CMPT 407: Computational Complexity **Summer 2017**

Lecture 2: Poly-time Hierarchy - 23 26 May

Lecturer: Valentine Kabanets *Scribe: Lily Li*

2.1 Space Hierarchy

Similar to the Time Hierarchy theorems, it simply states that with more time we can do more work.

Theorem 2.1 *For any $n \leq s \leq S \leq 2^n/n$ such that $S \geq 10 \cdot s$, we have*

$$\mathsf{SIZE}(s) \subsetneq \mathsf{SIZE}(S)$$

Proof: First we will prove the following:

Theorem 2.2 *(Shannon-Lupanov) For all sufficiently large n, the maximum circuit complexity of an n-variate Boolean function is*

$$\frac{2^n}{n}\left(1+\Theta\left(\frac{\log n}{n}\right)\right)$$

More of the proof to come... ■

We see this using a counting argument. Consider the total number of boolean formulas: there are 2^n possible clauses and each clause could be in the formula so there are 2^{2^n} boolean formulas.

2.2 SAT Menagerie

Consider different types of SAT:

1. Circuit SAT - these look like general digraph.
2. Formula SAT - these look like trees (only one output from each node).
3. CNF SAT - we have a CNF formula.
4. $k-\mathsf{SAT}$ - each clause in the CNF has at most $k-\mathsf{SAT}$

Theorem 2.3 *All the above (including the last one with $k \geq 3$) are* NP*-complete.*

Proof: We already showed that Circuit-SAT is NP-complete. Next we will show that $3-\mathsf{SAT}$ is NP-complete to complete the proof. It should be easy to see that SAT is in NP. To show that SAT is NP-hard reduce Circuit-SAT to $3-\mathsf{SAT}$. Take a circuit $C(x_1, ..., x_n)$ as an input. We want to make an instance of $3-\mathsf{SAT}$ associated with C. Let $\phi(x_1, ..., x_{n^d})$ be our $3-\mathsf{SAT}$ instance. Label the gates of C as $g_1, ..., g_m$. Associate to each gate a set of 3-CNF clauses. Then conjunct all the clauses (don't forget the output gate). Yay! You have created a valid $3 - \mathsf{SAT}$ instance. ■

Consider the following trivial NP-complete language:

$$L_u = \{(M, x, 1^k) : NTMM \text{ accepts } x \text{ in } \leq k \text{ steps}\}$$

Proposition 2.4 *L_u is* NP*-complete.*

Proof:

■

Once SAT was proved NP-complete many more were proved to be NP-complete as well. Such as $\mathsf{NAE} - \mathsf{SAT}$ (not all equal SAT) as follows: (omitting proof that $\mathsf{NAE} - \mathsf{SAT} \in \mathsf{NP}$)

Other language of interest is coNP. Where languages in NP are those which have easy to verify *yes* instances coNP have easy to verify *no* instances. Not that NP and coNP are *NOT* disjoint since P is in both. It is unknown if $\mathsf{coNP} = \mathsf{NP}$.

Proposition 2.5 TAUT *is* coNP*-complete*

Proof:

■

2.3 Introduction to Polynomial Hierarchy

Polynomial hierarchy PH is the generalization of the classes P, NP, and coNP. There are an infinite number of subclasses in PH which are conjectured to be distinct (stronger version of $\mathsf{P} \neq \mathsf{NP}$. Three definitions of PH are as follows:

1. defined as the set of languages defined via polynomial-time predicates combined with a constant number of alternating for all and exists quantifiers, generalizing the definitions of NP and coNP.
2. defined interms **alternating TM** which generalize NTM.
3. defined using **oracle TM**.

Theorem 2.6 *(Fortnow)* $\mathsf{SAT} \notin \mathsf{TimeSpace}(n^{1.1}, n^{0.1})$. *This means you have $O(n^{1.1})$ time and $O(n^{0.1})$ space, then you definitely cannot solve* SAT*.*

Proof: We will use the following two ideas: (1) NTime-Hierarchy and (2) Poly Hierarchy (alternation).

First (2) denoted by PH. Note $\mathsf{NP}, \mathsf{coNP} \subset \mathsf{PH}$, where $\mathsf{NP} = \exists \bar{x} : \phi(x_1, ..., x_n)$ and $\mathsf{coNP} = \forall \bar{x} : \phi(x_1, ..., x_n)$, since PH is the set of first order logical statements with any number alternating $\forall$ and $\exists$. ■

2.4 Nondeterministic Time Hierarchy

Much like deterministic time hierarchy and space hierarchy, we need some way to say that with more time on a nondeterministic machine we can do more work. This is formalized in the following theorem. However, this time our standard diagonalization trick will not work because it is not know if $\mathsf{NP} = \mathsf{coNP}$.

Theorem 2.7 NTime ***Hierarchy Theorem*** *For every proper complexity function $f(n) \geq n$ and $g(n) \in \omega(f(n+1))$, we have*

$$\mathsf{NTime}(f(n)) \subsetneq \mathsf{NTime}(g(n))$$

Proof: This is requires a different tool than diagonalization. The difficulty is due to the lack of closure for complementation in NP (not known if coNP = NP). Enumerate all NTM $M_1, M_2, ...$ which are clocked to run in less than $f(n)$ steps over the unary alphabet (this strengthens the theorem). Let $L(M_i)$ be the language of machine M_i. Let $t(n)$ be a fast growing function split the natural numbers into intervals $[1, ..., t(n)], [t(n)+1, ..., t^{(2)}(n)], ...$ indexed $I_1, I_2, ...$ where interval $I_i = [t^{i-1}(n), ..., t^i(n)]$ where $t^k(n)$ is the composition of $t(n)$, k times. Define NTM D which diagonalizes all M_i. ∎

2.5 Decision to Search

Consider the difference between SAT (a decision problem) and the corresponding search problem SAT-search which asks for an assignment.

Theorem 2.8 $\mathsf{SAT} \in \mathsf{P} \implies \mathsf{SAT} - \mathsf{SEARCH}$ *is poly-time.*

Proof: Surprisingly intuitive! Image a SAT instance $\phi(x_1, ..., x_n)$ that you want to find an assignment for. Randomly assign $x_1 = 1$ and ask the polynomial SAT for the decidability of $\phi(1, ..., x_n)$. ∎

Suppose that we are given that $\mathsf{SAT} \in \mathsf{Time}(n^c)$ then we can actually find an explicit SAT algorithm that runs in $O(n^{2+c})$.

2.6 Polynomial Hierarchy

Definition 2.9 *For $i \geq 1$, a language L is in $\sup_2^P$ if there exists a polynomial-time TM M and a polynomial q such that*

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \cdots Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, ..., u_i) = 1$$

where Q_i is a $\forall$ or a $\exists$ depending if i is even or odd. ***Polynomial hierarchy (PH)*** *is* $\mathsf{PH} = \cup_i \sum_2^P$.

The polynomial hierarchy does not collapse.

Theorem 2.10 *The following hold:*

1. *For every $i \geq 1$, if $\sum_i^P = \prod_i^P$ then $\mathsf{PH} = \sum_i^P$ i.e. the hierarchy collapses to the ith level.*
2. *If $\mathsf{P} = \mathsf{NP}$ then $PH = P$ i.e. the hierarchy collapses to* P.

2.7 Alternating TM

2.8 Oracle Machines