Today

- PH collapse hypothesis.
- Circuit complexity
- Karp-Lipton Theorem

PH collapse hypothesis

Jargon: Hierarchy "collapses" if $\Sigma_i^P = \Pi_i^P$.

Hypothesis: Hierarchy does not "collapse", i.e., For every i, $\Sigma_i^P \neq \Pi_i^P$.

Why "collapse"? Next proposition explains.

Collapse of the PH

Proposition: For $i \leq j$,
$\Sigma_i^P = \Pi_i^P \Rightarrow \Sigma_j^P = \Pi_j^P = \Sigma_i^P = \Pi_i^P$.

Proof:

- By induction on j. True for $j = i$. Let $j > i$ and assume true for $j - 1$.
- Let $A \in \Sigma_j^P$ and let $B \in \Pi_{j-1}^P$ s.t.
  $x \in A \Leftrightarrow \exists y$ s.t. $(x, y) \in B$.
- By induction $B \in \Sigma_i^P$ and so $\exists C \in \Pi_{i-1}^P$ s.t. $(x, y) \in B \Leftrightarrow \exists z$ s.t. $(x, y, z) \in C$.
- So $x \in A$ iff $\exists y, z$ s.t. $(x, y, z) \in C$. Thus $A \in \Sigma_i^P$.

PH collapse hypothesis

Why do we like it?

- Can't prove it false!
- It implies many other things we believe.
- Examples:
  - NP has randomized polynomial time algorithms implies hierarchy collapses.
  - NP has sparse complete language implies hierarchy collapses.
- Today's example: NP has small circuits implies hierarchy collapses.

Circuit complexity/Non-uniform computation

- Does solving a problem become much easier if we only have to design an algorithm to work for one fixed n at a time?
- Certainly, if the language is unary!
- But not necessarily if languages are binary!
- How do we measure running time in this case?
  - Design a family of "algorithms": one for each n and study runtime as function of n.

  - Equivalently: design a family of "circuits", one for each n and study circuit size as function of n.
  - To meaningfully study questions such as "Is NP=P?", restrict circuit size to be polynomial in n.

Boolean circuits

- Circuit is a DAG (directed acyclic graph).
- Node categories:
  - Input gates: Distinct labels 1 to n.
  - Output gates: Distinct labels 1 to m.
  - Computation gates: AND, OR, NOT.
- Wires: Run between gates.
  - Input gates have no wires coming in.
  - Computation gates have one (if NOT), or two (if OR/AND), wires coming in.
  - Output gates have no wires going out.
- Size = # of gates. (Sometimes allow unbounded fan-in OR/AND gates: in such case size = # wires.)

- Circuit computes a function $f : \{0,1\}^n \to \{0,1\}^m$.
- Our interest: E.g. smallest circuit deciding SAT ($m = 1$).

Turing machines with advice

- Alternate interpretation of non-uniform computation: Give "advice" to a Turing machine.

- Fix a polynomial p. Let $a_1, a_2, \ldots$ with $a_n \in \{0,1\}^{p(n)}$ be advice strings. Given $x \in \{0,1\}^n$, an advice Turing machine M uses the advice a_n to determine if $x \in L$ or not.

Defn: $L \in P/_{\text{poly}}$ if there exists a polynomial time bounded Turing machine M, polynomial p and advice strings $a_1, \ldots, a_n, \ldots$ with $|a_n| \le p(n)$ such that for every $x \in \{0,1\}^*$,

$$x \in L \Leftrightarrow M(x, a_{|x|}) = 1.$$

Can think of a_n as describing circuit, and $M(x,a)$ computes value of circuit a on input x. Conversely, given any advice a and poly-time TM M, can build poly-sized circuit that determines value of M on input x and advice a. Thus $P/_{\text{poly}}$ is the class of languages with polynomial sized circuit family.

Circuit complexity

- Given Boolean function family $\{f_n\}_n$ with $f_n : \{0,1\}^n \to \{0,1\}$ show lower bounds on smallest circuit computing f_n.

- Hope: Can show NP $\neq$ P by showing NP $\not\subseteq$ P$/_{\text{poly}}$.

- Wait - what?

- P$/_{\text{poly}}$ includes undecidable languages!

- Why should it not just contain NP, if it is so powerful!

- Karp-Lipton: Non-uniformity is not too powerful in deciding uniform languages. Specifically:

Thm: If NP $\subseteq$ P$/_{\text{poly}}$ then PH collapses.

Karp-Lipton

Assume M is an advice TM deciding SAT.

Defn: a_n is GOOD if $M(\phi, a_n)$ decides $\phi \in$SAT?.

Karp-Lipton Lemmas:

Lemma 1: GOOD is in Π_2^P.

(Wonderful: we have shown NP is in PH!)

Lemma 2: If NP $\subseteq$ P/$_{\mathrm{poly}}$ and GOOD is in Π_i^P, then $\Sigma_{i+2}^P = \Sigma_{i+1}^P$.

Note: deliberately ignoring the fact that we know GOOD is very low. We don't need it to collapse the hierarchy.

Proof of Lemma 1

Lemma 1: GOOD is in Π_2^P.

Proof: a_n is GOOD, if

$$\forall \psi, \quad M(\psi, a_n) = 1 \Rightarrow \exists \alpha \text{ s.t. } \psi(\alpha) = 1$$

$$M(\psi, a_n) = 0 \Rightarrow \forall \beta \psi(\beta) = 0.$$

Equivalently:

$$\forall \psi, \beta \exists \alpha \text{ s.t. } ((M(\psi, a_n) = 0) \vee \psi(\alpha) = 1)$$

$$\wedge M(\psi, a_n) = 1) \vee \psi(\beta) = 0)).$$

Proof of Lemma 2

Lemma 2: If NP $\subseteq$ P/$_{\mathrm{poly}}$ and GOOD is in Π_i^P, then $\Sigma_{i+2}^P = \Sigma_{i+1}^P$.

Proof: Will show $(i+2)$-QBF in Σ_{i+1}^P. Assume for simplicity that i is odd.

Basic idea: Given formula ϕ where we wish to decide if

$$\exists \mathbf{x}_1 \forall \mathbf{x}_2 \ldots \exists \mathbf{x}_i \phi(\mathbf{x}_1, \ldots, \mathbf{x}_i) = 1,$$

we'll quantify over $\mathbf{x}_1$ to $\mathbf{x}_{i-1}$ and let $\psi(\mathbf{x}_i) = \phi(\mathbf{x}_1, \ldots, \mathbf{x}_i)$ be the remaining formula. We'll then use a GOOD string a_n and determine if $M(\psi, a_n) = 1$.

How do we find a GOOD string? We guess it along with $\mathbf{x}_1$ and in parallel to the computation determing if ϕ is a YES instance, we'll check if a_n is GOOD.

Formal Proof

Σ_{i+1}^P computation for ϕ:

- GUESS $\mathbf{x}_1, a_n$
- FORALL Verify a_n is GOOD
  Verify $\forall \mathbf{x}_2, \exists \mathbf{x}_3, \ldots, \forall \mathbf{x}_{i-1}$
  $M(\psi, a_n) = 1$ where $\psi(\cdot) = \phi(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})$.

Non-uniform complexity

Why would it be easier to show NP $\not\subseteq$ P$/_{\text{poly}}$ than to show NP $\neq$ P.

- Circuit lower bounds more combinatorial.
- Can show circuit lower bounds by counting.
- Other sophisticated techniques available.
- Unfortunately: No explicit functions (in NP) with superlinear lower bound.
- Better lower bounds exist for high complexity; but based on diagonalization.