Tractability vs. Intractability
Thus, for any nondeterministic Turing machine $M$ that runs in some polynomial time $p(n)$, we can devise an algorithm that takes an input $w$ of length $n$ and produces $E_{M,w}$. The running time is $O(p^2(n))$ on a multitape deterministic Turing machine and...

WTF, man. I just wanted to learn how to program video games.
**Decision Problem:** Given a problem instance $I$, answer “yes” or “no.”

**Instance:** A particular specialization (input) of a problem.

**Solution:** Correct answer ("yes" or "no") for the given instance.

- $I$ is a **yes-instance** if the correct answer is “yes.”
- $I$ is a **no-instance** if the correct answer is “no.”

**Instance Size:** number of bits to represent the instance.
### Examples

**Problem: Cycle**

**Instance:** An undirected graph \( G = (V, E) \).

**Question:** Does \( G \) contain a cycle?

**Problem: Hamiltonian Cycle**

**Instance:** An undirected graph \( G = (V, E) \).

**Question:** Does \( G \) contain a Hamiltonian cycle?

**Problem: Eulerian Cycle**

**Instance:** An undirected graph \( G = (V, E) \).

**Question:** Does \( G \) contain an Eulerian cycle?

**Hamiltonian cycle:** A cycle that passes through every node exactly once.

**Eulerian cycle:** A cycle that passes through every edge exactly once.
Optimization Problem: Given a problem instance $I$, minimize a cost function subject to constraints.

Instance: A particular specialization (input) of a problem.

Instance Size: number of bits to represent the instance.

Seems like more complicated.
Problem: TSP-Optimization

Instance: A graph $G = (V, E)$ and weights $w : E \rightarrow \mathbb{Z}_+$.  
Find: A Hamiltonian cycle $H$ with minimum weight $w(H) = \sum_{e \in H} w(e)$?

Problem: TSP-Value

Instance: A graph $G = (V, E)$ and weights $w : E \rightarrow \mathbb{Z}_+$.  
Find: The minimum weight of all Hamiltonian cycles?

Problem: TSP-Decision

Instance: A graph $G = (V, E)$, weights $w : E \rightarrow \mathbb{Z}_+$ and target $T \in \mathbb{Z}_+$.  
Question: Is there a Hamiltonian cycle $H$ with weight at most $T$?
Equivalence for exact solutions

TSP-Optimization $\implies$ TSP-Value: simply sum the weights in any optimal Hamiltonian cycle.

TSP-Value $\implies$ TSP-Decision: simply compare the optimal value with $T$.

TSP-Decision $\implies$ TSP-Value: Any Hamiltonian cycle has weight at most $U = \sum_e w(e)$. Binary search the optimal value using TSP-Decision.

TSP-Value $\implies$ TSP-Optimization: For every edge $e$, delete it and run TSP-Value. If the optimal value changes after the deletion, put $e$ back.

Focusing on decision problems from now on.
Complexity class $\mathbb{P}$

**Algorithm Solving a Decision Problem:** An algorithm $A$ solves a decision problem $\Pi$ if $A$ finds the correct answer ("yes" or "no") for every instance of $\Pi$ in finite time.

**Polynomial-time Algorithm:** An algorithm $A$ is polynomial time if its complexity is a polynomial of its input (instance) size.

**Complexity Class $\mathbb{P}$:** A class of decision problems that can be solved by some polynomial time algorithm.

So far, most algorithms we studied are in $\mathbb{P}$, except ???

Can identify both yes-instance and no-instance in polytime.
Complexity class \( \text{NP} \)

**Motivation:** Solving may be hard; verifying yes-instance seems easy.

**Certificate:** claimed solution, or more generally extra hints for verification.

Existence of certificate is enough; no need to find it in polytime.

**Complexity Class \( \text{NP} \):** A class of decision problems that can be verified in polytime, i.e., \( \exists \) some polytime verification algorithm \( V \) such that

- \( \forall \) yes-instance, \( \exists \) a poly-size certificate \( C \) such that \( V(I, C) = 1 \);
- \( \forall \) no-instance, \( \forall \) poly-size certificate \( C \), \( V(I, C) = 0 \).

\( \text{NP} = \text{Nondeterministic Polynomial} \neq \text{Non-polynomial} \)

**Theorem.** \( \text{NP} \subseteq \text{DEXP} \).
Theorem. \( P \subseteq NP \).

Proof. Simply ignore the certificate and run the polytime algorithm to solve \( P \).

Hamiltonian Cycle \( \in NP \).

Verification algorithm: \( O(n^2) \) time.
THE $P = NP$ question

Philosophically: is proving (computationally) harder than verifying?

Touched upon by many prominent scientists, e.g.,

- Jack Edmonds
- Steve Nash
THE P = NP question

Philosophically: is proving (computationally) harder than verifying?

Touched upon by many prominent scientists, e.g.,

- Jack Edmonds
- Steve Nash

Claimed to be “proved” hundreds of times.
THE $P = NP$ question

Philosophically: is proving (computationally) harder than verifying?

Touched upon by many prominent scientists, e.g.,
- Jack Edmonds
- Steve Nash

Claimed to be “proved” hundreds of times.

Claimed to be “disproved” hundreds of times.
THE $P = NP$ question

Philosophically: is proving (computationally) harder than verifying?

Touched upon by many prominent scientists, e.g.,

- Jack Edmonds
- Steve Nash

Claimed to be “proved” hundreds of times.

Claimed to be “disproved” hundreds of times.

Still open. Worth 1 Million.
Complexity class $\text{co-NP}$

**Complexity Class** $\text{co-NP}$: Complement of $\text{NP}$.

**Theorem.** $P \subseteq \text{NP} \cap \text{co-NP}$.

**Theorem.** $\text{NP} \neq \text{co-NP} \implies P \neq \text{NP}$. 
Why “polynomial”? 

Small polynomial $\approx$ efficient; large polynomial is rare and usually reduced to small polynomial afterwards.

Closed under addition, multiplication, and composition. Leads to appealing theory.

Robust across different computational models.

Reasonable case of illustration.
Polynomial reduction

For a decision problem $\Pi$:

- $\mathcal{I}(\Pi)$: the set of all instances of $\Pi$
- $\mathcal{I}_y(\Pi)$: the set of all yes-instances of $\Pi$
- $\mathcal{I}_n(\Pi)$: the set of all no-instances of $\Pi$

**Karp’s polynomial reduction:** $\Pi_1$ is polynomial reducible to $\Pi_2$, denoted as $\Pi_1 \leq_P \Pi_2$, if there exists a function $f : \mathcal{I}(\Pi_1) \rightarrow \mathcal{I}(\Pi_2)$ such that:

- $f$ is polytime computable (in terms of its input size)
- $l \in \mathcal{I}_y(\Pi_1) \iff f(l) \in \mathcal{I}_y(\Pi_2)$

**Cook’s polynomial reduction:** $\Pi_1$ is polynomial reducible to $\Pi_2$, if we can solve $\Pi_1$ in polytime, plus calling $\Pi_2$ polynomially many times.
Importance of polynomial reduction

**Theorem.** If $\Pi_1 \leq_P \Pi_2$ and $\Pi_2 \in P$, then $\Pi_1 \in P$.

**Corollary.** If $\Pi_1 \leq_P \Pi_2$ and $\Pi_1 \notin P$, then $\Pi_2 \notin P$.

**Theorem.** If $\Pi_1 \leq_P \Pi_2$ and $\Pi_2 \leq_P \Pi_3$, then $\Pi_1 \leq_P \Pi_3$.

**Informally.** $\leq_P$ is an order for comparing “hardness:”

If $\Pi_1 \leq_P \Pi_2$ and $\Pi_1$ is hard, then $\Pi_2$ is also hard.
Examples

Problem: Clique

Instance: An undirected graph $G = (V, E)$ and integer $k \in [1, |V|]$.

Question: Does $G$ contain a clique of size at least $k$?

Clique: complete subgraph.

Problem: Vertex Cover

Instance: An undirected graph $G = (V, E)$ and integer $k \in [1, |V|]$.

Question: Does $G$ contain a vertex cover of size at most $k$?

Vertex cover: a subset of nodes so that each edge has at least one endpoint in it.

Problem: Set Cover

Instance: A set $U$, a collection of subsets $S = \{S_i \subseteq U : 1 \leq i \leq m\}$, and integer $k \in [1, m]$.

Question: Does $S$ contain a subset of size at most $k$ whose union is $U$?
Proofs

Clique $\leq_P$ vertex cover: A clique of size at least $k$ in $G$ iff a vertex cover of size at most $n - k$ in the complement graph $G^c$.

Vertex cover $\leq_P$ set cover: $U = E$, $S_i := \{e \in E : i \in e\}$. 

Complexity Class **NP-hard**: the set of decision problems $\Pi$ such that for all $\Pi' \in \text{NP}$, $\Pi' \leq_P \Pi$.

**Complexity Class NPC**: $\text{NP-hard} \cap \text{NP}$.

By definition, $\text{NPC} \subseteq \text{NP}$.

**Theorem.** $\text{P} \cap \text{NPC} \neq \emptyset \implies \text{P} = \text{NP}$.

But, we do not know if $\text{P} \cap \text{NPC} \neq \emptyset$.

In fact, even $\text{NPC} \neq \emptyset$ is not obvious...
Problem: CNF-SAT

Instance: A boolean formula $B$ in $n$ boolean variables $x_1, \ldots, x_n$ such that $B$ is a conjunction (logical “and”) of $m$ clauses, each of which is the disjunction (logical “or”) of literals (a boolean variable or its negation).

Question: Is there a truth assignment of $x$ such that $B$ is true?

Theorem. CNF-SAT $\in$ NPC.

Proof. Easy part: CNF-SAT $\in$ NP.

Hard part: CNF-SAT $\in$ NP-hard.
**Theorem.** $\Pi_1 \in \text{NP-hard}$ and $\Pi_1 \leq_P \Pi_2 \implies \Pi_2 \in \text{NP-hard}$. If additionally $\Pi_2 \in \text{NP}$, then $\Pi_2 \in \text{NPC}$. 
Theorem. \( \Pi_1 \in \text{NP-hard} \) and \( \Pi_1 \leq_P \Pi_2 \implies \Pi_2 \in \text{NP-hard} \). If additionally \( \Pi_2 \in \text{NP} \), then \( \Pi_2 \in \text{NPC} \).

Theorem. \( \text{CNF-SAT} \in \text{NPC} \).

Theorem. If \( \Pi_1 \leq_P \Pi_2 \) and \( \Pi_2 \leq_P \Pi_3 \), then \( \Pi_1 \leq_P \Pi_3 \).