Notes on Complexity Theory: Fall 2005 Last updated: October, 2005

Lecture 9

Jonathan Katz

1 Counting and $\#\mathcal{P}$

Last time we introduced the class $\#\mathcal{P}$ and defined $\#\mathcal{P}$-completeness. We continue with our discussion of the hardness of computing the permanent.

1.1 Computing the Permanent is $\#\mathcal{P}$-Complete

The permanent of a square matrix $A = \{a_{i,j}\}$ is defined as:

$$\sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i,\pi(i)} ,$$

where S_n is the set of permutations on n elements. You might recognize that this formula is very similar to the formula defining the *determinant* of a matrix; the difference is that in the case of the determinant there is an extra factor of $(-1)^{\sigma(\pi)}$ in the sum, where $\sigma(\pi)$ is the sign of π. Nevertheless, although the determinant can be computed in polynomial time, computing the permanent (even of a 0-1 matrix) is $\#\mathcal{P}$-complete.

We should say a word about why the problem is in $\#\mathcal{P}$ (since it does not seem to directly correspond to a counting problem). The reason is that computing the permanent is equivalent to (at least) two other problems on graphs. In particular: (1) Computing the number of perfect matchings in a bipartite graph with n vertices in each component is equivalent to computing the permanent of the $n \times n$ matrix $A = \{a_{i,j}\}$ defined by $a_{i,j} = 1$ iff there is an edge from vertex i (in the left component) to vertex j (in the right component). (2) Computing the number of cycle covers in an n-vertex directed graph (with self-loops) is equivalent to computing the permanent of the adjacency matrix of the graph. (A *cycle cover* in an n-vertex graph is a set of edges such that each vertex has exactly one incoming edge and one outgoing edge in the set. It is not hard to see that any cycle cover corresponds to a permutation π on $[n]$ such that $(i, \pi(i))$ is an edge for all i [and hence the connection to computing the permanent].) Continuing the discussion from the last subsection, we remark that determining *existence* of a perfect matching (or of a cycle cover) can be done in polynomial time; it is *counting* the number of solutions that is hard.

The following result is due to Valiant [3]:

Theorem 1 *Computing the permanent of a 0-1 matrix is $\#\mathcal{P}$-complete.*

The proof is rather technical, so we will skip it. (The reader is referred to [2, Chap. 18] for a proof, and you may also ask me for a high-level proof sketch which will help in understanding the former.)

2 Approximate Counting, and the Relation of $\mathcal{NP}$ to $\#P$

$\#\mathcal{P}$ is clearly not weaker than $\mathcal{NP}$, since if we can count solutions we can certainly tell if any exist. Although $\#\mathcal{P}$ is (in some sense) "harder" than $\mathcal{NP}$, we show that it is not *too* much harder in

the following sense: any problem in $\#\mathcal{P}$ can be probabilistically *approximated* in polynomial time using an $\mathcal{NP}$ oracle. (This is reminiscent of the problem of reducing search to decision, except that here we are reducing *counting* the number of witness to the decision problem of whether or not a witness exists. Also, we are only obtaining an approximation, and we use randomization.) Specifically, we focus on the $\#\mathcal{P}$-complete problem $\#SAT$. Let $\#SAT(\phi)$ denote the number of satisfying assignments of a boolean formula ϕ. We show that for any polynomial p there exists a PPT algorithm A such that

$$\Pr\left[\#SAT(\phi)\cdot\left(1-\frac{1}{p(|\phi|)}\right)\leq A^{\mathcal{NP}}(\phi)\leq\#SAT(\phi)\cdot\left(1+\frac{1}{p(|\phi|)}\right)\right]\geq 1-2^{-|\phi|}\,; \tag{1}$$

that is, A approximates $\#SAT(\phi)$ to within a factor $(1\pm\frac{1}{p(|\phi|)})$ with high probability. Our proof of this assertion proceeds in a number of steps. We first prove that finding a *constant-factor* approximation suffices:

Proposition 2 *Assume there exists a* PPT *algorithm B such that for some constant $i\geq 0$ we have*

$$\Pr\left[\#SAT(\phi)\cdot 2^{-i}\leq B^{\mathcal{NP}}(\phi)\leq\#SAT(\phi)\cdot 2^{i}\right]\geq 1-2^{-|\phi|}.$$

Then for any polynomial p there exists a PPT *algorithm A such that (1) holds.*

Proof We prove the Proposition by describing A as if it were given oracle access to B (it is clear that when A is given oracle access to $\mathcal{NP}$ it can simulate the actions of B itself). We actually prove a slightly different result: that if B outputs a "good" approximation with probability 1, then A outputs a "good" approximation with probability 1 as well (extending this to prove the proposition as stated is left as an easy exercise).

Note that if $i=0$ then B computes $\#SAT(\phi)$ exactly and we are done. So assume $i>0$ and set $q(n)=i\cdot p(n)$ (which is polynomial). Construct A as follows: on input ϕ construct the formula

$$\phi'\stackrel{\text{def}}{=}\bigwedge_{i=1}^{q(|\phi|)}\phi(\vec{x}^i)\,,$$

where the $\vec{x}^i$ denote independent sets of variables. Note that if N' (resp., N) is the number of satisfying assignments of ϕ' (resp., ϕ), then $N'=N^{q(|\phi|)}$. Now, by calling $B(\phi')$ we obtain a t such that $N'\cdot 2^{-i}\leq t\leq N'\cdot 2^{i}$, implying:

$$2^{-i}\cdot N^{q(|\phi|)}\leq t\leq 2^{i}\cdot N^{q(|\phi|)}.$$

A then outputs the value $t^{1/q(|\phi|)}$, which lies in the range

$$\left[2^{-1/p(|\phi|)}\cdot N,\quad 2^{1/p(|\phi|)}\cdot N\right]\subseteq\left[\left(1-\frac{1}{p(|\phi|)}\right)\cdot N,\quad\left(1+\frac{1}{p(|\phi|)}\right)\cdot N\right],$$

as desired. In the last step, we use the following inequalities which hold for all $x\geq 1$:

$$\left(\frac{1}{2}\right)^{1/x}\geq\left(1-\frac{1}{x}\right)\quad\text{and}\quad 2^{1/x}\leq\left(1+\frac{1}{x}\right).$$

■

So, we may now restrict ourselves to finding a (probabilistic) constant-factor approximation for $\#SAT$. Before continuing, we define the notion of a *promise problem* (which crops up in many other contexts as well):

Definition 1 A *promise problem* is defined by disjoint sets Π_Y, Π_N, where the only requirements for an algorithm "solving" the problem are for inputs in one of these two sets (promise problems are therefore a generalization of languages where $L = \Pi_Y$ and $\Pi_N = \{0,1\}^* \setminus \Pi_Y$). For example:

- The class promise-$\mathcal{P}$ consists of promise problems (Π_Y, Π_N) for which there exists a poly-time algorithm M such that:

$$x \in \Pi_Y \Rightarrow M(x) = 1 \quad \text{and} \quad x \in \Pi_N \Rightarrow M(x) = 0.$$

- The class promise-$\mathcal{RP}$ consists of promise problems (Π_Y, Π_N) for which there exists a PPT algorithm M such that:

$$x \in \Pi_Y \Rightarrow \Pr[M(x) = 1] \geq \frac{1}{2} \quad \text{and} \quad x \in \Pi_N \Rightarrow \Pr[M(x) = 1] = 0.$$

(Note that nothing is required in either case when $x \notin \Pi_Y \cup \Pi_N$.) ♢

We show that approximating $\#SAT$ to within a constant factor can be reduced to solving the following promise problem $\mathsf{Gap} = (\Pi_Y, \Pi_N)$ regarding $\#SAT$:

$$\begin{aligned} \Pi_Y &\stackrel{\text{def}}{=} \{(\phi, k) \mid \#SAT(\phi) > 8k\} \\ \Pi_N &\stackrel{\text{def}}{=} \{(\phi, k) \mid \#SAT(\phi) < k/8\}. \end{aligned}$$

Proposition 3 *There exists a (deterministic) polynomial-time algorithm B which approximates $\#SAT$ to within a factor of 64 given access to an oracle that solves* Gap.

Proof We show how to approximate $\#SAT$ to within a factor of 64, given an oracle M solving Gap. On input ϕ, run the following algorithm:

- Set $i = 0$.
- While $M((\phi, 8^i)) = 1$, increment i.
- Return $8^{i-\frac{1}{2}}$.

Let i^* be the value of i at the end of the algorithm, and set $\alpha = \log_8 \#SAT(\phi)$. In the second step, we know that $M((\phi, 8^i))$ outputs 1 as long as $\#SAT(\phi) > 8^{i+1}$ or, equivalently, $\alpha > i+1$. So we end up with an i^* satisfying $i^* \geq \alpha - 1$. We also know that $M((\phi, 8^i))$ will output 0 whenever $i > \alpha + 1$ and so the algorithm above must stop at the first (integer) i to satisfy this. Thus, $i^* \leq \alpha + 2$. Putting this together, we see that our output value satisfies:

$$\#SAT/64 < 8^{i^* - \frac{1}{2}} < 64 \cdot \#SAT,$$

as desired. (Note that we assume nothing about the behavior of M when $(\phi, 8^i) \notin \Pi_Y \cup \Pi_N$.) ■

Our original problem of approximating $\#\mathcal{P}$ using an $\mathcal{NP}$ oracle has thus been reduced to solving Gap using an oracle for SAT. The reductions we have shown in the previous two propositions are deterministic, but the reduction that follows will be probabilistic. We have already discussed

(briefly) that the proof of Proposition 2 easily generalizes when B may err. The proof of Proposition 3, however, seems (at first) to rely on the fact that the oracle solving Gap is always correct (but in the reduction that follows, we will only solve Gap probabilistically). The key point is that as long as the error probability of the oracle is bounded[1] by a constant (say), we may use error reduction (and repeated calls to the oracle) to reduce the probability of error to a negligible quantity. Then applying a union bound over all the (polynomially-many) queries to the oracle, we see that with all but negligible probability the oracle does not make any mistakes during a run of the algorithm. Thus, we get a reduction which works with all but negligible probability.

The idea to solve Gap using an oracle for SAT is as follows: given an instance (ϕ, k) we know that there are either very many (i.e., more than $8k$) or very few (i.e., fewer than $k/8$) solutions. If we could take a random $1/k$ fraction of the assignments, with high probability in the first case there will still exist a satisfying assignment while in the second case there likely will not. We will take a random $1/k$ fraction of the satisfying assignments using (what else?) pairwise-independent hashing, relying on the following result:

Proposition 4 *Let $H_{n,m}$ be a family of pairwise-independent hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$, and let $\varepsilon > 0$. Let $S \subseteq \{0,1\}^n$ be arbitrary with $|S| \geq \varepsilon^{-3} \cdot 2^m$. Then:*

$$\Pr_{h \in H_{n,m}} \left[(1-\varepsilon) \cdot \frac{|S|}{2^m} \leq |\{x \in S \mid h(x) = 0^m\}| \leq (1+\varepsilon) \cdot \frac{|S|}{2^m} \right] > 1 - \varepsilon.$$

Proof Define for each $x \in S$ an indicator random variable δ_x such that $\delta_x = 1$ iff $h(x) = 0^m$ (and 0 otherwise). Note that the δ_x are pairwise independent random variables with expectation 2^{-m} and variance $2^{-m} \cdot (1 - 2^{-m})$. Let $Y \stackrel{\text{def}}{=} \sum_{x \in S} \delta_x = |\{x \in S \mid h(x) = 0^m\}|$. The expectation of Y is $|S|/2^m$, and its variance is $\frac{|S|}{2^m} \cdot (1 - 2^{-m})$ (using pairwise independent of the δ_x). Using Chebychev's inequality, we obtain:

$$\begin{aligned} \Pr\left[(1-\varepsilon) \cdot \mathbf{Exp}[Y] \leq Y \leq (1+\varepsilon) \cdot \mathbf{Exp}[Y]\right] &= \Pr\left[|Y - \mathbf{Exp}[Y]| \leq \varepsilon \cdot \mathbf{Exp}[Y]\right] \\ &\geq 1 - \frac{\mathbf{Var}[Y]}{(\varepsilon \cdot \mathbf{Exp}[Y])^2} \\ &= 1 - \frac{(1-2^{-m}) \cdot 2^m}{\varepsilon^2 \cdot |S|}, \end{aligned}$$

which is greater than $1 - \varepsilon$ for $|S|$ as stated in the proposition. ■

We now show the main result of this section:

Theorem 5 *There exists a* PPT *algorithm G which probabilistically solves $\mathsf{Gap} = (\Pi_Y, \Pi_N)$ given access to an oracle for SAT. Specifically, G satisfies:*

$$\begin{aligned} (\phi, k) \in \Pi_Y &\Rightarrow \Pr\left[G^{SAT}(\phi, k) = 1\right] > \frac{1}{2} \\ (\phi, k) \in \Pi_N &\Rightarrow \Pr\left[G^{SAT}(\phi, k) = 1\right] < \frac{1}{4}. \end{aligned}$$

Proof We describe G on input (ϕ, k). First note that if $k = 1$ then a reduction to SAT is immediate (since ϕ is unsatisfiable iff $(\phi, 1) \in \Pi_N$). So, assume $k > 1$ and set $m = \lfloor \log k \rfloor$. Let ϕ

[1] This is for the case of one-sided error. For two-sided error we need the "gap" in acceptance probabilities to be bounded by, e.g., a constant.

have n variables. Choose a random h from a family of pairwise-independent hash functions $H_{n,m}$ and let $\phi'(\vec{x})$ denote the proposition $\phi(\vec{x}) \wedge (h(\vec{x}) = 0^m)$ (this can either be Karp-reduced to a SAT formula, or can be directly converted to one when specific hash families are used). Finally, G queries ϕ' to its SAT oracle and outputs 1 iff ϕ' is satisfiable.

We claim that if $(\phi, k) \in \Pi_Y$ then ϕ' is satisfiable with "high" probability, while if $(\phi, k) \in \Pi_N$ then ϕ' is satisfiable with "low" probability:

Case 1: $\#SAT(\phi) > 8k$**.** Let $S_\phi = \{\vec{x} \mid \phi(\vec{x}) = 1\}$. Then $|S_\phi| > 8k \geq 8 \cdot 2^m$. So:

$$\begin{aligned} \Pr\left[\phi' \in SAT\right] &= \Pr\left[\{\vec{x} \in S_\phi : h(\vec{x}) = 0^m\} \neq \emptyset\right] \\ &\geq \Pr\left[\,|\{\vec{x} \in S_\phi : h(\vec{x}) = 0^m\}| \geq 4\right] \;\geq\; \frac{1}{2}, \end{aligned}$$

which we obtain by applying Proposition 4 with $\varepsilon = \frac{1}{2}$.

Case 2: $\#SAT(\phi) < k/8$**.** Let S_ϕ be as before. Now $|S_\phi| < k/8 \leq 2^m/4$. So:

$$\begin{aligned} \Pr\left[\phi' \in SAT\right] &= \Pr\left[\{x \in S_\phi : h(\vec{x}) = 0^m\} \neq \emptyset\right] \\ &\leq \sum_{\vec{x} \in S_\phi} \Pr\left[h(\vec{x}) = 0^m\right] \\ &< \frac{2^m}{4} \cdot 2^{-m} \;=\; \frac{1}{4}, \end{aligned}$$

where we have applied a union bound in the second step. ■

The above — showing that we can solve Gap (and thus approximate $\#SAT$) using an oracle for SAT — is the main result of this section. However, it is also interesting to note that we can solve SAT using an oracle for Gap, by relying on the following lemma:

Lemma 6 *There is a (deterministic) polynomial-time machine M which takes as input a boolean formula ϕ and outputs a boolean formula ϕ', such that:*

$$\#SAT(\phi') = 15 \cdot \#SAT(\phi).$$

Specifically, if ϕ is satisfiable then ϕ' has at least 15 solutions.

Proof Introduce new variables x_1, x_2, x_3, x_4 (not in ϕ), and set $\phi' = \phi \bigwedge (x_1 \vee x_2 \vee x_3 \vee x_4)$. ■

3 Reducing SAT to Unique SAT

In the spirit of the final paragraph of the previous section (noting that SAT can be reduced to an oracle solving Gap), we further explore the idea of reducing SAT to ever-weaker oracles solving promise problems. Along the way we will also address the following question: does it become any easier to decide whether a formula ϕ is satisfiable if we are *guaranteed* that ϕ has at most one solution? Equivalently,[2] does it become any easier to *find* a solution if we are guaranteed that ϕ has exactly[3] one solution? Toward this end, define the promise problem uniqueSAT as follows:

$$\begin{aligned} \Pi_Y &\stackrel{\text{def}}{=} \{\phi : \#SAT(\phi) = 1\} \\ \Pi_N &\stackrel{\text{def}}{=} \{\phi : \#SAT(\phi) = 0\}. \end{aligned}$$

[2] Equivalence is left as an exercise.

[3] As is the case, e.g., for Sudoku.

The questions introduced above can then be re-phrased as asking whether uniqueSAT is easier than SAT. We will show now that the answer is "no," in the sense that an oracle for uniqueSAT can be used to (probabilistically) solve SAT.

We first show that if we can solve uniqueSAT (and thus distinguish formulae having no solution and those having one solution) then we can also distinguish between formulae having no solution and those having "few" solutions.

Lemma 7 *Define the promise problem fewSAT as follows:*

$$\begin{aligned} \Pi_Y &\stackrel{\text{def}}{=} \{\phi : 1 \leq \#SAT(\phi) < 100\} \\ \Pi_N &\stackrel{\text{def}}{=} \{\phi : \#SAT(\phi) = 0\}. \end{aligned}$$

There is a deterministic polynomial-time algorithm to solve fewSAT given access to an oracle for uniqueSAT. The algorithm can additionally output $\#SAT(\phi)$ when $\phi \in \Pi_Y$. (The constant 100 in the above is arbitrary, and could be replaced with any polynomial function of $|\phi|$.)

Proof On input a formula ϕ we construct, for $1 \leq i < 100$, a formula ϕ_i such that:

- If ϕ has fewer than i solutions, then ϕ_i is unsatisfiable.
- If ϕ has *exactly* i solutions, then ϕ_i has a *unique* solution.

We can then solve fewSAT by calling our oracle for uniqueSAT on each of the ϕ_i (and outputting 1 iff any the answers are 1). The number of solutions to ϕ (assuming we indeed output 1) is simply the largest i for which our oracle outputs 1.

It remains to describe the construction of ϕ_i: In words, take i independent copies of ϕ (with independent variable sets $\{\vec{x}^j\}$) and require that the solutions be ordered lexicographically; i.e.,

$$\phi_i \stackrel{\text{def}}{=} \left(\bigwedge_{j=1}^{i} \phi(\vec{x}^j) \right) \wedge \left(\bigwedge_{j=1}^{i-1} \vec{x}^j <_{lex} \vec{x}^{j+1} \right).$$

■

With the above in place we may now easily prove the following result:

Theorem 8 (Valiant-Vazirani [4]) *There exists a* PPT *algorithm U which probabilistically solves SAT given access to an oracle for uniqueSAT, That is:*

$$\begin{aligned} \phi \in SAT &\Rightarrow \Pr[U^{\text{unique}SAT}(\phi) = 1] \geq \frac{1}{2} \\ \phi \notin SAT &\Rightarrow \Pr[U^{\text{unique}SAT}(\phi) = 1] = 0. \end{aligned}$$

Thus, if uniqueSAT is solvable in polynomial time then $\mathcal{NP} \subseteq \mathcal{RP}$.

Proof Given formula ϕ, algorithm U proceeds as follows:

1. Let n denote the number of variables in ϕ. Guess a value $i \in \{0, \ldots, n-1\}$. We will view this as a guess that (assuming ϕ is satisfiable)

   $$2^i \leq \#SAT(\phi) \leq 2^{i+1}.$$

   Note that if ϕ is satisfiable then this guess is indeed correct with probability at least $1/n$.

2. If we guess $i \leq 3$ then we are guessing that there are a constant number of solutions to ϕ, and we can proceed as in Lemma 7 to reduce the problem to uniqueSAT.

3. Otherwise, $i > 3$. As in the proof of Theorem 5, choose at random a hash function $h \in H_{n',i-3}$ and set
$$\phi'(\vec{x}) = \phi(\vec{x}) \wedge (h(\vec{x}) = 0^{i-3})$$
(where $\vec{x}$ denotes the variables in ϕ). Next, proceed as in Lemma 7 assuming that ϕ' has between 4 and 24 solutions, inclusive.

Note that if the original ϕ is not satisfiable, then we never output 1. Thus, we need only to show that when ϕ *is* satisfiable we output 1 with inverse polynomial probability (we can then increase our chances of acceptance by iterating the algorithm). Recall that our guess i is correct with probability at least $1/n$. When our guess is correct and $i \leq 3$, we always output 1. When our guess is correct and $i > 3$, let N (resp., N') denote the number of solutions to ϕ (resp., ϕ'). Proposition 4 (with $\varepsilon = 1/2$) tells us that with probability greater than $1/2$ (over choice of h) we have:
$$\frac{1}{2} \cdot \frac{N}{2^{i-3}} \leq N' \leq \frac{3}{2} \cdot \frac{N}{2^{i-3}}.$$
Since our guess i is correct, we know $2^i \leq N \leq 2^{i+1}$; if the above holds, then $4 \leq N' \leq 24$ (and we will indeed output 1). We conclude that when our guess is correct U outputs 1 with probability at least $1/2$; putting everything together, when ϕ is satisfiable U outputs 1 with probability at least $1/2n$. Applying (one-sided) error reduction gives the result of the theorem. ■

Examining the proof, we see the theorem holds even if uniqueSAT is solvable by an $\mathcal{RP}$ algorithm (which may sometimes output 0 when $\phi \in \Pi_Y$, but never outputs 1 when $\phi \in \Pi_N$). Thus:

Corollary 9 *If unique$SAT \in$ promise-$\mathcal{RP}$ then $\mathcal{NP} \subseteq \mathcal{RP}$.*

By modifying the proof we can show that unique$SAT \in$ promise-$\mathcal{BPP}$ implies $\mathcal{NP} \subseteq \mathcal{BPP}$. (We remark that $\mathcal{NP} \subseteq \mathcal{BPP}$ implies $\mathcal{NP} \subseteq \mathcal{RP}$.)

Bibliographic Notes

Sections 2 and 3 are loosely based on [1, Lect. 10].

References

[1] O. Goldreich. Introduction to Complexity Theory (July 31, 1999).

[2] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1995.

[3] L.G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science* 8: 189–201, 1979.

[4] L.G. Valiant and V.V. Vazirani. NP is as Easy as Detecting Unique Solutions. *Theoretical Computer Science* 47(1): 85–93, 1986.