Algorithm Design and Implementation
Principle of Algorithms XII

NP Problem I

Guoqiang Li
May 25, 2020
School of Software, Shanghai Jiao Tong University
Poly-Time Reductions
Algorithm design patterns and antipatterns

Algorithm design patterns.
Algorithm design patterns.

- Greedy.
- Divide and conquer.
- Dynamic programming.
- Duality.
- Reductions.
- Local search.
- Randomization.
Algorithm design patterns and antipatterns

Algorithm design patterns.

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

Algorithm design antipatterns.

- NP-completeness.
Algorithm design patterns.

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

Algorithm design antipatterns.

- NP-completeness. $O(n^k)$ algorithm unlikely.
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
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.
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
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.
Q. Which problems will we be able to solve in practice?
Q. Which problems will we be able to solve in practice?

A working definition. Those with poly-time algorithms.
Q. Which problems will we be able to solve in practice?

A working definition. Those with poly-time algorithms.

Theory. Definition is broad and robust.
  - Turing machine, word RAM, uniform circuits, . . .
Q. Which problems will we be able to solve in practice?

A working definition. Those with poly-time algorithms.

Theory. Definition is broad and robust.
- Turing machine, word RAM, uniform circuits, . . .

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 poly-time algorithms.

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

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

**Requirement.** 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?
Classify problems

**Requirement.** 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.
Suppose we could solve problem $Y$ in polynomial time. What else could we solve in polynomial time?
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 $Y$. 

![Diagram of poly-time reductions](image)
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 of $Y$ sent to oracle $\Rightarrow$ instances of $Y$ must be of polynomial size.
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 of $Y$ sent to oracle $\Rightarrow$ instances of $Y$ must be of polynomial size.

**Novice mistake.** Confusing $X \leq_P Y$ with $Y \leq_P X$. 
Suppose that $X \leq_P Y$. Which of the following can we infer?

A. If $X$ can be solved in polynomial time, then so can $Y$.
B. $X$ can be solved in poly time iff $Y$ can be solved in poly time.
C. If $X$ cannot be solved in polynomial time, then neither can $Y$.
D. If $Y$ cannot be solved in polynomial time, then neither can $X$. 
Which of the following poly-time reductions are known?

A. \( \text{FIND-MAX-FLOW} \leq_P \text{FIND-MIN-CUT} \).
B. \( \text{FIND-MIN-CUT} \leq_P \text{FIND-MAX-FLOW} \).
C. Both A and B.
D. Neither A nor B.
Poly-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.
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.
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.
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.
Packing and Covering Problems
Independent Set. Given a graph $G = (V, E)$ and an integer $k$, is there a subset of $k$ (or more) vertices such that no two are adjacent?
**Independent Set.** Given a graph $G = (V, E)$ and an integer $k$, is there a subset of $k$ (or more) vertices such that no two are adjacent?

**Example.** Is there an independent set of size $\geq 6$?

**Example.** Is there an independent set of size $\geq 7$?
Vertex Cover. Given a graph $G = (V, E)$ and an integer $k$, is there a subset of $k$ (or fewer) vertices such that each edge is incident to at least one vertex in the subset?
Vertex Cover. Given a graph $G = (V, E)$ and an integer $k$, is there a subset of $k$ (or fewer) vertices such that each edge is incident to at least one vertex in the subset?

Example. Is there a vertex cover of size $\leq 4$?
Example. Is there a vertex cover of size $\leq 3$?
Consider the following graph $G$. Which are true?

A. The white vertices are a vertex cover of size 7.
B. The black vertices are an independent set of size 3.
C. Both A and B.
D. Neither A nor B.
Vertex cover and independent set reduce to one another

**Theorem**

\[ \text{Independent Set} \equiv_P \text{Vertex Cover}. \]
Theorem

Independent Set $\equiv_P$ Vertex Cover.

Proof. We show $S$ is an independent set of size $k$ iff $V - S$ is a vertex cover of size $n - k$. 
Theorem

\[ \text{Independent Set} \equiv_P \text{Vertex Cover}. \]

Proof. 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**

*Independent Set* $\equiv_p$ *Vertex Cover*.

**Proof.** 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 $(u, v) \in E$. 


Vertex cover and independent set reduce to one another

**Theorem**

Independent Set $\equiv_P$ Vertex Cover.

**Proof.** 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 $(u, v) \in E$.
- $S$ independent $\Rightarrow$ either $u \notin S$, or $v \notin S$, or both.
  $\Rightarrow$ either $u \in V - S$, or $v \in V - S$, or both.
Vertex cover and independent set reduce to one another

\textbf{Theorem}

\[ \text{Independent Set} \equiv_P \text{Vertex Cover}. \]

\textbf{Proof.} 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 \((u, v) \in E\).
- \(S\) independent \(\Rightarrow\) either \(u \notin S\), or \(v \notin S\), 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**

$\text{Independent Set} \equiv_p \text{Vertex Cover}$

**Proof.** We show $S$ is an independent set of size $k$ iff $V - S$ is a vertex cover of size $n - k$. 

$\Rightarrow$

- Let $V - S$ be any independent set of size $n - k$.
- $S$ is of size $k$.
- Consider an arbitrary edge $(u, v) \in E$.
- $V - S$ is a vertex cover $\Rightarrow$ either $u \in V - S$, or $v \in V - S$, or both.
- $\Rightarrow$ either $u \notin S$, or $v \notin S$, or both.
- Thus, $S$ is an independent set.

$\Leftarrow$
Vertex cover and independent set reduce to one another

**Theorem**

\[ \text{Independent Set} \equiv_p \text{Vertex Cover} \]

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

\( \Leftarrow \)

- Let \( V - S \) be any independent set of size \( n - k \).
- \( S \) is of size \( k \).
- Consider an arbitrary edge \( (u, v) \in E \).
Vertex cover and independent set reduce to one another

**Theorem**

\[ \text{Independent Set} \equiv_P \text{Vertex Cover}. \]

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

\[ \leftarrow \]

- Let \( V - S \) be any independent set of size \( n - k \).
- \( S \) is of size \( k \).
- Consider an arbitrary edge \( (u, v) \in E \).

\[ V - S \text{ is a vertex cover} \Rightarrow \text{either } u \in V - S, \text{ or } v \in V - S, \text{ or both.} \]

\[ \Rightarrow \text{either } u \not\in S, \text{ or } v \not\in S, \text{ or both.} \]
Theorem

Independent Set \equiv_P \text{Vertex Cover.}

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

\( \Leftarrow \)

- Let \( V - S \) be any independent set of size \( n - k \).
- \( S \) is of size \( k \).
- Consider an arbitrary edge \((u, v) \in E\).

\( V - S \) is a vertex cover \( \Rightarrow \) either \( u \in V - S \), or \( v \in V - S \), or both.

\( \Rightarrow \) either \( u \notin S \), or \( v \notin S \), or both.

- Thus, \( S \) is an independent set.
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 $U$?
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.
- 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.

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

a set cover instance
Given the universe $U = \{1, 2, 3, 4, 5, 6, 7\}$ and the following sets, which is the minimum size of a set cover?

A. 1
B. 2
C. 3
D. None of the above.

$U = \{1, 2, 3, 4, 5, 6, 7\}$
$S_a = \{1, 4, 6\}$  $S_b = \{1, 6, 7\}$
$S_c = \{1, 2, 3, 6\}$  $S_d = \{1, 3, 5, 7\}$
$S_e = \{2, 6, 7\}$  $S_f = \{3, 4, 5\}$
Vertex cover reduces to set cover

**Theorem**

\[ \text{Vertex Cover} \leq_P \text{Set Cover} \]
Theorem

Vertex Cover \( \leq_P \) Set Cover.

Proof.
Vertex cover reduces to set cover

Theorem

Vertex Cover $\leq_P$ Set Cover.

Proof. Given a Vertex Cover instance $G = (V, E)$ and $k$, we construct a Set Cover instance $(U, S, k)$ that has a set cover of size $k$ iff $G$ has a vertex cover of size $k$. 

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

**Example:**

- **Vertex cover instance** ($k = 2$)
  - $U = \{1, 2, 3, 4, 5, 6, 7\}$
  - $S_a = \{3, 7\}$
  - $S_b = \{2, 4\}$
  - $S_c = \{3, 4, 5, 6\}$
  - $S_d = \{5\}$
  - $S_e = \{1\}$
  - $S_f = \{1, 2, 6, 7\}$
Vertex cover reduces to set cover

Theorem

Vertex Cover $\leq_P$ Set Cover.

Proof. Given a Vertex Cover instance $G = (V, E)$ and $k$, we construct a 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 $v \in V$: $S_v = \{e \in E : e$ incident to $v \}$. 

vertex cover instance $(k = 2)$

set cover instance $(k = 2)$

$U = \{1, 2, 3, 4, 5, 6, 7\}$
$S_a = \{3, 7\}$
$S_c = \{3, 4, 5, 6\}$
$S_e = \{1\}$
$S_b = \{2, 4\}$
$S_d = \{5\}$
$S_f = \{1, 2, 6, 7\}$
Vertex cover reduces to set cover

Lemma

\[ G = (V, E) \] contains a vertex cover of size \( k \) iff \( (U, S, k) \) contains a set cover of size \( k \).
Vertex cover reduces to set cover

**Lemma**

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

**Proof.** ⇒

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 instance

$(k = 2)$

set cover instance

$(k = 2)$

$U = \{1, 2, 3, 4, 5, 6, 7\}$

$S_a = \{3, 7\}$

$S_b = \{2, 4\}$

$S_c = \{3, 4, 5, 6\}$

$S_d = \{5\}$

$S_e = \{1\}$

$S_f = \{1, 2, 6, 7\}$
Lemma

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

Proof. $\Leftarrow$

Let $Y \subseteq S$ be a set cover of size $k$ in $(U, S, k)$.
- Then $X = \{v : S_v \in Y\}$ is a vertex cover of size $k$ in $G$.

vertex cover instance
$(k = 2)$

set cover instance
$(k = 2)$
Constraint Satisfaction Problems
Literal. A Boolean variable or its negation: \( x_i \) or \( \bar{x}_i \).
Satisfiability

Literal. A Boolean variable or its negation: $x_i$ or $\overline{x}_i$.

Clause. A disjunction of literals: $C_j = x_1 \lor \overline{x}_2 \lor x_3$
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): \( \Phi = C_1 \land C_2 \land C_3 \land C_4 \)
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):** \( \Phi = C_1 \land C_2 \land C_3 \land C_4 \)

**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).

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

**yes instance:** \( x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false}, x_4 = \text{false} \)
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): $\Phi = C_1 \land C_2 \land C_3 \land C_4$

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).
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):** $\Phi = C_1 \land C_2 \land C_3 \land C_4$

**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).

$$\Phi = (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4)$$

**yes instance:** $x_1 = \text{true}$, $x_2 = \text{true}$, $x_3 = \text{false}$, $x_4 = \text{false}$
**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):** $\Phi = C_1 \land C_2 \land C_3 \land C_4$

**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).

$$\Phi = (\overline{x}_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4)$$

**yes instance:** $x_1 = \text{true}, \ x_2 = \text{true}, \ x_3 = \text{false} \ x_4 = \text{false}$

**Key application.** Electronic design automation (EDA).
Satisfiability is hard

Scientific hypothesis. There does not exists a poly-time algorithm for 3-SAT.
Scientific hypothesis. There does not exist a poly-time algorithm for 3-SAT.

P vs. NP This hypothesis is equivalent to $P \neq NP$ conjecture.
Scientific hypothesis. There does not exist a poly-time algorithm for 3-SAT.

P vs. NP This hypothesis is equivalent to $P \neq NP$ conjecture.

Computer Scientists have so much funding and time and can't even figure out the boolean satisfiability problem. SAT!
Theorem

3-SAT $\leq_P$ Independent Set.
3-satisfiability reduces to independent set

**Theorem**

$3\text{-SAT} \leq_P \text{Independent Set.}$

*Proof.*
3-satisfiability reduces to independent set

**Theorem**

\[ 3\text{-SAT} \leq_P \text{Independent Set.} \]

**Proof.**

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.

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

Theorem

$3$-SAT $\leq_P$ Independent Set.

Proof.
3-satisfiability reduces to independent set

**Theorem**

3-SAT $\leq_p$ Independent Set.

**Proof.**

$\implies$ Consider any satisfying assignment for $\Phi$.

- Select one true literal from each clause/triangle.
- This is an independent set of size $k = |\Phi|$.

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

\[ 3\text{-SAT} \leq_P \text{Independent Set}. \]

**Proof.**
3-satisfiability reduces to independent set

**Theorem**

\[3\text{-SAT} \leq_P \text{Independent Set.}\]

**Proof.**

\[\iff \text{Let } S \text{ be independent set of size } k.\]

- \(S\) must contain exactly one node in each triangle.
- Set these literals to \text{true}
3-satisfiability reduces to independent set

Theorem

\[ 3\text{-SAT} \leq_P \text{Independent Set}. \]

Proof.

\[ \iff \] Let \( S \) be independent set of size \( k \).

- \( S \) must contain exactly one node in each triangle.
- Set these literals to true and remaining literals consistently.
- All clauses in \( \Phi \) are satisfied.

\[ \Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4) \]
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.
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 $X \leq_P Z$. 
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 $X \leq_P Z$.

Proof idea. Compose the two algorithms.
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 $X \leq_P Z$.

Proof idea. Compose the two algorithms.

Example. 3-SAT $\leq_P$ Independent Set $\leq_P$ Vertex Cover $\leq_P$ Set Cover.
Decision problem. Does there exist a vertex cover of size $\leq k$?
Decision problem. Does there exist a vertex cover of size \( \leq k \)?

Search problem. Find a vertex cover of size \( \leq k \).
Decision, search and 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.
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.

Goal. Show that all three problems poly-time reduce to one another.
Vertex cover. Does there exist a vertex cover of size $\le k$?

Find vertex cover. Find a vertex cover of size $\le k$. 
Vertex cover. Does there exist a vertex cover of size $\leq k$?

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

Theorem. Vertex cover \(\equiv_P\) Find vertex cover.
Search problems VS. Decision problems

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

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

Theorem. Vertex cover $\equiv_P$ Find vertex cover.

Proof.
Search problems VS. Decision problems

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

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

**Theorem.** Vertex cover $\equiv_P$ Find vertex cover.

**Proof.**

$\leq_P$. Decision problem is a special case of search problem.
Search problems VS. Decision problems

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

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

**Theorem.** Vertex cover $\equiv_P$ Find vertex cover.

**Proof.**

$\leq_P$. Decision problem is a special case of search problem.

$\geq_P$. To find a vertex cover of size $\leq k$:
Vertex cover. Does there exist a vertex cover of size $\leq k$?

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

**Theorem.** Vertex cover $\equiv_P$ Find vertex cover.

**Proof.**

$\leq_P$. Decision problem is a special case of search problem.

$\geq_P$. 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 $G - \{v\}$. 
Find vertex cover. Find a vertex cover of size $\leq k$.

Find min vertex cover. Find a vertex cover of minimum size.
Optimization problems VS. Search problems VS. Decision problems

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

Find min vertex cover. Find a vertex cover of minimum size.

**Theorem.** Find vertex cover $\equiv_P$ Find min vertex cover.
Find vertex cover. Find a vertex cover of size $\leq k$.

Find min vertex cover. Find a vertex cover of minimum size.

**Theorem.** Find vertex cover $\equiv_P$ Find min vertex cover.

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

Find min vertex cover. Find a vertex cover of minimum size.

**Theorem.** Find vertex cover $\equiv_P$ Find min vertex cover.

**Proof.**

$\leq_P$. Search problem is a special case of optimization problem.
Find vertex cover. Find a vertex cover of size $\leq k$.

Find min vertex cover. Find a vertex cover of minimum size.

**Theorem.** Find vertex cover $\equiv_P$ Find min vertex cover.

**Proof.**

$\leq_P$. Search problem is a special case of optimization problem.

$\geq_P$. To find vertex cover of minimum size:
Find vertex cover. Find a vertex cover of size $\leq k$.

Find min vertex cover. Find a vertex cover of minimum size.

**Theorem.** Find vertex cover $\equiv_P$ Find min vertex cover.

**Proof.**

$\leq_P$. Search problem is a special case of optimization problem.

$\geq_P$. To find vertex cover of minimum size:

- Binary search (or linear search) for size $k^*$ of min vertex cover.
- Solve search problem for given $k^*$. 
Sequencing Problems
Hamilton cycle. Given an undirected graph $G = (V, E)$, does there exist a cycle $\Gamma$ that visits every node exactly once?
Hamilton cycle. Given an undirected graph $G = (V, E)$, does there exist a cycle $\Gamma$ that visits every node exactly once?

no
Directed Hamilton cycle. Given a directed graph $G = (V, E)$, does there exist a directed cycle $\Gamma$ that visits every node exactly once?

**Theorem**

Directed Hamilton cycle $\leq_P$ Hamilton cycle.
Directed Hamilton cycle. Given a directed graph $G = (V, E)$, does there exist a directed cycle $\Gamma$ that visits every node exactly once?

Theorem

**Directed Hamilton cycle $\leq_P$ Hamilton cycle.**

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

![Diagram of directed graph $G$ and $G'$ with labels $a, b, c, d, e, v, v_{in}, v_{out}, a_{out}, b_{out}, c_{out}, d_{in}, e_{in}$]
Directed Hamilton cycle reduces to Hamilton cycle

**Lemma**

\[ G \text{ has a directed Hamilton cycle iff } G' \text{ has a Hamilton cycle.} \]
Directed Hamilton cycle reduces to Hamilton cycle

Lemma

\[ G \text{ has a directed Hamilton cycle iff } G' \text{ has a Hamilton cycle.} \]

Proof.
Lemma

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

Proof.

$\Rightarrow$

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

**Lemma**

\[ G \text{ has a directed Hamilton cycle iff } G' \text{ has a Hamilton cycle.} \]

**Proof.**

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

\[ \Leftarrow \]
- Suppose \( G' \) has an undirected Hamilton cycle \( \Gamma' \).
- \( \Gamma' \) must visit nodes in \( G' \) using one of following two orders:
  - \( \ldots, \text{black, white, blue, black, white, blue, black, white, blue,} \ldots \)
  - \( \ldots, \text{black, blue, white, black, blue, white, black, blue, white,} \ldots \)
- Black nodes in \( \Gamma' \) comprise either a directed Hamilton cycle \( \Gamma \) in \( G \), or reverse of one.
Theorem

$3$-SAT $\leq_p$ Directed Hamilton cycle.

Proof. Given an instance $\Phi$ of 3-SAT, we construct an instance $G$ of Directed Hamilton cycle that has a Hamilton cycle iff $\Phi$ is satisfiable.

Construction overview. Let $n$ denote the number of variables in $\Phi$. We will construct a graph $G$ that has $2^n$ Hamilton cycles, with each cycle corresponding to one of the $2^n$ possible truth assignments.
Theorem

3-SAT \leq_P Directed Hamilton cycle.

Proof.
Theorem

3-SAT \leq_p Directed Hamilton cycle.

Proof.

Given an instance \( \Phi \) of 3-SAT, we construct an instance \( G \) of Directed Hamilton cycle that has a Hamilton cycle iff \( \Phi \) is satisfiable.
3-satisfiability reduces to directed Hamilton cycle

Theorem

3-SAT $\leq_P$ Directed Hamilton cycle.

Proof.

Given an instance $\Phi$ of 3-SAT, we construct an instance $G$ of Directed Hamilton cycle that has a Hamilton cycle iff $\Phi$ is satisfiable.

Construction overview. Let $n$ denote the number of variables in $\Phi$. We will construct a graph $G$ that has $2^n$ Hamilton cycles, with each cycle corresponding to one of the $2^n$ possible truth assignments.
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 variables $x_i = \text{true}$
Which is truth assignment corresponding to Hamilton cycle below?

A. $x_1 = true, x_2 = true, x_3 = true$
B. $x_1 = true, x_2 = true, x_3 = false$
C. $x_1 = false, x_2 = false, x_3 = true$
D. $x_1 = false, x_2 = false, x_3 = false$
Construction. Given 3-SAT instance \( \Phi \) with \( n \) variables \( x_i \) and \( k \) clauses.

- For each clause: add a node and 2 edges per literal.
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 2 edges per literal.
3-satisfiability reduces to directed Hamilton cycle

Lemma

$\Phi$ is satisfiable iff $G$ has a Hamilton cycle.
3-satisfiability reduces to directed Hamilton cycle

**Lemma**

Φ is satisfiable iff G has a Hamilton cycle.

*Proof.*
3-satisfiability reduces to directed Hamilton cycle

Lemma

Φ is satisfiable iff G has a Hamilton cycle.

Proof.

⇒

- Suppose 3-SAT instance Φ has satisfying assignment x*.
- Then, define Hamilton cycle Γ 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)
Lemma

\( \Phi \) is satisfiable iff \( G \) has a Hamilton cycle.

Proof.
3-satisfiability reduces to directed Hamilton cycle

**Lemma**

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

**Proof.**

$\implies$

- 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 $x_i^* = \text{true}$ if $\Gamma'$ traverses row $i$ left-to-right; otherwise, set $x_i^* = \text{false}$.
- traversed in “correct” direction, and each clause is satisfied.
Graph Coloring
Home reading!
Numerical Problems
Subset sum. Given $n$ natural numbers $w_1, \ldots, w_n$ and an integer $W$, is there a subset that adds up to exactly $W$?


Remark. With arithmetic problems, input integers are encoded in binary. Poly-time reduction must be polynomial in binary encoding.
Subset sum. Given $n$ natural numbers $w_1, \ldots, w_n$ and an integer $W$, is there a subset that adds up to exactly $W$?

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


Yes. $215 + 355 + 355 + 580 = 1505$. 

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


Yes. $215 + 355 + 355 + 580 = 1505$.

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

$3$-SAT $\leq_P$ Subset sum.
Theorem

$3$-SAT $\leq_P$ Subset sum.

Proof. Given an instance $\Phi$ of $3$-SAT, we construct an instance of 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 having $n + k$ digits:

- Include one digit for each variable $x_i$ and one digit 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.

\[
\begin{align*}
C_1 &= \neg x_1 \vee x_2 \vee x_3 \\
C_2 &= x_1 \vee \neg x_2 \vee x_3 \\
C_3 &= \neg x_1 \vee \neg x_2 \vee \neg x_3
\end{align*}
\]
3-satisfiability reduces to subset sum

Lemma

Φ is satisfiable iff there exists a subset that sums to W.
3-satisfiability reduces to subset sum

**Lemma**

\[ \Phi \text{ is satisfiable iff there exists a subset that sums to } W. \]

**Proof.**
3-satisfiability reduces to subset sum

**Lemma**

Φ is satisfiable iff there exists a subset that sums to W.

**Proof.⇒** Suppose 3-SAT instance Φ has satisfying assignment x*.

- If \( x_i^* = true \), select integer in row \( x_i \); otherwise, select integer in row \( \neg x_i \).
- Each \( x_i \) digit sums to 1.
- Since Φ is satisfiable, each \( C_j \) digit sums to at least 1 from \( x_i \) and \( \neg x_i \) rows.
- Select dummy integers to make \( C_j \) digits sum to 4.

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

3-SAT instance

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

Subset sum instance

\[
\begin{array}{cccc}
C_1 & 0 & 0 & 1 & 0 & 0 & 100 \\
C_2 & 0 & 0 & 2 & 0 & 0 & 200 \\
C_3 & 0 & 0 & 0 & 0 & 1 & 10 \\
W & 0 & 0 & 0 & 2 & 0 & 20 \\
& 0 & 0 & 0 & 0 & 1 & 1 \\
& 0 & 0 & 0 & 0 & 2 & 2 \\
& 111,444
\end{array}
\]

dummies to get clause columns to sum to 4
3-satisfiability reduces to subset sum

**Lemma**

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

**Proof.**
3-satisfiability reduces to subset sum

**Lemma**

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

**Proof.** Suppose there exists a subset \( S^* \) that sums to \( W \).

- Digit \( x_i \) forces subset \( S^* \) to select either row \( x_i \) or row \( \neg x_i \) (but not both).
- If row \( x_i \) selected, assign \( x_i^* = \text{true} \); otherwise, assign \( x_i^* = \text{false} \).

Digit \( C_j \) forces subset \( S^* \) to select at least one literal in clause.

<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>0</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>
</tbody>
</table>

\( W \) is the sum of the weights of the literals in the clause.

3-SAT instance

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

Subset sum instance

\[
W \begin{bmatrix}
1 & 1 & 1 & 4 & 4 & 4
\end{bmatrix} + 111,444
\]
Subset sum. Given \( n \) natural numbers \( w_1, \ldots, w_n \) and an integer \( W \), is there a subset that adds up to exactly \( W \)?

Knapsack. Given a set of items \( X \), weights \( u_i \geq 0 \), values \( v_i \geq 0 \), a weight limit \( U \), and a target value \( V \), is there a subset \( S \subseteq X \) such that:

\[
\sum_{i \in S} u_i \leq U, \quad \sum_{i \in S} v_i \geq V
\]
Subset sum. Given \( n \) natural numbers \( w_1, \ldots, w_n \) and an integer \( W \), is there a subset that adds up to exactly \( W \)?

Knapsack. Given a set of items \( X \), weights \( u_i \geq 0 \), values \( v_i \geq 0 \), a weight limit \( U \), and a target value \( V \), is there a subset \( S \subseteq X \) such that:

\[
\sum_{i \in S} u_i \leq U, \quad \sum_{i \in S} v_i \geq V
\]

Recall. \( O(nU) \) dynamic programming algorithm for knapsack.
**Subset sum**. Given $n$ natural numbers $w_1, \ldots, w_n$ and an integer $W$, is there a subset that adds up to exactly $W$?

**Knapsack**. Given a set of items $X$, weights $u_i \geq 0$, values $v_i \geq 0$, a weight limit $U$, and a target value $V$, is there a subset $S \subseteq X$ such that:

$$\sum_{i \in S} u_i \leq U, \quad \sum_{i \in S} v_i \geq V$$

**Recall**. $O(nU)$ dynamic programming algorithm for knapsack.

**Challenge**. Prove subset sum $\leq_P$ Knapsack.
Poly-time reductions

constraint satisfaction

3-SAT

1. INDEPENDENT-SET
   - 3-SAT poly-time reduces to INDEPENDENT-SET
   - VERTEX-COVER
   - SET-COVER

2. DIR-HAM-CYCLE
   - HAM-CYCLE

3. 3-COLOR

4. SUBSET-SUM
   - KNAPSACK

packing and covering
sequencing
partitioning
numerical
Karp's 20 poly-time reductions from satisfiability

**FIGURE 1** - Complete Problems

Dick Karp (1972)
1985 Turing Award