Reduction, NP and NPC

Cormen et. al. 34 NP Completeness

Major Transition

So far we have studied certain **algorithmic patterns**

Greedy,
Divide and conquer,
Dynamic programming

to develop efficient algorithms.

Now we want to classify and quantify **problems** that cannot be solved efficiently.

Our tool for doing this is another algorithmic pattern: **reduction**

Reduction transforms the input and output of an algorithm so that it can be used to solve a different problem.

Reductions

An person A is handed an **empty** kettle and is asked to make tea. A fills up the kettle, boils the water and makes tea.

A person B is handed a **full** kettle and is asked to make tea.

What do you think the person B does?

B empties the kettle and hands it to A ☺

B knows that A can make tea, but A starts with an empty kettle, so B transforms the problem so that it fits A's algorithm.

Reductions

Sustainability Mantra: Reuse, Reduce, Recycle …

If we have a solution to one problem and we can use this solution to solve another problem, we do not need to write a new program, we can **reuse** the existing code, and **reduce** the new problem (change its input (and output)), so it can use the existing code to solve it.

eg 1: We have a **max heap**, but we need a **min heap**. How can we use the max heap to perform min heap operations, without changing the max heap code?

```
InsertMinHeap(x): InsertMaxHeap(-x);

ExtractMinHeap(): x=ExtractMaxHeap();  return –x;
```

Arbitrage

We have a set of currencies and conversion rates from each currency to each other currency.

Is there a cyclic sequence of currency exchanges that provides a profit? eg:

loss: \$ → £ → € → ¥ → \$: 0.63*1.14*106*0.013 = 0.981

profit: \$ → £ → € → ¥ → \$: 0**.64***1.14*106*0.013 = 1.005

$$\begin{tikzcd}
\$ \arrow[r, "0.63"] & \pounds \arrow[d, "1.14"] \\
\yen \arrow[u, "0.013"] & \text{€} \arrow[l, "106"]
\end{tikzcd}$$

$$\begin{tikzcd}
\$ \arrow[r, "\mathbf{0.64}"] & \pounds \arrow[d, "1.14"] \\
\yen \arrow[u, "0.013"] & \text{€} \arrow[l, "106"]
\end{tikzcd}$$

Arbitrage

We have a set of currencies and conversion rates from each currency to each other currency.

Is there a cyclic sequence of currency exchanges that provides a profit?

Is there an algorithm that we studied, that we can use to solve his problem?

Bellman Ford: finding negative cycles in a graph

How would we reduce to that algorithm?

rate → -(log rate)

* → +

profit if product>1 → profit if sum<0

Try it for: 2 ¼ 4 ¼ and for: 2 ½ 4 ¼ and 2 1 4 ¼

Arbitrage

Bellman Ford: finding negative cycles in a graph

How would we reduce to that algorithm?

rate → -(log rate)

* → +

profit if product>1 → profit if sum<0

Try it for: 2 ¼ 4 ¼ and for: 2 ½ 4 ¼ and 2 1 4 ¼

2* ¼ * 4 * ¼ = ½ → -1 + 2 + -2 + 2 = 1 LOSS

2* ½ *4* ¼ = 1 → -1 + 1 + -2 + 2 = 0

2* 1 *4 * ¼ = 2 → -1 + 0 + -2 + 2 = -1 PROFIT

Algorithm Design Patterns and Problem Patterns

Algorithm design patterns	Example
▪ Greedy.	O(n log n) interval scheduling
▪ Divide-and-conquer.	O(n log n) closest point pair
▪ Dynamic programming.	O(n m) negative cycles
▪ Reductions.	
▪ Approximation algorithms.	
▪ Randomized algorithms.	

Problem patterns	
▪ NP-completeness.	Existence of a polynomial algorithm unknown.
▪ Undecidability.	Provable that no algorithm exists.

Classifying problems

Q. Which problems will we be able to solve in practice?
Those with polynomial-time algorithms. But some polynomial algorithms have such a high degree that they are practically intractable (e.g. primes, $O((\log n)^6)$. (log n: the number of digits in the number n)

Some problems provably require exponential-time.

- Towers of Hanoi
- Generate all permutations of 1 to n

But some algorithms have exponential worst case, but are still tractable for the type of problems we usually solve with them (Simplex algorithm for Linear Programming)

Grey area: A number of fundamental problems have defied classification for decades. We don't know of a polynomial algorithm for them, but neither can prove that no polynomial algorithm exists.

Primes Problem

PRIMES problem: given an int $p > 1$, is p prime?
PRIMES: $X = \{ 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, ... \}$
Algorithm:
[Agrawal-Kayal-Saxena, 2002] $p(|s|) = |s|^{12}$.
Later improved to $|s|^6$ [Pomerance, Lenstra]
(See wikipedia: AKS primality test)

- For a long time people were not sure if the primes problem had a polynomial solution.

Polynomial-Time Reductions

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

Reduction. Problem Y **polynomially reduces to** problem X if arbitrary instances of problem Y can be solved using:

- Polynomial number of standard computational steps to transform a Y input to an X input, and an X result to a Y result, plus
- Polynomial number of calls to black box that solves problem X.
- Because polynomial (plus or times) polynomial is polynomial

Notation. $Y \leq_P X$. We can think of it as "X is potentially more complex than Y".

Remark

- We pay for the time to write down instances sent to black box $\Rightarrow$ inputs of X must also be of polynomial size.

Polynomial-Time Reduction

Purpose. Classify problems according to **relative** difficulty.

Design algorithms. If $Y \leq_P X$ and X can be solved in polynomial-time, then Y can also be solved in polynomial time.

Q: If $Y \leq_P X$ and X cannot be solved in polynomial-time, can Y be solved in polynomial time, or not?

Hint: can you shoot a fly with a cannon?

A: It depends on the problem, see the coming slides

Interval Scheduling

You have a resource (hotel room, printer, lecture room, telescope, manufacturing facility, professor...)

There are requests to use the resource in the form of start time s_i and finish time f_i, such that $s_i < f_i$

Objective: grant as many requests as possible.
Two requests i and j are **compatible** if they don't overlap, i.e.

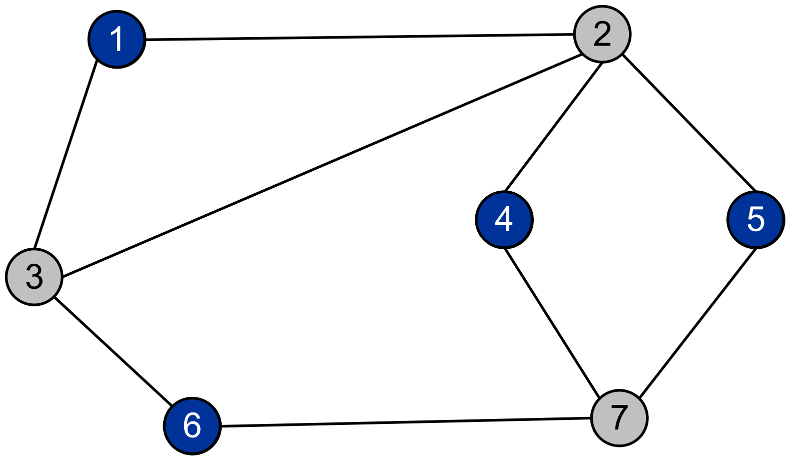
$$f_i \leq s_j \text{ or } f_j \leq s_i$$

Solution? Complexity?

Greedy, sort by finish time,
pick first compatible interval, repeat
O(n logn)

Independent Set

Input. Graph.
Goal. Find maximum cardinality independent set:



subset of nodes such that no two are joined by an edge

Can you formulate interval scheduling as an independent set problem?

Yes, interval = node, edge if two intervals overlap.
We call this an interval graph.

If so, how could you solve the interval scheduling problem?

Transform it to an independent set problem.

Independent set problem

- There is no known efficient way to solve the independent set problem.
- **But we just said: we can formulate interval scheduling as independent set problem..... ???**
- **We used the solution of a more complex problem X to solve a simpler problem Y, by reducing Y polynomially to X**
- What does "no efficient way" mean?
- The only solution we have so far is trying all sub sets and finding the largest independent one.
- How many sub sets of a set of n nodes are there?

Polynomial-Time Reduction

Purpose. Classify problems according to relative difficulty.

Design algorithms. If $Y \leq_P X$ and X can be solved in polynomial-time, then Y can also be solved in polynomial time.

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

WHY?

Otherwise we have a contradiction:
if X has a polynomial solution then Y has.

Establish equivalence. If $X \leq_P Y$ and $Y \leq_P X$ they are as hard, notation: $X \equiv_P Y$.

Reduction Strategies

- Reduction by equivalence.
- Reduction from special case to general case.
- Transitivity of reductions
- Reduction by encoding with gadgets.

The Independent Set Problem

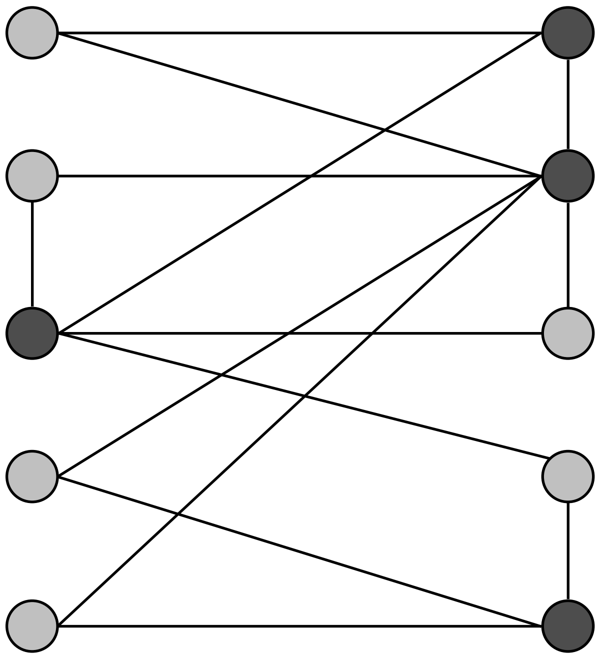
INDEPENDENT SET: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge **at most one** of its endpoints is in S?

(i.e., no edges join nodes in S)

Is there an independent set of size ≥ 6? Yes.

Is there an independent set of size ≥ 7? No.

We have turned the independent set problem into a decision (Yes/No) problem by introducing |S| >= k.

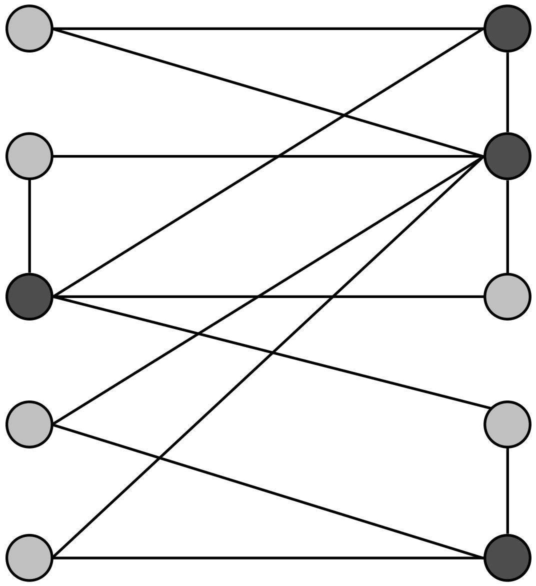


● independent set

The Vertex Cover Problem

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?
(i.e., the nodes in S cover all edges in E)

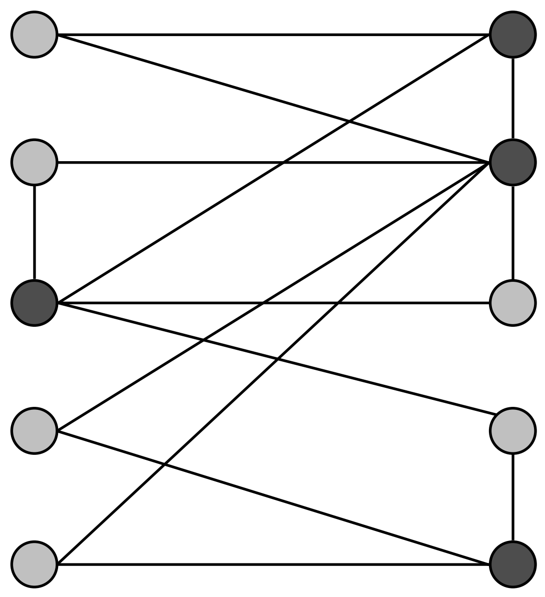
Is there a vertex cover of size ≤ 4? Yes.
Is there a vertex cover of size ≤ 3? No.



● vertex cover

Vertex Cover and Independent Set

Claim. Let G=(V, E) be a graph. Then S is an independent set iff V–S is a vertex cover.



○ independent set

● vertex cover

Vertex Cover and Independent Set

Claim. Let $G=(V, E)$ be a graph. Then S is an independent set iff $V - S$ is a vertex cover.

$\Rightarrow$

- Let S be an independent set.
- Consider an arbitrary edge (u, v).
- S independent $\Rightarrow u \notin S$ or $v \notin S \Rightarrow u \in V - S$ or $v \in V - S$.
- Thus, $V - S$ covers (u, v).

$\Leftarrow$

- Let $V - S$ be a vertex cover.
- Consider two nodes $u \in S$ and $v \in S$.
- Observe that $(u, v) \notin E$ since $V - S$ is a vertex cover, and therefore all edges have at least one node in $V - S$.
- Thus, no two nodes in S are joined by an edge $\Rightarrow S$ independent set. ▪

Vertex Cover and Independent Set

S is an independent set iff V – S is a vertex cover.

This shows:

VERTEX-COVER $\leq_P$ INDEPENDENT-SET and
INDEPENDENT-SET $\leq_P$ VERTEX-COVER.

from which we conclude:

VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.

Reduction from Special Case to General Case

In this case we do not prove $Y \equiv_P X$, we prove $Y \leq_P X$

Eg: Vertex cover versus set cover

Vertex cover is phrased in terms of graphs

Set cover is phrased, more generally, in terms of sets

Set Cover

SET COVER: Given a set U of elements, a collection $S_1, S_2, \ldots,$ S_m of subsets of U, and an integer k, is there a collection of at most k of these subsets whose union is equal to U?

Example:

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

k = 2

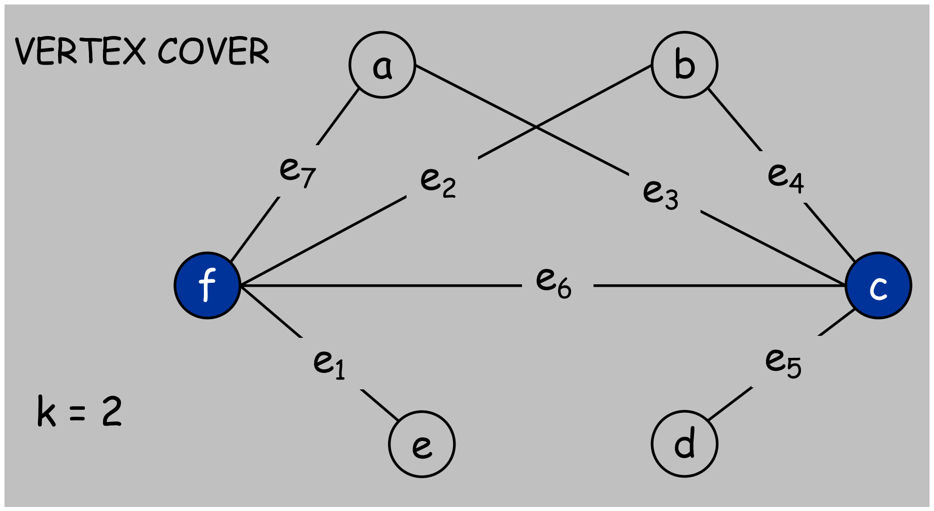
$S_1 = \{3, 7\}$	$S_4 = \{2, 4\}$
$S_2 = \{3, 4, 5, 6\}$	$S_5 = \{5\}$
$S_3 = \{1\}$	$S_6 = \{1, 2, 6, 7\}$

Vertex Cover Reduces to Set Cover

Claim. VERTEX-COVER $\le_P$ SET-COVER.

Proof. Given a VERTEX-COVER instance $G = (V, E)$, k, we reduce it to a set cover instance with U=E and a subset of edges S_v for each node v in V

- Create SET-COVER instance:
  - $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$
- Set-cover of size $\le k$ iff vertex cover of size $\le k$.
- Hence we have shown that VERTEX-COVER $\le_P$ SET-COVER ▪

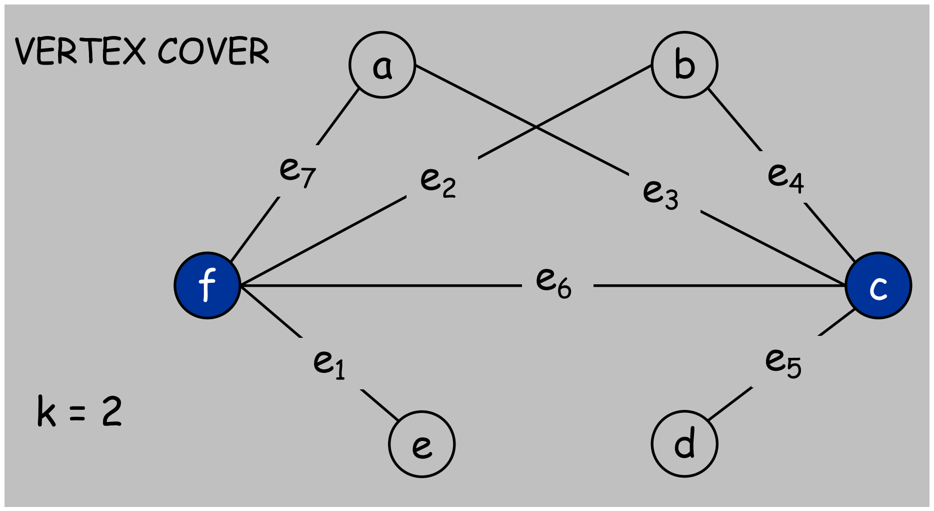


Vertex Cover Reduces to Set Cover

Claim. VERTEX-COVER $\leq_P$ SET-COVER.

Proof. Given a VERTEX-COVER instance $G = (V, E)$, k, we reduce it to a set cover instance with $U=E$ and a subset S_v for each v in V

- Create SET-COVER instance:
  - $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$
- Set-cover of size $\leq k$ iff vertex cover of size $\leq k$.
- Hence we have shown that VERTEX-COVER $\leq_P$ SET-COVER ▪



SET COVER

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

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

The Satisfiability Problem

Literal: A Boolean value or variable or its negation. x_i or $\overline{x_i}$ $(\overline{x_i} = not\ x_i)$

Clause: A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$

Conjunctive normal form (CNF):
A propositional formula Φ that is a conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT: Given a CNF Φ, does it have a satisfying truth assignment?
3-SAT: SAT where each clause contains exactly 3 literals.
Any logical formula can be expressed in CNF.

Example: $\left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(x_2 \vee x_3\right) \wedge \left(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}\right)$

Yes: x_1 = true, x_2 = true x_3 = false, or $x_1=1, x_2=1, x_3=0$

What about $(x_2 \vee x_3)$? $x_2 \vee x_3 \vee 0$

What is a necessary and sufficient condition for a CNF to be true?

Notice: for the CNF to be true all clauses must have at least one literal evaluating to true

SAT

In its general form, SAT concerns an expression in CNF (Conjunctive Normal Form). It's many clauses, ANDed ($\wedge$) together. Each clause is a disjunction, a number of literals (variables or negated ($\neg$) variables) ORed ($\vee$) together.

$(a \vee b \vee c) \wedge (a \vee b \vee \neg c \vee d) \wedge (c \vee d) \wedge (b)$

SAT => 3SAT

Say we have an arbitrary expression:

$(a \lor b \lor c) \land (a \lor b \lor \neg c \lor d) \land (c \lor d) \land (b)$

It's CNF, but not 3-CNF. We want 3-CNF, because we want to work with 3SAT, which is easier than SAT. We want each disjunctive clause to be three exactly elements. How can we convert CNF to 3-CNF? There are several cases to consider:

- clauses with one element
- clauses with two elements
- clauses with three elements
- clauses with more than three elements

Clause with one element

If we have a clause with one element, (*b*), we introduce a new variable *x*, and rewrite the single-element clause thus:

$(b) \equiv (b \lor x) \land (b \lor \neg x)$

If x=0 (false), then this reduces to

$(b) \equiv (b \lor 0) \land (b \lor \neg 0)$
$(b) \equiv (b) \land (b \lor \neg 0)$
$(b) \equiv (b) \land (b \lor 1)$
$(b) \equiv (b) \land (1)$
$(b) \equiv (b)$

If x=1 (true), then this reduces to

$(b) \equiv (b \lor 1) \land (b \lor \neg 1)$
$(b) \equiv (1) \land (b \lor \neg 1)$
$(b) \equiv (b \lor \neg 1)$
$(b) \equiv (b \lor 0)$
$(b) \equiv (b)$
$\therefore$, $(b) \equiv (b \lor x) \land (b \lor \neg x)$.

Hooray! We've converted a one-element clause into two two-element clauses.Or, you can just convert (b) to (b ∨ 0), if you allow boolean constants instead of literals. Or even (b ∨ b).

Clause with 2 elements

If we have a clause with two elements, $(c \lor d)$, we introduce a new variable y, and rewrite the single-element clause thus:

$$(c \lor d) \equiv (c \lor d \lor y) \land (c \lor d \lor \neg y)$$

If y=0 (false), then this reduces to

$$(c \lor d) \equiv (c \lor d \lor 0) \land (c \lor d \lor \neg 0)$$
$$(c \lor d) \equiv (c \lor d) \land (c \lor d \lor \neg 0)$$
$$(c \lor d) \equiv (c \lor d) \land (c \lor d \lor 1)$$
$$(c \lor d) \equiv (c \lor d) \land (1)$$
$$(c \lor d) \equiv (c \lor d)$$

If y=1 (true), then this reduces to

$$(c \lor d) \equiv (c \lor d \lor 1) \land (c \lor d \lor \neg 1)$$
$$(c \lor d) \equiv (1) \land (c \lor d \lor \neg 1)$$
$$(c \lor d) \equiv (c \lor d \lor \neg 1)$$
$$(c \lor d) \equiv (c \lor d \lor 0)$$
$$(c \lor d) \equiv (c \lor d)$$

$\therefore$,

$$(c \lor d) \equiv (c \lor d \lor y) \land (c \lor d \lor \neg y).$$

Hooray! We've converted a two-element clause into two three-element clauses.

Clause with more than 3 elements

Lemma: if the formula $(A \vee B)$ is satisfiable, that is, if there is a truth-assignment to A and B such that $(A \vee B)$ is true, then the formula $(A \vee z) \wedge (B \vee \neg z)$, where z is a new variable, is satisfiable. This is true even if A and B are not simple variables, but instead expressions.

Think of it this way:

- If $(A \vee B)$ is true, then either A is true, or B is true (or maybe both).
- If A is true, then let $z=0$, which yields: $(A \vee 0) \wedge (B \vee \neg 0)$, which is true.
- If B is true, then let $z=1$, which yields: $(A \vee 1) \wedge (B \vee \neg 1)$, which is true.
- If A and B are both false, then $(A \vee B)$ is false. $(A \vee z) \wedge (B \vee \neg z) \Rightarrow (0 \vee z) \wedge (0 \vee \neg z) \Rightarrow z \wedge \neg z \Rightarrow$ false.

Clause with more than 3 elements - 2

Or, as a truth table:

A	B	$A \vee B$	$A \vee z$	$B \vee \neg z$	$(A \vee z) \wedge (B \vee \neg z)$
0	0	0	z	$\neg z$	0
0	1	1	z	1	z
1	0	1	1	$\neg z$	$\neg z$
1	1	1	1	1	1

Clause with more than 3 elements - 3

The last column is satisfiable, by an appropriate assignment to z, in all rows except the first one, where $A \lor B$ itself is not satisfiable. Therefore, the formulas $A \lor B$ and $(A \lor z) \land (B \lor \neg z)$ have *equal* satisfiability.

This lets us break a lengthy clause into two smaller clauses. We can pull the last two components of the lengthy clause into a separate clause of length 3, replacing them with a new variable. This reduces the length of the first clause by one element. Rinse & repeat.

- $(a \lor b \lor c \lor d) \Rightarrow (a \lor b \lor z) \land (c \lor d \lor \neg z)$
- $(a \lor b \lor c \lor d \lor e) \Rightarrow (a \lor b \lor c \lor z) \land (d \lor e \lor \neg z)$
- $(a \lor b \lor c \lor d \lor e \lor f) \Rightarrow (a \lor b \lor c \lor d \lor z) \land (e \lor f \lor \neg z)$

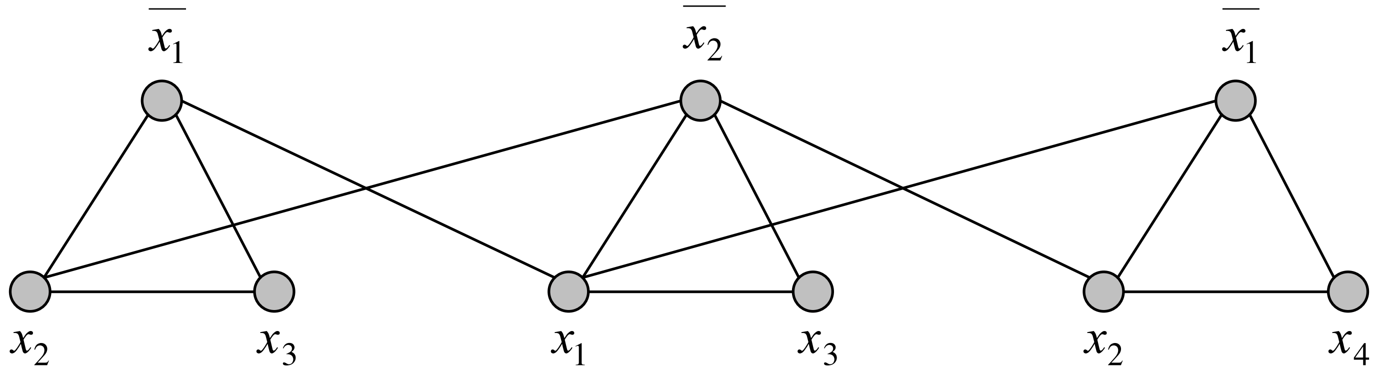
3-SAT Reduces to Independent Set

Claim. 3-SAT $\leq_P$ INDEPENDENT-SET.

Proof. Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT-SET that has an independent set of size k iff Φ is satisfiable.

Construction.

- G contains 3 vertices for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect a literal to each of its negations.
- We call these connected triangles **gadgets**, creating a useful instance of independent set.



$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4\right)$$

3-SAT Reduces to Independent Set

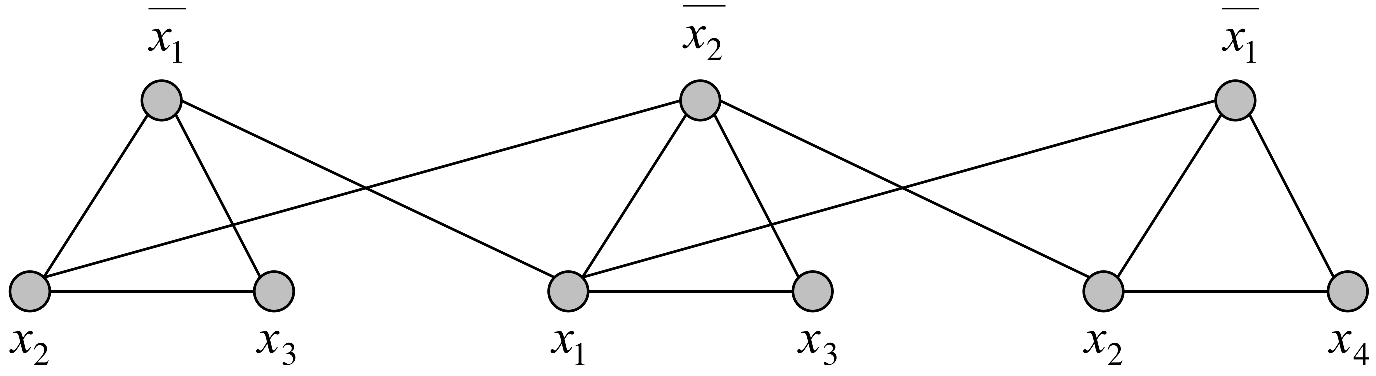
Claim. G contains independent set of size k = $|\Phi|$ iff Φ is satisfiable.

number of clauses, i.e. triangles

Proof. $\Rightarrow$ Let S be independent set of size k.

1) S must contain exactly one vertex in each triangle, and cannot contain both x_i and $\overline{x_i}$ (they are connected)
2) Set these literals to true, there are no conflicts, because (1)
3) All clauses are satisfied.

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



$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4\right)$$

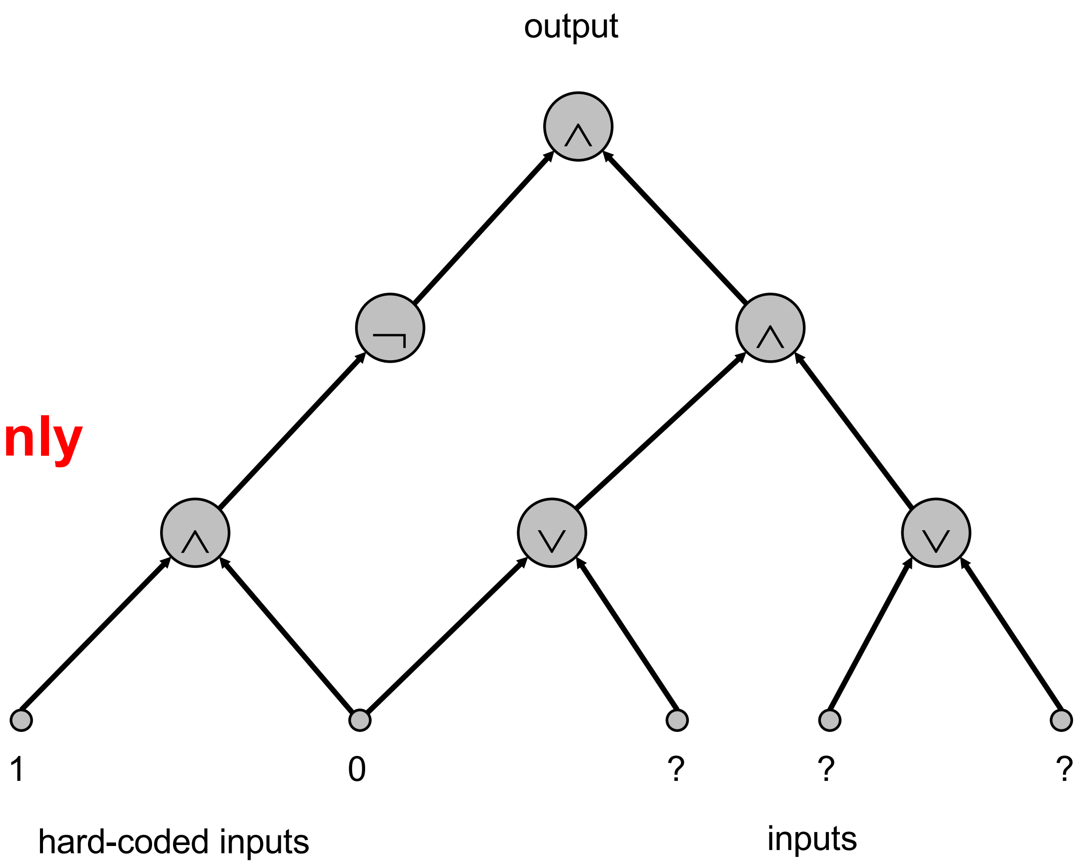
Circuit Satisfiability

CIRCUIT-SAT. A combinational circuit is a directed acyclic graph built out of AND ($\wedge$), OR($\vee$), and NOT($\neg$) nodes. Given such a circuit, is there a way to set the circuit inputs so that the output is 1?

yes: 1 0 1

Is that the only option ?

No,
1 1 0
1 1 1



CIRCUIT-SAT $\leq_P$ 3-SAT

- Let K be any circuit.
- Create a 3-SAT variable x_i for each circuit element i.
- Make 3-SAT clauses; compute values for each circuit-SAT node, eg.:

  $x_2 = \neg x_3 \Rightarrow$ add 2 clauses: $x_2 \vee x_3$, $\overline{x_2} \vee \overline{x_3}$

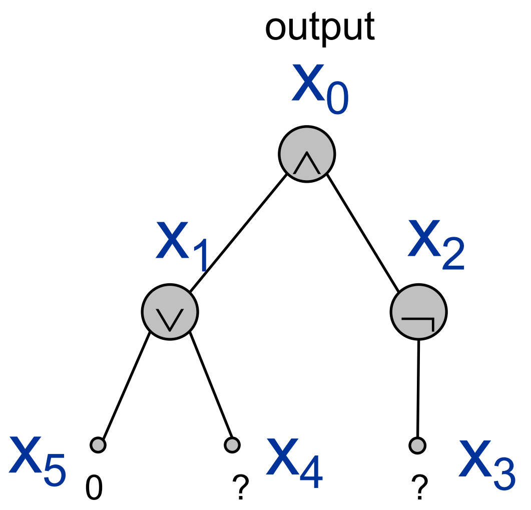
  $x_1 = x_4 \vee x_5 \Rightarrow$ add 3 clauses: $x_1 \vee \overline{x_4}$, $x_1 \vee \overline{x_5}$, $\overline{x_1} \vee x_4 \vee x_5$

  $x_0 = x_1 \wedge x_2 \Rightarrow$ add 3 clauses: $\overline{x_0} \vee x_1$, $\overline{x_0} \vee x_2$, $x_0 \vee \overline{x_1} \vee \overline{x_2}$

- Hard-coded input values and output value.
  - $x_5 = 0 \Rightarrow$ add 1 clause: $\overline{x_5}$
  - $x_0 = 1 \Rightarrow$ add 1 clause: x_0

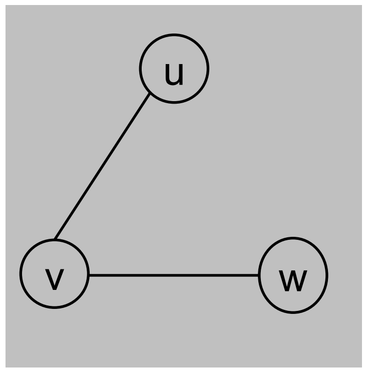
- Final step: turn clauses of length < 3 into clauses of length exactly 3. **HOW?**

  x → x or 0 or 0

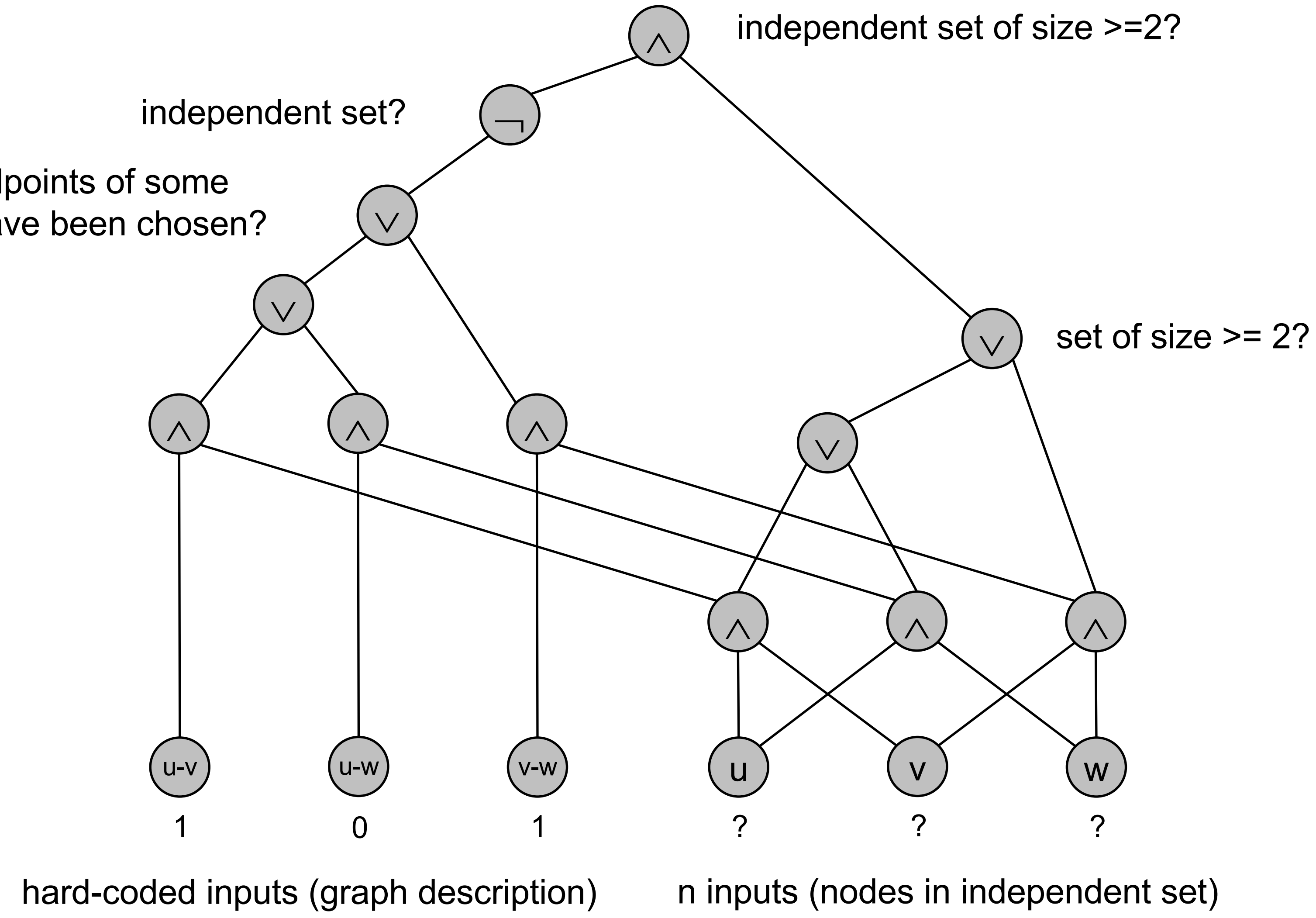


Independent Set $\leq_p$ CircSAT

Example. Construction below creates a circuit C whose inputs can be set so that C outputs true iff graph G has an independent set of size >=2.



G = (V, E), n = 3


independent set of size >=2?
independent set?
both endpoints of some edge have been chosen?
set of size >= 2?
u-v
u-w
v-w
u
v
w
1
0
1
?
?
?
hard-coded inputs (graph description)
n inputs (nodes in independent set)

Try it for {u,v}

Try it for {u,w}

Independent Set $\leq_p$ CircSAT

To show that any Independent Set$_k$ problem $\leq_p$ CircSAT we construct a circuit *C* with inputs and one output:

inputs:

1: **graph description**, in terms of all its edges (1 bit per node pair)

2: **set description**, in terms of the nodes in the set (1 bit per node)

output: one output bit: yes/no

The components of the circuit:

1: define the graph in terms of the edges it contains

2: check whether the set has a pair of nodes that is an edge in the graph.
If there is such a pair, the set is not independent

3: count whether there are at least k nodes in the set

Review and Conclusions

Basic reduction strategies.

- Equivalence: INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
- Special case to general case: VERTEX-COVER $\le_P$ SET-COVER.
- Encoding with gadgets: 3-SAT $\le_P$ INDEPENDENT-SET.

Transitivity. If $X \le_P Y$ and $Y \le_P Z$, then $X \le_P Z$.
Proof idea. Compose the two algorithms.

Example:

3-SAT $\le_P$ INDEPENDENT-SET $\le_P$ VERTEX-COVER $\le_P$ SET-COVER.

Equivalence through transitivity. If $X \le_P Y$ and $Y \le_P Z$, and $Z \le_P X$ then they are all equivalent.

Example:

C-SAT $\le_P$ 3-SAT $\le_P$ INDEPENDENT-SET $\le_P$ C-SAT

Why are NP problems called “NP”?

The act of searching for a solution can be viewed as a Non Deterministic Search over all possible solutions.

The Non Deterministic search guesses all choices of the search at the same time, thereby allowing us to find the right guess in one step (think choice vector e.g. of the knapsack problem with span=n).

This brings the search time in an exponential sized search space down to a polynomial, if we could make all guesses right.

So NP stands for **Non Deterministic Polynomial**

Example 3-Satisfiability

SAT. Given a Conjunctive Normal Form (ands of or-expressions) formula Φ, is there a satisfying assignment?
3_SAT: 3 variables in each or-expression, n: the number of variables in Φ

Solution is a choice vector: An assignment of truth values to the n Boolean variables.

$$\left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(x_1 \vee x_2 \vee x_4\right) \wedge \left(\overline{x_1} \vee \overline{x_3} \vee \overline{x_4}\right)$$

Possible solution vector

$$x_1 = 1,\ x_2 = 1,\ x_3 = 0,\ x_4 = 1$$

3-SAT is an problem NP (parlance 3-SAT is in NP, because NP is the class of NP problems
).

P, NP, EXP

P. Problems for which there is a polynomial-time algorithm.

NP. Problems for which there is a polynomial length choice vector, or polynomial time certifiers.

EXP. Problems for which there is an exponential-time algorithm.

Theorem 1. P $\subseteq$ NP.

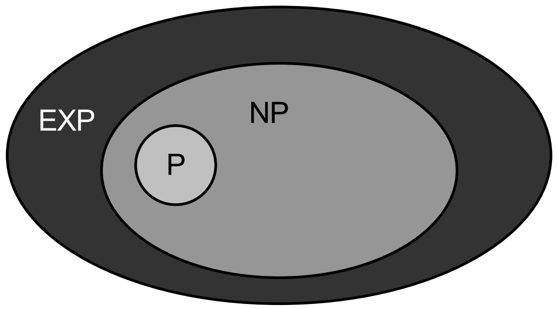
Theorem 2. NP $\subseteq$ EXP.

Later theory/algorithms courses (e.g. cs520) will discuss the proofs of these theorems.

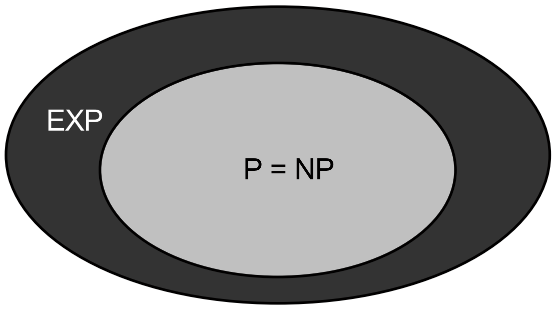
The One Million Dollar CS Question: P = NP?

P = NP?

- Clay mathematics institute $1 million prize.



If P ≠ NP



If P = NP

would break RSA cryptography
(and potentially collapse economy)

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

Consensus opinion on P = NP? Probably no.

NP Completeness

- Polynomial time reductions ($X \leq_P Y$)
- The class NP – problems with polynomial time certifiers
- NP complete problem – problem in NP such that every other NP problem has a polynomial reduction to it.
- Examples of NP-complete problems:

  Circuit-SAT, 3-SAT, Independent Set

Circuit SAT is NPC [Cook 1971, Levin 1973]

Take an arbitrary problem X in NP
Show it can be reduced in polynomial time to Circuit SAT

Intuitive Argument:
Any algorithm, that takes a fixed number of n input bits and produces an output bit (yes/no answer), can be represented by a circuit of **and-s**, o**r-s**, and **not-s,** after all, that is how we can view any computer program in execution.

If the program takes a number of steps polynomial in n, then the circuit has polynomial size.

(The nasty details are in how an algorithm is translated into a circuit :)

3-SAT is NP-Complete

Theorem. 3-SAT is NP-complete.

Proof. Enough to show that CIRCUIT-SAT $\le_P$ 3-SAT since 3-SAT is in NP.

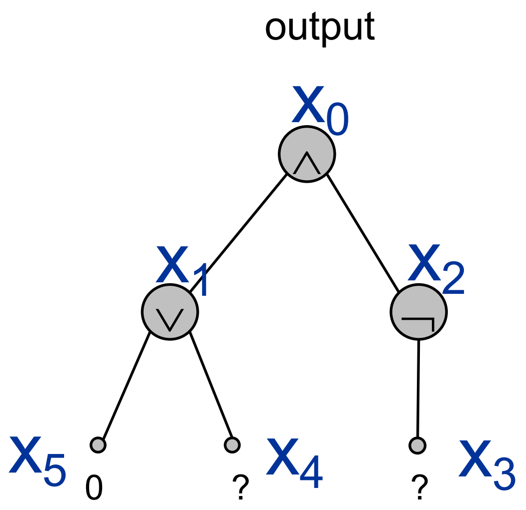
- We did earlier in this lecture (showed that 3-SAT $\le_P$ CIRCUIT-SAT).
- Let K be any circuit.
- Create a 3-SAT variable x_i for each circuit element output (i.wire) i.
- Make 3-SAT clauses compute values for each circuit-SAT node, eg.:

  $x_2 = \neg x_3 \Rightarrow$ add 2 clauses: $x_2 \vee x_3$, $x_2 \vee x_3$

  $x_1 = x_4 \vee x_5 \Rightarrow$ add 3 clauses: $x_1 \vee \overline{x_4}$, $x_1 \vee \overline{x_5}$, $\overline{x_1} \vee x_4 \vee x_5$

  $x_0 = x_1 \wedge x_2 \Rightarrow$ add 3 clauses: $\overline{x_0} \vee x_1$, $\overline{x_0} \vee x_2$, $x_0 \vee \overline{x_1} \vee \overline{x_2}$

- Hard-coded input values and output value.
  - $x_5 = 0 \Rightarrow$ add 1 clause: x_5
  - $x_0 = 1 \Rightarrow$ add 1 clause: x_0

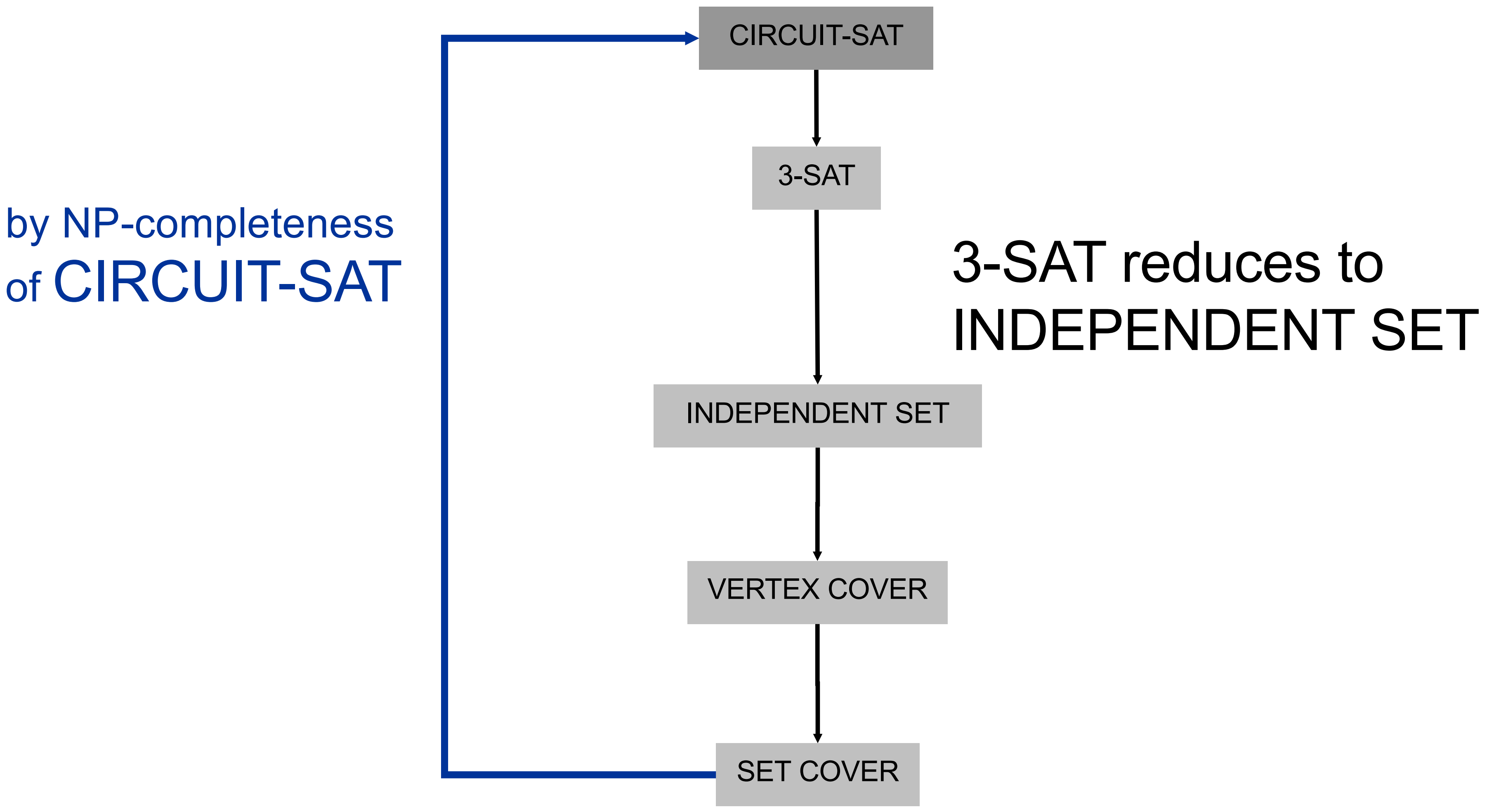


- Final step: turn clauses of length < 3 into clauses of length exactly 3.
- HOW?

add "or 0"-s

NP-Completeness

Observation. All problems below are NP-complete and polynomially reduce to one another!


CIRCUIT-SAT
3-SAT
INDEPENDENT SET
VERTEX COVER
SET COVER
by NP-completeness of CIRCUIT-SAT
3-SAT reduces to INDEPENDENT SET

Extent and Impact of NP-Completeness

Extent of NP-completeness. [Papadimitriou 1995]

- Prime intellectual export of CS to other disciplines.
- 6,000 citations per year
- Broad applicability and classification power.
- "Captures vast domains of computational, scientific, mathematical endeavors, and seems to roughly delimit what mathematicians and scientists have been aspiring to compute feasibly."