Recall:
\[
\text{TIME}(f(n)) = \{ \text{set of languages decidable by a Turing machine with running time } O(f(n)) \text{ on length-} n \text{ inputs} \}
\]
\[
P = \text{TIME}(n) \cup \text{TIME}(n^2) \cup \text{TIME}(n^3) \cup \ldots
\]
\[
= \text{polynomial-time decidable languages}
\]
\[
\text{NTIME}(f(n)) = \{ \text{languages decidable by a } O(f(n)) \text{-time nondet T.M.} \}
\]
\[
\text{NP} = \bigcup \text{NTIME}(n^c)
\]
\[
= \text{languages with a polynomial-time verifier}
\]

Conjecture: \( P \neq \text{NP} \).

Theorem: Let SAT = \( \{ \langle \phi \rangle \mid \phi \text{ is a boolean formula with a satisfying assignment} \} \)

If SAT \( \in \) P, then \( P = \text{NP} \).

Proof: Use the Cook-Levin theorem: SAT is \( \text{NP-complete} \).
That is SAT \( \in \) NP, and for any NP language \( L \), \( L \leq \text{SAT} \)
by a poly-time reduction. Then
\[
\begin{array}{c}
\text{input } x \in L \\
\text{reduction to SAT} \\
f(x) \\
P\text{-time SAT alg.} \\
\text{Yes} \\
f(x) \in \text{SAT} \\
\text{No} \\
x \notin L
\end{array}
\]
is a polytime algorithm for \( L \).

More reductions

Def: \( k\text{-SAT} = \{ \langle \phi \rangle \mid \phi \text{ is a boolean formula in conjunctive normal form, i.e., an AND of OR clauses, in which each clause involves exactly } k \text{ literals; and } \phi \text{ is satisfiable} \} \).

Examples: \( (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_1) \in 2\text{-SAT} \)
\( (x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_2 \vee \overline{x}_3) \in 3\text{-SAT} \), ...

Claim: 2-SAT \( \in \) P, i.e., can be decided in poly-time.
Claim: For \( k \geq 3 \), \( k\text{-SAT} \leq \text{poly}3\text{-SAT} \).

In particular, 3-SAT is \( \text{NP-complete} \).

Proof: Replace a clause \( (a_1 \vee a_2 \vee a_3 \vee a_4) \) by \( (a_1 \vee a_2 \vee \overline{a}_3) \wedge (\overline{a}_2 \vee a_3 \vee a_4) \)
for a new variable \( \overline{a}_3 \). In general if the \( j \)th clause is...
$C_j = (a_{j,1}, V a_{j,2}, V \ldots, V a_{j,k})$

replace it with

$(a_{j,1}, V a_{j,2}, V z_{j,1}) \land (z_{j,1}, V a_{j,3}, V z_{j,2}) \land (z_{j,2}, V a_{j,4}, V z_{j,3}) \land \ldots \land (z_{j,k-1}, V a_{j,k-1}, V a_{j,k})$.

Then the new formula is satisfiable iff the original formula was.

**VERTEX-COVER** = \{<G, k> | G is an undirected graph for which some vertex subset of size ≤ k is incident to every edge\}.

**Example**

![Graph](image)

**Size-3 vertex cover**

**Theorem**: VERTEX-COVER is NP-complete.

**Proof**: Clearly VERTEX-COVER is in NP; given a subset of vertices, we can verify that it covers all edges.

Let us reduce 3-SAT to VERTEX-COVER.

**Example**: Map \( \phi = (x_1 \lor x_2 \lor x_3) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_3}) \land (x_1 \lor x_2 \lor x_3) \) to

\[ G = \]

\[ k = \# \text{variables} + 2 \cdot \# \text{clauses} \]

Each literal has 2 vertices, connected by an edge. Each clause has a triangle.

**Claim 1**: \( \phi \) is satisfiable \( \Rightarrow \) G has a cover of size k.

**Proof**: Put in the true literals at the top vertices. For each clause, put in everything except one true vertex in the triangle. The covers the variable and clause gadgets' edges, and the edges between them.

**Claim 2**: G has cover of size k \( \Rightarrow \) \( \phi \) is satisfiable.

**Proof sketch**: Any cover must include \( \geq 1 \) vertex in each variable gadget & \( \geq 2 \) in each clause gadget. Since \( k = \# \text{vars} + 2 \cdot \# \text{clauses} \), it must be exact. Read off the assignment as above.
Approximation algorithms

- **VERTEX-COVER** is the decision version (yes/no version) of the problem of finding the minimum-size vertex cover for a given graph (MIN-VERTEX-COVER).
- approximation algorithms goal: find nearly optimal solutions

**Theorem**: There is a poly-time algorithm that takes a graph and outputs a vertex cover with size at most twice that of the smallest cover.

**Proof**: Input: $G=(V,E)$

$S \leftarrow \emptyset$

While an uncovered edge $e=(u,v)$ remains,

$S \leftarrow S \cup \{u,v\}$

Output $S$.

The algorithm is efficient and outputs some vertex cover $S$. Let $S^* \subseteq V$ be an optimal vertex cover. Then $|S| \leq 2|S^*|$, since for every edge for which we added both endpoints to $S$, at least one of the endpoints must be in $S^*$, too.

**Inapproximability of MIN-VERTEX-COVER**

**Theorem (Hastad 2001)**: Unless $P=NP$, there is no polynomial time vertex cover algorithm that always outputs a cover of size $\leq \frac{7}{6}$ times the optimal size.

**Theorem (Dinur & Safra 2002)**: Unless $P=NP$, no poly-time algorithm has performance ratio better than $10\sqrt{2} - 1 = 1.556$.

**Conjecture (Khot & Regev 2003)**: Unless $P=NP$, no poly-time algorithm has performance ratio better than 2. [I.e., the above trivial 2-approximation algorithm is optimal!]

- this is a corollary of the so-called Unique Games Conjecture (Khot’01)

**Complexity theory goal**: Characterize optimal approximation ratios for NP-complete optimization problems.

- How can such theorems be proved? Need reductions with gaps:
  - satisfiable $\rightarrow$ G has vertex cover with size $\leq k$
  - unsatisfiable $\rightarrow$ any vertex cover has size $\geq \frac{7}{6}k$.

**PCP Theorem (Arora, Lund, Motwani, Sudan, Szegedy ’92)**
or 1 (true)

Each variable has only two possible assignments: 0 (false)
Furthermore, add inequalities 0 ≤ x ≤ 1 to ensure that
x_1 + (1 - x_2) + x_5 ≥ 1.
For each clause alike (x_1 ∨ x_2 ∨ x_3), add inequalities
to show that if a NP hard, reduce from 3-SAT.
Since we can easily check a proposed integer solution, the

**Proof:** Integer Programming is NP-Complete.

**Theorem:** Integer Programming

**Theorem:** Linear Programming

*Feasible integer solutions* tell if there are no points for the
feasible region.

**Example:** \( x \geq \frac{1}{2}, 2x + y \leq 1 \)

*Infeasible* if there is no solution to all
inequalities with rational coefficients.

Linear programming is set of linear

*Infeasible* if there is no solution to all
inequalities with rational coefficients.

Linear programming is a set of linear

*Infeasible* if there is no solution to all
inequalities with rational coefficients.

Linear programming is a set of linear
Def: \( \text{CLIQUE} = \{ <G, k> \mid G \text{ is an undirected graph with a } k\)-clique, i.e., a set of } k \text{ vertices any two of which are connected by an edge} \}\)

Ex:

![Diagram of a 3-clique and a 4-clique](image)

Theorem: CLIQUE is NP-complete.

Proof:

First, CLIQUE \(NP\); here is a poly-time verifier:

\[
\begin{align*}
\text{On input } & <G, k>, c : \\
\text{1. Test that } & c \text{ is a set of } k \text{ nodes of } G \\
\text{2. Test that } & G \text{ includes an edge between every pair of vertices in } c. \\
\text{3. If both pass, accept, else reject.}
\end{align*}
\]

(To get a nondeterministic Turing machine, guess \(c\) first.)

To show that CLIQUE is NP-hard, we want to show that every NP language \(L\) reduces in poly-time to CLIQUE. But since every \(L\) already reduces to 3-SAT (3-SAT is NP-hard) it is enough to reduce 3-SAT to CLIQUE.

Reduction: Input: a \(k\)-clause 3-SAT formula 

\[\phi = \left( a_1 \lor b_1 \lor c_1 \right) \land \left( a_2 \lor b_2 \lor c_2 \right) \land \cdots \land \left( a_k \lor b_k \lor c_k \right)\]

Output: \(<G, k>, \text{ where } G \text{ is given as follows} >

3 vertices for every clause
edges between all pairs of vertices for different clauses, except those for opposite literals, e.g., \(x_2\) and \(\overline{x_2}\)

Example: 

\[\phi = (x_1 \lor x_1 \lor x_2) \land (x_1 \lor \overline{x_2} \lor \overline{x_1}) \land (x_1 \lor x_2 \lor \overline{x_2})\]

\[G:\]

![Diagram of a reduction](image)

Claim: \(\phi\) is satisfiable iff \(G\) has a \(k\)-clique.
Graph coloring problems

**Def**: A coloring of a graph is an assignment of colors to its nodes so that no two adjacent nodes are assigned the same color.

**Theorem**: Let \( \text{SCOLOR} = \{ <G> \mid \text{the nodes of } G \text{ can be colored with three colors so no two nodes joined by an edge have the same color} \} \)

Then \( \text{SCOLOR} \) is an NP-complete problem.

**Proof sketch**: \( \text{SCOLOR} \in \text{NP} \) clearly.

For NP-hardness, reduce from 3SAT.

Use the gadgets

\[
\begin{align*}
\text{palette} & \quad \text{variable} \quad \text{clause gadget} \\
\end{align*}
\]

**Theorem** (Blum & Karger '97): Assuming \( G \) is a 3-colorable graph, there is a polynomial-time algorithm that colors it using at most \( n \log n \) colors.

**Theorem** (Khanna, Linial, Safra '93): If \( P \neq \text{NP} \), there is no poly-time algorithm that colors 3-colorable graphs using 4 colors.

---

*huge gap!*