Notes for Lecture 6

1 Kannan’s Theorem

Although it is open to prove that the polynomial hierarchy is not contained in $\text{P}/\text{poly}$, 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 $L \not\in \text{SIZE}(O(p(n)))$.

Note that Theorem 1 is not saying that $\Sigma_4 \not\subseteq \text{P}/\text{poly}$, 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 $L \not\in \text{SIZE}(p(n))$, 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 $L \in \text{P}$ such that $L \not\in \text{DTIME}(p(n))$, but this doesn’t mean that $\text{P} \not= \text{P}$.)

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 $L \not\in \text{SIZE}(O(p(n)))$.

**Proof:** We consider two cases:

- if $\text{3SAT} \not\in \text{SIZE}(O(p(n)))$; then we are done because $\text{3SAT} \in \text{NP} \subseteq \Sigma_2$.
- if $\text{3SAT} \in \text{SIZE}(O(p(n)))$, then $\text{NP} \subseteq \text{P}/\text{poly}$, so by the Karp-Lipton theorem we have $\Sigma_4 = \Sigma_2$, and the language $L \in \Sigma_4 - \text{SIZE}(O(p(n)))$ given by Theorem 1 is in $\Sigma_2$.

$\square$

2 Counting Classes

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

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

Observe that an $\text{NP}$-relation $R$ naturally defines an $\text{NP}$ language $L_R$, where $L_R = \{x : \exists y. (x, y) \in R\}$, and every $\text{NP}$ language can be defined in this way. Therefore problems in $\#\text{P}$ can always be seen as the problem of counting the number of witnesses for a given instance of an $\text{NP}$ 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 ≤_{par} \#B) if there is a polynomial time transformation f such that for all x, \{|y, (x, y) ∈ A| = |\{z : (f(x), z) ∈ B\}|\}.

Often this definition is a little too restrictive and we use the following definition instead.

Definition 5 \#A ≤ \#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 \#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| ≤ 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 7 \#3SAT is \#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 \(\phi\) such that the number of satisfying assignments for \(\phi\) 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 \(\phi\) 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 \(\phi\) with \(n + m\) variables and \(4m\) clauses. So there is a parsimonious reduction from \#CSAT to \#3SAT.

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

## 3 Complexity of counting problems

We will prove the following theorem:

Theorem 8 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

\[ (1 − \epsilon)\#A(x) ≤ v ≤ (1 + \epsilon)\#A(x) \]

in time polynomial in \(|x|\) and in \(\frac{1}{\epsilon}\), using an oracle for \textbf{NP}. 


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

Another remark concerns the following result.

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

This implies that \#CSAT is \(\Sigma_k\)-hard for every \(k\), i.e., \#CSAT lies outside the polynomial hierarchy, unless the hierarchy collapses. Recall that BPP lies inside \(\Sigma_2\), and hence approximating \#CSAT can be done in \(\Sigma_3\). Therefore, approximating \#CSAT cannot be equivalent to computing \#CSAT exactly, unless the polynomial hierarchy collapses.

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 \#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 \(C^k = C_1 \land C_2 \land \cdots \land C_k\) 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

\[
\frac{1}{c} t^k \leq A(C^k) \leq c t^k
\]

\[
\frac{1}{c} t^{1/k} \leq A(C^k)^{1/k} \leq c^{1/k} t.
\]

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 \(\text{P}^{NP}\).

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 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 BPP \(\subseteq \Sigma_2\) and that approximate counting is doable in BPP^{NP} it does not necessarily follow that approximate counting is in \(\Sigma_3\), although in this case it does because the proof that BPP \(\subseteq \Sigma_2\) relativizes; (ii) we have defined BPP, \(\Sigma_3\), 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 BPP \(\subseteq \Sigma_2\) holds also for promise problems.
4 Using an approximate comparison procedure

Suppose that we had available an approximate comparison procedure \( a\text{-comp} \) with the following properties:

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

Given \( a\text{-comp} \), we can construct an algorithm that 2-approximates \( \#CSAT \) as described below:

- Input: \( C \)
- compute:
  - \( a\text{-comp}(C, 0) \)
  - \( a\text{-comp}(C, 1) \)
  - \( a\text{-comp}(C, 2) \)
  - ... 
  - \( a\text{-comp}(C, n) \)
- if \( a\text{-comp} \) outputs \text{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 \text{NP} oracle.
  - Query to the oracle and output an exact value.
- else
  - Suppose that it outputs \text{YES} for \( t = 1, \ldots, i - 1 \) and \text{NO} for \( t = i \)
  - Output \( 2^i \)

We need to show that this algorithm approximates \( \#CSAT \) within a factor of 2. If \( a\text{-comp} \) answers \text{NO} from the first time, the algorithm outputs the right answer because it checks for the answer explicitly. Now suppose \( a\text{-comp} \) says \text{YES} for all \( t = 1, 2, \ldots, i - 1 \) and says \text{NO} for \( t = i \). Since \( a\text{-comp}(C, i - 1) \) outputs \text{YES}, \( \#CSAT(C) \geq 2^{i-1} \), and also since \( a\text{-comp}(C, 2^i) \) outputs \text{NO}, \( \#CSAT(C) < 2^{i+1} \). The algorithm outputs \( a = 2^i \). Hence,

\[
\frac{1}{2} a \leq \#CSAT(C) < 2 \cdot a
\]  

and the algorithm outputs the correct answer within a factor of 2.

Thus, to establish the theorem, it is enough to give a \( \text{BPP}^{\text{NP}} \) implementation of the \( a\text{-comp} \) procedure.