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.

- von Neumann (1953)
- Nash (1955)
- Gödel (1956)
- Cobham (1964)
- Edmonds (1965)
- Rabin (1966)

**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 problem $Y$ 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 $Y$.

---

Polynomial-time reductions

**Desiderata’.** Suppose we could solve problem $Y$ 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 \( X \not\leq_P Y \) and \( Y \) 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 \( \geq 7 \)?

![Independent set](image)

**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 \( \leq 3 \)?

![Vertex cover](image)
**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 \( n - k \).

\[ \begin{array}{c}
\text{independent set of size 6} \\
\text{vertex cover of size 4}
\end{array} \]

\[ \text{let } S = \{3, 7\} \quad S_b = \{2, 4\} \\
S_c = \{3, 4, 5, 6\} \quad S_d = \{5\} \\
S_e = \{1\} \quad S_f = \{1, 2, 6, 7\} \\
k = 2
\]

**Set cover**

**SET-COVER.** Given a set \( U \) of elements, a collection \( S \) of subsets of \( U \), and an integer \( k \), are there \( \leq k \) of these subsets whose union is equal to \( U \)?

**Sample application.**

\( m \) available pieces of software.

\( U \) of \( n \) capabilities that we would like our system to have.

The \( i \)th piece of software provides the set \( S_i \subseteq U \) of capabilities.

Goal: achieve all \( n \) capabilities using fewest pieces of software.

\[ U = \{1, 2, 3, 4, 5, 6, 7\} \\
S_a = \{3, 7\} \\
S_b = \{2, 4\} \\
S_c = \{3, 4, 5, 6\} \\
S_d = \{5\} \\
S_e = \{1\} \\
S_f = \{1, 2, 6, 7\} \\
k = 2
\]

**Set cover instance**
Vertex cover reduces to set cover

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

**Pf.** Given a \( \text{VERTEX-COVER} \) instance \( G = (V, E) \) and \( k \), we construct a 
\( \text{SET-COVER} \) instance \( (U, S, k) \) that has a set cover of size \( k \) iff \( G \) has a vertex cover of size \( k \).

**Construction.**
- Universe \( U = E \).
- Include one subset for each node \( v \in V \) : \( S_v = \{ e \in E : e \text{ incident to } v \} \).

---

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

---

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 (CNF). A propositional formula \( \Phi \) that is a conjunction of clauses.

\( \Phi = C_1 \land C_2 \land C_3 \land C_4 \)

SAT. Given a 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).

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

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

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

Pf. \( \Rightarrow \) Let \( S \) be independent set of size \( k \).
\( \bullet \) \( S \) must contain exactly one node in each triangle.
\( \bullet \) Set these literals to \text{true} (and remaining variables consistently).
\( \bullet \) Truth assignment is consistent and all clauses are satisfied.

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

Review

Basic reduction strategies.
\( \bullet \) Simple equivalence: INDEPENDENT-SET \( \not\leq_P \) VERTEX-COVER.
\( \bullet \) Special case to general case: VERTEX-COVER \( \leq_P \) SET-COVER.
\( \bullet \) 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. $\text{VERTEX-COVER} = \text{p} \text{ FIND-VERTEX-COVER}$.

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. $\text{VERTEX-COVER} = \text{p} \text{ FIND-VERTEX-COVER} = \text{p} \text{ OPTIMAL-VERTEX-COVER}$.

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 $V$?

Theorem. **3-SAT** $\leq_P$ **HAM-CYCLE.**

**Pf.** Given a graph $G = (V, E)$, 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:
  - ..., black, white, blue, black, white, blue, black, white, blue, ...
  - ..., black, blue, white, black, blue, white, black, blue, white, ...
- Black nodes in $\Gamma'$ make up directed Hamilton cycle $\Gamma$ in $G$, or reverse of one.

3-satisfiability reduces to directed Hamilton cycle

**Theorem.** **3-SAT** $\leq_P$ **DIR-HAM-CYCLE.**

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

Construction overview. Let $n$ denote the number of variables in $\Phi$. We will 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 $x_i = true$.

**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)

**Pf.** $\Leftarrow$
- Suppose $G$ has a Hamilton cycle $\Gamma$.
- If $\Gamma$ enters clause node $C_j$, it must depart on mate edge.
  - nodes immediately before and after $C_j$ are connected by an edge $e \in E$
  - 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 $\Gamma'$ in $G - \{C_1, C_2, \ldots, C_k\}$.
- Set $x^* = true$ 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.
3-satisfiability reduces to longest path

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

**Theorem.** $3$-SAT $\leq_p$ LONGEST-PATH.

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

**Pf 2.** Show **HAM-CYCLE** $\leq_p$ **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 $\leq D$?

- Optimal TSP tour
  - 13,509 cities in the United States
  - http://www.math.uwaterloo.ca/tsp

- 11,849 holes to drill in a programmed logic array
  - http://www.math.uwaterloo.ca/tsp
Traveling salesperson problem

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

http://www.math.uwaterloo.ca/tsp

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 an instance \( G = (V, E) \) of \( \text{HAM-CYCLE} \), create \( n = |V| \) 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

**8. INTRACTABILITY I**

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

**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?

\[
X = \{ x_1, x_2, x_3 \}, \quad Y = \{ y_1, y_2, y_1 \}, \quad Z = \{ z_1, z_2, z_3 \}
\]

\[
T_1 = \{ x_1, y_1, z_2 \}, \quad T_2 = \{ x_1, y_2, z_1 \}, \quad T_3 = \{ x_1, y_2, z_2 \}
\]

\[
T_4 = \{ x_2, y_2, z_3 \}, \quad T_5 = \{ x_2, y_2, z_3 \}, \quad T_6 = \{ x_3, y_1, z_3 \}, \quad T_7 = \{ x_3, y_1, z_1 \}, \quad T_8 = \{ x_3, y_2, z_1 \}
\]

An instance of 3d-matching (with \( n = 3 \))

### 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.

**Number of clauses**

- \( \text{clause 1 tips} \)
- \( \text{clause 2 tips} \)
- \( \text{clause 3 tips} \)

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 \( 2^k \) core elements and \( 2^k \) 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 \( x_i = \text{true} \)) or all blue ones (corresponding to \( x_i = \text{false} \)).

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 \( x_3 \).

3-satisfiability reduces to 3-dimensional matching

Construction. (part 3)
- There are \( 2n^k \) tips: \( n^k \) 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 \( 2n^k \) triples, connected to every tip.

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. \(X = \text{black}, Y = \text{white}, \text{and } Z = \text{blue}\).

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 black, white, and blue so that no adjacent nodes have the same color?
Application: register allocation

Register allocation. Assign program variables to machine registers 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; 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 $k \geq 3$.

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 clause $C_j$ add a gadget of 6 nodes and 13 edges.
(v) For each clause $C_j$, add a gadget of 6 nodes and 13 edges.

3-satisfiability reduces to 3-colorability

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

Pf. $\Rightarrow$ Suppose graph $G$ is 3-colorable.
- WLOG, assume that node $T$ is colored black, $F$ is white, and $B$ is blue.
- Consider assignment that sets all black literals to true (and white to false).
- (iv) ensures each literal is colored either black or white.
- (ii) ensures that each literal is white if its negation is black (and vice versa).

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

3-satisfiability reduces to 3-colorability
3-satisfiability reduces to 3-colorability

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

**Pf.** $\Rightarrow$ Suppose graph $G$ is 3-colorable.
- WLOG, assume that node $T$ is colored black, $F$ is white, and $B$ is blue.
- Consider assignment that sets all black literals to true (and white to false).
- (iv) ensures each literal is colored either black or white.
- (ii) ensures that each literal is white if its negation is black (and vice versa).
- (v) ensures at least one literal in each clause is black.

simulate 6-node gadget

$C_f = x_1 \lor \overline{x}_2 \lor x_3$

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 black and all false literals white.
- Pick one true literal; color node below that node white, and node below that blue.
- Color remaining middle row nodes blue.
- Color remaining bottom nodes black or white, as forced.

simulate 6-node gadget

$C_f = x_1 \lor \overline{x}_2 \lor x_3$

Polynomial-time reductions

simulate 6-node gadget

constraint satisfaction

$3$-SAT poly-time reduces to $\text{3-SAT}$

$\text{3-SAT}$ poly-time reduces to $\text{1-SAT}$

$\text{INDEPENDENT-SET}$ $\text{DIR-HAM-CYCLE}$ $\text{GRAPH-3-COLOR}$ $\text{SUBSET-SUM}$

$\text{VERTEX-COVER}$ $\text{HAMP-CYCLE}$ $\text{PLANAR-3-COLOR}$ $\text{SCHEDULING}$

packing and covering sequencing partitioning numerical constraint satisfaction
8. Intractability

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

Subset sum

**Theorem.** 3-SAT ≤ₚ SUBSET-SUM.

**Pf.** Given an instance \( \Phi \) of 3-SAT, we construct an instance of SUBSET-SUM that has a solution if and only if \( \Phi \) 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 \( \implies \) 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>\neg x_1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<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>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>\neg x_3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>
<tr>
<td>( W )</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>11,444</td>
</tr>
</tbody>
</table>

Subset-SUM instance
3-satisfiability reduces to subset sum

**Lemma.** \( \Phi \) is satisfiable iff there exists a subset that sums to \( W \).

**Pf.** \( \Rightarrow \) Suppose \( \Phi \) is satisfiable.

- Choose integers corresponding to each \( \text{true} \) 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.

\[
\begin{align*}
C_1 &= \neg x_1 \lor x_2 \lor x_3 \\
C_2 &= x_1 \lor \neg x_2 \lor x_3 \\
C_3 &= \neg x_1 \lor \neg x_2 \lor \neg x_3
\end{align*}
\]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( W )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1 00,010</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1 00,101</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0 10,100</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1 10,011</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1 1,110</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0 1,001</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0 111,444</td>
</tr>
</tbody>
</table>

\( W = 111,444 \) subset sum instance

**Pf.** \( \Leftarrow \) 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 \( x_i = \text{true} \) iff row \( x_i \) selected. \( \blacksquare \)

### 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 \( v_1, \ldots, v_n \), 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.
  - \( v_1 = w_1, v_2 = w_2, \ldots, v_n = w_n, v_{n+1} = 2 \sum w_i - W, v_{n+2} = \sum w_i + W \)
- 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. \( \blacksquare \)
Scheduling with release times

**Theorem.** \textsc{Subset-Sum} $\leq_p \text{Schedule}$.  

**Pf.** Given \textsc{Subset-Sum} instance $w_1, \ldots, w_n$ and target $W$, construct an instance of \textsc{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 ($d_j = 1 + \sum w_j$).
- 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. 

- must schedule jobs 1 to n either here or here
- or here
- $1 + \sum w_j$
- must schedule job 0 here

---

**Polynomial-time reductions**

- 3-Sat poly time reduces to \textsc{Independent-Set}
- \textsc{Independent-Set} poly time reduces to \textsc{Directed-Ham-Cycle}
- 3-Sat poly time reduces to \textsc{Graph-3-Color}
- 3-Sat poly time reduces to \textsc{Sub-Set}
- \textsc{Sub-Set} poly time reduces to \textsc{Scheduling}

**Karp’s 21 NP-complete problems**

- Dick Karp (1972)
- 1985 Turing Award