Notes for Lecture 6

1 Kannan's Theorem

Although it is open to prove that the polynomial hierarchy is not contained in $\mathbf{P}/poly$, it is not hard to prove the following result.

Theorem 1 *For every polynomial $p()$, there is a language $L \in \Sigma_4$ such that $L \notin \mathbf{SIZE}(O(p(n)))$.*

Note that Theorem 1 is not saying that $\Sigma_4 \not\subseteq \mathbf{P}/poly$, because for that to be true we would have to be able to construct a single language L such that for every polynomial p we have $L \notin \mathbf{SIZE}(p(n))$, instead of constructing a different language for each polynomial. (This is an important difference: the time hierarchy theorem gives us, for every polynomial $p()$, a language $L \in \mathbf{P}$ such that $L \notin \mathbf{DTIME}(p(n))$, but this doesn't mean that $\mathbf{P} \neq \mathbf{P}$.)

Kannan observed the following consequence of Theorem 1 and of the Karp-Lipton theorem.

Theorem 2 *For every polynomial $p()$, there is a language $L \in \Sigma_2$ such that $L \notin \mathbf{SIZE}(O(p(n)))$.*

Proof: We consider two cases:

- if $3SAT \notin \mathbf{SIZE}(O(p(n)))$; then we are done because $3SAT \in \mathbf{NP} \subseteq \Sigma_2$.
- if $3SAT \in \mathbf{SIZE}(O(p(n)))$, then $\mathbf{NP} \subseteq \mathbf{P}/poly$, so by the Karp-Lipton theorem we have $\Sigma_4 = \Sigma_2$, and the language $L \in \Sigma_4 - \mathbf{SIZE}(O(p(n)))$ given by Theorem 1 is in Σ_2.

□

2 Counting Classes

Recall that R is an $\mathbf{NP}$-*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 3 *If R is an $\mathbf{NP}$ relation, then $\#R$ is the problem that, given x, asks how many y satisfy $(x, y) \in R$.*

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

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

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

Definition 4 *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 5 *$\#A \leq \#B$ if there is a polynomial time algorithm for $\#A$ given an oracle that solves $\#B$.*

$\#CSAT$ is the problem where given a circuit, we want to count the number of inputs that make the circuit output 1.

Theorem 6 *$\#CSAT$ is $\#\mathbf{P}$-complete under parsimonious reductions.*

PROOF: Let $\#R$ be in $\#\mathbf{P}$ 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 7 *$\#3SAT$ is $\#\mathbf{P}$-complete.*

PROOF: We show that there is a parsimonious reduction from $\#CSAT$ to $\#3SAT$. 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 $\#CSAT$ to $\#3SAT$. $\square$

Notice that if a counting problem $\#R$ is $\#\mathbf{P}$-complete under parsimonious reductions, then the associated language L_R is $\mathbf{NP}$-complete, because $\#3CSAT \leq_{par} \#R$ implies $CSAT \leq L_R$. On the other hand, with the less restrictive definition of reducibility, even some counting problems whose decision version is in $\mathbf{P}$ are $\#\mathbf{P}$-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 $\#\mathbf{P}$-complete.

3 Complexity of counting problems

We will prove the following theorem:

Theorem 8 *For every counting problem $\#A$ in $\#\mathbf{P}$, there is a probabilistic algorithm C that on input x, computes with high probability a value v such that*

$$(1 - \epsilon)\#A(x) \leq v \leq (1 + \epsilon)\#A(x) \tag{1}$$

in time polynomial in $|x|$ and in $\frac{1}{\epsilon}$, using an oracle for $\mathbf{NP}$.

The theorem says that $\#\mathbf{P}$ can be approximate in $\mathbf{BPP^{NP}}$. We remark that approximating $\#CSAT$ is $\mathbf{NP}$-hard, and so to compute an approximation we need at least the power of $\mathbf{NP}$. Theorem 8 states that the power of $\mathbf{NP}$ and randomization is sufficient.

Another remark concerns the following result.

Theorem 9 (Toda) *For every k, $\Sigma_k \subseteq \mathbf{P}^{\#\mathbf{P}}$.*

This implies that $\#CSAT$ is Σ_k-hard for every k, i.e., $\#CSAT$ lies outside the polynomial hierarchy, unless the hierarchy collapses. Recall that $\mathbf{BPP}$ lies inside Σ_2, and hence approximating $\#CSAT$ can be done in Σ_3. Therefore, approximating $\#CSAT$ cannot be equivalent to computing $\#CSAT$ 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 $\#CSAT$.

  If we have an approximation algorithm for $\#CSAT$, we can extend it to any #A in $\#\mathbf{P}$ using the parsimonious reduction from #A to $\#CSAT$.

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

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

  $$\frac{1}{c}\#CSAT(C) \leq A(C) \leq c\#CSAT(C). \tag{2}$$

  Given a circuit C, we can construct $C^k = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ where each C_i is a copy of C constructed using fresh variables. If C has t satisfying assignments, C^k has t^k satisfying assignments. Then, giving C^k to the algorithm we get

  $$\begin{aligned}\frac{1}{c}t^k &\leq A(C^k) \leq ct^k \\ \frac{1}{c}^{1/k}t &\leq A(C^k)^{1/k} \leq c^{1/k}t.\end{aligned}$$

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

- For a circuit C that has $O(1)$ satisfying assignments, $\#CSAT(C)$ can be computed in $\mathbf{P^{NP}}$.

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

[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 $\mathbf{BPP^{NP}}$ 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.

4 Using an approximate comparison procedure

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

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

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

- Input: C
- compute:
  - `a-comp`$(C, 0)$
  - `a-comp`$(C, 1)$
  - `a-comp`$(C, 2)$
  - $\vdots$
  - `a-comp`(C, n)
- 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 $\#CSAT$ 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 `a-comp`$C, i-1)$ outputs YES, $\#CSAT(C) \geq 2^{i-1}$, and also since `a-comp`$(C, 2^i)$ outputs NO, $\#CSAT(C) < 2^{i+1}$. The algorithm outputs $a = 2^i$. Hence,

$$\frac{1}{2}a \leq \#CSAT(C) < 2 \cdot a \tag{3}$$

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

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