The Gap Between P and NP

Let $NPL = NP - (P \cup NPC)$. Is $NPL = \emptyset$?
**Ladner’s Theorem**

**Theorem:** If P $\neq$ NP, then NPL $\neq \emptyset$.

**Lemma:** Let B be any decidable language that is not in P. There exists a language D that is in P and that has the following property: Let $A = D \cap B$. Then $A \notin P$, $A \leq_P B$, but it is not true that $B \leq_P A$.

**Proof:** Suppose that B is any NP-complete language. Unless P = NP, B is not in P. So there must exist a language D that is in P, and from which we can compute $A = D \cap B$. A must be in NP since membership in D can be decided in polynomial time and membership in B can be verified in polynomial time. So, using the lemma, we have:

- $A \notin P$, but
- It is not true that $B \leq_P A$. Since B is in NP but is not deterministic, polynomial-time reducible to A, A is not NP-complete.

So A is an example of an NP language that is neither in P nor NP-complete. Thus NPL $\neq \emptyset$. 
The Gap Between P and NP

If $\text{NPL} \neq \emptyset$, what does it contain?

Some candidates:

- COMPOSITES, now known in P.
- LINEAR-PROGRAMMING, now known in P.
- GRAPH-ISOMORPHISM.
Small Differences Matter

- Circuit problems
- SAT problems
- Path problems
- Covering problems
- Map coloring problems
- Linear programming problems
- Diophantine equation problems
Two Similar Circuit Problems

- EULERIAN-CIRCUIT, in which we check that there is a circuit that visits every edge exactly once, is in P.

- HAMILTONIAN-CIRCUIT, in which we check that there is a circuit that visits every vertex exactly once, is NP-complete.
Two Similar SAT Problems

• 2-SAT = \{<w> : w is a wff in Boolean logic, w is in 2-conjunctive normal form and w is satisfiable\} is in P.

  \((\neg P \lor R) \land (S \lor \neg T)\)

• 3-SAT = \{<w> : w is a wff in Boolean logic, w is in 3-conjunctive normal form and w is satisfiable\} is NP-complete.

  \((\neg P \lor R \lor T) \land (S \lor \neg T \lor \neg W)\)
Two Similar Path Problems

• SHORTEST-PATH = \{<G, u, v, k>: G is an undirected graph, u and v are vertices in G, k \geq 0, and there exists a path from u to v whose length is at most k\} is in P.

• LONGEST-PATH = \{<G, u, v, k>: G is an undirected graph, u and v are vertices in G, k \geq 0, and there exists a path with no repeated edges from u to v whose length is at least k\} is NP-complete.
Two Similar Path Problems

\[ \text{shortest-path}(G = (V, E), u, v: \text{vertices}, k) = \]
1. Mark \( u \).
2. For \( i = 1 \) to \( \min(k, |E|) \) do:
   For each marked vertex \( n \) do:
     For each edge from \( n \) to some other vertex \( m \) do:
       Mark \( m \).
3. If \( v \) is marked then accept. Else reject.

\text{Shortest-path} runs in \( O(|G|^3) \) time.
Two Similar Covering Problems

• An **edge cover** $C$ of a graph $G$ is a subset of the edges of $G$ with the property that every vertex of $G$ is an endpoint of one of the edges in $C$.

• A **vertex cover** $C$ of a graph $G$ is a subset of the vertices of $G$ with the property that every edge of $G$ touches one of the vertices in $C$. 
Two Similar Covering Problems

- **EDGE-COVER** = \{<G, k>: G is an undirected graph and there exists an edge cover of G that contains at most k edges\} is in P.

- **VERTEX-COVER** = \{<G, k>: G is an undirected graph and there exists a vertex cover of G that contains at most k vertices\} is NP-complete.
Three Similar Coloring Problems

Color a planar map so that no two adjacent regions (countries, states, or whatever) have the same color.

(a)    (b)    (c)

How many colors are required?
Three Similar Coloring Problems

• 2-COLORABLE = \{<m> : m can be colored with 2 colors}\}.

• 3-COLORABLE = \{<m> : m can be colored with 3 colors}\}.

• 4-COLORABLE = \{<m> : m can be colored with 4 colors}\}.
Three Similar Coloring Problems

• \(2\)-COLORABLE = \{\langle m \rangle : m \text{ can be colored with 2 colors}\}.

A map is 2-colorable iff it does not contain any point that is the junction of an odd number of regions.

• \(3\)-COLORABLE = \{\langle m \rangle : m \text{ can be colored with 3 colors}\}.

THREE-COLORABLE is NP-complete.

• \(4\)-COLORABLE = \{\langle m \rangle : m \text{ can be colored with 4 colors}\}.
Chromatic Number

The *chromatic number* of a graph is the smallest number of colors required to color its vertices, subject to the constraint that no two adjacent vertices may be assigned the same color.

$$\text{CHROMATIC-NUMBER} = \{<G, k> : G \text{ is an undirected graph whose chromatic number is no more than } k\}.$$  

**CHROMATIC-NUMBER** is NP-complete.
Two Similar Linear Programming Problems

- **LINEAR-PROGRAMMING** = \{<a set of linear inequalities \(Ax \leq b\)> : there exists a **rational** vector \(X\) that satisfies all of the inequalities\} is in P.

- **INTEGER-PROGRAMMING** = \{<a set of linear inequalities \(Ax \leq b\)> : there exists an **integer** vector \(X\) that satisfies all of the inequalities\} is NP-complete.
A Hierarchy of Diophantine Equation Problems

A Diophantine equation is a polynomial equation in any number of variables, all with integer coefficients. A Diophantine problem then is, “Given a system of Diophantine equations, does it have an integer solution?”

\[ 4x^3 + 7xy + 2z^2 - 23xz = 0 \]
A Hierarchy of Diophantine Equation Problems

- The general Diophantine problem is undecidable.

- If all equations have the form $ax^2 + by = c$, then the problem is NP-complete.

- If all the variables are of degree 1 or all equations have the form $ax^k = c$, then the problem is in P.

A farmer buys 100 animals for $100.00. The animals include at least one cow, one pig, and one chicken, but no other kind. If a cow costs $10.00, a pig costs $3.00, and a chicken costs $0.50, how many of each did he buy?
The Class co-NP

$L \in \text{co-NP}$ iff $\neg L \in \text{NP}$.

- A language $L$ is in NP iff a **qualifying** certificate can be checked efficiently.
- A language $L$ is in co-NP iff a **disqualifying** certificate, can be checked efficiently.

Example:

- $\text{TSP-DECIDE} \in \text{NP}$.
- $\neg \text{TSP-DECIDE} \in \text{co-NP}$. 
Relating NP and co-NP

Theorem: If NP ≠ co-NP then P ≠ NP.

Proof: The class P is closed under complement. So, if P = NP, then NP must also be closed under complement. If NP ≠ co-NP then NP is not closed under complement. So it cannot equal P.

We do not know whether NP = co-NP implies that P = NP. It is possible that NP = co-NP but that that class is nevertheless larger than P.
Relating NP and co-NP

**Theorem:** NP = co-NP iff there exists some language $L$ such that $L$ is NP-complete and $\neg L$ is also in NP.

**Proof:** We prove the two directions of the claim separately:

If NP = co-NP then there exists some language $L$ such that $L$ is NP-complete and $\neg L$ is also in NP:

There exists at least one language $L$ (for example, SAT) that is NP-complete. By definition, $\neg L$ is in co-NP. If NP = co-NP then $\neg L$ must also be in NP.
Relating NP and co-NP

If there exists some language $L$ such that $L$ is NP-complete and $\neg L$ is also in NP then $NP = co-NP$:

Suppose that some language $L$ is NP-complete and $\neg L$ is also in NP. Then we can prove:

- $NP \subseteq co-NP$
- $co-NP \subseteq NP$
Relating NP and co-NP

**NP ⊆ co-NP:** Let $L_1$ be any language in NP. Since $L$ is NP-complete, there exists a polynomial-time reduction $R$ from $L_1$ to $L$.

$R$ is also a polynomial time reduction from $\neg L_1$ to $\neg L$.

Since $\neg L$ is in NP, there exists a nondeterministic polynomial-time Turing machine $M$ that decides it. So we can decide $\neg L_1$ in nondeterministic polynomial polynomial time by first running $R$ and then running $M$.

So $\neg L_1$ is in NP and its complement, $L_1$, is in co-NP.

Thus every language in NP is also in co-NP.
Relating NP and co-NP

\textbf{co-NP} \subseteq \textbf{NP}: Let \( L_1 \) be any language in co-NP. Then \( \neg L_1 \) is in NP.

Since \( L \) is NP-complete, there exists a polynomial-time reduction \( R \) from \( \neg L_1 \) to \( L \). \( R \) is also a reduction from \( L_1 \) to \( \neg L \).

Since, by assumption, \( \neg L \) is in NP, there exists a nondeterministic polynomial-time Turing machine \( M \) that decides it. So we can decide \( L_1 \) in nondeterministic polynomial time by first running \( R \) and then \( M \).

So \( L_1 \) is in NP.

Thus every language in co-NP is also in NP.
Inherently Hard Languages Exist

- There exists a set of hierarchy theorems that show that adding resources (in terms of either time or space) increases the set of languages that can be decided.

- There exist some specific decidable languages that can be shown to be hard in the sense that no efficient algorithm to decide them exists.
Hierarchy Theorems

While it is possible that particular pairs of complexity classes may collapse, it is not possible that all of them do.

Unfortunately:

- The theorems do not tell us what languages lie where in the hierarchy. They are proved by diagonalization so they show only that some language must exist. They are not constructive.

- They do not relate deterministic complexity classes to nondeterministic ones. So, for example, they say nothing about whether P = NP.

- They do not relate time complexity classes to space complexity classes.
Deterministic Time Hierarchy Theorem

**Theorem:** Increasing the amount of time by at least a logarithmic factor makes a difference.

**Proof:** By construction of a Turing machine that can do the following two things:

- Compute the value of a \textit{timereq} function, on a given input, and store that value, in binary, on its tape.
- Efficiently simulate another Turing machine for a specified number of steps.
Time-Constructible Functions

A function $t(n)$ from the positive integers to the positive integers is *time-constructible* iff:

- $t(n)$ is at least $\mathcal{O}(n \log n)$, and
- the function that maps the unary representation of $n$ (i.e., $1^n$) to the binary representation of $t(n)$ can be computed in $\mathcal{O}(t(n))$ time.

If $\text{timereq}(M)$ is time-constructible, it is possible to write its value, for a particular $n$, in some number of steps that will not dominate the number of steps executed by $M$ itself.

Time-constructible functions: polynomials, $n \log n$, $2^n$, $n!$. 
Efficient Bounded Simulation

*BSim* accepts as input a Turing machine $M$, an input string $w$, and a time bound $b$. It views its tape as having three tracks:

- Track 1 will hold the current value of $M'$'s tape (including the position of its read/write head).

- Track 2 will hold $M'$'s current state followed by $M'$'s description.

- Track 3 will hold a counter that is initially set to be the time bound $b$. As each step of $M$ is simulated, the counter will be decremented by 1. The simulation will halt if the counter ever reaches 0 (or if $M$ naturally halts).
Efficient Bounded Simulation

We need to avoid the $\mathcal{O}(n^2)$ price that normally occurs when we simulate a $k$-tape TM by a 1-tape one.

The key idea is to keep the tracks lined up.

| Track 1: | ababbbbaaabb | state, $<M>$ |
| Track 2: |               |              |
| Track 3: |               | counter      |
Efficient Bounded Simulation

To simulate one step of $M$:

- Update track 1 as specified by $M'$'s transition function. Doing this requires moving at most one square on track 1, so it takes constant time.
- Update $M'$'s state on track 2. Doing this requires time that is a function of the length of the state description, which is bounded by $|<M>|$. So it takes $O(|<M>|)$ time.
- Move the contents of track 2 one square to the right or to the left, depending on which way $M'$'s read/write head moved. Doing this takes time that is a function only of $M$. So it also takes $O(|<M>|)$ time.
- Manage the counter.
Efficient Bounded Simulation

Managing the counter:

• Decrement the counter by 1 and check for 0. This can be done in constant time.

• Shift the counter left or right one square so that it remains lined up with $M$’s read/write head. The number of steps required to do this is a function of the length of the counter. The maximum value of the counter is the original bound, $b$. Since the counter is represented in binary, its maximum length is $\log b$. So this step takes $\log b$ time.
Efficient Bounded Simulation

\textit{BSim} runs $M$ for no more than $b$ steps.

Each step takes $\mathcal{O}(|<M>|)$ time to do the computation plus $\mathcal{O}(\log b)$ time to manage the counter.

So \textit{BSim} can simulate $b$ steps of $M$ in $\mathcal{O}(b\cdot(|<M>|+\log b))$ time.
Deterministic Time Hierarchy Theorem

**Theorem:** For any time-constructible function $t(n)$, there exists a language $L_{t(n)\text{hard}}$ that is deterministically decidable in $O(t(n))$ time but that is not deterministically decidable in $\sigma(t(n)/\log t(n))$ time.

**Proof:** We describe $L_{t(n)\text{hard}}$ by specifying a TM $M_{t(n)\text{hard}}$ that decides it in $O(t(n))$ time.

$M_{t(n)\text{hard}}$ is defined by diagonalization to guarantee that it is different from any $\sigma(t(n)/\log t(n))$ time TM.
Let $n$ be $|w|$. Compute $t(n)$. Write it, in binary, on the tape.

Divide that number by $\log t(n)$. Store $\lceil t(n)/\log t(n) \rceil$, in binary, on the tape. Call this number $b$.

Check to see that $w$ is of the form $\langle M \rangle 10^*$. If not, reject.

Check that $|\langle M \rangle| < \log b$. If not, reject.

Reformat the tape into three tracks as required by $BSim$.

Run $BSim$. In other words, simulate $M$ on $w$ (which is of the form $\langle M \rangle 10^*$) for $t(n)/\log t(n)$ steps.

If $M$ did not halt in that time, reject.

If $M$ did halt and it accepted, reject.

If $M$ did halt and it rejected, accept.
$M_{t(n)\text{hard}}(w)$

1. Let $n$ be $|w|$. Compute $t(n)$. Write it, in binary, on the tape.
2. Divide that number by $\log t(n)$. Store $\lfloor t(n)/\log t(n) \rfloor$, in binary, on the tape. Call this number $b$.
3. Check to see that $w$ is of the form $<M>10^*$. If not, reject.
4. Check that $|<M>| < \log b$. If not, reject.
5. Reformat the tape into three tracks as required by $BSim$.
6. Run $BSim$. In other words, simulate $M$ on $w$ (which is of the form $<M>10^*$) for $t(n)/\log t(n)$ steps.
7. If $M$ did not halt in that time, reject.
8. If $M$ did halt and it accepted, reject.
9. If $M$ did halt and it rejected, accept.

$M_{t(n)\text{hard}}$ runs in $\mathcal{O}(t(n))$ time. On input $<M, 10^*>$, it simulates $t(n)/\log t(n)$ steps of $M$, using $\mathcal{O}(|\log t(n)|)$ time for each one.
We Can’t Do Much Better

Suppose that $M_{t(n)\text{easy}}$ decides $L_{t(n)\text{hard}}$ in time that is $o(t(n)/\log t(n))$. Then, on all inputs of length greater than some constant $k$, $M_{t(n)\text{easy}}$ must halt in fewer than $t(n)/\log t(n)$ steps. But, in that case, we can show that it is not equivalent to $M_{t(n)\text{hard}}$ and so does not decide $L_{t(n)\text{hard}}$:

Let $w = <M_{t(n)\text{easy}}>^{10^p}$, where $p$ is chosen so that $w$ is long enough and so that $|<M_{t(n)\text{easy}}>|$ is short relative to the entire length of $w$.

Then, on input $w = <M_{t(n)\text{easy}}>^{10^p}$, the simulation of $M_{t(n)\text{easy}}$ on $<M_{t(n)\text{easy}}>^{10^p}$ will run to completion. If $M_{t(n)\text{easy}}$ accepts, $M_{t(n)\text{hard}}$ rejects. And vice versa. This contradicts the assumption that $M_{t(n)\text{easy}}$ decides $L_{t(n)\text{hard}}$. 
EXPTIME

For any language $L$, $L \in \text{EXPTIME}$ iff there exists some deterministic Turing machine $M$ that decides $L$ and $\text{timereq}(M) \in \mathcal{O}(2^{(n^k)})$ for some positive integer $k$.

Example:

$\text{CHESS} = \{<b>: b \text{ is a configuration of an } n \times n \text{ chess board and there is a guaranteed win for the current player}\}$. 
Chess and Sudoku

**SUDOKU** = \{<b>: b is a configuration of an \( n \times n \) grid and b has a solution under the rules of Sudoku}\}.

How many paths in a certificate?

**CHESS** = \{<b>: b is a configuration of an \( n \times n \) chess board and there is a guaranteed win for the current player}\}.

How many paths in a certificate?
CHESS

A

B

C

D

E

F

G

max

min

max
EXPTIME-Completeness

Suppose that:

1. \( L \) is in EXPTIME.
2. Every language in EXPTIME is deterministic, polynomial-time reducible to \( L \).

\( L \) is **EXPTIME-hard** iff it possesses property 2.

If it also possesses property 1, it is **EXPTIME-complete**.

CHESS is EXPTIME-complete if we add pieces as well as rows and columns.
A Time Hierarchy

\[ P \subseteq NP \subseteq PSPACE \subseteq EXPTIME. \]

It is not known which of these inclusions is proper. However, it follows from the Deterministic Time Hierarchy Theorem that:

\[ P \neq EXPTIME. \]

So at least one of them is. It is thought that all of them are.

A consequence of the fact that \( P \neq EXPTIME \) is that we know that there are decidable problems for which no efficient (i.e., polynomial time) decision procedure exists.
Harder than EXPTIME

Some problems are even harder than the EXPTIME-complete problems, such as CHESS.

FOLtheorem = \{<A, w> : A is a decidable set of axioms in first-order logic, w is a sentence in first-order logic, and w is entailed by A\} is not decidable.

Presburger arithmetic, a theory of the natural numbers with just the function plus is decidable. But it is intractable. [Fischer and Rabin 1974] showed that any algorithm that decides whether a sentence is a theorem of Presburger arithmetic must have time complexity at least $O(2^{2^{cn}})$. 
Moving Beyond Decision Problems

• **The Class FP:** A binary relation $Q$ is in FP iff there is deterministic polynomial time **algorithm** that, given an arbitrary input $x$, can find some $y$ such that $(x, y) \in Q$.

• **The Class FNP:** A binary relation $Q$ is in FNP iff there is a deterministic polynomial time **verifier** that, given an arbitrary input pair $(x, y)$, determines whether $(x, y) \in Q$. Equivalently, $Q$ is in FNP iff there is a nondeterministic polynomial time algorithm that, given an arbitrary input $x$, can find some $y$ such that $(x, y) \in Q$.

$FP \subseteq FNP.$

$FP = FNP$ iff $P = NP.$
The Function Version May be Harder than the Decision Version

PRIMES = \{w : w \text{ is the binary encoding of a prime number}\} is in P.

But the problem of finding the factors of a composite number has no known polynomial time solution.