Undergraduate Complexity Theory
Lecture 14: Ladner's Theorem and Mahaney's Theorem

Marcythm

July 17, 2022

1 Lecture Notes

Theorem 1.1 (Ladner's Theorem)**.** *Assume* $\mathsf{P} \neq \mathsf{NP}$*, then* $\exists L \in \mathsf{NP}$ *s.t.* $L \notin \mathsf{P}$ *and* L *is not* NP*-complete.*

Today we are going to prove a weaker version of the Ladner's Theorem, with a stronger assumption that called E.T.H.

Definition 1.2 (Exponential Time Hypothesis)**.** $\exists \delta > 0 : \mathsf{3SAT} \notin \mathsf{TIME}(2^{\delta n})$

Proof. The idea to construct a not-so-hard language is based on padding: take hard problem $\mathsf{3SAT}$, kind of water it down halfway s.t. with the padding technique to make the input length kind of artificially longer s.t. will give some extra time to the algo but not so much.

Define $L :=$ "$\mathsf{3SAT}$ padded to length $2^{\sqrt{n}}$" $= \{\langle x, 1^{2^{\sqrt{|x|}}} \rangle : x \in \mathsf{3SAT}\}$

3 things to check:

1. $L \in \mathsf{NP}$. Prove this by directly giving an verifier.
2. $L \notin \mathsf{P}$. Prove this by contra: $\mathsf{3SAT}$ will have a subexp algo, which contradicts E.T.H.
3. L is not NP-complete. Prove this by contra: $\mathsf{3SAT} \leq_m^P L$, since L can be decided in time $N^{O(\log N)}$, $\mathsf{3SAT}$ can be solved in subexp time.

□

Definition 1.3. A language $L \in \{0,1\}^+$ is *sparse* if $|L \cap \{0,1\}^n| \leq O(n^c)$ for some c.

Q1: Maybe exists algo A solving SAT correctly, and running in poly(n) time on "almost all" inputs. Here "almost all" means $L = \{x : A(x)$ takes more than poly(n) time$\}$ is sparse.

Q2: Maybe exists algo A solving SAT in poly(n) time, and correctly on "almost all" inputs.

The Mahaney's Theorem says nope if assuming $\mathsf{P} \neq \mathsf{NP}$.

Theorem 1.4 (Mahaney's Theorem)**.** *Assume* $\mathsf{P} \neq \mathsf{NP}$*, then sparse* NP*-complete language not exists.*

Proof. idea: By contrapos. Suppose L is sparse and NP-hard, we'll show $\mathsf{SAT} \in \mathsf{P}$.

By assumption we know $\mathsf{SAT} \leq_m^{\mathsf{P}} L$, i.e. exists a reduction R s.t. $\phi \in \mathsf{SAT} \iff R(\phi) \in L$ which runs in poly time and $|R(\phi)| = O(|\phi|^s)$. Since L is sparse, suppose $|L \cap \{0,1\}^n| \leq n^r$, then we can derive the key property: let $T = n^{rs}$, for $T+1$ formulas $\phi_0, \phi_1, \ldots, \phi_T$ with length n, by Pigeon Hole's Principle, either exists two formulas ϕ_i, ϕ_j s.t. $R(\phi_i) = R(\phi_j)$, or $\exists i : \phi_i \notin \mathsf{SAT}$ since all formulas after reduction are distinct, while there are only at most T such strings in L.

We use the property this way: roughly, similar to the search-to-decision reduction, for instance ϕ, derive the full search tree like $\phi = \phi[x_1 \mapsto 0] \vee \phi[x_1 \mapsto 1] = \ldots$, and in each depth of the search tree, use the key property to restrict the size of all clauses in $T =$ poly(n).

More specifically, to check which case a formula ϕ_0 belongs to, use the following strategy: denotes current T formulas as $\phi_1, \ldots, \phi_T$, construct $\psi_1 = \phi_0 \vee \phi_1, \ldots, \psi_T = \phi_0 \vee \phi_T$, then calculate R for all ψs. If they are all distinct, then must exists a formula $\psi_i \notin \mathsf{SAT}$, thus ϕ_0 is unsatisfiable and we can drop it. If $R(\psi_i) = R(\psi_j)$,

i.e. ψ_i and ψ_j have the same satisfiability, then we can drop either ϕ_i or ϕ_j: if ϕ_0 is satisfiable, then drop which one won't affect the result; otherwise, if ϕ_0 is unsatisfiable, then ϕ_i and ϕ_j have the same satisfiability, thus drop which is the same.

All the steps above can be done in polynomial time, so the overall algorithm complexity is poly too. □