NP completeness and computational tractability
Part II

Grand challenge: Classify Problems According to Computational Requirements

Q. Which problems will we be able to solve in practice?
A. working definition. [Cobham 1964, Edmonds 1965, Rabin 1966]
Those with polynomial-time algorithms.

<table>
<thead>
<tr>
<th></th>
<th>Yes</th>
<th>Probably no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Longest path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matching</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3D-matching</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min cut</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max cut</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-SAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-SAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planar 4-color</td>
<td></td>
<td>Planar 3-color</td>
</tr>
<tr>
<td>Planar 3-color</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bipartite vertex cover</td>
<td>Vertex cover</td>
<td></td>
</tr>
<tr>
<td>Primality testing</td>
<td>Factoring</td>
<td></td>
</tr>
</tbody>
</table>

Classify Problems

Desiderata. Classify problems according to those that can be solved in polynomial-time and those that cannot.
For any nice function T(n)
- There are problems that require more than T(n) time to solve.
Frustrating news: Huge number of fundamental problems have defied classification for decades.

NP-completeness: Show that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one really hard problem.

Polynomial-Time Reduction

Desiderata'. 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 oracle that solves problem Y.

Notation. X \ \leq_P \ Y,

Basic Reduction Strategies

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

**Basic reduction strategies.**
- Simple equivalence: \( \textsc{Independent-set} \leq_p \textsc{Vertex-cover} \)
- Special case to general case: \( \textsc{Vertex-cover} \leq_p \textsc{Set-cover} \)
- Encoding with gadgets: \( \textsc{3-Sat} \leq_p \textsc{Independent-set} \)

**Transitivity.** If \( X \leq_p Y \) and \( Y \leq_p Z \), then \( X \leq_p Z \).

**Pf idea.** Compose the two algorithms.

**Ex:** \( \textsc{3-Sat} \leq_p \textsc{Independent-set} \leq_p \textsc{Vertex-cover} \leq_p \textsc{Set-cover} \).

### Self-Reducibility

**Decision problem.** Does there exist a vertex cover of size \( k \)?

**Search problem.** Find vertex cover of minimum cardinality.

**Self-reducibility.** Search problem \( \leq_p \) decision version.
- Applies to all (NP-complete) problems we discuss.
- Justifies our focus on decision problems.

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

### Definition of NP

**Certification algorithm intuition.**
- Certifier views things from "managerial" viewpoint.
- Certifier doesn’t determine whether \( s \in X \) on its own; rather, it checks a proposed proof \( t \) that \( s \in X \).

**Def.** Algorithm \( C(s, t) \) is a **certifier** for problem \( X \) if for every string \( s \), \( s \in X \) if and only if there exists a string \( t \) such that \( C(s, t) = \text{yes} \).

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

**Remark.** NP stands for nondeterministic polynomial-time.

### Decision Problems

**NP -- another definition**

**Nondeterministic Turing machines.**
- At any point in a computation, the machine may proceed according to several possibilities.
- Machine accepts if there is a computation branch that ends in an accepting state.

**Example: NTM for Clique**
- On input \( (G, k) \) where \( G \) is a graph:
  - Nondeterministically select a subset \( S \) of \( k \) nodes of \( G \).
  - Test whether \( G \) contains all edges connecting nodes in \( S \).
  - If yes, accept; else reject.

**Theorem:** A language is in NP if it is decided by some nondeterministic polynomial time Turing machine.
NP -- equivalence of definitions

Theorem: A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.

Proof: Let A be a language in NP.

\[ \Rightarrow \] Let C(s) be a certificate for A that runs in time \( n^k \).
Construct nondeterministic TM N that on input s of length n does:
- Nondeterministically select string t of length at most \( n^k \).
- Run C(s, t).
- If C(s, t) accepts, accept, otherwise reject.

\[ \Leftarrow \] Suppose N is a NTM that decides A. Construct verifier C that on input (s, t) does the following:
- Simulate N on input s, treating each symbol of t as a description of the nondeterministic choice to make at each step.
- If this branch of N’s computation accepts, accept, else reject.

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 certificate.

Claim. \( P \subseteq NP \).
Proof. Consider any problem X in P.
- By definition, there exists a poly-time algorithm A(s) that solves X.
- Certificate: t \( = \) s, certifier C(s, t) = A(s).

Claim. \( NP \subseteq EXP \).
Proof. Consider any problem X in NP.
- By definition, there exists a poly-time certificate C(s, t) for X.
- To solve input s, run C(s, t) on all strings t with \( |t| \leq p(|s|) \).
- Return yes, if C(s, t) returns yes for any of these.

The Main Question: P Versus NP

Does \( P = NP \)? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]
- Is the decision problem as easy as the certification problem?
- Clay $1 million prize.

If yes: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, …
If no: No efficient algorithms possible for 3-COLOR, TSP, SAT, …

Consensus opinion on \( P = NP \): Probably no.

NP-Complete

NP-complete. A problem Y in NP with the property that for every problem X in NP, \( X \leq_p Y \).

Theorem. Suppose Y is an NP-complete problem. Then Y is solvable in poly-time iff \( P = NP \).
Proof. \( \Rightarrow \) If \( P = NP \) then Y can be solved in poly-time since Y is in NP.
Proof. \( \Leftarrow \) Suppose Y 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 \( P = NP \).

Fundamental question. Do there exist "natural" NP-complete problems?

Circuit Satisfiability

CIRCUIT-SAT. Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?
The "First" NP-Complete Problem

**Theorem.** CIRCUIT-SAT is NP-complete. [Cook 1971, Levin 1973]

**Pf.** (sketch+)

Consider some problem X in NP. It has a poly-time certifier C(s, t).
To determine whether s is in X, need to know if there exists a certificate t of length p(|s|) such that C(s, t) = yes.

View C(s, t) as an algorithm, i.e. Turing machine on |s| + p(|s|) bits (input s, certificate t).
- Assumptions about TM:
  - It moves its head all the way to left and writes blank in leftmost tape cell right before halting.
  - Once it halts, it stays in same configuration for all future steps.
- Convert TM it into a poly-size circuit K.
  - first |s| bits are hard-coded with s
  - remaining p(|s|) bits represent bits of t

Construct circuit K that is satisfiable iff C(s, t) = yes.

Establishing NP-Completeness

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

**Recipe to establish NP-completeness of problem Y.**

1. Show that Y is in NP.
2. Choose an NP-complete problem X.
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.

Some NP-Complete Problems

Six basic genres of NP-complete problems and paradigmatic examples.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Practice. Most NP problems are either known to be in P or NP-complete.

Notable exceptions. Factoring, graph isomorphism.

Sequencing Problems

**Basic genres.**

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Hamiltonian Cycle

**HAMILTONIAN-CYCLE:** given an undirected graph \(G = (V, E)\), does there exist a simple cycle \(I\) that contains every node in \(V\).

YES: vertices and faces of a dodecahedron.
Hamiltonian Cycle

**HAM-CYCLE:** given an undirected graph $G = (V, E)$, does there exist a simple cycle $C$ that contains every node in $V$.

Claim. **HAM-CYCLE** is in $NP$.

Proof. Given a directed graph $G = (V, E)$, construct an undirected graph $G'$ with $3n$ nodes.

Directed Hamiltonian Cycle

**DIR-HAM-CYCLE:** given a digraph $G = (V, E)$, does there exist a simple directed cycle $C'$ that contains every node in $V$.

Claim. **DIR-HAM-CYCLE** is in $P$.

Proof. Given a directed graph $G = (V, E)$, construct an undirected graph $G'$ with $3n$ nodes.

3-SAT Reduces to Directed Hamiltonian Cycle

Claim. **3-SAT** reduces to **DIR-HAM-CYCLE**.

Proof. Given a 3-SAT instance $\Phi$ with $n$ variables $x_i$ and $k$ clauses.

Construction. Given 3-SAT instance $\Phi$ with $n$ variables $x_i$ and $k$ clauses.

For each clause: add a node and 6 edges.
3-SAT Reduces to Directed Hamiltonian Cycle

Claim. \( \phi \) is satisfiable iff \( G \) has a Hamiltonian cycle.

Pf. 
- Suppose 3-SAT instance has satisfying assignment \( x^* \).
- Then, define Hamiltonian cycle in \( G \) as follows:
  - if \( x^*_i = 1 \), traverse row \( i \) from left to right
  - if \( x^*_i = 0 \), traverse row \( i \) from right to left
  - for each clause \( C_j \), there will be at least one row \( i \) in which we are going in "correct" direction to splice node \( C_j \) into tour.

Pf. 
- Suppose \( G \) has a Hamiltonian cycle \( \Gamma \).
  - If \( \Gamma \) enters clause node \( C_j \), it must depart on mate edge.
  - that, nodes immediately before and after \( C_j \) are connected by an edge \( e \) in \( G \)
  - removing \( C_j \) from cycle, and replacing it with edge \( e \) yields Hamiltonian cycle on \( G - \{ C_j \} \).
  - Continuing in this way, we are left with Hamiltonian cycle \( \Gamma' \) in \( G - \{ C_1, C_2, \ldots, C_k \} \).
  - Set \( x^*_i = 1 \) iff \( \Gamma' \) traverses row \( i \) left to right.
  - Since \( \Gamma' \) visits each clause node \( C_j \), at least one of the paths is traversed in "correct" direction, and each clause is satisfied.

Longest Path

SHORTEST-PATH. Given a digraph \( G = (V, E) \), does there exists a simple path of length at most \( k \) edges?

LONGEST-PATH. Given a digraph \( G = (V, E) \), does there exists a simple path of length at least \( k \) edges?

Prove that LONGEST-PATH is NP-complete.

The Longest Path

Lyrics. Copyright © 1988 by Daniel J. Barrett.
Music. Sung to the tune of "The Longest Time" by Billy Joel.
http://www.cs.princeton.edu/~wayne/cs423/lectures/longest-path.mp3

Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path!
If you said P is NP tonight,
There would still be papers left to write,
I have a weakness,
I’m addicted to completeness,
And I keep searching for the longest path.
The algorithm I would like to see
Is of polynomial degree,
But it’s elusive:
Nobody has found conclusive
Evidence that we can find a longest path.
I have been hard working for so long.
I swear it’s right, and he marks it wrong.
Some how I’ll feel sorry when it’s done:
GPA 2.1
Is more than I hope for.
Garey, Johnson, Karp and other men (and women)
Tried to make it order N log N.
Am I a mad fool
If I spend my life in grad school,
Forever following the longest path?
Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path.

\( ^* \) Recorded by Dan Barrett while a grad student at Johns Hopkins during a difficult algorithms final.

Traveling Salesperson Problem

TSP. Given a set of \( n \) cities and a pairwise distance function \( d(u, v) \), is there a tour of length \( \leq D \)?

All U.S. data is from U.S. Bureau of the Census, unless noted.
Reference: http://www.census.gov

Reference: http://www.tsp.gatech.edu

Traveling Salesperson Problem

TSP. Given a set of \( n \) cities and a pairwise distance function \( d(u, v) \), is there a tour of length \( \leq D \)?

Optimal TSP tour
Reference: http://www.tsp.gatech.edu
Traveling Salesperson Problem

TSP: Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length at most $D$?

**HAM-CYCLE:** Given a graph $G = (V, E)$, does there exist a simple cycle that contains every node in $V$?

**Claim.** $\text{HAM-CYCLE} \leq_{P} \text{TSP}$.

**Proof.**
1. Given an instance $G = (V, E)$ of $\text{HAM-CYCLE}$, create $n$ cities with distance function $d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \notin E \end{cases}$.
2. TSP instance has tour of length at most $n$ if and only if $G$ is Hamiltonian.

**Remark.** TSP instance in reduction satisfies 3-inequality.

**Partioning Problems**

- Basic genres:
  - Packing problems: SET-PACKING, INDEPENDENT SET.
  - Covering problems: SET-COVER, VERTEX COVER.
  - Constraint satisfaction problems: SAT, 3-SAT.
  - Sequencing problems: HAMILTONIAN-CYCLE, TSP.
  - Partitioning problems: 3D-MATCHING, 3-COLOR.
  - Numerical problems: SUBSET-SUM, KNAPSACK.

3-Dimensional Matching

**3D-MATCHING.** Given $n$ instructors, $n$ courses, and $n$ times, and a list of the possible courses and times each instructor is willing to teach, is it possible to make an assignment so that all courses are taught at different times?

<table>
<thead>
<tr>
<th>Instructor</th>
<th>Course</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wayne</td>
<td>COS 423</td>
<td>MW 11-12:20</td>
</tr>
<tr>
<td>Wayne</td>
<td>COS 226</td>
<td>TTh 11-12:20</td>
</tr>
<tr>
<td>Wayne</td>
<td>COS 126</td>
<td>TTh 11-12:20</td>
</tr>
<tr>
<td>Tardos</td>
<td>COS 423</td>
<td>TTh 11-12:20</td>
</tr>
<tr>
<td>Tardos</td>
<td>COS 226</td>
<td>TTh 11-12:20</td>
</tr>
<tr>
<td>Kleinberg</td>
<td>COS 423</td>
<td>MW 11-12:20</td>
</tr>
<tr>
<td>Kleinberg</td>
<td>COS 226</td>
<td>MW 11-12:20</td>
</tr>
</tbody>
</table>

**3D-MATCHING.** Given disjoint sets $X$, $Y$, and $Z$, each of size $n$ and a set $T \subseteq X \times Y \times Z$ of triples, does there exist a set of $n$ triples in $T$ such that each element of $X \cup Y \cup Z$ is in exactly one of these triples?
Graph Coloring

Basic genres:
- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Register Allocation

Register allocation. Assign program variables to machine register so that no more than k registers are used and no two program variables that are needed at the same time are assigned to the same register.

Interference graph. Nodes are program variables names, edge between u and v if there exists an operation where both u and v are “live” at the same time.

Observation. [Chaitin 1982] Can solve register allocation problem iff interference graph is k-colorable.

Fact. 3-COLOR \( \leq_p \) k-REGISTER-ALLOCATION for any constant k \( \leq 3 \).

3-Colorability

Claim. 3-SAT \( \leq_p \) 3-COLOR.

Pf. Given 3-SAT instance \( \Phi \), we construct an instance of 3-COLOR that is 3-colorable iff \( \Phi \) is satisfiable.

Construction:
1. For each literal, create a node.
2. Create 3 new nodes T, F, B; connect them in a triangle, and connect each literal to B.
3. Connect each literal to its negation.
4. For each clause, add gadget of 6 nodes and 13 edges.

3-Colorability

Claim. Graph is 3-colorable iff \( \Phi \) is satisfiable.

Pf. Suppose graph is 3-colorable.
1. Consider assignment that sets all T literals to true.
2. (ii) ensures each literal is T or F.
3. (iii) ensures a literal and its negation are opposites.
4. (iv) ensures at least one literal in each clause is T.
3-Colorability

Claim. Graph is 3-colorable iff \( \phi \) is satisfiable.

Pf. \( \Rightarrow \) Suppose graph is 3-colorable.

1. Consider assignment that sets all T literals to true.
2. (ii) ensures each literal is T or F.
3. (iii) ensures a literal and its negation are opposites.
4. (iv) ensures at least one literal in each clause is T.

\[ C_i = x_1 \lor \overline{x}_2 \lor x_3 \]

\( C_1 = \{x_1, x_2, x_3\} \)

\( C_2 = \{x_2, x_3, x_4\} \)

\( C_3 = \{x_1, x_3, x_4\} \)

\( \text{iff all are red} \)

\( \text{contradiction} \)

Numerical Problems

Basic genres:
- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3-COLOR, 3D-MATCHING.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Subset Sum

Construction. Given 3-SAT instance \( \phi \) with \( n \) variables and \( k \) clauses, form \( 2n + 2k \) decimal integers, each of \( n+k \) digits, as illustrated below.

Claim. \( \phi \) is satisfiable iff there exists a subset that sums to \( W \).

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

\[ C_1 = x_1 \lor \overline{x}_2 \lor x_3 \]

\[ C_2 = x_2 \lor \overline{x}_1 \lor x_3 \]

\[ C_3 = \overline{x}_1 \lor \overline{x}_2 \lor x_3 \]

\( \text{iff all are red} \)

\( \text{contradiction} \)

Scheduling With Release Times

Construction. Given 3-SAT instance \( \phi \) with \( n \) variables and \( k \) clauses, form \( 2n + 2k \) decimal integers, each of \( n+k \) digits, as illustrated below.

Claim. \( \phi \) is satisfiable iff there exists a subset that sums to \( W \).

Pf. No carries possible.

\[ C_1 = x_1 \lor \overline{x}_2 \lor x_3 \]

\[ C_2 = x_2 \lor \overline{x}_1 \lor x_3 \]

\[ C_3 = \overline{x}_1 \lor \overline{x}_2 \lor x_3 \]

\( \text{iff all are red} \)

\( \text{contradiction} \)

\( \text{iff all are red} \)

\( \text{contradiction} \)
Polynomial-Time Reductions

Dick Karp (1972) 1985 Turing Award

3-SAT reduces to INDEPENDENT SET

Planar 3-Colorability

PLANAR-3-COLOR: Given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color?

YES instance.

Def. A graph is planar if it can be embedded in the plane in such a way that no two edges cross.
Application: VLSI circuit design, computer graphics.

Kuratowski's Theorem. An undirected graph \( G \) is non-planar iff it contains a subgraph homeomorphic to \( K_5 \) or \( K_{3,3} \).

Planarity Testing

Planarity testing. [Hopcroft-Tarjan 1974] \( O(n) \).

Remark. Many intractable graph problems can be solved in poly-time if the graph is planar; many tractable graph problems can be solved faster if the graph is planar.
Planar 3-Colorability

Claim. 3-COLOR in \#P PLANAR-3-COLOR.

Proof sketch: Given instance of 3-COLOR, draw graph in plane, letting edges cross if necessary.
- Replace each edge crossing with the following planar gadget W.
  - in any 3-coloring of W, opposite corners have the same color
  - any assignment of colors to the corners in which opposite corners have the same color extends to a 3-coloring of W

Planar k-Colorability

PLANAR-2-COLOR. Solvable in linear time.
PLANAR-3-COLOR. NP-complete.
PLANAR-4-COLOR. Solvable in O(1) time.

Theorem. [Appel-Haken, 1976] Every planar map is 4-colorable.

- Resolved century-old open problem.
- Used 50 days of computer time to deal with many special cases.
- First major theorem to be proved using computer.

False intuition. If PLANAR-3-COLOR is hard, then so is PLANAR-4-COLOR and PLANAR-5-COLOR.

Asymmetry of NP

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

Ex 1. SAT vs. NON-SATISFIABLE.
- Can prove a CNF formula is satisfiable by giving such an assignment.
- How could we prove that a formula is not satisfiable?

Ex 2. 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. SAT is NP-complete and \#SAT = \#SAT, but how do we classify NON-SATISFIABLE?

not even known to be in NP
Fundamental question. Does NP = co-NP?

- Do yes instances have succinct certificates iff no instances do?
- Consensus opinion: no.

Theorem. If NP = co-NP, then P = NP.

Proof 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.

Good Characterizations

- Observation. P \subseteq NP \cap co-NP.
  - Proof of max-flow min-cut theorem led to stronger result that max-flow and min-cut are in P.
- Sometimes finding a good characterization seems easier than finding an efficient algorithm.

Fundamental open question. Does P = NP \cap co-NP?

- Mixed opinions.
  - Many examples where problem found to have a non-trivial good characterization, but only years later discovered to be in P.
    - linear programming [Khachiyan, 1979]
    - primality testing [Agrawal-Kayal-Saxena, 2002]

Fact. Factoring is in NP \cap co-NP, but not known to be in P.

Input. \( s = 437,677 \)
Certificate. \( t = 17, 22, 3, 36,473 \)

PRIMES is in NP \cap co-NP

Theorem. PRIMES is in NP \cap co-NP.

Proof. We already know that PRIMES is in co-NP, so it suffices to prove that PRIMES is in NP.

Pratt's Theorem. An odd integer \( s \) is prime iff there exists an integer \( 1 < t < s \) s.t.

\[
\begin{align*}
    t^s &\equiv 1 \pmod{s}, \\
    t^{(s-1)/2} &\equiv 1 \pmod{s}, \\
    t^{(s-1)/3} &\equiv 1 \pmod{s}, \\
    \vdots
\end{align*}
\]

for all prime divisors \( p \) of \( s-1 \)

Certifier.
- Check \( s-1 = 2 \times 3 \times 36,473 \).
- Check \( 17s-1 = 1 \pmod{s} \).
- Check \( 17(s-1)/2 = 437,676 \pmod{s} \).
- Check \( 17(s-1)/3 = 329,415 \pmod{s} \).
- Check \( 17(s-1)/36,473 = 305,452 \pmod{s} \).

Input: \( s = 437,677 \)
Certificate: \( t = 17, 22, 3, 36,473 \)

Primality Testing and Factoring

We established: PRIMES \subseteq COMPOSITES \subseteq FACTOR.

Natural question: Does FACTOR \subseteq PRIMES?

Consensus opinion. No.

State-of-the-art.
- PRIMES is in P, proved in 2001.
- FACTOR not believed to be in P.

RSA cryptosystem.
- Based on dichotomy between complexity of two problems.
- To use RSA, must generate large primes efficiently.
- To break RSA, suffixes to find efficient factoring algorithm.
Some Philosophical Remarks

P ≠ NP captures important philosophical phenomenon: recognizing the correctness of an answer is often easier than coming up with the answer.

P ≠ NP asks if exhaustive search can be avoided.

If P = NP, then there is an algorithm that finds mathematical proofs in time polynomial in the length of the proof.

Theorems = \{ (\psi, n) : \psi \text{ has a formal proof of length at most } n \text{ in axiomatic system } A \}

Theorems is in NP

In fact, Theorems is NP-complete.

The Riemann Hypothesis

Considered by many mathematicians to be the most important unresolved problem in pure mathematics

Conjecture about the distribution of zeros of the Riemann zeta-function

1 Million dollar prize offered by Clay Institute

3D Bin Packing is NP-Complete

There is a finite and not unimaginably large set of boxes, such that if we knew how to pack those boxes into the trunk of your car, then we'd also know a proof of the Riemann Hypothesis. Indeed, every formal proof of the Riemann Hypothesis with at most (say) a million symbols corresponds to some way of packing the boxes into your trunk, and vice versa. Furthermore, a list of the boxes and their dimensions can be feasibly written down.

Courtesy of Scott Aaronson

Why do we believe P different from NP if we can't prove it?

- The empirical argument: hardness of solving NP-complete problems in practice.
- There are "vastly easier" problems than NP-complete ones (like factoring) that we already have no idea how to solve in P.
- P=NP would mean that mathematical creativity could be automated. "God would not be so kind!" Scott Aaronson

- We will add to this list later...

Why is it so hard to prove P different from NP?

- Because P is different from NP.
- Because there are lots of clever, non-obvious polynomial time algorithms. For example, proof that 3SAT is hard will have to fail for 2-SAT. Proof that 3-coloring planar graphs is hard will have to fail for 4-coloring planar graphs. Etc Etc.

- We'll add to this list later...