approximating the permanent can be done with a BPP algorithm.