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>

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 \( X \leq_p Y \) and \( Y \) cannot be solved in polynomial time, then \( X \) cannot be solved in polynomial time.

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

**Bottom line.** Reductions classify problems according to relative difficulty.

---

**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 \( \geq 7 \) ?

---

**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 \( \leq 3 \) ?
Theorem. \( \textsc{Vertex-Cover} \equiv_p \textsc{Independent-Set} \).

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 \( (u, v) \).

\[ \Leftarrow \]
- \( V - S \) 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 \( (u, v) \not\in E \) since \( V - S \) is a vertex cover.
- Thus, no two nodes in \( S \) are joined by an edge \( \Rightarrow S \) independent set.

Set cover

\textsc{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.
- 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_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
\end{align*}

\[ \text{a set cover instance} \]
Vertex cover reduces to set cover

Theorem. \textsc{Vertex-Cover} \(\leq_p \textsc{Set-Cover} \).

Pf. Given a \textsc{Vertex-Cover} instance \( G = (V, E) \) and \( k \), 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 \( k \).

Construction.

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

\[
\begin{array}{c}
\text{vertex cover instance} \\
(k = 2)
\end{array}
\quad
\begin{array}{c}
\text{set cover instance} \\
(k = 2)
\end{array}
\]

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

8. Intractability

- 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).

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

*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

**Theorem.** $3\text{-SAT} \leq_p \text{INDEPENDENT-SET}$.

**Pf.** Given an instance $\Phi$ of 3-SAT, we construct an instance $(G, k)$ of INDEPENDENT-SET that has an independent set of size $k = |\Phi|$ 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.

$k = 3$

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

---

**Review**

**Basic reduction strategies.**

- Simple equivalence: $\text{INDEPENDENT-SET} \not\leq_p \text{VERTEX-COVER}$.
- Special case to general case: $\text{VERTEX-COVER} \leq_p \text{SET-COVER}$.
- Encoding with gadgets: $3\text{-SAT} \leq_p \text{INDEPENDENT-SET}$.

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

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

**Ex.** $3\text{-SAT} \leq_p \text{INDEPENDENT-SET} \leq_p \text{VERTEX-COVER} \leq_p \text{SET-COVER}$. 
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 \( G - \{v\} \).

(delete \( v \) and all incident edges)

**Bottom line.** \( \textsc{Vertex-Cover} =_{p} \textsc{Find-Vertex-Cover} \).

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.** \( \textsc{Vertex-Cover} =_{p} \textsc{Find-Vertex-Cover} =_{p} \textsc{Optimal-Vertex-Cover} \).

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 \( V \)?

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:
  
  \[ \ldots, \text{black}, \text{white}, \text{blue}, \text{black}, \text{white}, \text{blue}, \text{black}, \text{white}, \text{blue}, \ldots \]
  \[ \ldots, \text{black}, \text{blue}, \text{white}, \text{black}, \text{blue}, \text{white}, \text{black}, \text{blue}, \text{white}, \ldots \]
- Black nodes in \( \Gamma' \) make up directed Hamilton cycle \( \Gamma \) in \( G \), or reverse of one.

Directed Hamilton cycle reduces to Hamilton cycle

**Theorem.** \( \text{DIR-HAM-CYCLE} \leq_p \text{HAM-CYCLE} \).

**Pf.** Given a digraph \( G = (V, E) \), construct a graph \( G' \) with 3n nodes.

3-satisfiability reduces to directed Hamilton cycle

**Theorem.** \( 3\text{-SAT} \leq_p \text{DIR-HAM-CYCLE} \).

**Pf.** Given an instance \( \Phi \) of 3-SAT, we construct an instance of \( \text{DIR-HAM-CYCLE} \) 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 2n Hamilton cycles which correspond in a natural way to 2n 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 $G - \{C_1, C_2, ..., C_k\}$.
- Set $x_i^* = true$ 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 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.

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

**Pf.** $\Rightarrow$
- 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 $G - \{C_1, C_2, ..., C_k\}$.
- Set $x_i^* = true$ 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.

$\blacksquare$
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.** $3$-$SAT \leq_p$ $\text{LONGEST-PATH}$.

**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 salesman 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.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 \( \leq D \)?

---

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

- **3-SAT** poly-time reduces to **INDEPENDENT-SET**
- **3-SAT** poly-time reduces to **DIR-HAM-CYCLE**
- **3-SAT** poly-time reduces to **GRAPH-3-COLOR**
- **3-SAT** poly-time reduces to **SUBSET-SUM**
- **3-SAT** poly-time reduces to **SET-COVER**
- **3-SAT** poly-time reduces to **TSP**
- **INDEPENDENT-SET** poly-time reduces to **DIR-HAM-CYCLE**
- **INDEPENDENT-SET** poly-time reduces to **GRAPH-3-COLOR**
- **INDEPENDENT-SET** poly-time reduces to **SUBSET-SUM**
- **INDEPENDENT-SET** poly-time reduces to **SET-COVER**
- **INDEPENDENT-SET** poly-time reduces to **TSP**
- **INDEPENDENT-SET** poly-time reduces to **HAM-CYCLE**
- **INDEPENDENT-SET** poly-time reduces to **PLANAR-3-COLOR**
- **INDEPENDENT-SET** poly-time reduces to **SCHEDULING**

---

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 226</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>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?

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.

### 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.

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

\[
X = \{x_1, x_2, x_3\}, \quad Y = \{y_1, y_2, y_3\}, \quad Z = \{z_1, z_2, z_3\}
\]

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

\[
T_4 = \{x_2, y_2, z_3\}, \quad T_5 = \{x_2, y_3, z_2\}, \quad T_6 = \{x_2, y_3, z_3\}
\]

\[
T_7 = \{x_3, y_1, z_3\}, \quad T_8 = \{x_3, y_1, z_1\}, \quad T_9 = \{x_3, y_2, z_1\}
\]

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 $x_i = \text{true}$) 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 $2nk$ tips: $nk$ covered by blue/grey 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.

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

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

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

**Pf.** \(\Leftarrow\) 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 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.** \( 3 \text{-COLOR} \leq_p K \text{-REGISTER-ALLOCATION} \) for any constant \( k \geq 3 \).

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

**Construction.**
1. Create a graph \( G \) with a node for each literal.
2. Connect each literal to its negation.
3. Create 3 new nodes \( T, F, \) and \( B \); connect them in a triangle.
4. Connect each clause \( C_i \), add a gadget of 6 nodes and 13 edges.
5. For each clause \( C_i \), add a gadget of 6 nodes and 13 edges.

\[ 	ext{to be described later} \]

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

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

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

- WLOG, assume that node \( T \) is colored \( \text{black} \), \( F \) is \( \text{white} \), and \( B \) is \( \text{blue} \).
- Consider assignment that sets all \( \text{black} \) literals to \( \text{true} \) (and \( \text{white} \) to \( \text{false} \)).
- (iv) ensures each literal is colored either \( \text{black} \) or \( \text{white} \).
- (ii) ensures that each literal is \( \text{white} \) if its negation is \( \text{black} \) (and vice versa).
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.

\[
C_j = x_1 \lor \overline{x_2} \lor x_3
\]

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 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.

\[
C_j = x_1 \lor \overline{x_2} \lor x_3
\]

Polynomial-time reductions
8. **INTRACTABILITY I**

- 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 Φ of 3-SAT, we construct an instance of SUBSET-SUM that has solution if and only if Φ is satisfiable.

---

**3-satisfiability reduces to subset sum**

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

- Include one digit for each variable xᵢ and for each clause Cⱼ.
- Include two numbers for each variable xᵢ.
- Include two numbers for each clause Cⱼ.
- Sum of each xᵢ digit is 1;
- sum of each Cⱼ digit is 4.

**Key property.** No carries possible ⇒ each digit yields one equation.

---

**Example.** Given the 3-SAT instance:

\[
\begin{align*}
\neg x_1 & \lor \neg x_2 & \lor x_3 \\
\neg x_1 & \lor x_2 & \lor \neg x_3 \\
\neg x_1 & \lor \neg x_2 & \lor \neg x_3
\end{align*}
\]

We can encode this as a subset sum problem by assigning each variable and clause a unique number:

<table>
<thead>
<tr>
<th>x₁</th>
<th>x₂</th>
<th>x₃</th>
<th>C₁</th>
<th>C₂</th>
<th>C₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</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>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>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The subset sum is:

\[
\{ 1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344 \}
\]

The sum is 3754.

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

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

My hobby

**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.** SUBSET-SUM \( \leq_P \) PARTITION.

**Pf.** Let \( W, w_1, \ldots, w_n \) be an instance of SUBSET-SUM.

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

\[
\begin{aligned}
v_{n+1} &= 2 \sum w_j - W \\
v_{n+2} &= \sum w_j + W \\
\end{aligned}
\]

Randall Munro
http://xkcd.com/287
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 \( d_j = 1 + \sum w_j \).
- 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. □

### Ex.

<table>
<thead>
<tr>
<th>( j )</th>
<th>( t_j )</th>
<th>( r_j )</th>
<th>( d_j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>10</td>
<td>19</td>
</tr>
</tbody>
</table>

**Polynomial-time reductions**

Constraint satisfaction

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

packing and covering

sequencing

partitioning

numerical

### Karp’s 21 NP-complete problems

Dick Karp (1972)

1985 Turing Award

**Dick Karp**

Â© 1972 Dick Karp

Figure 1: Complete Problems