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. $k-$SAT - each clause in the CNF has at most $k-$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 $3-$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 $3-$SAT. Take a circuit $C(x_1, ..., x_n)$ as an input. We want to make an instance of $3-$SAT associated with C. Let $\phi(x_1, ..., x_{n^d})$ be our $3-$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 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 2.6 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 $\mathsf{coNP} = \mathsf{NP}$). 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 and split the natural numbers into intervals $[1, ..., t(n)], [t(n)+1, ..., t^{(2)}(n)], ...$ indexed $I_1, I_2, ...$ where interval $I_i = [t^{i-1}(n)+1, ..., t^i(n)]$ and $t^k(n)$ is the composition of $t(n)$, k times. Define NTM D which diagonalizes all M_i as follows: give machine M_i interval i to work with for simplicity label this interval $[u_0, ..., u_m]$. The values of D for I_i is defined as: on input 1^{u_i} for $0 \leq i < m$, D accepts if M_i accepts $1^{u_{i+1}}$. On input 1^{u_m}, D accepts if M_i rejects 1^{u_0}. ■

Proposition 2.7 *There exists a universal NTM U which simulates a given NTM M on input x so that, if M takes time t on x, the the simulation by U takes at most $c_M \cdot t$ time for some constant c_M.*

2.4 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 $\phi(1, ..., x_n)$. If it says yes, then proceed on and randomly assign x_2. Otherwise assign $x_1 = 0$ and randomly assign x_2. ■

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 $O(n^{2+c})$.

Theorem 2.9 *(Levin's Universal Search) Suppose* $\mathsf{SAT} \in \mathsf{Time}(n^c)$ *for a known constant* $c > 0$ *for some black box algorithm. Then there exists an explicit polytime algorithm that solves* SAT *in time* $O(n^{c+2} \cdot \log^2 n + t_0(n) \cdot n)$*, where* $t_0(n) \in \mathsf{poly}(n)$ *is the time needed to check if a given assignment satisfies a specified* SAT *instance of size* n*.*

Proof: SAT-search is solved in $O(n^{c+1})$ by the above algorithm and using the black-box. The explicit algorithm to solve SAT takes input ϕ of size n and loops from 1 to n as follows: simulate TM M_i on ϕ for $O(n^{c+1} \cdot \log^2 n)$ steps. If M_i produces a satisfying assignment for ϕ except and halt. If the loop terminates and no satisfying assignment was found, reject. Why does this work? Think about it! ■

2.5 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 in-terms **alternating TM** which generalize NTM.
3. defined using **oracle TM**.

Theorem 2.10 *(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*.*

We will use the following two ideas: (1) NTime-Hierarchy and (2) Poly Hierarchy (alternation). Before we begin the proof we need some tools.

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 $\exists$.

Let a k-ary relation R be **polynomially balanced** if, for every tuple $(a_1, ..., a_k) \in R$, the lengths of all a_i's are polynomially related to each other.

Definition 2.11 *For any $i \geq 1$, a language $L \in \sum_i^p$ iff there is a polynomially balanced $(i+1)$-ary relation R such that*

$$L = \{x : \exists y_1 \forall y_2 \exists y_3 \cdots Q_i y_i R(x, y_1, ..., y_i)\}$$

where $Q_i \in \{\forall, \exists\}$ depending on the parity of i.

Definition 2.12 *For any $i \geq 1$, a language $L \in \prod_i^p$ iff there is a polynomially balanced $(i+1)$-ary relation R such that*

$$L = \{x : \forall y_1 \exists y_2 \forall y_3 \cdots Q_i y_i R(x, y_1, ..., y_i)\}$$

where $Q_i \in \{\forall, \exists\}$ depending on the parity of i.

Remark that $\mathsf{NP} \in \sum_1^p$ and $\mathsf{coNP} \in \prod_1^p$. In general $\prod_i^p = \mathsf{co}\sum_i^p$. $\mathsf{PH} = \cup_{i \geq 0} \sum_i^p$. Examples of PH include: $\mathsf{UNIQUE} - \mathsf{SAT} \in \sum_2^p$, $\mathsf{MIN} - \mathsf{CIRCUIT} \in \prod_2^p$.

Theorem 2.13 $\mathsf{PH} \subseteq \mathsf{PSPACE}$

Proof:

■

Another definition of PH does things recursively: $\sum_0^p = \prod_0^p = \mathsf{P}$. For all $i \geq 0$ define $\sum_i^p = \mathsf{NP}^{\prod_{i-1}^p}$ and $\prod_i^p = \mathsf{coNP}^{\sum_{i-1}^p}$.

Theorem 2.14 *This alternate definition is same as the previous one.*

Proof: By induction on i. ■

Proof: (Fortnow). ■