Undergraduate Complexity Theory
Lecture 28: Why is P vs. NP difficult?

Marcythm

July 30, 2022

1 Lecture Notes

P vs. NP:

1. discussed by Gödel in 1956 letter to von Neumann
2. formalized 1970
3. open for 60+ years …

progress towards proving it = negligible (?), why so hard?

Theorems about why P vs. NP is hard! Negative results we can prove:

1. HALTS is not computable
2. $\mathsf{EXP} \nsubseteq \mathsf{P}$, T.H.T.: $\exists A \in \mathsf{TIME}(T(n)) \setminus \mathsf{TIME}(o(T(n)/\log T(n)))$.

Both results use "diagonalization" / "simulation": simulate and do the opposite.

Hallmark of "simulation results": they hold equally well if both machines get the same oracle. e.g.

1. For any language A, $\exists L$ solvable by a time $T(n)$ A-oracle TM, but not by a $o(T(n)/\log T(n))$ one.
2. For any language A, $\mathsf{NP}^A \subseteq \mathsf{PSPACE}^A$.

"simulation / diagonalization arguments tend to go through word for word in any A-oracle world"

Theorem 1.1 (Baker-Gill-Solovay '75)**.** *$\exists$ language A, B s.t.* $\mathsf{P}^A = \mathsf{NP}^A, \mathsf{P}^B \neq \mathsf{NP}^B$.

This is a negative result about proof technique.

Proof. For A, TQBF is a valid choice.

$$\mathsf{NP}^{\mathsf{TQBF}} \subseteq \mathsf{NPSPACE} = \mathsf{PSPACE} \subseteq \mathsf{P}^{\mathsf{TQBF}}$$

Basic idea for B: make B some kind of sparse language, then a NTM can just like guess the location of strings in B, but a DTM cam not do so.

Given B, define $L_B = \{1^n : \exists x \in B, |x| = n\}$, $B_n := B \cap \{0,1\}^n$.

Claim 1.2. $\forall B : L_B \in \mathsf{NP}^B$.

Remaining task: design B s.t. $L_B \notin \mathsf{P}^B$. Construct B by diagonalization. Intuition: for each n, B_n will either be $\varnothing$ or very sparse. Every oracle-TM $M^?$ has an encoding $\langle M \rangle \in \{0,1\}^*$, thus has a bijection to $\mathbb{N}$. So for $i \in \mathbb{N}$ we'll write M_i for the ith oracle-TM. (Notice that one TM can have many different encodings, so we can just pick one with sufficiently large length.)

Design B in stages $i = 0, 1, \ldots$, the ith stage will be designed to beat M_i, i.e. ensure M_i doesn't decide L_B in poly time (actually much stronger, not in time $2^n/10$). At stage i:

1. pick suffciently large n s.t. haven't made any decision about B_n yet.
2. simulate $M_i^?(1^n)$, if it makes an oracle query to some string y that has not been decided whether in B yet, answer no, and irrevocably decide $y \notin B$.

Here we want to ensure M_i^B's answer about $1^n \overset{?}{\in} L_B$ (after $2^n/10$ steps) is wrong.

If $M_i(1^n)$ accepts, it thinks $1^n \in L_B \iff B_n \neq \varnothing$, then we just irrevocably decide $B_n = \varnothing$.

If $M_i(1^n)$ rejects, it thinks $1^n \notin L_B \iff B_n = \varnothing$, but it can't ask all strings with length n within time $2^n/10$, so there must be many strings not decided yet. Just irrevocably pick one, declare it is in B. □

In '70s, people kept trying to prove $\mathsf{P} \neq \mathsf{NP}$ anyway. In '80s, a new strategy became popular: try to prove a harder statement, since they really hate to reason about TMs. e.g. tried to show NP doesn't have poly-size circuit family.

[Håstad '88] $\exists L \in \mathsf{NP}$ s.t. L doesn't have poly size, constant depth circuit families. But in fact this language is also in P: the parity function.

[2017] Maybe $L \in \mathsf{NP}$ has poly-size log-depth circuits (maybe also for $L \in \mathsf{NEXP}$, unknown.)

[1994 Razborov-Rudich] Observed all known circuit lower bounds followed "natural" proof strategy. ref

Theorem 1.3. *Assuming well-believed hypothesis H, $\nexists$ "natural" proof that NP has no poly-size circuits.*

Here H = "good pesudorandom generators exist", is true if factoring product of two random n-bit primes is "hard", i.e. no 2^{n^ϵ} size circuits $\forall \epsilon > 0$. In other words, there are efficiently generatable random instances of hard problems, which is talked two lectures ago. ("random SAT instances are hard")

2 Reading

2.1 sipser 9.2 (Relativization)

Limits of the Diagonalization Method: it's a simulation of one TM by another, where the simulating TM can determine the behavior of the other TM and then behave differently. Thus if we can prove $\mathsf{P} = \mathsf{NP}$ by diagonalization, also $\mathsf{P}^L = \mathsf{NP}^L$ for all language L, but we can construct language A s.t. $\mathsf{P}^A \neq \mathsf{NP}^A$; Similarly, if we can prove $\mathsf{P} \neq \mathsf{NP}$ by diagonalization, also $\mathsf{P}^L \neq \mathsf{NP}^L$ for all language L, but we can construct language B s.t. $\mathsf{P}^B = \mathsf{NP}^B$.

Theorem 2.1 (Baker-Gill-Solovay '75)**.** *An oracle A exists whereby $\mathsf{P}^A \neq \mathsf{NP}^A$; An oracle B exists whereby $\mathsf{P}^B = \mathsf{NP}^B$.*

In summary, the relativization method tells us that to solve the P versus NP question, we must analyze computations, not just simulate them.