- Theme: (Failed) approaches to proving $P \neq NP$.
- Diagonalization: Power & Problems
- Relativization
- Baker-Gill-Solovay
- Circuit Complexity

E.g., Would like a complete map of complexity?
Unfortunately: only two tools so far - Algorithms & Diagonalization.

- Diagonalization can prove:
  - Problems undecidable.
  - Space hierarchy, time hierarchy.
  - Ladner’s theorem (between any two classes is an infinitely dense hierarchy).
  - But can it resolve $NP = P$?

Aside: Bird’s eyview of Ladner’s theorem

- Suppose $P \neq NP$.
- Let $L_1 \in P$ and $L_2$ be NP-complete.
- Let $n_1 = 1$ and $n_i = 2^{n_{i-1}}$.
- Let $L = L_1$ for strings of length $[n_{i-1}, n_i)$ for odd $i$, and $L = L_2$ for strings of length $[n_{i-1}, n_i)$ for even $i$.
- $L \in P$? Probably not.
- Is $L$ NP-complete? Probably not.
- Ladner’s theorem picks a more careful choice of $n_i$’s (by “lazy diagonalization”), to eliminate the “Probably’s” above.

Won’t cover theorem in detail.
• Can it resolve $\text{NP} \overset{?}{=} \text{P}$?

• Question raised in the seventies.

• Baker-Gill-Solovay: No!

• Err.... some caveats ....

---

**B-G-S Proposition**

**Prop:** If diagonalization shows $C_1 \not\subseteq C_2$, then for every $A$, $C_1^A \not\subseteq C_2^A$.

**Jargon:** $C_1 \not\subseteq C_2$ relativizes.

**Proof (of Prop/Jargon):**

- Exists machine in $C_1$ that can simulate any machine in $C_2$. (Since diagonalization works.)

- Augment this machine into an oracle machine.

- Machine now shows that $C_1^A$ diagonalizes $C_2^A$.

---

**BGS Lemmas**

**Lemma 1** There exists an oracle $A$ such that $\text{NP}^A = \text{P}^A$.

**Proof:** Take some language that is sufficiently powerful. Example: Let $A$ be any PSPACE-complete language. Then $\text{NP}^A = \text{NPSPACE} = \text{PSPACE} = \text{P}^A$.
Lemma 2 There exists an oracle $B$ such that $\text{NP}^B \neq \text{P}^B$.

Proof:

- Insert proof here.

---

Circuits

Defn; Circuit over $\{\text{Not}, \lor, \land\}$ is a directed acyclic graph, with nodes of one of the following types:

- **Input Nodes:** In-degree zero, labelled with el’t of $\{x_1, \ldots, x_n\}$.

- **Computation nodes:** Labelled by one of $\{\text{Not}, \lor, \land\}$, with Not nodes having in-deg. one, and others having in-deg. two.

- **$m$-Output nodes:** In-degree one, Out-degree zero, labelled by distinct elements of $\{y_1, \ldots, y_m\}$.

Circuit computes function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ in the natural way: Given assignment in $\{0,1\}$ to $x_1, \ldots, x_n$, propagate assignment along circuit (labelling vertices as appropriate functions of incoming edges, and edges as equal to the label of the vertices they come from) and then reading off the assignment to the $m$ output vertices $y_1, \ldots, y_m$ in order.

---

BGS Lemmas

Proof makes sense only when specialized (to say P vs. NP).

- Otherwise, it is pedagogy, not mathematics.

- Only rules out very specific proofs. Minor variations not accepted!

- Often misinterpreted, misrepresented, misrepresented etc.

- Alternate approach: Try combinatorics.
Circuits only compute functions on fixed input length. How to compare their power vs. that of a Turing machine?

Idea: Use a family of circuits \( \{ C_n \} \), where \( C_n : \{0, 1\}^n \rightarrow \{0, 1\} \).

\( \{ C_n \} \) decides \( L \) if for every \( n \), \( L \cap \{0, 1\}^n = C_n^{-1}(1) \).

Circuit complexity of language \( L \) is the function \( f(n) = |C_n| \), where \( C_n \) is smallest circuit to accept \( L \cap \{0, 1\}^n \).

What languages can be computed by family of polynomial sized circuits? Exponential sized circuits? Doubly exponential sized circuits?

Non-uniformity

Jargon: Ability to do completely different things on different inputs in referred to as non-uniformity.

Other models:

- Formulae: Circuits where DAG is a tree (root = unique output).

- Branching programs: Diff. model: given by DAG, with one source, two sinks (labelled 0/1), every node (except sinks) have out-deg. 2 and are labelled by element of \( \{x_1, \ldots, x_n\} \). Edges labelled 0/1. Assignment to \( x_1, \ldots, x_n \) specifies path from source to sink. Function value = label of sink.

Captured broadly by "two input machine with advice". \( M(x, y) \) is a two-input machine. Advice is a sequence of strings \( a = (a_1, \ldots, a_n, \ldots) \) with \( a_n \in \{0, 1\}^{t(n)} \).

If \( M \) is in class \( C \) and \( L = M \) with advice \( a \), then \( L \) is said to be in \( C/\ell \). Most relevant class \( P/poly \).

\( C \subseteq C/\ell \) and so \( P \subseteq P/poly \).

\( P/poly = \) all languages with poly-sized circuit complexity.

Let \( M(x, y) \) compute output of circuit \( y \) on input \( x \) - decides circuit \( y \).

On the other hand, any advice \( a_n \) can be hardwired into circuit \( C_n \) and then it suffices to simulate a Turing Machine computation with a circuit. Can do so trivially with
Some basic hopes/intuitions

- Size of circuit (\# of nodes) \(\approx\) running time?
- Width of branching program \(\approx\) space complexity?
- Er ... every language has a circuit family of exponential size \(|C_n| = 2^{O(n)}\); and a branching program of constant width.
- Every unary language has a constant sized circuit! (including undecidable ones).
- Highly contradictory? Refinement of belief: For NP-complete languages, circuit complexity is exponential.

Circuits and combinatorics

- Set of functions computable by circuits of given size may yield to combinatorial methods. E.g., it is easy to show that \(\exists\) functions that require circuits of size \(2^{\Omega(n)}\) etc.
- But give an explicit function? Of moderate-low uniform complexity? Hard. Best uniform to non-uniform contrast: There exists a function in doubly exponential time that does not have polynomial sized (or even sub-exponential sized) circuits. (Kannan.)
- For functions in \(\text{PSPACE}\) (or even \(\text{EXP}\)), best known lower bound on circuit size ...

4.5\(n\). for formulae, \(n^3\), and for branching programs \(n^2\) ...
Next two lectures

Branching program lower bounds;

Circuit lower bound for parity.