Complexity Theory
Chapter 17

Problem: “Given a graph, is it connected?”
Each particular graph is an instance.
The size of the instance, $n$, is the number of bits needed to specify it.
An algorithm is polynomial-time if it uses at most $kn^2$ steps, for some constants $k,c$.

P is the class of all problems that have polynomial-time algorithms.

### Definition of P

P. Problems for which there is a poly-time algorithm.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
<th>Algorithm</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTIPLE</td>
<td>Is $x$ a multiple of $y$?</td>
<td>Grade school division</td>
<td>$51, 17$</td>
<td>$51, 16$</td>
</tr>
<tr>
<td>RELPRIME</td>
<td>Are $x$ and $y$ relatively prime?</td>
<td>Euclid (300 BCE)</td>
<td>$34, 39$</td>
<td>$34, 51$</td>
</tr>
<tr>
<td>PRIMES</td>
<td>Is $x$ prime?</td>
<td>AKS (2002)</td>
<td>$31$</td>
<td>$11$</td>
</tr>
<tr>
<td>EDIT-DISTANCE</td>
<td>Is the edit distance between $x$ and $y$ less than 5?</td>
<td>Dynamic programming</td>
<td>Neither</td>
<td>neither</td>
</tr>
<tr>
<td>LSOLVE</td>
<td>Is there a vector $x$ that satisfies $Ax = b$?</td>
<td>Gauss-Edmonds elimination</td>
<td>$0, 1, 2, 4$</td>
<td>$3, 1, 5$</td>
</tr>
</tbody>
</table>

**EXPTIME (or EXP)**

- Problems that can be solved in exponential time ($O(2^{f(N)})$ for some polynomial function $f(N)$)

### Definition of NP

NP. Problems whose solutions can be verified in polynomial time.

For these problems, an algorithm takes the problem plus a potential solution and verifies that it’s indeed a solution in polynomial time $O(N^k)$ for some constant $k$.

Certification algorithm intuition:
- Certifier views things from “managerial” viewpoint.
- Certifier doesn’t solve the problem on its own; rather, it checks a proposed solution $t$ is indeed a solution.

Def. Algorithm $C(s, t)$ is a certifier for a decision problem $X$ if for every instance $s$, it has a yes answer iff there exists a certificate $t$ such that $C(s, t) = \text{yes}$.

NP. Problems for which there exists a poly-time certifier.

Remark. NP stands for nondeterministic polynomial-time, which is an equivalent definition of the same class of problems.
Certifiers and Certificates: Composite

**COMPOSITES.** Given an integer \( s \), is \( s \) composite?

**Certificate.** A nontrivial factor \( t \) of \( s \). Note that such a certificate exists iff \( s \) is composite. Moreover \( |t| \leq |s| \).

**Certifier.**

```java
boolean C(s, t) {
    if (t <= 1 or t >= s)
        return false
    else if (remainder(s, t) == 0)
        return true
    else
        return false
}
```

**Instance.** \( s = 437,669 \).

**Certificate.** \( t = 541 \) or \( 809 \).

**Conclusion.** COMPOSITES is in NP.

Every problem in P is also in NP.

---

Certifiers and Certificates: Satisfiability

**SAT.** Given a formula \( \Phi \), is there a satisfying assignment?

**Certificate.** An assignment of truth values to the \( n \) boolean variables.

**Certifier.** Check that each clause in \( \Phi \) has at least one true literal.

**Ex.**

\[
((\pi \lor x_1 \lor x_3) \land (\pi \lor x_1 \lor x_3) \land (\pi \lor x_1 \lor x_3) \land (\pi \lor x_1 \lor x_3))
\]

**Instance s**

\[
\begin{align*}
\pi & = 1, \\
x_1 & = 1, \\
x_2 & = 0, \\
x_3 & = 1
\end{align*}
\]

**Certificate t**

**Conclusion.** SAT is in NP.

---

Certifiers and Certificates: Hamiltonian Cycle

**HAM-CYCLE.** Given an undirected graph \( G = (V, E) \), does there exist a simple cycle \( C \) that visits every node?

**Certificate.** A permutation of the \( n \) nodes.

**Certifier.** Check that the permutation contains each node in \( V \) exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.

**Conclusion.** HAM-CYCLE is in NP.

---

P, NP, EXP

**P.** Decision problems for which there is a poly-time algorithm.

**EXP.** Decision problems for which there is an exponential-time algorithm.

**NP.** Decision problems for which there is a poly-time certifier.

**Claim.** \( P \subseteq NP \).

**Pf.** Consider any problem \( X \) in \( P \).

\[ \text{By definition, there exists a poly-time algorithm } A(s) \text{ that solves } X. \]

\[ \text{Certificate: } t = \varepsilon, \text{ certifier } C(s, t) = A(s). \]

**Claim.** \( NP \subseteq EXP \).

**Pf.** Consider any problem \( X \) in \( NP \).

\[ \text{By definition, there exists a poly-time certifier } C(s, t) \text{ for } X. \]

\[ \text{To solve input } s, \text{ run } C(s, t) \text{ on all strings } t \text{ with } |t| \leq p(|s|), \text{ where } |t| \text{ and } |s| \text{ are the sizes of } s \text{ and } t, \text{ respectively.} \]

\[ \text{Return } yes, \text{ if } C(s, t) \text{ returns } yes \text{ for any of these.} \]

---

The Main Question: P Versus NP

**Does P = NP?** [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

\[ \text{Is solving a problem as easy as verifying its solution (upto a poly-time difference)?} \]

\[ \text{Clay$1 million prize.} \]

\[ \text{If yes: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, ...} \]

\[ \text{If no: No efficient algorithms possible for 3-COLOR, TSP, SAT, ...} \]

**Consensus opinion on P = NP?** Probably no.
Polynomial-Time Reduction

Suppose we could solve $X$ in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem $X$ polynomial reduces to problem $Y$ if arbitrary instances of problem $X$ can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to the blackbox that solves problem $Y$.

Notation. $X \leq_P Y$.

That is, if the code of $Y$ is $B$, we may obtain the code $A$ which uses $B$ to solve $X$ (the time spent by $B$ is not cared).

Purpose. Classify problems according to relative difficulty.

Design algorithms. If $X \leq_P Y$ and $Y$ can be solved in polynomial-time, then $X$ can also be solved in polynomial time. That is, if $Y$ is easy, so is $X$.

Establish intractability. If $X \leq_P Y$ and $X$ cannot be solved in polynomial-time, then $Y$ cannot be solved in polynomial time. That is, if $X$ is hard, so is $Y$.

Establish equivalence. If $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$. up to cost of reduction.

Reduction By Simple Equivalence

Basic reduction strategies:

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Clique

Clique: Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each pair $(x, y)$ of points in $S$, $(x, y)$ is an edge of $E$?

Ex. Is there an independent set of size $\geq 6$? Yes. Ex. Is there an independent set of size $\geq 7$? No.

Independent Set

INDEPENDENT SET: Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in $S$?

Ex. Is there an independent set of size $\geq 6$? Yes. Ex. Is there an independent set of size $\geq 7$? No.

Vertex Cover

VERTEX COVER: Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in $S$?

Ex. Is there a vertex cover of size $\leq 4$? Yes. Ex. Is there a vertex cover of size $\leq 3$? No.
Vertex Cover and Independent Set

Claim. \textsc{Vertex-Cover} \equiv \textsc{Independent-Set}.

\textbf{Pf.} We show \(S\) is an independent set iff \(V - S\) is a vertex cover.

\[\begin{align*}
\text{Claim. } & \textsc{Vertex-Cover} \equiv \textsc{Independent-Set}. \\
\text{Pf.} & \quad \text{We show } S \text{ is an independent set iff } V - S \text{ is a vertex cover.} \\
& \Rightarrow \\
& \quad \text{Let } S \text{ be any independent set.} \\
& \quad \text{Consider an arbitrary edge } (u, v). \\
& \quad S \text{ independent } \Rightarrow u \in S \text{ or } v \in S \Rightarrow u \in V - S \text{ or } v \in V - S. \\
& \quad \text{Thus, } V - S \text{ covers } (u, v). \\
& \Leftarrow \\
& \quad \text{Let } V - S \text{ be any vertex cover.} \\
& \quad \text{Consider two nodes } u \in S \text{ and } v \in S. \\
& \quad \text{Observe that } (u, v) \notin E \text{ since } V - S \text{ is a vertex cover.} \\
& \quad \text{Thus, no two nodes in } S \text{ are joined by an edge } \Rightarrow S \text{ independent set.}\]

Set Cover

\textsc{Set-Cover}: Given a set \(U\) of elements, a collection \(S_1, S_2, \ldots, S_m\) of subsets of \(U\), and an integer \(k\), does there exist a collection of \(\leq k\) of these sets whose union is equal to \(U\)?

Sample application.
- \(m\) available pieces of software.
- \(U\) of \(n\) capabilities that we would like our system to have.
- The \(i\)th piece of software provides the set \(S_i \subseteq U\) of capabilities.
- Goal: achieve all \(n\) capabilities using fewest pieces of software.

\[\begin{align*}
U &= \{1, 2, 3, 4, 5, 6, 7\} \\
k &= 2 \\
S_1 &= \{3, 7\} \\
S_2 &= \{2, 4\} \\
S_3 &= \{3, 4, 5, 6\} \\
S_4 &= \{5\} \\
S_5 &= \{1\} \\
S_6 &= \{1, 2, 6, 7\}
\end{align*}\]

Vertex Cover Reduces to Set Cover

Claim. \textsc{Vertex-Cover} \leq_p \textsc{Set-Cover}.

\textbf{Pf.} Given a \textsc{Vertex-Cover} instance \(G = (V, E), k\), we construct a \textsc{set-cover} instance whose size equals the size of the vertex cover instance.

Construction.
- Create \textsc{Set-Cover} instance:
  - \(k = k\), \(U = E\), \(S_v = \{e \in E : e \text{ incident to } v\}\).
  - Set-cover of size \(\leq k\) iff vertex cover of size \(\leq k\).

\[\begin{align*}
\text{Vertex-Cover} & \quad \textsc{Set-Cover} \\
U &= \{1, 2, 3, 4, 5, 6, 7\} \\
k &= 2 \\
S_1 &= \{3, 7\} \\
S_2 &= \{2, 4\} \\
S_3 &= \{3, 4, 5, 6\} \\
S_4 &= \{5\} \\
S_5 &= \{1\} \\
S_6 &= \{1, 2, 6, 7\}
\end{align*}\]

NP-Hard and NP-Complete

\textbf{NP-hard}: A problem \(Y\) is \(NP\)-hard if, for every problem \(X\) in \(NP\), \(X \leq_p Y\). A problem \(Y\) is \(NP\)-complete, if it is \(NP\)-hard and in \(NP\).

\textbf{Theorem}. Suppose \(Y\) is an \(NP\)-complete problem. Then \(Y\) is in \(P\) iff \(P = NP\).

\textbf{Pf. \Leftarrow} Suppose \(Y\) is in \(P\), i.e., can be solved in poly-time.
- Let \(X\) be any problem in \(NP\). Since \(X \leq_p Y\), we can solve \(X\) in poly-time. This implies \(NP \subseteq P\).
- We already know \(P \subseteq NP\). Thus \(P = NP\). \(\blacksquare\)

\textbf{Fundamental question}. Do there exist "natural" \(NP\)-complete problems?
NP-hard

NP-complete

NP

P

Hamilton cycle
Graph 3-coloring
Satisfiability
Maximum clique

Halting problem
Factoring
Graph isomorphism
Minimum circuit size

Graph connectivity
Primality testing
Matrix determinant
Linear programming

HALTING PROBLEM

SAT

SAT is NP-Complete

- The Cook-Levin theorem (or just Cook’s theorem) proves that SAT is NP-complete
- For details, see: http://en.wikipedia.org/wiki/Cook-Levin_theorem
- Need to show:
  - SAT is in NP
  - All other problems in NP can be reduced to SAT

Establishing NP-Completeness

Remark. Once we establish first “natural” NP-complete problem, others fall like dominoes.

Recipe to establish NP-completeness of problem Y.
- Step 1. Show that Y is in NP.
- Step 2. Choose an NP-complete problem X.
- Step 3. Prove that \( X \leq_P Y \).

Justification. If X is an NP-complete problem, and Y is a problem in NP with the property that \( X \leq_P Y \) then Y is NP-complete.

Pf. Let W be any problem in NP. Then \( W \leq_P X \leq_P Y \).

- By transitivity, \( W \leq_P Y \).
- Hence Y is NP-complete.

3-SAT is NP-Complete

Theorem. 3-SAT is NP-complete.

Pf. Suffices to show that SAT \( \leq_P \) 3-SAT since 3-SAT is in NP.
- Let K be any circuit representing the formula of SAT.
- Create a 3-SAT variable \( x_i \) for each circuit element i.
- Make circuit compute correct values at each node:
  - \( x_i = x_j \) \( \Rightarrow \) add 2 clauses: \( x_i, x_j \) and \( \overline{x_i}, \overline{x_j} \)
  - \( x_i = \overline{x_j} \) \( \Rightarrow \) add 3 clauses: \( \overline{x_i}, x_j, x_i \cup x_j \)
  - \( x_i = x_j \) \( \Rightarrow \) add 3 clauses: \( x_i, \overline{x_j}, x_i \cup \overline{x_j} \)
- Hard-coded input values and output value.
  - \( x_0 = 0 \) \( \Rightarrow \) add 1 clause: \( \overline{x_0} \)
  - \( x_0 = 1 \) \( \Rightarrow \) add 1 clause: \( x_0 \)
- Final step: turn clauses of length < 3 into clauses of length exactly 3 by introducing new variables.

3-SAT Reduces to Independent Set

Claim. 3-SAT \( \leq_P \) INDEPENDENT-SET.

Pf. Given an instance \( \Phi \) of 3-SAT, we construct an instance \( (G, k) \) of INDEPENDENT-SET that has an independent set of size \( k \) iff \( \Phi \) is satisfiable.

Construction.
- \( G \) contains 3 vertices for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.

Ex:
- \( C_1 = x_1 \vee \overline{x_2} \vee x_3 \)
- \( C_2 = \overline{x_1} \vee x_2 \vee \overline{x_3} \)
- \( C_3 = \overline{x_1} \vee \overline{x_2} \vee x_3 \)

Yes: \( x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false} \).
3-SAT Reduces to Independent Set

**Claim.** \( G \) contains independent set of size \( k = |\varnothing| \) iff \( \varnothing \) is satisfiable.

**Pf.** \( \Rightarrow \) Let \( S \) be independent set of size \( k \).
- \( S \) must contain exactly one vertex in each triangle.
- Set these literals to true. ... and any other variable in a consistent way.
- Truth assignment is consistent and all clauses are satisfied.

**Pf.** \( \Leftarrow \) Given satisfying assignment, select one true literal from each triangle. This is an independent set of size \( k \).

**Review**

- **Basic reduction strategies.**
  - Simple equivalence: \( \text{INDEPENDENT-SET} \equiv \text{VERTEX-COVER} \).
  - Special case to general case: \( \text{VERTEX-COVER} \leq_p \text{SET-COVER} \).
  - Encoding with gadgets: \( \text{3-SAT} \leq_p \text{INDEPENDENT-SET} \).

- **Transitivity.** If \( X \leq_p Y \) and \( Y \leq_p Z \), then \( X \leq_p Z \).
  - Pf idea. Compose the two algorithms.
- **Ex:** \( \text{3-SAT} \leq_p \text{INDEPENDENT-SET} \leq_p \text{VERTEX-COVER} \leq_p \text{SET-COVER} \).

**Detection, Reporting, and Optimization Problems**

- For the subset sum problem:
  - Detection—Is there a subset of the numbers that adds up to a specific value \( k \)? (Decision Problems)
  - Reporting—Find a subset of the numbers that adds up to the specific value \( k \) (if such a subset exists)
  - Optimization—Find a subset of the numbers with a total as close to the specific value \( k \) as possible

**Detection \( \leq_p \) Reporting**

- (Detection is reducible to reporting)
- Suppose algorithm \( \text{Report}(k) \) returns a subset with values that add up to \( k \)
- \( \text{Detect}(k) \) calls \( \text{Report}(k) \) and returns true if \( \text{Report}(k) \) finds such a subset

**Reporting \( \leq_p \) Optimization**

- (Reporting is reducible to optimization)
- Suppose algorithm \( \text{Optimize}(k) \) returns a subset with sum as close as possible to \( k \)
- \( \text{Report}(k) \) calls \( \text{Optimize}(k) \) and returns the subset if its total value is \( k \)

**Reporting \( \leq_p \) Detection**

- (Reporting is reducible to detection)
- Suppose algorithm \( \text{Detect}(k) \) returns true if there is a subset with sum \( k \)
Reporting \( \leq_p \) Detection Algorithm

1. Call Detect\( k \) on the whole set to see if a solution is even possible
2. If a solution is possible, for each value \( V_i \) in the set:
   a. Remove \( V_i \) from the set and call Detect\( k \) for the remaining set to see if there is a subset with total value \( k \)
   b. If Detect\( k \) returns false, restore \( V_i \) to the set, and continue the loop at Step 2
   c. If Detect\( k \) returns true, leave \( V_i \) out of the set, and continue the loop at Step 2

When the loop finishes, the remaining values make a set with total value \( k \)

Optimization \( \leq_p \) Reporting

- (Optimization is reducible to reporting)
- Suppose algorithm Report\( k \) returns a subset with total value \( k \) (if one exists)

1. For \( i = 0 \) to \( N \), where \( N \) is the number of items in the set:
   a. If Report\( k + i \) returns a subset, Optimize\( k \) returns that subset.
   b. If Report\( k - i \) returns a subset, Optimize\( k \) returns that subset.
   c. Continue the loop in Step 1.

Vertex Cover Problem

- Detection: Does there exist a vertex cover of size \( \leq k \)?
- Reporting: Find a vertex cover of size \( \leq k \).
- Optimization problem. Find vertex cover of minimum cardinality.

- Self-reducibility: Detection, Reporting, and Optimization Problems are all equivalent \( (r_3) \)
- Applies to all (NP-complete) problems.
  - Justifies our focus on decision problems.

Vertex Cover Problem

- Ex: Reduce Optimization to Detection
  - To find min cardinality vertex cover.
    - (Binary) search for cardinality \( k^* \) of min vertex cover.
    - Find a vertex \( v \) such that \( G - \{ v \} \) has a vertex cover of size \( \leq k^* - 1 \).
      - any vertex in any min vertex cover will have this property
        - Include \( v \) in the vertex cover.
        - Recursively find a min vertex cover in \( G - \{ v \} \).

NP-Complete Problems

- More than 3,000 known NP-complete problems
  - Art gallery problem—Find the minimum number of guards needed
  - Bin packing—Pack objects in the fewest bins possible
  - Bottleneck TSP—Find a Hamiltonian path with minimum largest link cost
  - Chromatic number (or vertex coloring)—Given a graph, find the smallest number of colors needed to color the graph’s nodes. (The graph is not necessarily planar.)
  - Clique—in a graph, find the largest clique (mutually connected nodes)
  - Clique cover problem—Given a number \( k \), find a way to partition a graph into \( k \) cliques.

NP-Complete Problems, Part 2

- Degree-constrained spanning tree—Find a spanning tree with a given maximum degree
- Dominating set—Given a graph, find a set of nodes \( S \) so that every other node is adjacent to one of the nodes in the set \( S \)
- Hamiltonian cycle—Determine whether there is a path through a graph that visits every node exactly once and then returns to its starting point
- Hamiltonian path (HAM)—Determine whether there is a path through a graph that visits every node exactly once
- Longest path—Find the longest path that doesn’t revisit any nodes
- Traveling Salesman Problem (TSP)—Determine whether there is a path through a graph that visits every node exactly once with minimal distance.
NP-Complete Problems, Part 3

• Knapsack—Given a knapsack with a capacity and a set of objects with weights and values, find the set of objects with the largest possible value that fits in the knapsack.
• Maximum independent set—Find the largest set of nodes where no two nodes in the set are connected by a link
• Maximum leaf spanning tree—Find a spanning tree that has the maximum possible number of leaves
• Minimum leaf spanning tree—Find a spanning tree that has the minimum possible number of leaves
• Minimum degree spanning tree—Find a spanning tree with the minimum possible degree

NP-Complete Problems, Part 4

• Partitioning—Given a set of integers, find a way to divide the values into two sets with the same total value
• Satisfiability (SAT)—Given a boolean expression containing variables, find an assignment of true and false to the variables to make the expression true
• Three-satisfiability (3SAT)—Given a boolean expression in 3CNF; find an assignment of true and false to the variables to make the expression true
• Subset sum—Given a set of integers, find a subset with a given total value

NP-Complete Problems, Part 5

• Traveling salesman problem (TSP)—Given a list of cities and the distances between them, find the shortest possible route that visits all the cities and returns to the starting city
• Unbounded knapsack—Similar to the knapsack problem, except that you can select any item multiple times
• Vehicle routing—Given a set of customers and a fleet of vehicles, find the most efficient routes for the vehicles to visit all the customers
• Vertex cover—Find a minimal set of vertices so that every link in the graph touches one of the selected vertices

Subset Sum

SUBSET-SUM. Given a set of natural numbers \( w_1, \ldots, w_n \) and an integer \( W \), is there a subset that sums to \( W \)?

Ex: \( \{ 1, 4, 6, 16, 256, 1040, 1041, 1093, 1284, 1344 \} \), \( W = 3754 \).

Yes: \( 1 \cdot 16 + 64 \cdot 1040 + 1093 + 1284 = 3754 \).

Remark. With arithmetic problems, input integers are encoded in binary. Polynomial reduction must be polynomial in the size of binary encoding.

Claim. 3-SAT \( \leq_p \) SUBSET-SUM.

PF. Given an instance \( \phi \) of 3-SAT, we construct an instance of SUBSET-SUM that has solution iff \( \phi \) is satisfiable.

Observation. All problems below are NP-complete and polynomial reduce to one another!
Set Partition

**PARTITION.** Given a set of natural numbers \(w_1, \ldots, w_n\), is there a subset that adds up to exactly half sum of all \(w_i\)?

**Claim.** \(\text{PARTITION} \leq_P \text{SUBSET-SUM} \).

**Pf.** \(\text{PARTITION} \) is a special of \(\text{SUBSET-SUM} \).

**Claim.** \(\text{SUBSET-SUM} \leq_P \text{PARTITION} \).

**Pf.**

Bin Packing

**BIN-PACKING.** Given a set \(S\) of real numbers \(w_1, \ldots, w_n\), \(0 < w_i \leq 1\), and integer \(K\), is there a partition of \(S\) into \(K\) subsets such that each subset adds up no more than 1?

**Claim.** \(\text{PARTITION} \leq_P \text{BIN-PACKING} \).

**Pf.**

The Knapsack Problem

**Input**
- Capacity \(K\)
- \(n\) items with weights \(w_i\) and values \(v_i\)

**Goal**
- Output a set of items \(S\) such that
  - the sum of weights of items in \(S\) is at most \(K\)
  - and the sum of values of items in \(S\) is maximized

Asymmetry of NP

**Asymmetry of NP.** We only need to have short proofs of yes instances.

**Ex 1.** \(\text{SAT vs. UNSAT} \).
- Can prove a CNF formula is satisfiable by giving such an assignment.
- How could we prove that a formula is not satisfiable?

**Ex 2.** \(\text{HAM-CYCLE vs. NO-HAM-CYCLE} \).
- Can prove a graph is Hamiltonian by giving such a Hamiltonian cycle.
- How could we prove that a graph is not Hamiltonian?

**Remark.** \(\text{SAT}\) is NP-complete, but how do we classify \(\text{UNSAT}\)?

NP and co-NP

**NP.** Decision problems for which there is a poly-time certifier.
- \(\text{SAT, HAM-CYCLE, COMPOSITES}\)

**Def.** Given a decision problem \(X\), its complement \(\overline{X}\) is the same problem with the yes and no answers reverse.

**Ex.** \(X = \{ 0, 1, 4, 6, 8, 9, 10, 12, 14, 15, \ldots \} \) -- composite numbers
- \(X' = \{ 2, 3, 5, 7, 11, 13, 17, 23, 29, \ldots \} \) -- primes

**Equivalence:** Since \(X \cup X' = \mathbb{N}\), we have \(X = \overline{X'}\).

**co-NP.** Complements of decision problems in NP.
- \(\text{TAUTOLEGNY, NO-HAM-CYCLE, PRIMES}\)

NP = co-NP?

**Fundamental question.** Does \(NP = co-NP\)?
- Do yes instances have succinct certificates if no instances do?
- Consensus opinion: no.

**Theorem.** If \(NP = co-NP\), then \(P = NP\).

**Pf idea.**
- \(P\) is closed under complementation.
- If \(P = NP\), then \(NP\) is closed under complementation.
- In other words, \(NP = co-NP\).
- This is the contrapositive of the theorem.