8. **Intractability**

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems
8. INTRACTABILITY I

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems
Algorithm design patterns and antipatterns

Algorithm design patterns.

- Greedy.
- Divide and conquer.
- Dynamic programming.
- Duality.
- Reductions.
- Local search.
- Randomization.

Algorithm design antipatterns.

- NP-completeness. $O(n^k)$ algorithm unlikely.
- PSPACE-completeness. $O(n^k)$ certification algorithm unlikely.
- Undecidability. No algorithm possible.
Classify problems according to computational requirements

Q. Which problems will we be able to solve in practice?

A working definition. Those with polynomial-time algorithms.

Theory. Definition is broad and robust.

Practice. Poly-time algorithms scale to huge problems.
### Classify problems according to computational requirements

**Q.** Which problems will we be able to solve in practice?

**A working definition.** Those with polynomial-time algorithms.

<table>
<thead>
<tr>
<th>yes</th>
<th>probably no</th>
</tr>
</thead>
<tbody>
<tr>
<td>shortest path</td>
<td>longest path</td>
</tr>
<tr>
<td>min cut</td>
<td>max cut</td>
</tr>
<tr>
<td>2-satisfiability</td>
<td>3-satisfiability</td>
</tr>
<tr>
<td>planar 4-colorability</td>
<td>planar 3-colorability</td>
</tr>
<tr>
<td>bipartite vertex cover</td>
<td>vertex cover</td>
</tr>
<tr>
<td>matching</td>
<td>3d-matching</td>
</tr>
<tr>
<td>primality testing</td>
<td>factoring</td>
</tr>
<tr>
<td>linear programming</td>
<td>integer linear programming</td>
</tr>
</tbody>
</table>
Classify problems

Desiderata. Classify problems according to those that can be solved in polynomial time and those that cannot.

Provably requires exponential time.
  • Given a constant-size program, does it halt in at most \( k \) steps?
  • Given a board position in an \( n \)-by-\( n \) generalization of checkers, can black guarantee a win?

Frustrating news. Huge number of fundamental problems have defied classification for decades.
**Polynomial-time reductions**

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

**Reduction.** Problem $X$ polynomial-time (Cook) 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 $Y$.

---

**Diagram:**

- **Instance $I$** (of $X$) is input to the **Algorithm for $Y$**.
- The algorithm runs in polynomial time and may make a polynomial number of calls to another algorithm that solves instances of $Y$ in a single step.
- The output of the algorithm is a **Solution $S$ to $I$**.

---

**Note:**

- Polynomials are a fundamental tool in computational complexity theory, allowing for efficient algorithms.
- Reductions are a key concept in complexity theory, enabling the classification of problems into complexity classes.
Polynomial-time reductions

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

Reduction. Problem $X$ polynomial-time (Cook) 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. $X \leq_p Y$.

Note. We pay for time to write down instances sent to oracle $\Rightarrow$ instances of $Y$ must be of polynomial size.

Caveat. Don't mistake $X \leq_p Y$ with $Y \leq_p X$. 
Polynomial-time reductions

**Design algorithms.** If $X \leq_p Y$ and $Y$ can be solved in polynomial time, then $X$ can be solved in polynomial time.

**Establish intractability.** If $X \leq_p Y$ and $X$ cannot be solved in polynomial time, then $Y$ cannot be solved in polynomial time.

**Establish equivalence.** If both $X \leq_p Y$ and $Y \leq_p X$, we use notation $X \equiv_p Y$. In this case, $X$ can be solved in polynomial time iff $Y$ can be.

**Bottom line.** Reductions classify problems according to relative difficulty.
8. INTRACTABILITY I

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems
**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 $\geq 6$?  
**Ex.** Is there an independent set of size $\geq 7$?

![Graph diagram with independent set of size 6]
**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 $\leq 4$?

**Ex.** Is there a vertex cover of size $\leq 3$?

![Graph with vertex cover and independent set]
Vertex cover and independent set reduce to one another

**Theorem.** \( \text{VERTEX-COVER} \equiv_p \text{INDEPENDENT-SET}. \)

**Pf.** We show \( S \) is an independent set of size \( k \) iff \( V - S \) is a vertex cover of size \( n - k \).
Vertex cover and independent set reduce to one another

**Theorem.** \( \text{VERTEX-COVER} \equiv_p \text{INDEPENDENT-SET} \).

**Pf.** We show \( S \) is an independent set of size \( k \) iff \( V - S \) is a vertex cover of size \( n - k \).

\[ \Rightarrow \]

- Let \( S \) be any independent set of size \( k \).
- \( V - S \) is of size \( n - k \).
- Consider an arbitrary edge \((u, v)\).
- \( S \) independent \( \Rightarrow \) either \( u \not\in S \) or \( v \not\in S \) (or both)
  \( \Rightarrow \) either \( u \in V - S \) or \( v \in V - S \) (or both).
- Thus, \( V - S \) covers \((u, v)\).
Vertex cover and independent set reduce to one another

**Theorem.** \( \text{VERTEX-COVER} \equiv_p \text{INDEPENDENT-SET} \).

**Pf.** We show \( S \) is an independent set of size \( k \) iff \( V - S \) is a vertex cover of size \( n - k \).

\( \iff \)

- Let \( V - S \) be any vertex cover of size \( n - k \).
- \( S \) is of size \( k \).
- Consider two nodes \( u \in S \) and \( v \in S \).
- Observe that \( (u, v) \notin E \) since \( V - S \) is a vertex cover.
- Thus, no two nodes in \( S \) are joined by an edge \( \Rightarrow \) \( S \) independent set. \( \blacksquare \)
Set cover

**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.
- Set $U$ 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.

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

a set cover instance
Theorem. \textsc{Vertex-Cover} $\leq_p$ \textsc{Set-Cover}.

\textbf{Pf.} Given a \textsc{Vertex-Cover} instance $G = (V, E)$, we construct a \textsc{Set-Cover} instance $(U, S)$ that has a set cover of size $k$ iff $G$ has a vertex cover of size $k$.

\textbf{Construction.}

- Universe $U = E$.
- Include one set for each node $v \in V$: $S_v = \{ e \in E : e \text{ incident to } v \}$.

\begin{itemize}
  \item \textbf{Vertex cover instance (k = 2)}
  \item \textbf{Set cover instance (k = 2)}
\end{itemize}
**Lemma.** \( G = (V, E) \) contains a vertex cover of size \( k \) iff \( (U, S) \) contains a set cover of size \( k \).

**Pf.** \( \Rightarrow \) Let \( X \subseteq V \) be a vertex cover of size \( k \) in \( G \).

- Then \( Y = \{ S_v : v \in X \} \) is a set cover of size \( k \). •
**Vertex cover reduces to set cover**

**Lemma.** $G = (V, E)$ contains a vertex cover of size $k$ iff $(U, S)$ contains a set cover of size $k$.

**Pf.** $\iff$ Let $Y \subseteq S$ be a set cover of size $k$ in $(U, S)$.

- Then $X = \{ v : S \ni Y \}$ is a vertex cover of size $k$ in $G$.  

---

**Example:**

- **Vertex cover instance** $(k = 2)$

  - $U = \{ 1, 2, 3, 4, 5, 6, 7 \}$
  - $S_a = \{ 3, 7 \}$
  - $S_b = \{ 2, 4 \}$
  - $S_c = \{ 3, 4, 5, 6 \}$
  - $S_d = \{ 5 \}$
  - $S_e = \{ 1 \}$
  - $S_f = \{ 1, 2, 6, 7 \}$

- **Set cover instance** $(k = 2)$
8. Intractability I

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems
Satisfiability

Literal. A boolean variable or its negation. \( x_i \) or \( \overline{x_i} \)

Clause. A disjunction of literals. \( C_j = x_1 \lor \overline{x_2} \lor x_3 \)

Conjunctive normal form. A propositional formula \( \Phi \) that is the conjunction of clauses. \( \Phi = C_1 \land C_2 \land C_3 \land C_4 \)

SAT. Given CNF formula \( \Phi \), does it have a satisfying truth assignment? 3-SAT. SAT where each clause contains exactly 3 literals (and each literal corresponds to a different variable).

\[
\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)
\]

yes instance: \( x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false}, x_4 = \text{false} \)

Key application. Electronic design automation (EDA).
3-satisfiability reduces to independent set

**Theorem.** $3$-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 nodes for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.

$$
\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)
$$
3-satisfiability reduces to independent set

**Lemma.** $G$ contains independent set of size $k = |\Phi|$ iff $\Phi$ is satisfiable.

**Pf.** $\Rightarrow$ Let $S$ be independent set of size $k$.
- $S$ must contain exactly one node in each triangle.
- Set these literals to *true* (and remaining variables consistently).
- 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 $k$. ■

$k = 3$

$$\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)$$
Review

Basic reduction strategies.
- Simple equivalence: INDEPENDENT-SET $\equiv_p$ VERTEX-COVER.
- Special case to general case: VERTEX-COVER $\leq_p$ SET-COVER.
- Encoding with gadgets: $3$-Sat $\leq_p$ 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. $3$-Sat $\leq_p$ INDEPENDENT-SET $\leq_p$ VERTEX-COVER $\leq_p$ SET-COVER.
**Search problems**

**Decision problem.** Does there exist a vertex cover of size $\leq k$?

**Search problem.** Find a vertex cover of size $\leq k$.

**Ex.** To find a vertex cover of size $\leq k$:
- Determine if there exists a vertex cover of size $\leq k$.
- Find a vertex $v$ such that $G - \{v\}$ has a vertex cover of size $\leq k - 1$. (any vertex in any vertex cover of size $\leq k$ will have this property)
- Include $v$ in the vertex cover.
- Recursively find a vertex cover of size $\leq k - 1$ in $G - \{v\}$.

**Bottom line.** $\text{VERTEX-COVER} \equiv_P \text{FIND-VERTEX-COVER}$. 

*delete v and all incident edges*
Optimization problems

Decision problem. Does there exist a vertex cover of size $\leq k$?

Search problem. Find a vertex cover of size $\leq k$.

Optimization problem. Find a vertex cover of minimum size.

Ex. To find vertex cover of minimum size:
   • (Binary) search for size $k^*$ of min vertex cover.
   • Solve corresponding search problem.

Bottom line. $\textsc{Vertex-Cover} \equiv_p \textsc{Find-Vertex-Cover} \equiv_p \textsc{Optimal-Vertex-Cover}$. 
Section 8.5

8. Intractability I

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems
Hamilton cycle

**HAM-CYCLE.** Given an undirected graph $G = (V, E)$, does there exist a simple cycle $\Gamma$ that contains every node in $V$?
Hamilton cycle

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

```
no
```
Directed hamilton cycle reduces to hamilton cycle

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

**Theorem.** $\text{DIR-HAM-CYCLE} \leq_{P} \text{HAM-CYCLE}$. 

**Pf.** Given a digraph $G = (V, E)$, construct a graph $G'$ with $3n$ nodes.
Directed hamilton cycle reduces to hamilton cycle

**Lemma.** $G$ has a directed Hamilton cycle iff $G'$ has a Hamilton cycle.

**Pf.** $\Rightarrow$

- Suppose $G$ has a directed Hamilton cycle $\Gamma$.
- Then $G'$ has an undirected Hamilton cycle (same order).

**Pf.** $\Leftarrow$

- Suppose $G'$ has an undirected Hamilton cycle $\Gamma'$.
- $\Gamma'$ must visit nodes in $G'$ using one of following two orders:
  - ..., $B, G, R, B, G, R, B, G, R, B, ...$
- Blue nodes in $\Gamma'$ make up directed Hamilton cycle $\Gamma$ in $G$, or reverse of one. □
3-satisfiability reduces to directed hamilton cycle

**Theorem.** $3\text{-Sat} \leq_p \text{DIR-HAM-CYCLE}$.

**Pf.** Given an instance $\Phi$ of $3\text{-Sat}$, we construct an instance of $\text{DIR-HAM-CYCLE}$ that has a Hamilton cycle iff $\Phi$ is satisfiable.

**Construction.** First, create graph that has $2^n$ Hamilton cycles which correspond in a natural way to $2^n$ possible truth assignments.
3-satisfiability reduces to directed hamilton cycle

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

- Construct $G$ to have $2^n$ Hamilton cycles.
- Intuition: traverse path $i$ from left to right $\iff$ set variable $x_i = true$. 
3-satisfiability reduces to directed hamilton cycle

**Construction.** Given 3-SAT instance $\Phi$ with $n$ variables $x_i$ and $k$ clauses.
- For each clause, add a node and 6 edges.
3-satisfiability reduces to directed hamilton cycle

**Lemma.** $\Phi$ is satisfiable iff $G$ has a Hamilton cycle.

**Pf.** $\Rightarrow$

- Suppose 3-SAT instance has satisfying assignment $x^*$.  
- Then, define Hamilton cycle in $G$ as follows:
  - if $x^*_i = true$, traverse row $i$ from left to right
  - if $x^*_i = false$, 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 clause node $C_j$ into cycle (and we splice in $C_j$ exactly once)
3-satisfiability reduces to directed hamilton cycle

Lemma. Φ is satisfiable iff G has a Hamilton cycle.

Pf. ⇐

• Suppose G has a Hamilton cycle Γ.
• If Γ enters clause node C_j, it must depart on mate edge.
  - nodes immediately before and after C_j are connected by an edge e ∈ E
  - removing C_j from cycle, and replacing it with edge e yields Hamilton cycle on G – { C_j }
• Continuing in this way, we are left with a Hamilton cycle Γ’ in G – { C_1, C_2, ..., C_k }.
• Set x^*_{i} = true iff Γ’ traverses row i left to right.
• Since Γ visits each clause node C_j, at least one of the paths is traversed in "correct" direction, and each clause is satisfied.
3-satisfiability reduces to longest path

**LONGEST-PATH.** Given a directed graph $G = (V, E)$, does there exists a simple path consisting of at least $k$ edges?

**Theorem.** $3$-$\text{Sat} \leq_p \text{LONGEST-PATH}$.

**Pf 1.** Redo proof for $\text{DIR-HAM-CYCLE}$, ignoring back-edge from $t$ to $s$.

**Pf 2.** Show $\text{HAM-CYCLE} \leq_p \text{LONGEST-PATH}$.
Traveling salesperson problem

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

13,509 cities in the United States
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 $\leq D$?

[Image of a map with an optimal TSP tour marked in red.]

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 $\leq D$?
Traveling salesperson problem

**TSP.** Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$?
Hamilton cycle reduces to traveling salesperson problem

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

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

**Theorem.** $\text{HAM-CYCLE} \leq_p \text{TSP}$.

**Pf.**

- Given instance $G = (V, E)$ of $\text{HAM-CYCLE}$, create $n$ cities with distance function
  
  $$d(u, v) = \begin{cases} 
  1 & \text{if } (u, v) \in E \\
  2 & \text{if } (u, v) \notin E 
  \end{cases}$$

- TSP instance has tour of length $\leq n$ iff $G$ has a Hamilton cycle.

**Remark.** TSP instance satisfies triangle inequality: $d(u, w) \leq d(u, v) + d(v, w)$. 
Polynomial-time reductions

constraint satisfaction

3-Sat

INDENTED-SET

DIR-HAM-CYCLE

GRAPH-3-COLOR

SUBSET-SUM

VERTEX-COVER

HAM-CYCLE

PLANAR-3-COLOR

SCHEDULING

SET-COVER

TSP

packing and covering

sequencing

partitioning

numerical

3-Sat poly-time reduces to INDEPENDENT-SET
8. **Intractability I**

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- **partitioning problems**
- graph coloring
- numerical problems

Section 8.6
### 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 226</td>
<td>TTh 11–12:20</td>
</tr>
<tr>
<td>Wayne</td>
<td>COS 423</td>
<td>MW 11–12:20</td>
</tr>
<tr>
<td>Wayne</td>
<td>COS 423</td>
<td>TTh 11–12:20</td>
</tr>
<tr>
<td>Tardos</td>
<td>COS 423</td>
<td>TTh 3–4:20</td>
</tr>
<tr>
<td>Tardos</td>
<td>COS 523</td>
<td>TTh 3–4:20</td>
</tr>
<tr>
<td>Kleinberg</td>
<td>COS 226</td>
<td>TTh 3–4:20</td>
</tr>
<tr>
<td>Kleinberg</td>
<td>COS 226</td>
<td>MW 11–12:20</td>
</tr>
<tr>
<td>Kleinberg</td>
<td>COS 423</td>
<td>MW 11–12:20</td>
</tr>
</tbody>
</table>
3-dimensional matching

**3D-Matching.** Given 3 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?

\[
\begin{align*}
X &= \{ x_1, x_2, x_3 \}, & Y &= \{ y_1, y_2, y_3 \}, & Z &= \{ z_1, z_2, z_3 \} \\
T_1 &= \{ x_1, y_1, z_2 \}, & T_2 &= \{ x_1, y_2, z_1 \}, & T_3 &= \{ x_1, y_2, z_2 \} \\
T_4 &= \{ x_2, y_2, z_3 \}, & T_5 &= \{ x_2, y_3, z_3 \}, & T_7 &= \{ x_3, y_1, z_3 \}, & T_8 &= \{ x_3, y_1, z_1 \}, & T_9 &= \{ x_3, y_2, z_1 \}
\end{align*}
\]

an instance of 3d-matching (with $n = 3$)

**Remark.** Generalization of bipartite matching.
3-dimensional matching

**3D-MATCHING.** Given 3 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?

**Theorem.** $3$-**Sat** $\leq_p$ 3D-MATCHING.

**Pf.** Given an instance $\Phi$ of 3-SAT, we construct an instance of 3D-MATCHING that has a perfect matching iff $\Phi$ is satisfiable.
3-satisfiability reduces to 3-dimensional matching

**Construction.** (part 1)

- Create gadget for each variable $x_i$ with $2k$ core elements and $2k$ tip ones.

![Diagram showing a gadget for variable $x_i$ with $k = 4$](image)

- Number of clauses
- Clause 1 tips
- Clause 2 tips
- Clause 3 tips
- Core elements

*a gadget for variable $x_i$ (k = 4)*
3-satisfiability reduces to 3-dimensional matching

Construction. (part 1)

- Create gadget for each variable $x_i$ with $2k$ core elements and $2k$ tip ones.
- No other triples will use core elements.
- In gadget for $x_i$, any perfect matching must use either all gray triples (corresponding to $x_i = true$) or all blue ones (corresponding to $x_i = false$).
3-satisfiability reduces to 3-dimensional matching

**Construction.** (part 2)
- Create gadget for each clause $C_j$ with two elements and three triples.
- Exactly one of these triples will be used in any 3d-matching.
- Ensures any perfect matching uses either (i) grey core of $x_1$ or (ii) blue core of $x_2$ or (iii) grey core of $x_3$.

Clauses 1, 2, and 3:

$$C_1 = x_1 \lor \overline{x_2} \lor x_3$$
3-satisfiability reduces to 3-dimensional matching

**Construction.** (part 3)

- There are $2nk$ tips: $nk$ covered by blue/gray triples; $k$ by clause triples.
- To cover remaining $(n-1)k$ tips, create $(n-1)k$ cleanup gadgets: same as clause gadget but with $2nk$ triples, connected to every tip.

\[
C_1 = x_1 \lor \overline{x_2} \lor x_3
\]
3-satisfiability reduces to 3-dimensional matching

**Lemma.** Instance \((X, Y, Z)\) has a perfect matching iff \(\Phi\) is satisfiable.

Q. What are \(X, Y,\) and \(Z\)?
**3-satisfiability reduces to 3-dimensional matching**

**Lemma.** Instance \((X, Y, Z)\) has a perfect matching iff \(\Phi\) is satisfiable.

**Q.** What are \(X, Y,\) and \(Z\)?

**A.** \(X = \text{red}, Y = \text{green},\) and \(Z = \text{blue}\).

\[
C_1 = x_1 \lor \overline{x_2} \lor x_3
\]
3-satisfiability reduces to 3-dimensional matching

**Lemma.** Instance \((X, Y, Z)\) has a perfect matching iff \(\Phi\) is satisfiable.

**Pf.**  
⇒ If 3d-matching, then assign \(x_i\) according to gadget \(x_i\).

**Pf.**  
⇐ If \(\Phi\) is satisfiable, use any true literal in \(C_j\) to select gadget \(C_j\) triple. □
8. Intractability I

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems
3-colorability

**3-COLOR.** Given an undirected graph $G$, can the nodes be colored red, green, and blue so that no adjacent nodes have the same color?

*yes instance*
Application: 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. $3\text{-COLOR} \leq_p K\text{-REGISTER-ALLOCATION}$ for any constant $k \geq 3$. 

REGISTER ALLOCATION & SPILLING VIA GRAPH COLORING
G. J. Chaitin
IBM Research
P.O.Box 218, Yorktown Heights, NY 10598
3-satisfiability reduces to 3-colorability

**Theorem.** \( 3\text{-SAT} \leq_p 3\text{-COLOR} \).

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

Construction.

(i) Create a graph $G$ with a node for each literal.
(ii) Connect each literal to its negation.
(iii) Create 3 new nodes $T$, $F$, and $B$; connect them in a triangle.
(iv) Connect each literal to $B$.
(v) For each clause $C_j$, add a gadget of 6 nodes and 13 edges.

\[ \text{to be described later} \]
3-satisfiability reduces to 3-colorability

Lemma. Graph $G$ is 3-colorable iff $\Phi$ is satisfiable.

Pf. $\Rightarrow$ Suppose graph $G$ is 3-colorable.
   • Consider assignment that sets all $T$ literals to true.
   • (iv) ensures each literal is $T$ or $F$.
   • (ii) ensures a literal and its negation are opposites.
3-satisfiability reduces to 3-colorability

**Lemma.** Graph $G$ is 3-colorable iff $\Phi$ is satisfiable.

**Pf.** $\Rightarrow$ Suppose graph $G$ is 3-colorable.

- Consider assignment that sets all $T$ literals to true.
- (iv) ensures each literal is $T$ or $F$.
- (ii) ensures a literal and its negation are opposites.
- (v) ensures at least one literal in each clause is $T$.

$$C_j = x_1 \lor \overline{x_2} \lor x_3$$
3-satisfiability reduces to 3-colorability

**Lemma.** Graph $G$ is 3-colorable iff $\Phi$ is satisfiable.

**Pf.** $\Rightarrow$ Suppose graph $G$ is 3-colorable.

- Consider assignment that sets all $T$ literals to true.
- (iv) ensures each literal is $T$ or $F$.
- (ii) ensures a literal and its negation are opposites.
- (v) ensures at least one literal in each clause is $T$. 

$C_j = x_1 \lor \overline{x_2} \lor x_3$
3-satisfiability reduces to 3-colorability

**Lemma.** Graph $G$ is 3-colorable iff $\Phi$ is satisfiable.

**Pf.** $\Leftarrow$ Suppose 3-SAT instance $\Phi$ is satisfiable.

- Color all true literals $T$.
- Color node below green node $F$, and node below that $B$.
- Color remaining middle row nodes $B$.
- Color remaining bottom nodes $T$ or $F$ as forced. □

$a$ literal set to true in 3-SAT assignment

$$C_j = x_1 \lor \overline{x_2} \lor x_3$$
Polynomial-time reductions

constraint satisfaction

3-Sat

INDEPENDENT-SET

INDEPENDENT-SET poly-time reduces to INDEPENDENT-SET

DIR-HAM-CYCLE

VERTEX-COVER

HAM-CYCLE

TSP

SET-COVER

GRAPH-3-COLOR

PLANAR-3-COLOR

SUBSET-SUM

packing and covering

sequencing

partitioning

numerical
SECTION 8.8

8. INTRACTABILITY I

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems
Subset sum

**Subset-Sum.** Given natural numbers $w_1, \ldots, w_n$ and an integer $W$, is there a subset that adds up to exactly $W$?

**Ex.** \{ 1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344 \}, $W = 3754$.

**Yes.** $1 + 16 + 64 + 256 + 1040 + 1093 + 1284 = 3754$.

**Remark.** With arithmetic problems, input integers are encoded in binary. Poly-time reduction must be polynomial in binary encoding.
Subset sum

**Theorem.** $3$-$\text{SAT} \leq_p \text{SUBSET-SUM}$.

**Pf.** Given an instance $\Phi$ of $3$-$\text{SAT}$, we construct an instance of $\text{SUBSET-SUM}$ that has solution iff $\Phi$ is satisfiable.
3-satisfiability reduces to subset sum

Construction. Given 3-SAT instance $\Phi$ with $n$ variables and $k$ clauses, form $2n + 2k$ decimal integers, each of $n + k$ digits:

- Include one digit for each variable $x_i$ and for each clause $C_j$.
- Include two numbers for each variable $x_i$.
- Include two numbers for each clause $C_j$.
- Sum of each $x_i$ digit is 1;
  sum of each $C_j$ digit is 4.

Key property. No carries possible $\Rightarrow$ each digit yields one equation.

$C_1 = \neg x_1 \lor x_2 \lor x_3$

$C_2 = x_1 \lor \neg x_2 \lor x_3$

$C_3 = \neg x_1 \lor \neg x_2 \lor \neg x_3$

3-SAT instance

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Subset-Sum instance

<table>
<thead>
<tr>
<th>$W$</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>111,444</th>
</tr>
</thead>
</table>

$W = 111,444$
Lemma. $\Phi$ is satisfiable iff there exists a subset that sums to $W$.

Pf. $\Rightarrow$ Suppose $\Phi$ is satisfiable.

- Choose integers corresponding to each true literal.
- Since $\Phi$ is satisfiable, each $C_j$ digit sums to at least 1 from $x_i$ rows.
- Choose dummy integers to make clause digits sum to 4.

3-satisfiability reduces to subset sum

\[ C_1 = \neg x_1 \lor x_2 \lor x_3 \]
\[ C_2 = x_1 \lor \neg x_2 \lor x_3 \]
\[ C_3 = \neg x_1 \lor \neg x_2 \lor \neg x_3 \]

\[
\begin{array}{cccccc}
  & x_1 & x_2 & x_3 & C_1 & C_2 & C_3 \\
 x_1 & 1 & 0 & 0 & 0 & 1 & 0 & 100,010 \\
 \neg x_1 & 1 & 0 & 0 & 1 & 0 & 1 & 100,101 \\
 x_2 & 0 & 1 & 0 & 1 & 0 & 0 & 10,100 \\
 \neg x_2 & 0 & 1 & 0 & 0 & 1 & 1 & 10,011 \\
 x_3 & 0 & 0 & 1 & 1 & 1 & 0 & 1,110 \\
 \neg x_3 & 0 & 0 & 1 & 0 & 0 & 1 & 1,001 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
  & 0 & 0 & 0 & 1 & 0 & 0 & 100 \\
 & 0 & 0 & 0 & 2 & 0 & 0 & 200 \\
 & 0 & 0 & 0 & 0 & 1 & 0 & 10 \\
 & 0 & 0 & 0 & 2 & 0 & 0 & 20 \\
 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
 & 0 & 0 & 0 & 0 & 2 & 0 & 2 \\
 100,010 & 100,101 & 10,100 & 10,011 & 1,110 & 1,001 & \text{W} & 111,444
\end{array}
\]

\text{3-Sat instance}

dummies to get clause columns to sum to 4

\text{Subset-Sum instance}
3-satisfiability reduces to subset sum

Lemma. $\Phi$ is satisfiable iff there exists a subset that sums to $W$.

Pf. $\iff$ Suppose there is a subset that sums to $W$.

- Digit $x_i$ forces subset to select either row $x_i$ or $\neg x_i$ (but not both).
- Digit $C_j$ forces subset to select at least one literal in clause.
- Assign $x_i = true$ iff row $x_i$ selected. □

$$C_1 = \neg x_1 \lor x_2 \lor x_3$$
$$C_2 = x_1 \lor \neg x_2 \lor x_3$$
$$C_3 = \neg x_1 \lor \neg x_2 \lor \neg x_3$$

3-Sat instance

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\neg x_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\neg x_2$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$x_3$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$\neg x_3$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

|   | 0     | 0     | 0     | 1     | 0     | 0     | 100,010 |
|   | 0     | 0     | 0     | 2     | 0     | 0     | 100,101 |
|   | 0     | 0     | 0     | 0     | 1     | 0     | 10,100  |
|   | 0     | 0     | 0     | 2     | 0     | 0     | 10,011  |
|   | 0     | 0     | 0     | 0     | 0     | 1     | 1,110   |
|   | 0     | 0     | 0     | 0     | 0     | 2     | 1,001   |

W: 1 1 1 4 4 4 111,444

Subset-Sum instance
My hobby

EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS

CHOTCHKIES RESTAURANT

APPETIZERS

MIXED FRUIT  2.15
FRENCH FRIES  2.75
SIDE SALAD   3.35
HOT WINGS    3.55
MOZZARELLA STICKS  4.20
SAMPLER PLATE  5.80

SANDWICHES

BARBECUE   6.55

WE'D LIKE EXACTLY $15.05 WORTH OF APPETIZERS, PLEASE.

...EXACTLY? UHH...

HERE, THESE PAPERS ON THE KNAPSACK PROBLEM MIGHT HELP YOU OUT.

LISTEN, I HAVE SIX OTHER TABLES TO GET TO --

-- AS FAST AS POSSIBLE, OF COURSE. WANT SOMETHING ON TRAVELING SALESMAN?

Randall Munro
http://xkcd.com/c287.html
**Partition**

**Subset-Sum.** Given natural numbers \( w_1, \ldots, w_n \) and an integer \( W \), is there a subset that adds up to exactly \( W \)?

**Partition.** Given natural numbers \( v_1, \ldots, v_m \), can they be partitioned into two subsets that add up to the same value \( \frac{1}{2} \sum v_i \)?

**Theorem.** \( \text{Subset-Sum} \leq_p \text{Partition} \).

**Pf.** Let \( W, w_1, \ldots, w_n \) be an instance of \( \text{Subset-Sum} \).

- Create instance of \( \text{Partition} \) with \( m = n + 2 \) elements.
  - \( v_1 = w_1, v_2 = w_2, \ldots, v_n = w_n, \quad v_{n+1} = 2 \sum w_i - W, \quad v_{n+2} = \sum w_i + W \)
- Lemma: there exists a subset that sums to \( W \) iff there exists a partition since elements \( v_{n+1} \) and \( v_{n+2} \) cannot be in the same partition. □

| \( v_{n+1} = 2 \sum w_i - W \) | \( W \) | subset A |
| \( v_{n+2} = \sum w_i + W \) | \( \sum w_i - W \) | subset B |

72
Scheduling with release times

**Schedule.** Given a set of $n$ jobs with processing time $t_j$, release time $r_j$, and deadline $d_j$, is it possible to schedule all jobs on a single machine such that job $j$ is processed with a contiguous slot of $t_j$ time units in the interval $[r_j, d_j]$?

Ex.
Scheduling with release times

**Theorem.** \( \text{SUBSET-SUM} \leq_p \text{SCHEDULE}. \)

**Pf.** Given \( \text{SUBSET-SUM} \) instance \( w_1, \ldots, w_n \) and target \( W \), construct an instance of \( \text{SCHEDULE} \) that is feasible iff there exists a subset that sums to exactly \( W \).

**Construction.**

- Create \( n \) jobs with processing time \( t_j = w_j \), release time \( r_j = 0 \), and no deadline \( (d_j = 1 + \sum_j w_j) \).
- Create job 0 with \( t_0 = 1 \), release time \( r_0 = W \), and deadline \( d_0 = W + 1 \).
- Lemma: subset that sums to \( W \) iff there exists a feasible schedule. □
Polynomial-time reductions

- **3-SAT**
  - **INDEPENDENT-SET**
    - **VERTEX-COVER**
      - **SET-COVER**
  - **DIR-HAM-CYCLE**
    - **HAM-CYCLE**
      - **TSP**
  - **GRAPH-3-COLOR**
    - **PLANAR-3-COLOR**
  - **SUBSET-SUM**
    - **SCHEDULING**

**constraint satisfaction**

3-SAT poly-time reduces to INDEPENDENT-SET

packing and covering  sequencing  partitioning  numerical
Karp's 21 NP-complete problems

Dick Karp (1972)
1985 Turing Award

FIGURE 1 - Complete Problems
8. **Intractability II**

- $P$ vs. $NP$
- $NP$-complete
- $co-NP$
- $NP$-hard
Recap

3-Sat poly-time reduces to all of these problems (and many, many more)
8. INTRACTABILITY II

- $P$ vs. $NP$
- $NP$-complete
- $co-NP$
- $NP$-hard

Section 8.3
Decision problems

Decision problem.
• Problem $X$ is a set of strings.
• Instance $s$ is one string.
• Algorithm $A$ solves problem $X$: $A(s) = yes$ iff $s \in X$.

Def. Algorithm $A$ runs in polynomial time if for every string $s$, $A(s)$ terminates in at most $p(|s|)$ "steps", where $p(\cdot)$ is some polynomial.

Ex.
• Problem PRIMES = \{ 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, …. \}.
• Instance $s = 592335744548702854681$.
• AKS algorithm PRIMES in $O(|s|^8)$ steps.
## Definition of P

**P.** Decision 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><strong>MULTIPLE</strong></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><strong>REL-PRIME</strong></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><strong>PRIMES</strong></td>
<td>Is $x$ prime?</td>
<td>AKS (2002)</td>
<td>53</td>
<td>51</td>
</tr>
<tr>
<td><strong>EDIT-DISTANCE</strong></td>
<td>Is the edit distance between $x$ and $y$ less than 5?</td>
<td>dynamic programming</td>
<td>neither</td>
<td>acgggt ttttta</td>
</tr>
<tr>
<td><strong>L-SOLVE</strong></td>
<td>Is there a vector $x$ that satisfies $Ax = b$?</td>
<td>Gauss-Edmonds elimination</td>
<td>[0 1 1] , [4 2]</td>
<td>[1 0 0] , [1]</td>
</tr>
<tr>
<td><strong>ST-CONN</strong></td>
<td>Is there a path between $s$ and $t$ in a graph $G$?</td>
<td>depth-first search (Theseus)</td>
<td>[2 4 -2] , [0 3 15]</td>
<td>[0 1 1] , [1]</td>
</tr>
</tbody>
</table>
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 \( s \), \( s \in X \) iff there exists a string \( t \) such that \( C(s, t) = yes \).

"certificate" or "witness"

**Def.** **NP** is the set of problems for which there exists a poly-time certifier.

- \( C(s, t) \) is a poly-time algorithm.
- Certificate \( t \) is of polynomial size: \( |t| \leq p(|s|) \) for some polynomial \( p(\cdot) \)

**Remark.** **NP** stands for nondeterministic polynomial time.
Certifiers and certificates: composite

**COMPOSITES.** Given an integer $s$, is $s$ composite?

**Certificate.** A nontrivial factor $t$ of $s$. Such a certificate exists iff $s$ is composite. Moreover $|t| \leq |s|$.

**Certifier.** Check that $1 < t < s$ and that $s$ is a multiple of $t$.

| instance $s$ | 437669 |
| certificate $t$ | 541 or 809 |

437,669 = 541 × 809

**Conclusion.** COMPOSITES $\in$ NP.
Certifiers and certificates: 3-satisfiability

3-Sat. Given a CNF 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.

\[
\text{instance } s \quad \Phi = \left( \overline{x}_1 \lor x_2 \lor x_3 \right) \land \left( x_1 \lor \overline{x}_2 \lor x_3 \right) \land \left( \overline{x}_1 \lor x_2 \lor x_4 \right)
\]

\[
\text{certificate } t \quad x_1 = \text{true}, \ x_2 = \text{true}, \ x_3 = \text{false}, \ x_4 = \text{false}
\]

Conclusion. $\text{3-Sat} \in \text{NP}$. 
Certifiers and certificates: Hamilton path

**HAM-PATH.** Given an undirected graph $G = (V, E)$, does there exist a simple path $P$ 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.

**Conclusion.** $\text{HAM-PATH} \in \text{NP}$. 
**Definition of NP**

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
<th>Algorithm</th>
<th>yes</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-SOLVE</td>
<td>Is there a vector $x$ that satisfies $Ax = b$?</td>
<td>Gauss-Edmonds elimination</td>
<td>[0 1 1 1]</td>
<td>[1 0 0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[2 4 -2]</td>
<td>[1 1 1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[0 3 15]</td>
<td>[0 1 1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[2]</td>
<td>[1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[36]</td>
<td>[1]</td>
</tr>
<tr>
<td>COMPOSITES</td>
<td>Is $x$ composite?</td>
<td>AKS (2002)</td>
<td>51</td>
<td>53</td>
</tr>
<tr>
<td>FACTOR</td>
<td>Does $x$ have a nontrivial factor less than $y$?</td>
<td></td>
<td>(56159, 50)</td>
<td>(55687, 50)</td>
</tr>
<tr>
<td>SAT</td>
<td>Is there a truth assignment that satisfies the formula?</td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>3-COLOR</td>
<td>Can the nodes of a graph $G$ be colored with 3 colors?</td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>HAM-PATH</td>
<td>Is there a simple path between $s$ and $t$ that visits every node?</td>
<td></td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>
**Definition of NP**

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

“*NP captures vast domains of computational, scientific, and mathematical endeavors, and seems to roughly delimit what mathematicians and scientists have been aspiring to compute feasibly.*” — Christos Papadimitriou

“In an ideal world it would be renamed P vs VP.” — Clyde Kruskal
P, NP, and EXP

P. Decision problems for which there is a poly-time algorithm.
NP. Decision problems for which there is a poly-time certifier.
EXP. Decision problems for which there is an exponential-time algorithm.

Claim. P ⊆ NP.
Pf. Consider any problem \( X \in P \).
   • By definition, there exists a poly-time algorithm \( A(s) \) that solves \( X \).
   • Certificate \( t = \varepsilon \), certifier \( C(s, t) = A(s) \).  □

Claim. NP ⊆ EXP.
Pf. Consider any problem \( X \in NP \).
   • By definition, there exists a poly-time certifier \( C(s, t) \) 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 potential certificates.  □

Remark. Time-hierarchy theorem implies P ⊊ EXP.
The main question: P vs. NP

Q. How to solve an instance of 3-SAT with $n$ variables?
A. Exhaustive search: try all $2^n$ truth assignments.

Q. Can we do anything substantially more clever?
Conjecture. No poly-time algorithm for 3-SAT.

"intractable"
The main question: P vs. NP

Does P = NP? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]
Is the decision problem as easy as the certification problem?

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

Consensus opinion. Probably no.
Possible outcomes

\[ P \neq \text{NP}. \]

“I conjecture that there is no good algorithm for the traveling salesman problem. My reasons are the same as for any mathematical conjecture: (i) It is a legitimate mathematical possibility and (ii) I do not know.”

— Jack Edmonds 1966
Possible outcomes

\( P \neq NP. \)

“In my view, there is no way to even make intelligent guesses about the answer to any of these questions. If I had to bet now, I would bet that \( P \) is not equal to \( NP \). I estimate the half-life of this problem at 25–50 more years, but I wouldn’t bet on it being solved before 2100.”

— Bob Tarjan

“We seem to be missing even the most basic understanding of the nature of its difficulty…. All approaches tried so far probably (in some cases, provably) have failed. In this sense \( P = NP \) is different from many other major mathematical problems on which a gradual progress was being constantly done (sometimes for centuries) whereupon they yielded, either completely or partially.”

— Alexander Razborov
Possible outcomes

$P = NP$.

“$P = NP$. In my opinion this shouldn’t really be a hard problem; it’s just that we came late to this theory, and haven’t yet developed any techniques for proving computations to be hard. Eventually, it will just be a footnote in the books.” — John Conway
Other possible outcomes

\[ \text{P} = \text{NP}, \text{ but only } \Omega(n^{100}) \text{ algorithm for 3-SAT.} \]

\[ \text{P} \neq \text{NP}, \text{ but with } O(n^{\log^*n}) \text{ algorithm for 3-SAT.} \]

\[ \text{P} = \text{NP} \text{ is independent (of ZFC axiomatic set theory).} \]

“\text{It will be solved by either 2048 or 4096. I am currently somewhat pessimistic. The outcome will be the truly worst case scenario: namely that someone will prove \text{“P} = \text{NP because there are only finitely many obstructions to the opposite hypothesis”}; hence there will exists a polynomial time solution to SAT but we will never know its complexity! }” \quad \text{— Donald Knuth}
Millennium prize. $1 million for resolution of $P = NP$ problem.
Looking for a job?

**Some writers for the Simpsons and Futurama.**

- J. Steward Burns. *M.S. in mathematics (Berkeley '93).*
- David X. Cohen. *M.S. in computer science (Berkeley '92).*
- Al Jean. *B.S. in mathematics. (Harvard '81).*
- Ken Keeler. *Ph.D. in applied mathematics (Harvard '90).*
- Jeff Westbrook. *Ph.D. in computer science (Princeton '89).*
8. INTRACTABILITY II

- P vs. NP
- NP-complete
- co-NP
- NP-hard

Section 8.4
Polynomial transformation

**Def.** Problem $X$ polynomial (Cook) 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 $Y$.

**Def.** Problem $X$ polynomial (Karp) transforms to problem $Y$ if given any input $x$ to $X$, we can construct an input $y$ such that $x$ is a yes instance of $X$ iff $y$ is a yes instance of $Y$.

we require $|y|$ to be of size polynomial in $|x|$

**Note.** Polynomial transformation is polynomial reduction with just one call to oracle for $Y$, exactly at the end of the algorithm for $X$. Almost all previous reductions were of this form.

**Open question.** Are these two concepts the same with respect to $\mathbf{NP}$?

we abuse notation $\leq_p$ and blur distinction
NP-complete

**NP-complete.** A problem $Y \in \textbf{NP}$ with the property that for every problem $X \in \textbf{NP}$, $X \leq_p Y$.

**Theorem.** Suppose $Y \in \textbf{NP}$-complete. Then $Y \in \textbf{P}$ iff $\textbf{P} = \textbf{NP}$.

**Pf. $\Leftarrow$** If $\textbf{P} = \textbf{NP}$, then $Y \in \textbf{P}$ because $Y \in \textbf{NP}$.

**Pf. $\Rightarrow$** Suppose $Y \in \textbf{P}$.

- Consider any problem $X \in \textbf{NP}$. Since $X \leq_p Y$, we have $X \in \textbf{P}$.
- This implies $\textbf{NP} \subseteq \textbf{P}$.
- We already know $\textbf{P} \subseteq \textbf{NP}$. Thus $\textbf{P} = \textbf{NP}$.  

---

**Fundamental question.** Do there exist "natural" $\textbf{NP}$-complete problems?
Circuit satisfiability

**Circuit-Sat.** Given a combinational circuit built from AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?

![Circuit Diagram]

- **Output**
- **Hard-coded inputs:** 1, 0
- **Variable inputs:** ?

Yes: 1 0 1
The "first" NP-complete problem

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

The Complexity of Theorem-Proving Procedures
Stephen A. Cook
University of Toronto

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined. It is shown that the problem of determining tautologyhood has the same polynomial degree as the problem of determining whether the first of two given graphs is isomorphic to a subgraph of the second. Other examples are discussed. A method of measuring the complexity of proof procedures for the predicate calculus is introduced and discussed.

Throughout this paper, a set of strings means a set of strings on some fixed, large, finite alphabet I. This alphabet is large enough to include symbols for all sets described here. All Turing machines are deterministic recognition devices, unless the contrary is explicitly stated.

1. Tautologies and Polynomial Reducibility

Let us fix a formalism for the propositional calculus in which formulas are written as strings on I. Since we will require infinitely many propositional symbols (atoms), each such symbol will consist of a member of I followed by a number in binary notation to distinguish that symbol. Thus a formula of length n can only have about n/log n distinct function and predicate symbols. The logical connectives are & (and), ∨ (or), and ~ (not).

The set of tautologies (denoted by \{tautologies\}) is a certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem I will give evidence that \{tautologies\} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean, roughly speaking, that if tautologyhood could be decided instantly (by an "oracle") then these problems could be decided in polynomial time.

In order to use this result, we introduce query machines, which are like Turing machines with oracles in them.

A query machine is a multitape Turing machine with a distinguished tape called the query tape, and three distinguished states called the query state, yes state, and no state, respectively. An M is a query machine and T is a set of strings on which computation of M is a computation of M in which initially M is in the initial state and T is an input string w on its input tape, and each line M assumes the query state there is a string u on the query tape, and the next state M assumes in the yes state if wU and the no state if wU. We think of an "oracle", which knows T, placing M in the yes state or no state.

Definition

A set S of strings is P-reducible (P for polynomial) to a set T of strings if there is a polynomial-time query machine M and a polynomial Q(n) such that for each input string w, the computation of M with input w halts within Q(|w|) steps ([w] is the length of w), and ends in an accepting state iff wS. It is not hard to see that P-reducibility is a transitive relation. Thus the relation ∈ on

The Proofs of Theorem-Proving Procedures

Stephen A. Cook
University of Toronto
The "first" NP-complete problem

**Theorem.** \( \text{CIRCUIT-SAT} \in \text{NP-complete}. \)

**Pf sketch.**

- Clearly, \( \text{CIRCUIT-SAT} \in \text{NP}. \)
- Any algorithm that takes a fixed number of bits \( n \) as input and produces a *yes* or *no* answer can be represented by such a circuit.
- Moreover, if algorithm takes poly-time, then circuit is of poly-size.

\[
\begin{align*}
\text{Consider any problem } X \in \text{NP}. \quad & \text{It has a poly-time certifier } C(s, t): \\
& s \in X \text{ iff there exists a certificate } t \text{ of length } p(|s|) \text{ such that } C(s, t) = \text{yes}. \\
\text{View } C(s, t) \text{ as an algorithm with } |s| + p(|s|) \text{ input bits and convert it into a poly-size circuit } K.
\end{align*}
\]

- *first* \(|s|\) bits are hard-coded with \( s \)
- *remaining* \( p(|s|) \) bits represent (unknown) bits of \( t \)
- Circuit \( K \) is satisfiable iff \( C(s, t) = \text{yes} \).
Example

Ex. Construction below creates a circuit $K$ whose inputs can be set so that it outputs 1 iff graph $G$ has an independent set of size 2.

$G = (V, E), n = 3$

\[ \binom{n}{2} \text{ hard-coded inputs} \]
\[ \binom{n}{2} \text{ (graph description)} \]
\[ n \text{ inputs} \]
\[ (\text{nodes in independent set}) \]
Establishing NP-completeness

Remark. Once we establish first "natural" $\text{NP}$-complete problem, others fall like dominoes.

Recipe. To prove that $Y \in \text{NP}$-complete:

- Step 1. Show that $Y \in \text{NP}$.
- Step 2. Choose an $\text{NP}$-complete problem $X$.
- Step 3. Prove that $X \leq_p Y$.

Theorem. If $X \in \text{NP}$-complete, $Y \in \text{NP}$, and $X \leq_p Y$, then $Y \in \text{NP}$-complete.

Pf. Consider any problem $W \in \text{NP}$. Then, both $W \leq_p X$ and $X \leq_p Y$.

- By transitivity, $W \leq_p Y$.
- Hence $Y \in \text{NP}$-complete.

by definition of NP-complete
by assumption
3-satisfiability is NP-complete

**Theorem.** $3\text{-SAT} \in \text{NP}$-complete.

**Pf.**

- Suffices to show that $\text{CIRCUIT-SAT} \leq_p 3\text{-SAT}$ since $3\text{-SAT} \in \text{NP}$.
- Given a combinational circuit $K$, we construct an instance $\Phi$ of $3\text{-SAT}$ that is satisfiable iff the inputs of $K$ can be set so that it outputs 1.
3-satisfiability is NP-complete

**Construction.** Let \( K \) be any circuit.

**Step 1.** Create a 3-SAT variable \( x_i \) for each circuit element \( i \).

**Step 2.** Make circuit compute correct values at each node:

- \( x_2 = \overline{x_3} \) \( \Rightarrow \) add 2 clauses: \( x_2 \lor x_3, \overline{x_2} \lor \overline{x_3} \)
- \( x_1 = x_4 \lor x_5 \) \( \Rightarrow \) add 3 clauses: \( x_1 \lor \overline{x_4}, x_1 \lor \overline{x_5}, \overline{x_1} \lor x_4 \lor x_5 \)
- \( x_0 = x_1 \land x_2 \) \( \Rightarrow \) add 3 clauses: \( \overline{x_0} \lor x_1, \overline{x_0} \lor x_2, x_0 \lor \overline{x_1} \lor \overline{x_2} \)

**Step 3.** Hard-coded input values and output value.

- \( x_5 = 0 \) \( \Rightarrow \) add 1 clause: \( \overline{x_5} \)
- \( x_0 = 1 \) \( \Rightarrow \) add 1 clause: \( x_0 \)
3-satisfiability is NP-complete

Construction. [continued]

Step 4. Turn clauses of length 1 or 2 into clauses of length 3.

- Create four new variables $z_1$, $z_2$, $z_3$, and $z_4$.
- Add 8 clauses to force $z_1 = z_2 = false$:
  - $(\overline{z_1} \lor z_3 \lor z_4)$,
  - $(\overline{z_1} \lor z_3 \lor \overline{z_4})$,
  - $(\overline{z_1} \lor \overline{z_3} \lor z_4)$,
  - $(\overline{z_1} \lor \overline{z_3} \lor \overline{z_4})$
  - $(\overline{z_2} \lor z_3 \lor z_4)$,
  - $(\overline{z_2} \lor z_3 \lor \overline{z_4})$,
  - $(\overline{z_2} \lor \overline{z_3} \lor z_4)$,
  - $(\overline{z_2} \lor \overline{z_3} \lor \overline{z_4})$

- Replace any clause with a single term ($t_i$) with ($t_i \lor z_1 \lor z_2$).
- Replace any clause with two terms ($t_i \lor t_j$) with ($t_i \lor t_j \lor z_1$).
Lemma. $\Phi$ is satisfiable iff the inputs of $K$ can be set so that it outputs 1.

Pf. $\iff$ Suppose there are inputs of $K$ that make it output 1.
   - Can propagate input values to create values at all nodes of $K$.
   - This set of values satisfies $\Phi$.

Pf. $\implies$ Suppose $\Phi$ is satisfiable.
   - We claim that the set of values corresponding to the circuit inputs constitutes a way to make circuit $K$ output 1.
   - The 3-SAT clauses were designed to ensure that the values assigned to all nodes in $K$ exactly match what the circuit would compute for these nodes. $\blacksquare$
Implications of Karp

3-SAT poly-time reduces to INDEPENDENT-SET

CIRCUIT-SAT poly-time reduces to all of these problems (and many, many more)
Implications of Cook-Levin

3-SAT poly-time reduces to INDEPENDENT-SET

INDEPENDENT-SET

DIR-HAM-CYCLE

GRAPH-3-COLOR

SUBSET-SUM

VERTEX-COVER

HAM-CYCLE

PLANAR-3-COLOR

SCHEDULING

SET-COVER

TSP

All of these problems (and many, many more) poly-time reduce to CIRCUIT-SAT.
Implications of Karp + Cook-Levin

All of these problems are NP-complete; they are manifestations of the same really hard problem.
Some NP-complete problems

Basic genres of NP-complete problems and paradigmatic examples.

- Packing + covering problems: SET-COVER, VERTEX-COVER, INDEPENDENT-SET.
- Constraint satisfaction problems: CIRCUIT-SAT, SAT, 3-SAT.
- Sequencing problems: HAM-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, PARTITION.

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

Notable exceptions. FACTOR, GRAPH-ISOMORPHISM, NASH-EQUILIBRIUM.

Theory. [Ladner 1975] Unless P = NP, there exist problems in NP that are neither in P nor NP-complete.
More hard computational problems

**Garey and Johnson.** Computers and Intractability.

- Appendix includes over 300 NP-complete problems.
- Most cited reference in computer science literature.

---

**Most Cited Computer Science Citations**

This list is generated from documents in the CiteSeerx database as of January 17, 2013. This list is automatically generated and may contain errors. The list is generated in batch mode and citation counts may differ from those currently in the CiteSeerx database, since the database is continuously updated.


1. M R Garey, D S Johnson

2. T Cormen, C E Leiserson, R Rivest
   Introduction to Algorithms 1990
   7210

3. V Vapnik
   The nature of statistical learning theory 1998
   6580

4. A P Dempster, N M Laird, D B Rubin
   Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society, 1977
   6082

5. T Cover, J Thomas
   Elements of Information Theory 1991
   6075

6. D E Goldberg
   5998

7. J Pearl
   Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference 1988
   5582

8. E Gamma, R Helm, R Johnson, J Vlissides
   Design Patterns: Elements of Reusable Object-Oriented Software 1995
   4614

9. C E Shannon
   A mathematical theory of communication Bell Syst. Tech. J, 1948
   4118

10. J R Quinlan
    C4.5: Programs for Machine Learning 1993
   4018
More hard computational problems

- **Aerospace engineering.** Optimal mesh partitioning for finite elements.
- **Biology.** Phylogeny reconstruction.
- **Chemical engineering.** Heat exchanger network synthesis.
- **Chemistry.** Protein folding.
- **Civil engineering.** Equilibrium of urban traffic flow.
- **Economics.** Computation of arbitrage in financial markets with friction.
- **Electrical engineering.** VLSI layout.
- **Environmental engineering.** Optimal placement of contaminant sensors.
- **Financial engineering.** Minimum risk portfolio of given return.
- **Game theory.** Nash equilibrium that maximizes social welfare.
- **Mathematics.** Given integer $a_1, \ldots, a_n$, compute
  \[ \int_0^{2\pi} \cos(a_1 \theta) \times \cos(a_2 \theta) \times \cdots \times \cos(a_n \theta) \, d\theta \]
- **Mechanical engineering.** Structure of turbulence in sheared flows.
- **Medicine.** Reconstructing 3d shape from biplane angiocardiogram.
- **Operations research.** Traveling salesperson problem.
- **Physics.** Partition function of 3d Ising model.
- **Politics.** Shapley-Shubik voting power.
- **Recreation.** Versions of Sudoku, Checkers, Minesweeper, Tetris.
- **Statistics.** Optimal experimental design.
Extent and impact of NP-completeness

**Extent of NP-completeness.** [Papadimitriou 1995]
- Prime intellectual export of CS to other disciplines.
- 6,000 citations per year (more than "compiler", "OS", "database").
- Broad applicability and classification power.

**NP-completeness can guide scientific inquiry.**
- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager finds closed-form solution to 2D-ISING in tour de force.
- 19xx: Feynman and other top minds seek solution to 3D-ISING.
- 2000: Istrail proves 3D-ISING $\in$ NP-complete.

search for closed formula appears doomed
P vs. NP revisited

Overwhelming consensus (still). \( P \neq NP \).

Why we believe \( P \neq NP \).

“We admire Wiles' proof of Fermat's last theorem, the scientific theories of Newton, Einstein, Darwin, Watson and Crick, the design of the Golden Gate bridge and the Pyramids, precisely because they seem to require a leap which cannot be made by everyone, let alone a by simple mechanical device.” — Avi Wigderson
You NP-complete me
8. **Intractability II**

- P vs. NP
- NP-complete
- co-NP
- NP-hard

**Section 8.9**
Asymmetry of NP

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

**Ex 1. SAT vs. Tautology.**
- Can prove a CNF formula is satisfiable by specifying 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 specifying a permutation.
- How could we prove that a graph is not Hamiltonian?

**Q.** How to classify Tautology and NO-Hamilton-Cycle?
- **SAT** ∈ **NP**-complete and **SAT** ≡ _p_ Tautology.
- **HAM-CYCLE** ∈ **NP**-complete and **HAM-CYCLE** ≡ _p_ NO-Ham-Cycle.
- But neither Tautology nor NO-Ham-Cycle are known to be in NP.
NP and co-NP

NP. Decision problems for which there is a poly-time certifier.
Ex. SAT, HAMILTON-CYCLE, and COMPOSITE.

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

Ex. \( X = \{ 0, 1, 4, 6, 8, 9, 10, 12, 14, 15, \ldots \} \)
\[ \overline{X} = \{ 2, 3, 5, 7, 11, 13, 17, 23, 29, \ldots \} \]

co-NP. Complements of decision problems in \( \textbf{NP} \).
Ex. TAUTOLOGY, NO-HAMILTON-CYCLE, and PRIMES.
NP = co-NP?

Fundamental open 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.

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.
Good characterizations

**Good characterization.** [Edmonds 1965] $\text{NP} \cap \text{co-NP}$.

- If problem $X$ is in both $\text{NP}$ and $\text{co-NP}$, then:
  - for *yes* instance, there is a succinct certificate
  - for *no* instance, there is a succinct disqualifier
- Provides conceptual leverage for reasoning about a problem.

**Ex.** Given a bipartite graph, is there a perfect matching.
- If yes, can exhibit a perfect matching.
- If no, can exhibit a set of nodes $S$ such that $|\text{N}(S)| < |S|$.
We seek a good characterization of the minimum number of independent sets into which the columns of a matrix of $M_F$ can be partitioned. As the criterion of “good” for the characterization we apply the “principle of the absolute supervisor.” The good characterization will describe certain information about the matrix which the supervisor can require his assistant to search out along with a minimum partition and which the supervisor can then use with ease to verify with mathematical certainty that the partition is indeed minimum. Having a good characterization does not mean necessarily that there is a good algorithm. The assistant might have to kill himself with work to find the information and the partition.
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 nontrivial good characterization, but only years later discovered to be in $P$. 
Linear programming is in \( \text{NP} \cap \text{co-NP} \)

**Linear programming.** Given \( A \in \mathbb{R}^{m \times n}, \ b \in \mathbb{R}^m, \ c \in \mathbb{R}^n, \) and \( \alpha \in \mathbb{R}, \) does there exist \( x \in \mathbb{R}^n \) such that \( Ax \leq b, \ x \geq 0 \) and \( c^T x \geq \alpha? \)

**Theorem.** [Gale-Kuhn-Tucker 1948] \( \text{LINEAR-PROGRAMMING} \in \text{NP} \cap \text{co-NP}. \)

**Pf sketch.** If (P) and (D) are nonempty, then \( \max = \min. \)

\[
\begin{align*}
\text{(P)} \quad \max & \ c^T x \\
\text{s. t.} & \ Ax \leq b \\
& \ x \geq 0
\end{align*}
\]

\[
\begin{align*}
\text{(D)} \quad \min & \ y^T b \\
\text{s. t.} & \ A^T y \geq c \\
& \ y \geq 0
\end{align*}
\]

---

**Chapter XIX**

LINEAR PROGRAMMING AND THE THEORY OF GAMES

BY DAVID GALE, HAROLD W. KUHN, AND ALBERT W. TUCKER

The basic “scalar” problem of linear programming is to maximize (or minimize) a linear function of several variables constrained by a system of linear inequalities [Dantzig, II]. A more general “vector” problem calls for maximizing (in a sense of partial order) a system of linear functions of several variables subject to a system of linear inequalities and, perhaps, linear equations [Koopmans, III]. The purpose of this chapter is to establish theorems of duality and existence for general “matrix” problems of linear programming which contain the “scalar” and “vector” problems as special cases, and to relate these general problems to the theory of zero-sum two-person games.
Linear programming is in $\text{NP} \cap \text{co-NP}$

**Linear programming.** Given $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and $\alpha \in \mathbb{R}$, does there exist $x \in \mathbb{R}^n$ such that $Ax \leq b$, $x \geq 0$ and $c^T x \geq \alpha$?

**Theorem.** [Khachiyan 1979] $\text{LINEAR-PROGRAMMING} \in \text{P}$. 

---

**ЖУРНАЛ**

**ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И МАТЕМАТИЧЕСКОЙ ФИЗИКИ**

Том 20 × Январь 1980 × Февраль № 1

УДК 519.852

ПОЛИНОМИАЛЬНЫЕ АЛГОРИТМЫ В ЛИНЕЙНОМ ПРОГРАММИРОВАНИИ

А. Г. ХАЧИЯН

(Москва)

Построены точные алгоритмы линейного программирования, трудоемкость которых ограничена полиномом от длины двоичной записи задачи.
Primality testing is in $\text{NP} \cap \text{co-NP}$

**Theorem.** [Pratt 1975] $\text{PRIMES} \in \text{NP} \cap \text{co-NP}$.

---

**EVERY PRIME HAS A SUCCINCT CERTIFICATE**

VAUGHAN R. PRATT

**Abstract.** To prove that a number $n$ is composite, it suffices to exhibit the working for the multiplication of a pair of factors. This working, represented as a string, is of length bounded by a polynomial in $\log_2 n$. We show that the same property holds for the primes. It is noteworthy that almost no other set is known to have the property that short proofs for membership or nonmembership exist for all candidates without being known to have the property that such proofs are easy to come by. It remains an open problem whether a prime $n$ can be recognized in only $\log_2 n$ operations of a Turing machine for any fixed $x$.

The proof system used for certifying primes is as follows.

**Axiom.** $(x, y, 1)$.

**Inference Rules.**

$R_1 : (p, x, a), q \vdash (p, x, qa)$ provided $x^{(n-1)/q} \not\equiv 1 \pmod{p}$ and $q|p - 1$.

$R_2 : (p, x, p - 1) \vdash p$ provided $x^{p-1} \equiv 1 \pmod{p}$.

**Theorem 1.** $p$ is a theorem $\iff p$ is a prime.

**Theorem 2.** $p$ is a theorem $\Rightarrow p$ has a proof of $[4 \log_2 p]$ lines.
Theorem. [Pratt 1975] PRIMES $\in$ NP ∩ co-NP.

Pf sketch. An odd integer $s$ is prime iff there exists an integer $1 < t < s$ s.t.

$$t^{s-1} \equiv 1 \pmod{s}$$
$$t^{(s-1)/p} \not\equiv 1 \pmod{s}$$

for all prime divisors $p$ of $s-1$

**Certifier** ($s$)

- **Check** $s - 1 = 2 \times 2 \times 3 \times 36473$.
- **Check** $17^{s-1} = 1 \pmod{s}$.
- **Check** $17^{(s-1)/2} \equiv 437676 \pmod{s}$.
- **Check** $17^{(s-1)/3} \equiv 329415 \pmod{s}$.
- **Check** $17^{(s-1)/36,473} \equiv 305452 \pmod{s}$.

Use repeated squaring

**Instance** $s = 437677$

**Certificate** $t = 17, 2^2 \times 3 \times 36473$

Prime factorization of $s-1$ also need a recursive certificate to assert that 3 and 36,473 are prime.
Theorem. [Agrawal-Kayal-Saxena 2004] \( \text{PRIMES} \in \text{P} \).


PRIMES is in P

By Manindra Agrawal, Neeraj Kayal, and Nitin Saxena*

Abstract

We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.
Factoring is in $NP \cap co-NP$

**FACTORIZER.** Given an integer $x$, **find** its prime factorization.

**FACTOR.** Given two integers $x$ and $y$, does $x$ have a nontrivial factor $< y$?

**Theorem.** $\text{FACTOR} \equiv_p \text{FACTORIZER}$.

**Pf.**

- $\leq_p$ trivial.
- $\geq_p$ binary search to find a factor; divide out the factor and repeat. ▪

**Theorem.** $\text{FACTOR} \in NP \cap co-NP$.

**Pf.**

- Certificate: a factor $p$ of $x$ that is less than $y$.
- Disqualifier: the prime factorization of $x$ (where each prime factor is less than $y$), along with a Pratt certificate that each factor is prime. ▪
Is factoring in \( P \)?

**Fundamental question.** Is \( \text{FACTOR} \in P \).

**Challenge.** Factor this number.

74037563479561712828046796097429573142593188889231289
08493623263897276503402826627689199641962511784399589
43305021275853701189680982867331732731089309005525051
16877063299072396380786710086096962537934650563796359

**RSA-704**

($30,000 prize if you can factor)
Exploiting intractability

Modern cryptography.

- Ex. Send your credit card to Amazon.
- Ex. Digitally sign an e-document.
- Enables freedom of privacy, speech, press, political association.

RSA. Based on dichotomy between complexity of two problems.

- To use: generate two random $n$-bit primes and multiply.
- To break: suffices to factor a $2n$-bit integer.

\[
\begin{align*}
P & \not\equiv Q \pmod{M} \\
N &= PQ \\
ED &= 1 \pmod{(P-1)(Q-1)} \\
C &= M^k \pmod{N} \\
M &= C^d \pmod{N}
\end{align*}
\]

The RSA algorithm is the most widely used method of implementing public key cryptography and has been deployed in more than one billion applications worldwide.

RSA sold for $2.1$ billion or design a t-shirt
Factoring on a quantum computer

Theorem. [Shor 1994] Can factor an $n$-bit integer in $O(n^3)$ steps on a "quantum computer."

2001. Factored $15 = 3 \times 5$ (with high probability) on a quantum computer.
2012. Factored $21 = 3 \times 7$.

Fundamental question. Does $P = BQP$?
8. **Intractability II**

- $P$ vs. $NP$
- $NP$-complete
- $co-NP$
- $NP$-hard
A note on terminology

A TERMINOLOGICAL PROPOSAL

D. F. Knuth

While preparing a book on combinatorial algorithms, I felt a strong need for a new technical term, a word which is essentially a one-sided version of polynomial complete. A great many problems of practical interest have the property that they are at least as difficult to solve in polynomial time as those of the Cook-Karp class NP. I needed an adjective to convey such a degree of difficulty, both formally and informally; and since the range of practical applications is so broad, I felt it would be best to establish such a term as soon as possible.

The goal is to find an adjective $x$ that sounds good in sentences like this:

The covering problem is $x$.
It is $x$ to decide whether a given graph has a Hamiltonian circuit.
It is unknown whether or not primality testing is an $x$ problem.

Note. The term $x$ does not necessarily imply that a problem is in $\textbf{NP}$, just that every problem in $\textbf{NP}$ poly-time reduces to $x$. 
A note on terminology

Knuth's original suggestions.

- Hard.
- Tough.
- Herculean.
- Formidable.
- Arduous.

So common that it is unclear whether it is being used in a technical sense.

Assign a real number between 0 and 1 to each choice.
A note on terminology

Some English word write-ins.

- Impractical.
- Bad.
- Heavy.
- Tricky.
- Intricate.
- Prodigious.
- Difficult.
- Intractable.
- Costly.
- Obdurate.
- Obstinate.
- Exorbitant.
- Interminable.
A note on terminology

**Hard-boiled.** [Ken Steiglitz] In honor of Cook.


**Sisyphean.** [Bob Floyd] Problem of Sisyphus was time-consuming.

**Ulyssean.** [Don Knuth] Ulysses was known for his persistence.

“creative research workers are as full of ideas for new terminology as they are empty of enthusiasm for adopting it.”

— Donald Knuth
A note on terminology: acronyms

**PET.** [Shen Lin] Probably exponential time.
- If $P \neq NP$, provably exponential time.
- If $P = NP$, previously exponential time.

**GNP.** [Al Meyer] Greater than or equal to $NP$ in difficulty.
- And costing more than the GNP to solve.
A note on terminology: made-up words

**Exparent.** [Mike Paterson] Exponential + apparent.

**Perarduous.** [Mike Paterson] Through (in space or time) + completely.

**Supersat.** [Al Meyer] Greater than or equal to satisfiability.

**Polychronious.** [Ed Reingold] Enduringly long; chronic.
A note on terminology: consensus

NP-complete. A problem in NP such that every problem in NP poly-time reduces to it.

NP-hard. [Bell Labs, Steve Cook, Ron Rivest, Sartaj Sahni]
A problem such that every problem in NP polynomial-time reduces to it.

One final criticism (which applies to all the terms suggested) was stated nicely by Vaughan Pratt: "If the Martians know that P = NP for Turing Machines and they kidnap me, I would lose face calling these problems 'formidable'." Yes; if P = NP, there's no need for any term at all. But I'm willing to risk such an embarrassment, and in fact I'm willing to give a prize of one live turkey to the first person who proves that P = NP.