NP-Completeness and Intractability

CS 3AC3

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Acknowledgments: Material partially based on *Algorithm Design* by Jon Kleinberg and Éva Tardos (Chapter 8)

Classifying big O classes

Typical O(...): (notation $log\ n = log_2 n$)

- $O(log\ n)$ $O(log(log(n))\ldots$
- $O(n)$
- $O(n\ log\ n)$
- $O(n^2)$ $O(n^k)$
- $O(2^n)$ $O(k^n)$

Classification:

$O(log\ n)$, $O(n)$, $O(n\ log\ n)$ } desired

$O(n^k)$: acceptable

$O(2^n)$: UNACCEPTABLE

Classifying big O classes

Fact

For every $k \geq 0$ and every $\alpha > 1$, there exists n_0 such that for every $n > n_0$:

$$n^k < \alpha^n$$

Another classification:

$O(n^k)$: polynomial, i.e. **GOOD**

$O(\alpha^n)$: non-polynomial, i.e. **BAD**

The class **P**

Definition

The class **P** (from *Polynomial*) consists of those problems that are solvable in polynomial time. More specifically, they are problems that can be solved in $O(n^k)$ for some constant k. where n is the size of the input to the problem.

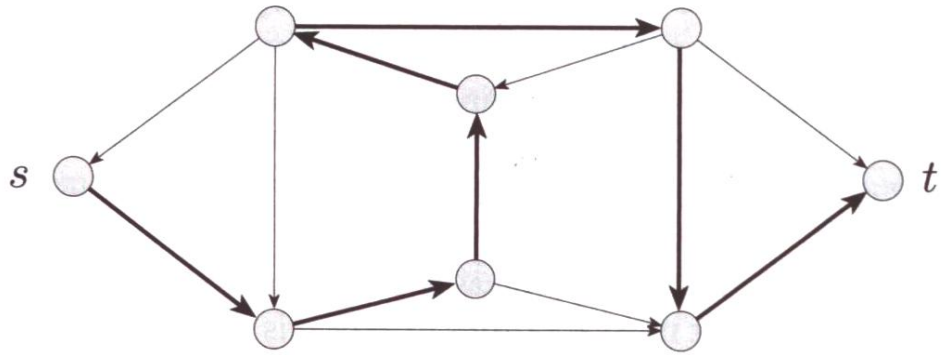
All problems except *independent set* and *competitive facility location* that were briefly mentioned at the end of Lecture Notes 1, considered so far in previous classes are in **P**

The class **P**: Why important and 'good'?

- While time $\Theta(n^{100})$ can reasonable be considered as intractable, there are few practical problems that require time of such degree polynomial.
- The polynomial time computable problems encountered in practice typically require much less time.
- Experience has shown that once a polynomial time algorithm for a problem is algorithm is discovered, more efficient algorithms often follow.
- Relationship between Turing Machines (abstract model of computations) and Random Access Machines (abstract model of executable programs) is polynomial.
- Relationship between Random Access Machines and programs in high level programming languages as Java, Haskell, etc. is linear.
- Relationships between all known models of computations (Turing Machines, Post Systems, Recursive Functions, etc.) are polynomial.

Verifiers

- A **Hamiltonian path** in a directed graph G is a path that goes through each node once.



- **Problem:** Does a directed graph contains a Hamiltonian path connecting two specified nodes?
- Exponential algorithm is easy, check all cases.
- Polynomial algorithm is not found.
- However, for a given path we can *verify* (in $O(n)$ time) if it is Hamiltonian!

Verifiers

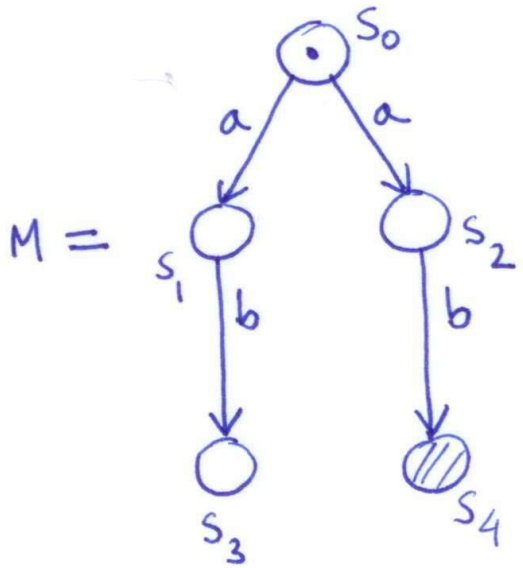
- "Does G **have** a Hamiltonian path from s to t?" - is polynomially **verifiable**.
- "Does G **have not** a Hamiltonian path from s to t?" - is **not** polynomially **verifiable**.

Definition

A **verifier** is an algorithm that can verify if a given instance is a solution or not.

Angelic vs Demonic Semantics

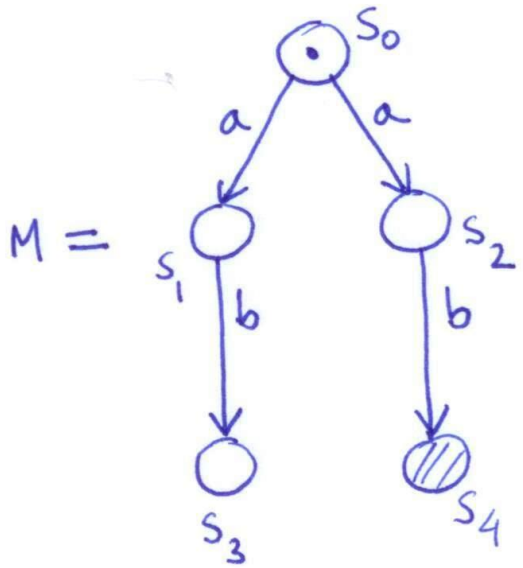
- Consider the following automaton:



- Which is true? $ab \in L(M)$ or $ab \notin L(M)$?

Angelic vs Demonic Semantics

- Consider the following automaton:

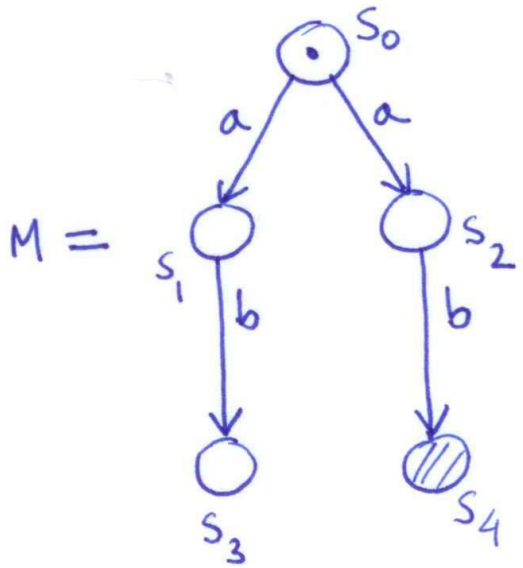


- Which is true? $ab \in L(M)$ or $ab \notin L(M)$?

  Usually it is assumed that $ab \in L(M)$. It is called **angelic semantics**.

Angelic vs Demonic Semantics

- Consider the following automaton:



- Which is true? $ab \in L(M)$ or $ab \notin L(M)$?

  Usually it is assumed that $ab \in L(M)$. It is called **angelic semantics**.

Angelic vs Demonic Semantics

- **Angelic**: At each state an *angel* will tell you where to go, so if there is a good choice you will make it. The only bad case is when all choices are bad.
- **Demonic**: At each state a *demon* will tell you where to go, so if there is a bad choice you will make it. The only good case is when all choices are good.
- The 'angelic' approach is the classical one, used in classical complexity theory. The 'demonic' approach is about 25 years old, used often in fault tolerant systems.

The class **NP**

Definition (with verifiers)

The class **NP** (from *Nondeterministic Polynomial*) consists of those problems that are **verifiable** in polynomial time. More specifically, they are problems that can be *verified* in $O(n^k)$ for some constant k. where n is the size of the input to the problem.

Hamiltonian Path from page 5 is such a problem!

Definition (with *nondeterministic* algorithms)

The class **NP** (from *Nondeterministic Polynomial*) consists of those problems that are *solvable* in polynomial time by *nondeterministic* algorithms. More specifically, they are problems that can be solved in $O(n^k)$ for some constant k. where n is the size of the input to the problem, by *nondeterministic* algorithms.

- The idea of *Nondeterministic Algorithms* is a simple consequence of *angelic* semantics.

Verifiers vs Nondeterministic Algorithms

- Nondeterministic algorithm (or nondeterministic Turing machine), if a solution exists, chooses the proper path to follow. Angelic semantics allows it.
- Every nondeterministic algorithm can be simulated by deterministic one (deterministic Turing machine), we just have to simulate all choices in an appropriate manner. If a solution exists, we will find it, but it may take at least exponential number of steps (all cases).

Proposition

Polynomial verifier $\iff$ *Polynomial Nondeterministic Algorithm.*

Proof.

($\Rightarrow$) Assume we have a polynomial verifier. If a solution does exist, we chose a proper choice and the apply verifier.
($\Leftarrow$) A part of the algorithm that has been used after proper choice is a verifier. $\square$

P vs NP and NP-completeness

- Clearly $\mathbf{P} \subseteq \mathbf{NP}$, since every (deterministic) algorithm is also nondeterministic algorithm.
- The problem if $\mathbf{P} = \mathbf{NP}$ or not, is an open million US dollars question (one of *millennium problems*).
- Informally, a problem is **NP**-complete, if it is in **NP** and it is "hard" as any problem in **NP**. If $\mathbf{P} \neq \mathbf{NP}$, **NP**-complete problems do not have polynomial solutions.
- For every problem in **NP** we have a (deterministic) algorithm, just apply verifier to all cases, but its complexity is at least exponential.
- We will show that if any **NP**-complete problem has a polynomial solution, then $\mathbf{P} = \mathbf{NP}$ (Cook-Levin Theorem)
- In practice, for **NP**-complete problems we are looking for approximate or good on average algorithms.

Polynomial-time reductions

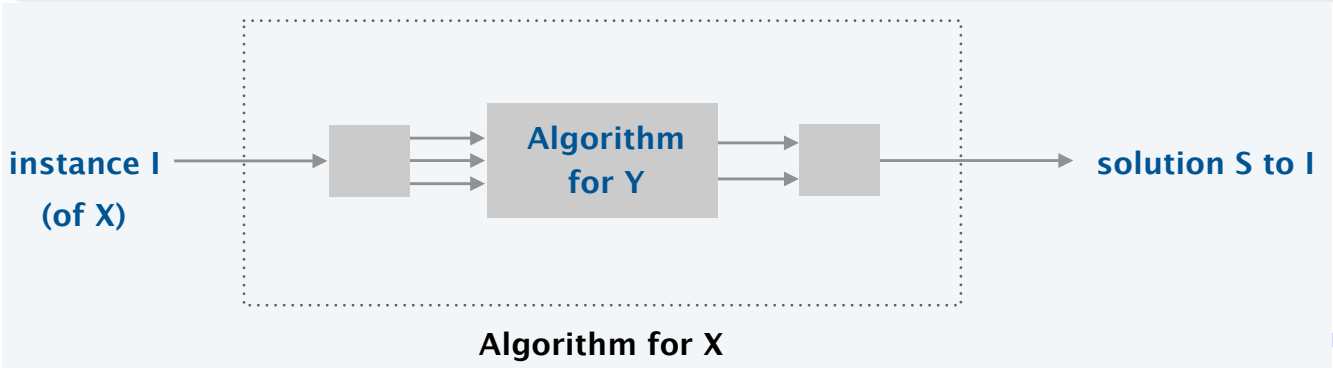
Desiderata. Suppose we could solve X in polynomial-time.
What else could we solve in polynomial time?

Definition (Reduction)

Let X and Y be two problems and assume that we already have a polynomial time solution to Y. Suppose that we have a procedure that transforms any instance α of X into some instance of β with the following characteristics:

1. The transformation takes polynomial time.
2. The answers are the same. that is, the answer for α is "yes" if and only if the answer to β is also "yes".

We call such a procedure a **polynomial-time reduction** algorithm.



Polynomial-time reductions

- Notation. $X \leq_P Y$ (X **is reduced to** Y).
- Note. We pay time for transformation.
- Caveat. Don't mistake $X \leq_p Y$ with $Y \leq_P X$.

Polynomial-time reductions

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

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

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

Bottom line. Reductions classify problems according to relative difficulty.

Polynomial-time reductions: examples

- We have already used reduction in this course:
  - Problem 4 (*CluNet*) from Assignment 1 can be reduced to *Stable marriage*, i.e. *CluNet* $\leq_P$ *Stable marriage*
  - *Bipartite Matching* has been reduced to *Max Flow*, i.e. *Bipartite Matching* $\leq_P$ *Max Flow*
  - *"Toy" Airline Scheduling* has been reduced to *Max Flow*, i.e. *"Toy" Airline Scheduling* $\leq_P$ *Max Flow*

  In all cases above transformation (cost of reduction) was linear.
- We will consider the following well known problems:
  - Independent set
  - Vertex cover
  - Set cover
  - 3-satisfiability
- We will show that:

3-satisfiability $\leq_P$ Independent set $\equiv_P$ Vertex cover $\leq_P$ Set cover

Polynomial-time reductions and intractability

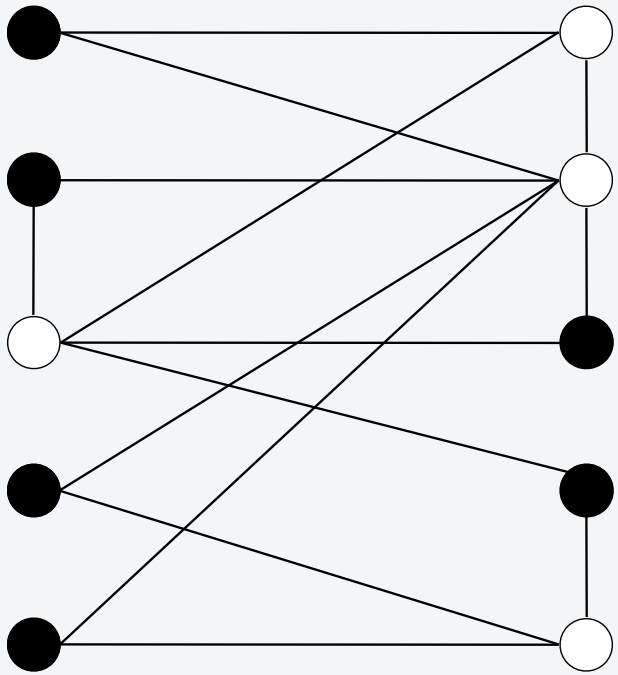
- **NP**-completeness is about showing how hard a problem is rather than how easy it is.
- We will use polynomial time reductions in the opposite way to show that a problem is **NP**-complete.
- If $X \leq_P Y$ and X cannot be solved in polynomial time, then Y cannot be solved in polynomial time.
- For **NP**-completeness, we cannot assume that there is absolutely no polynomial time algorithm for problem X.
- The proof methodology is similar however, in that we prove that problem Y is **NP**-complete on the assumption that problem X is also **NP**-complete.
- Hence, we will analyze some non trivial polynomial-time reductions in details.

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 ≥ 6?

Ex. Is there an independent set of size ≥ 7?



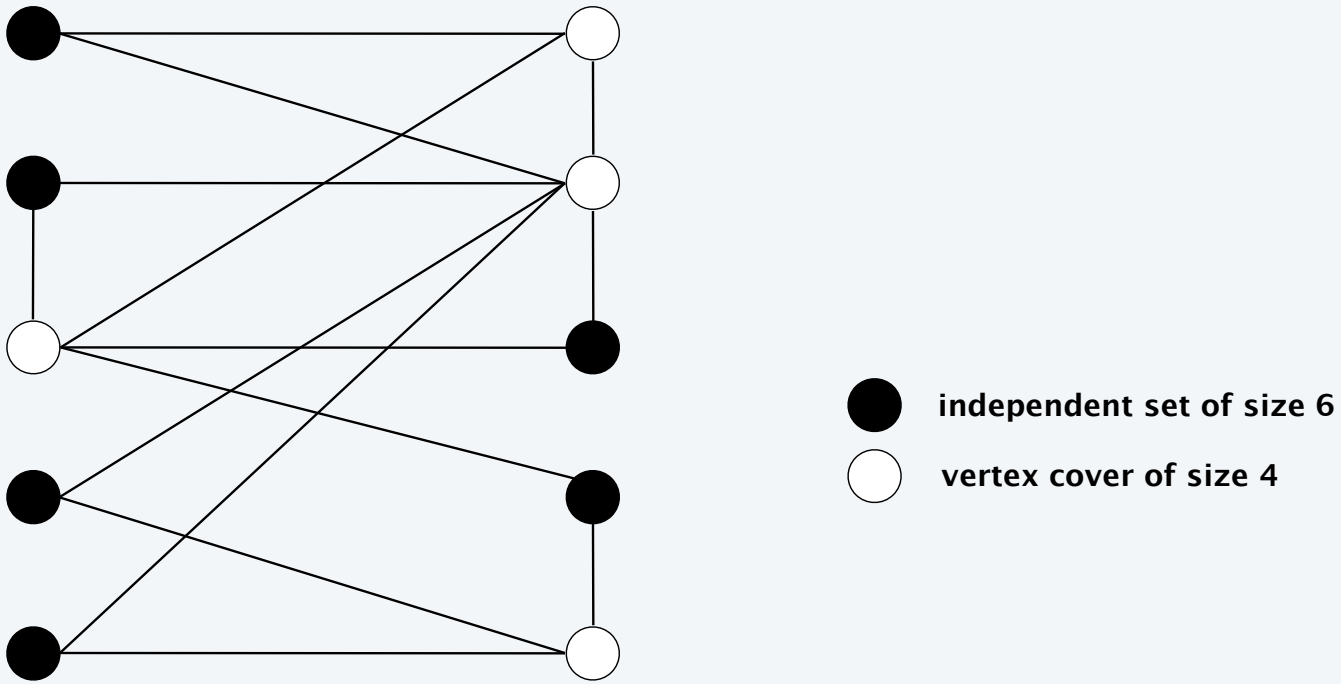
independent set of size 6

Vertex cover

VERTEX-COVER. Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?

Ex. Is there a vertex cover of size ≤ 4?

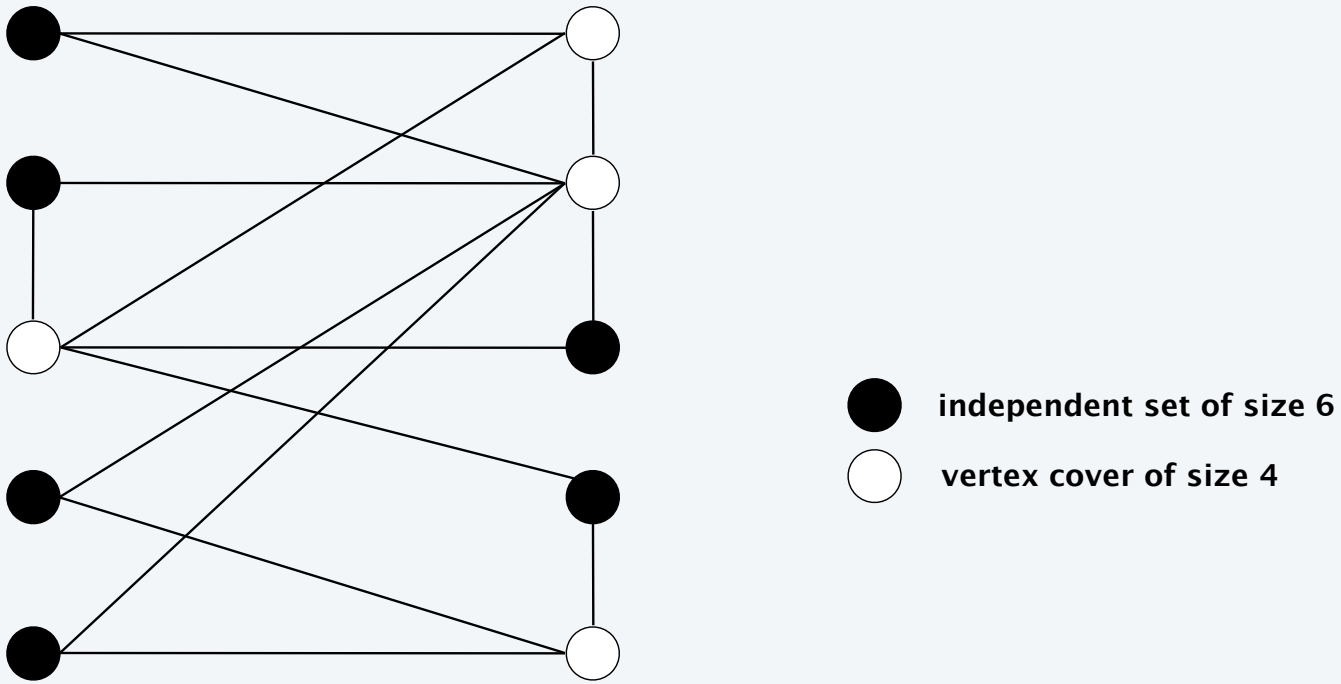
Ex. Is there a vertex cover of size ≤ 3?



Vertex cover and independent set reduce to one another

Theorem. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.

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



Vertex cover and independent set reduce to one another

Theorem. VERTEX-COVER $\equiv_P$ 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 $u \notin S$ 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. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.

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

Set cover

SET-COVER. Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k, does there exist a collection of $\leq k$ of these sets whose union is equal to U?

Sample application.

- m available pieces of software.
- Set U of n capabilities that we would like our system to have.
- The i^{th} piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all n capabilities using fewest pieces of software.

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

$S_1 = \{ 3, 7 \}$ $S_4 = \{ 2, 4 \}$

$S_2 = \{ 3, 4, 5, 6 \}$ $S_5 = \{ 5 \}$

$S_3 = \{ 1 \}$ $S_6 = \{ 1, 2, 6, 7 \}$

$k = 2$

a set cover instance

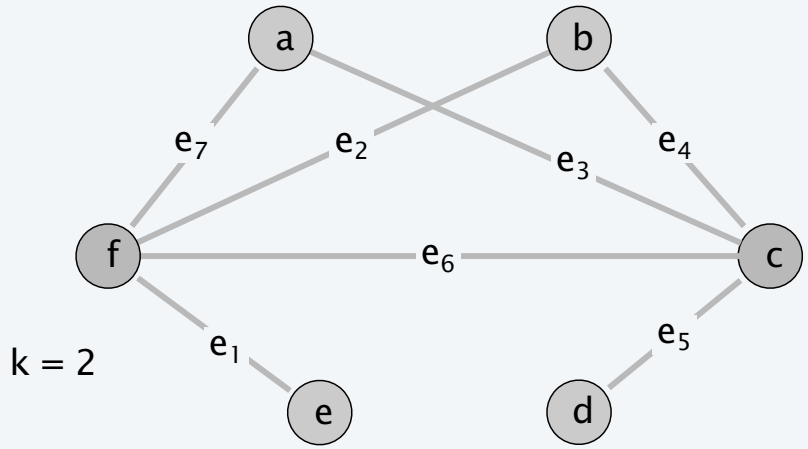
Vertex cover reduces to set cover

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

Pf. Given a VERTEX-COVER instance $G = (V, E)$, we construct a 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 set for each node $v \in V$: $S_v = \{e \in E : e \text{ incident to } v\}$.



vertex cover instance
(k = 2)

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

$S_a = \{3, 7\}$ $S_b = \{2, 4\}$

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

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

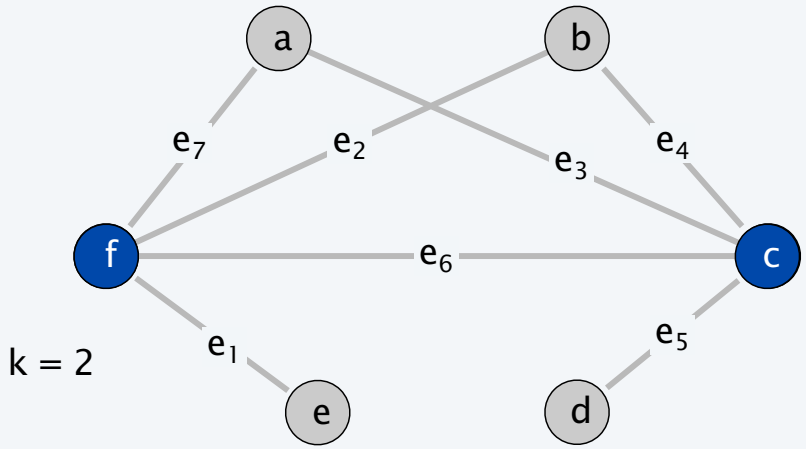
set cover instance
(k = 2)

Vertex cover reduces to set cover

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

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

- Then $Y = \{ S_v : v \in X \}$ is a set cover of size k. ▪



vertex cover instance
(k = 2)

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

$S_a = \{ 3, 7 \}$ $S_b = \{ 2, 4 \}$

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

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

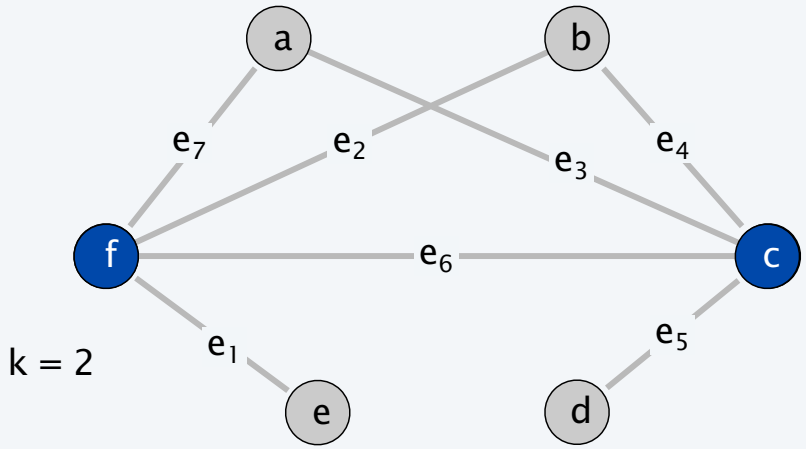
set cover instance
(k = 2)

Vertex cover reduces to set cover

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

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

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



vertex cover instance
(k = 2)

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

$S_a = \{ 3, 7 \}$ $S_b = \{ 2, 4 \}$

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

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

set cover instance
(k = 2)

Satisfiability

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

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

Conjunctive normal form. A propositional formula Φ that is the conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT. Given CNF formula Φ, 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).

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

yes instance: x_1 = true, x_2 = true, x_3 = false, x_4 = false

Key application. Electronic design automation (EDA).

3-satisfiability reduces to independent set

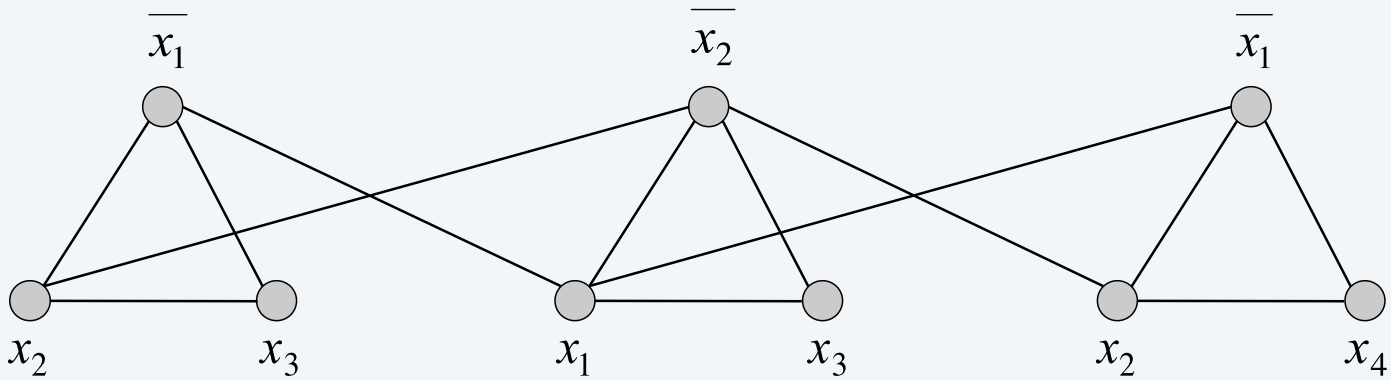
Theorem. 3-SAT $\leq_P$ INDEPENDENT-SET.

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

G



k = 3

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

3-satisfiability reduces to independent set

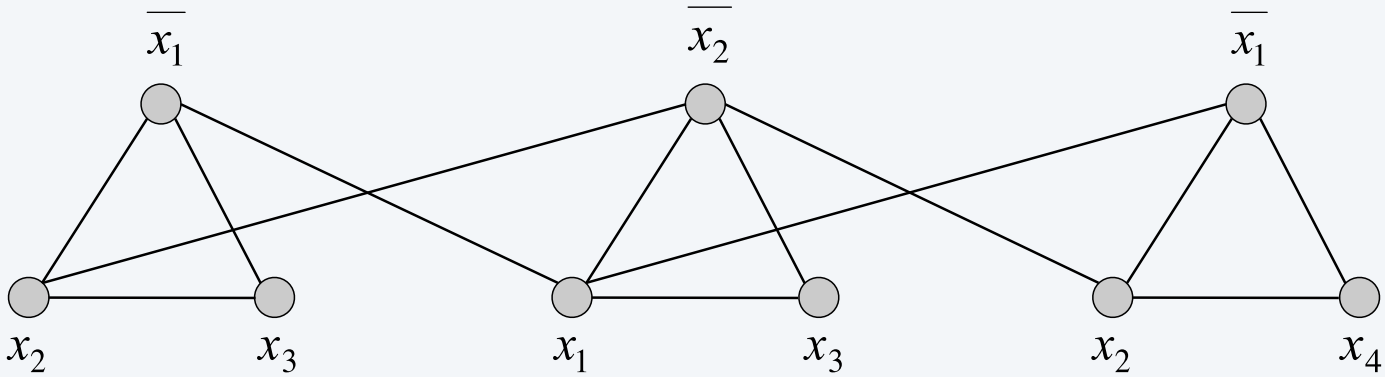
Lemma. G contains independent set of size $k = |\Phi|$ iff Φ is satisfiable.

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

- S must contain exactly one node in each triangle.
- Set these literals to *true* (and remaining variables consistently).
- Truth assignment is consistent and all clauses are satisfied.

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

G



k = 3

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

Review

Basic reduction strategies.

- Simple equivalence: INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
- Special case to general case: VERTEX-COVER $\leq_P$ SET-COVER.
- Encoding with gadgets: 3-SAT $\leq_P$ INDEPENDENT-SET.

Transitivity. If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.

Pf idea. Compose the two algorithms.

Ex. 3-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ SET-COVER.

P, NP, and EXP

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

Claim

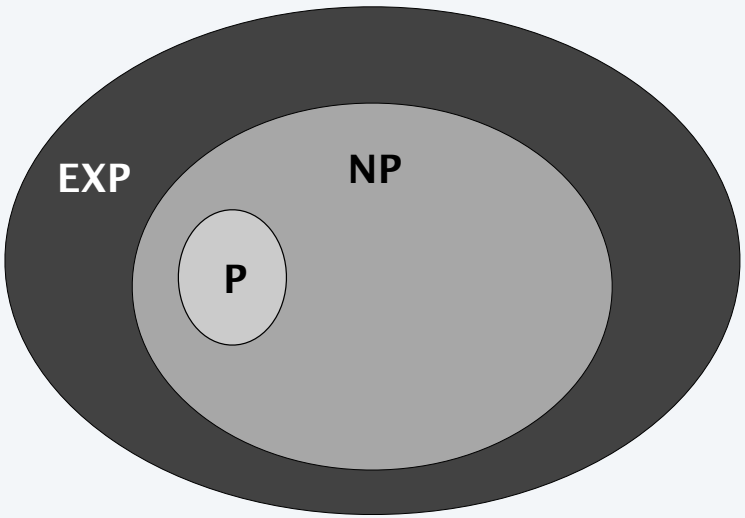
$\mathbf{P} \subseteq \mathbf{NP} \subsetneq \mathbf{EXP}$

Proof.

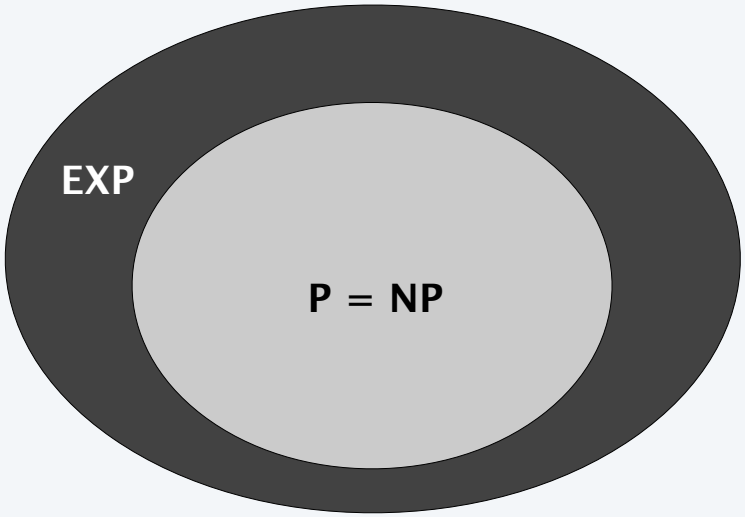
Clearly $\mathbf{P} \subseteq \mathbf{NP}$, since every (deterministic) algorithm is also nondeterministic algorithm. Since by considering all possible choices we can simulate every nondeterministic algorithm by deterministic one, but the complexity is exponential so $\mathbf{NP} \subseteq \mathbf{EXP}$. The property $\mathbf{NP} \neq \mathbf{EXP}$ follows from the observation that not every problem from **EXP** has a polynomial verifier. For example "Does G **have not** a Hamiltonian path from s to t?" clearly is in **EXP**, but it does not have a polynomial verifier. □

P vs NP again

Does **P** = **NP**?



If P ≠ NP



If P = NP

NP-completeness

Definition

A problem Y is **NP**-complete if it satisfies the following two conditions:

1. $Y \in \mathbf{NP}$
2. every $X \in \mathbf{NP}$ is *polynomially reducible* to Y, i.e. $X \leq_P Y$.

Let **NPC** denote the class of all **NP**-complete problems.

Theorem

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

Proof.

($\Leftarrow$) If $\mathbf{P} = \mathbf{NP}$, then $Y \in \mathbf{P}$ because $Y \in \mathbf{NP}$.
($\Rightarrow$) Suppose $Y \in \mathbf{P}$.

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

□

NP-completeness: basic tool

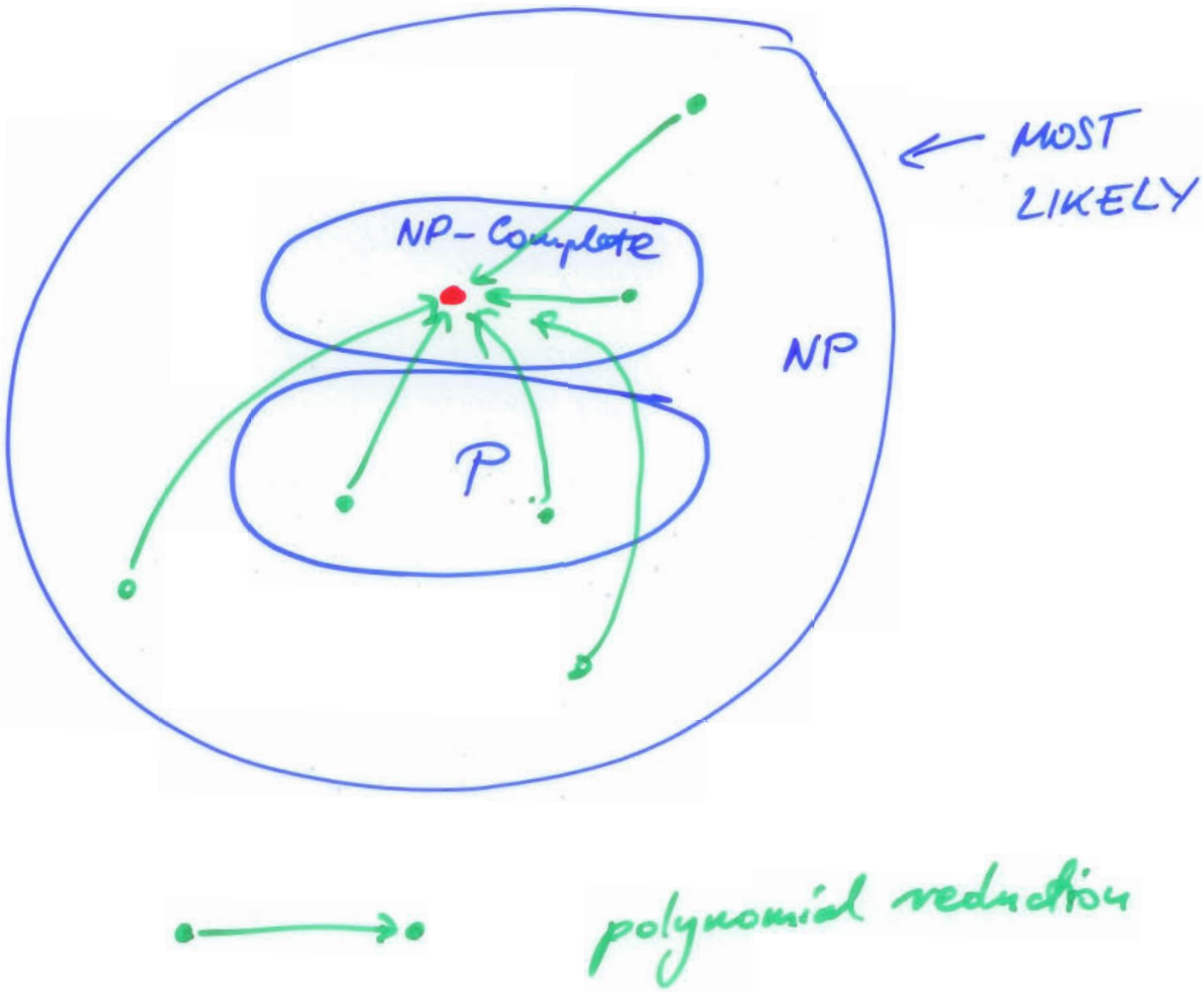
Theorem

If X is **NP***-complete, $X \in$* **NP***, and $X \leq_P Y$, then Y is* **NP***-complete.*

Proof.

Let A be any problem from **NP**, i.e. $A \in$ **NP**. Since X is **NP**-complete, then $A \leq_P X$. Hence we have $A \leq_P X \leq_P Y$. This is true for any $A \in$ **NP**, so by (2) of the definition of **NP**-completeness, Y is also **NP**-complete. □

Fundamental question. Do there exist "natural" **NP**-complete problems? If not, **NPC** is empty!



- If $P \cap (NP\text{-complete}) = \emptyset$ then $P = NP$.
- Does such 'red element' exist?

SAT Problem: Idea

- An example of a *Boolean formula*:

$$\Phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z}),$$

where $\overline{x}$ means $\neg x$, so $x = 0 \iff \overline{x} = 1$ and $x = 1 \iff \overline{x} = 0$.

Definition

A Boolean formula Φ is **satisfiable** if so some assignment of 0's and 1's to the variables makes the formula to eveluate to 1.

- $(\overline{x} \wedge y) \vee (x \wedge \overline{z}) = 1$ if $x = 0, y = 1, z = 0$.
  This formula is satisfiable.
- $(\overline{x} \wedge y) \wedge (x \wedge \overline{z})$ is never 1, always 0.
  This formula is not satisfiable.

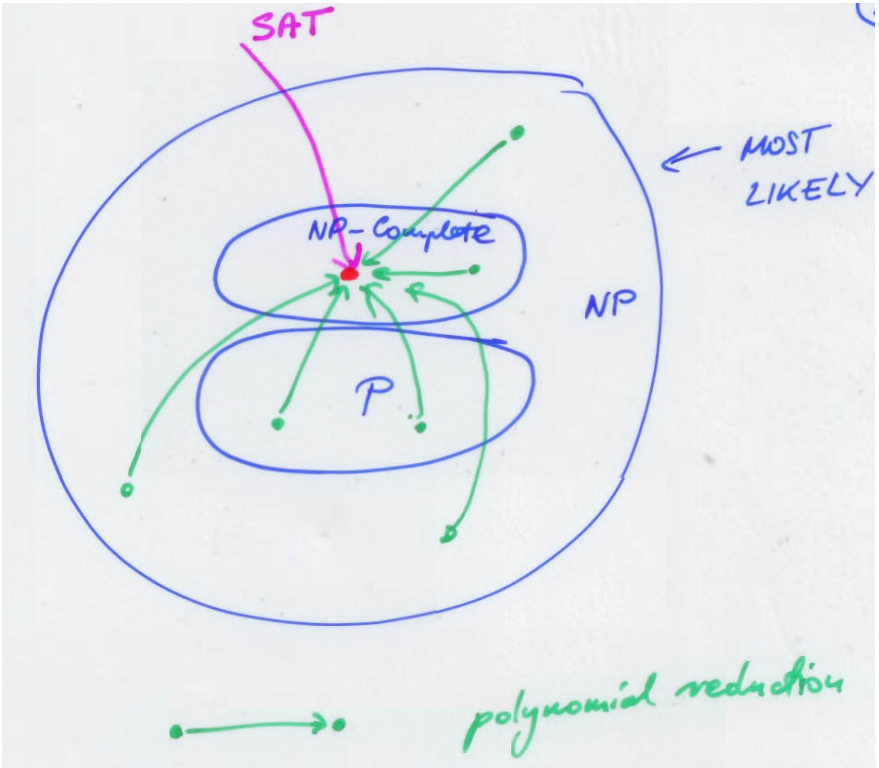
SAT Problem and Cook-Levin Theorem

Definition

The **satisfiability problem** (SAT) is to test whether Boolean formula is satisfiable.

Theorem (Cook-Levin)

SAT is **NP***-complete.*



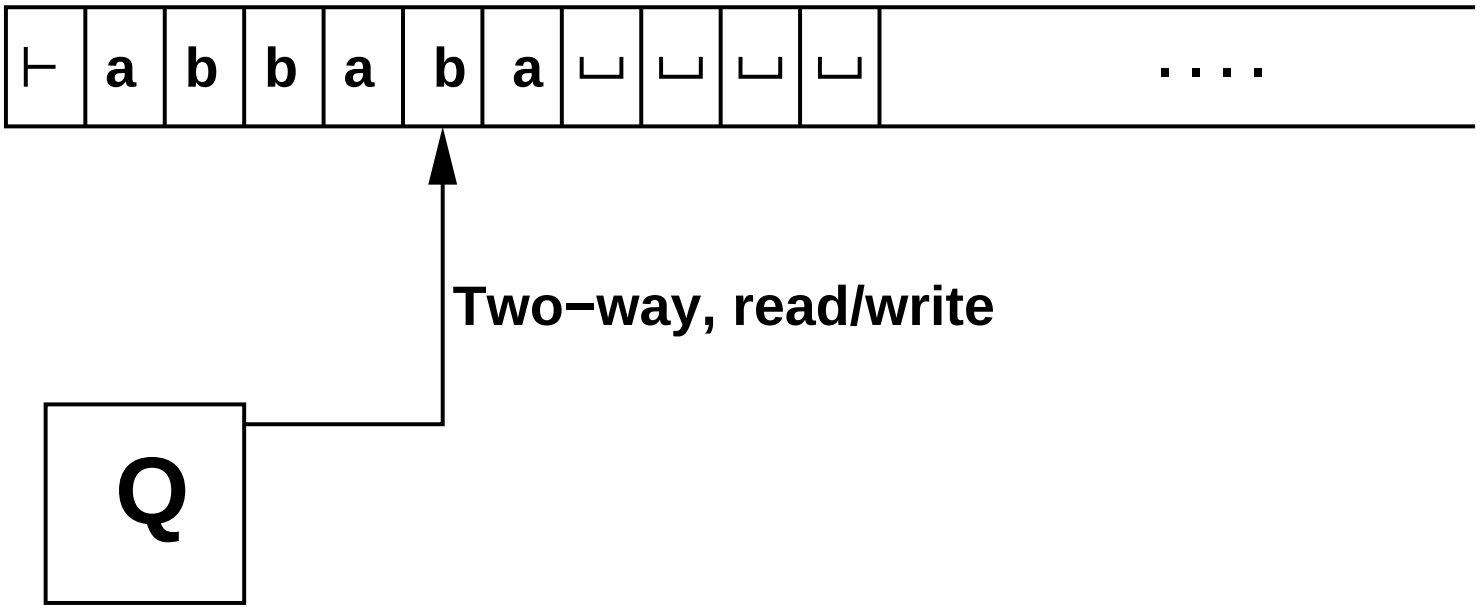
Cook-Levin Theorem: how to prove it?

Theorem (Cook-Levin)

SAT is **NP***-complete.*

- How such theorem can be proven? Especially how to prove that any problem in **NP** is polynomial time reducible to *SAT*. There is infinite number of problems and most of them has not even be formulated yet. How should we think?
- - Every finite entity can be described as a string.
  - Every book is a string.
  - Jorge Luis Borges, *Babel Library*, **1941**
    Each book consists of 409 pages; each page 40 lines; each line about 80 letters of black colour..."
    "There is no two identical books in the library"
    "Library is total... It describes everything and in all languages. Everything: detailed history of the past..."
  - Every book in Borges library is a string.
    Let L = all books in Borges library, the language L is finite! Is our knowledge finite?
  - Maybe encoding is a solution? Each program is a rewriting system, it eats sequences of bits (characters) and produces sequences of bits (characters).

Informal Description of Turing Machines



- Turing Machine (TM) can both write on the tape and read from it.
- The read-write head can move both to the left and to the right.
- The special states for rejecting and accepting take effect immediately.
- There are many slightly different definitions of Turing Machines.

Turing Machines

Turing Machines:

- *recognize* languages, i.e. they can tell if $x \in L$, for some language L.
- *decide* languages, i.e. they can tell if $x \in L$ or $x \notin L$, for some language L
- *loop*, i.e. there is x such that a Turing machine will run forever trying to analyze x.
- Turing Machine *halts* is it does not loop.
- In this course our Turing machines *decide* languages.

Time Complexity of Turing Machines

- Let M be a *deterministic* Turing machine that *halts on all inputs*, i.e. does not loop.

Definition

The **running time** or **time complexity** of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the *maximum* number of steps that M uses on any input of length n, and $\mathbb{N} = \{1, 2, \ldots\}$.

Important.

- Turing machines are very inefficient but their relationship with RAM programs (i.e. normal algorithms) is polynomial.
- For Turing machines we are always interested in if time complexity is polynomial or not, not in efficient practical complexity.

Codes

- All Turing machines may be *coded* as strings.
- All configurations of a given Turing machines can be coded as strings. We just need to code the current state, current state position, perhaps current tape number, etc.
- All algorithms can be coded as strings.
- All instances can be coded as strings.
- All problems can be coded as strings.
- Sets of codes are *formal languages* so theory of Turing Machines can be use with full power.
- Algorithm does not exist if its code does not exists!

Languages Defined by Problems

- For any concept C (i.e. graph, algorithm, instance of a solution, Turing machine, etc.), $\langle C \rangle$ denotes a **code of** C.
- Details of coding are not important, but we assume its existence, and one coding technique for a given concept.
- $\langle C \rangle$ is a **string of symbols**, i.e. $\langle C \rangle \in \Sigma^*$ for some finite Σ.
  $A = \{\langle G \rangle \mid G \text{ is a connected undirected graph } \}$
  $B = \{\langle G_{Ham}, (s,t) \rangle \mid G_{Ham} \text{ has a Hamiltonian path from } s \text{ to } t\}$
- Both A and B are **languages**, i.e. $A \subseteq \Sigma_A^*$, $B \subseteq \Sigma_B^*$, where Σ_A, Σ_B are alphabets of codings.

Codes and Problems

Claim

$X \in \mathbf{P} \iff \langle X \rangle \in \mathbf{P}$

Proof.

It requires more knowledge about Turing Machines, it is beyond the scope of this course. □

- The above result allows using Turing Machines to prove the Cook-Levin Theorem.

Cook-Levin Theorem

Theorem (Cook-Levin)

SAT is **NP***-complete.*

Proof. (idea).

Recall that a problem Y is **NP**-complete if it satisfies the following two conditions:

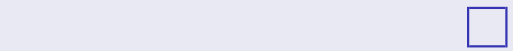
1. $Y \in \mathbf{NP}$
2. every $X \in \mathbf{NP}$ is *polynomially reducible* to Y, i.e. $X \leq_P Y$.

The proof consists of two parts.

1. $SAT \in \mathbf{NP}$. A nondeterministic polynomial time algorithm (angelic semantics) can guess an assignment to agiven formula Φ and accept if the assignment satisfies Φ.
2. The hard part of the proof is showing that any problem in **NP** is polynomial time reducible to SAT.

Proof. (continuation).

We construct a polynomial time reduction for each language L in **NP** to *SAT*. The reduction for L takes a string w and produces a Boolean formula Φ that simulates the nondeterministic polynomial (NP) Turing machine for L on input w. If the machine accepts, ϕ has a satisfying assignment that correspond to the accepting computation. If the machine doesn't accept, no assignment satisfies Φ. Therefore, $w \in L$ if and only if Φ is satisfiable.
The proof idea explores the fact that formula is a *propositional calculus formula*, so it can express $w \in L$ for any L!
A slightly different proof, in details, is given in the textbook.



Why **NP**-completeness?

- For most of the problems, it is usually easy to show that $B \in \mathbf{NP}$.
- In many cases we cannot find any polynomial solution, but we are unable to prove that $B \notin \mathbf{P}$ either.
- Proving that B is **NP**-complete is in most cases easier (or just possible) than $B \notin \mathbf{P}$.
- If B is **NP**-complete, it is practically considered as *non-polynomial*.
- Efficient solution must take particular properties into account.

Other *NP*-complete Problems

- SAT problem is the only problem that has been proven from the definition.
- All other problems have been proven **NP**-complete by using **polynomial reduction**.

Fact

*If X is **NP**-complete and $X \leq_P Y$, i.e. X is polynomially reducible to Y, then Y is **NP**-complete.*

- Since so far we only have SAT, the second problem X_2 must be a reduction of SAT, i.e $SAT \leq_P X_2$.

Conjunctive Normal Form (CNF)

- Conjunctive Normal Form (CNF):
  $(x \vee y) \wedge (y \vee \overline{x} \vee z) \wedge (\overline{z} \vee x) \in CNF.$
  $((x \vee y) \wedge z) \vee \overline{x} \notin CNF$

Theorem

Every Boolean formula Ψ can be polynomially transformed into Φ in CNF such that $\Psi \equiv \Phi$.

Theorem

The satisfiability problem for Boolean expressions in CNF is **NP***-complete.*

Proof.

From Theorem above! □

3-Conjunctive Normal Form (or 3-satisfiability)

- 3-Conjunctive Normal Form (3-CNF):
  $(\overline{t} \vee x \vee y) \wedge *y \vee \overline{x} \vee z) \in$ 3-CNF.
  $(x \vee y) \wedge (y \vee \overline{x} \vee z \vee t) \notin$ 3-CNF.

Theorem

SAT for 3-CNF is **NP***-complete.*

Proof.

We will polynomially reduce SAT for CNF to SAT for 3-CNF.
Let $k \geq 0$. Replace in CNF each $(x_1 \vee \ldots \vee x_k)$ by
$(x_1 \vee x_2 \vee y_1) \wedge (x_3 \vee \overline{y}_1 \vee y_2) \wedge (x_4 \vee \overline{y}_2 \vee y_3) \wedge \ldots$
$\wedge (x_{k-2} \vee \overline{y}_{k-4} \vee y_{k-3}) \wedge (x_{k-1} \vee x_k \vee \overline{y}_{k-3})$,
where $y_1, \ldots, y_{k-3}$ are new Boolean variables.
For example: $x_1 \vee x_2 \vee x_3 \vee x_4 \rightarrow (x_1 \vee x_2 \vee y_1) \wedge (x_3 \vee x_4 \vee \overline{y}_4)$.

- Transformation is polynomial.
- $x_1 \vee \ldots \vee x_k = 1 \iff \exists x_i . x_i = 1$
- If $x_i = 1$ then $y_1 = y_2 = \ldots = y_{i-2} = 1$, $y_{i-1} = y_i = \ldots = y_{k-3} = 0$ guarantees that the replacement has the value 1.

Hence SAT for CNF is reduced to SAT for 3-CNF. □

Other **NP**-complete problems

Since:

3-satisfiability $\leq_P$ Independent set $\equiv_P$ Vertex cover $\leq_P$ Set cover

and 3-satisfiability is **NP**-complete, then

- Independent set,
- Vertex cover and
- Set cover

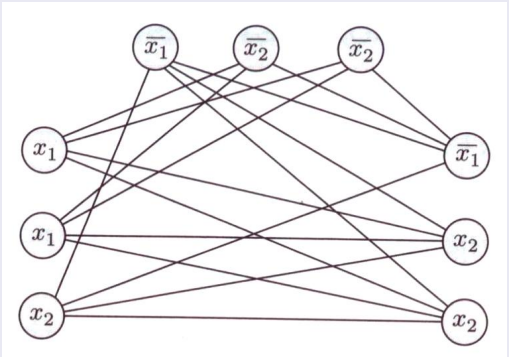
are **NP**-complete as well!

Theorem (Clique Problem)

Clique problem is **NP***-complete.*

Proof. (idea).

We will polynomially reduce SAT for 3-CNF to clique problem.



Idea of transformation:

$$\Phi = \overbrace{(x_1 \vee x_1 \vee x_2)}^{\Phi_1} \wedge \overbrace{(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_2)}^{\Phi_2} \wedge \overbrace{(\overline{x}_1 \vee x_2 \vee x_2)}^{\Phi_3}$$

$\Phi = 1$ for $x_1 = 0, x_2 = 1$, and $\Phi = \Phi_1 \wedge \Phi_2 \wedge \Phi_3$.

- No edge between nodes of Φ_i
- No edge between x and $\overline{x}$
- Transformation is polynomial.
- $\Phi = \Phi_1 \wedge \ldots \wedge \Phi_k$ is satisfiable $\iff$ the graph contains k-clique.

□

Decision problems vs 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. VERTEX-COVER $\equiv_P$ FIND-VERTEX-COVER.

Decision problems, 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.

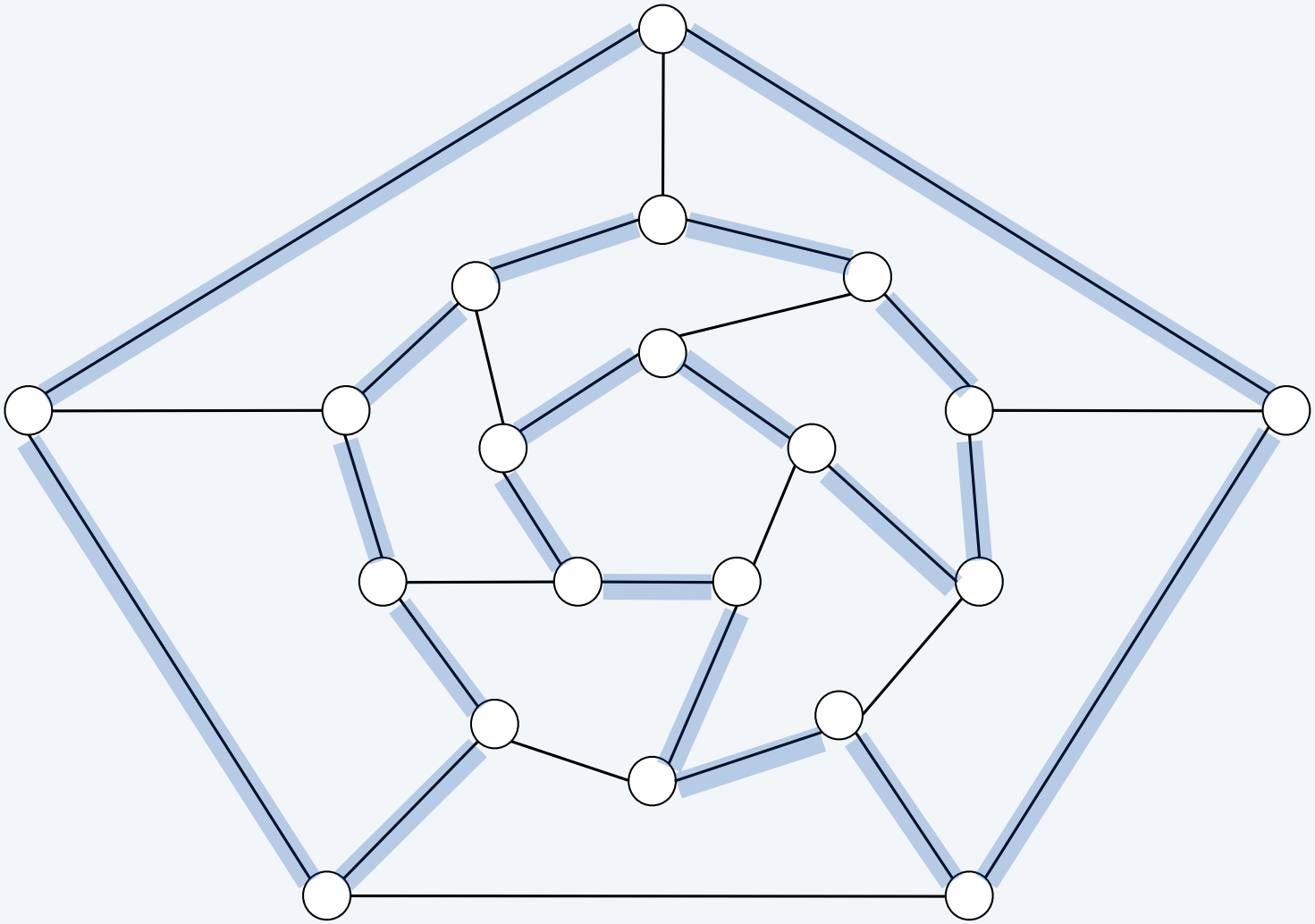
Ex. To find vertex cover of minimum size:

- (Binary) search for size k^* of min vertex cover.
- Solve corresponding search problem.

Bottom line. VERTEX-COVER $\equiv_P$ FIND-VERTEX-COVER $\equiv_P$ OPTIMAL-VERTEX-COVER.

NP-complete problems: Hamilton cycle

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



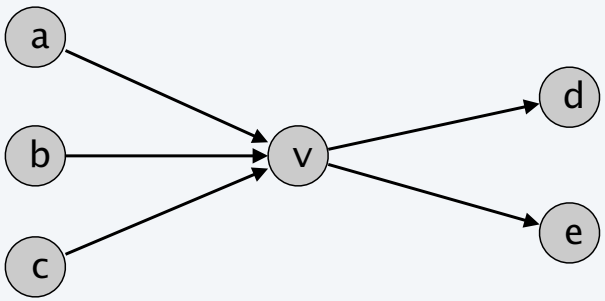
yes

NP-complete problems: directed Hamilton cycle

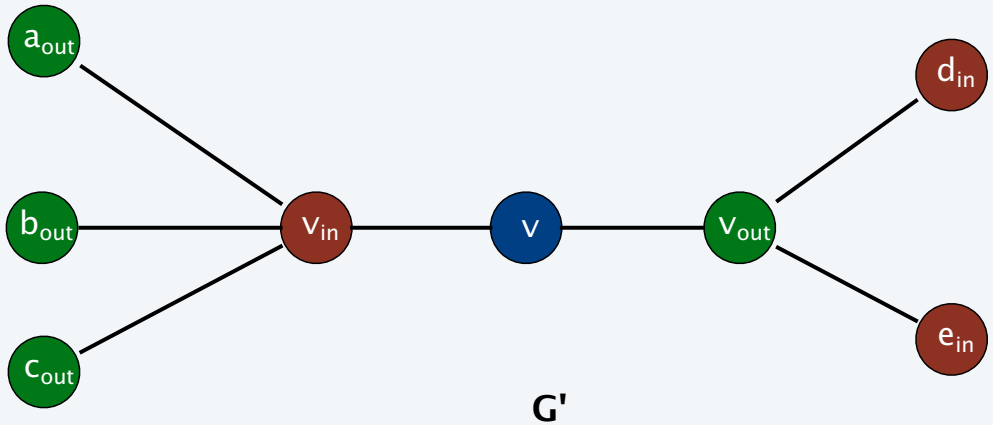
DIR-HAM-CYCLE: Given a digraph $G = (V, E)$, does there exist a simple directed cycle Γ that contains every node in V ?

Theorem. DIR-HAM-CYCLE $\leq_P$ HAM-CYCLE.

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



G



G'

NP-complete problems: directed Hamilton cycle

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

Pf. $\Rightarrow$

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

Pf. $\Leftarrow$

- Suppose G' has an undirected Hamilton cycle Γ'.
- Γ' must visit nodes in G' using one of following two orders:

  $\ldots, B, G, R, B, G, R, B, G, R, B, \ldots$

  $\ldots, B, R, G, B, R, G, B, R, G, B, \ldots$
- Blue nodes in Γ' make up directed Hamilton cycle Γ in G, or reverse of one. ▪

B = blue, G = green, R = red

Directed Hamilton Cycle and Hamilton Cycle are NP-Complete

Theorem

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

Proof.

Idea: Given an instance Φ of 3-SAT, we construct an instance of DIR-HAM-CYCLE that has a Hamilton cycle iff Φ is satisfiable. See Kleinberg-Tardos for details. □

Corollary

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

- Both directed Hamilton cycle and Hamilton cycle are NP-complete.

NPC-problems: Longest path

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

Theorem

HAM-CYCLE $\leq_P$ *LONGEST-PATH.*

Corollary

3-SAT $\leq_P$ *DIR-HAM-CYCLE* $\leq_P$ *HAM-CYCLE* $\leq_P$ *LONGEST-PATH.*

- LONGEST-PATH is NP-complete.

NPC-problems: Traveling Salesperson Problem (TSP)

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 Γ that contains every node in V ?

Theorem

HAM-CYCLE $\leq_P$ TSP.

Proof.

- Given instance $G = (V, E)$ of HAM-CYCLE, create n cities with distance function

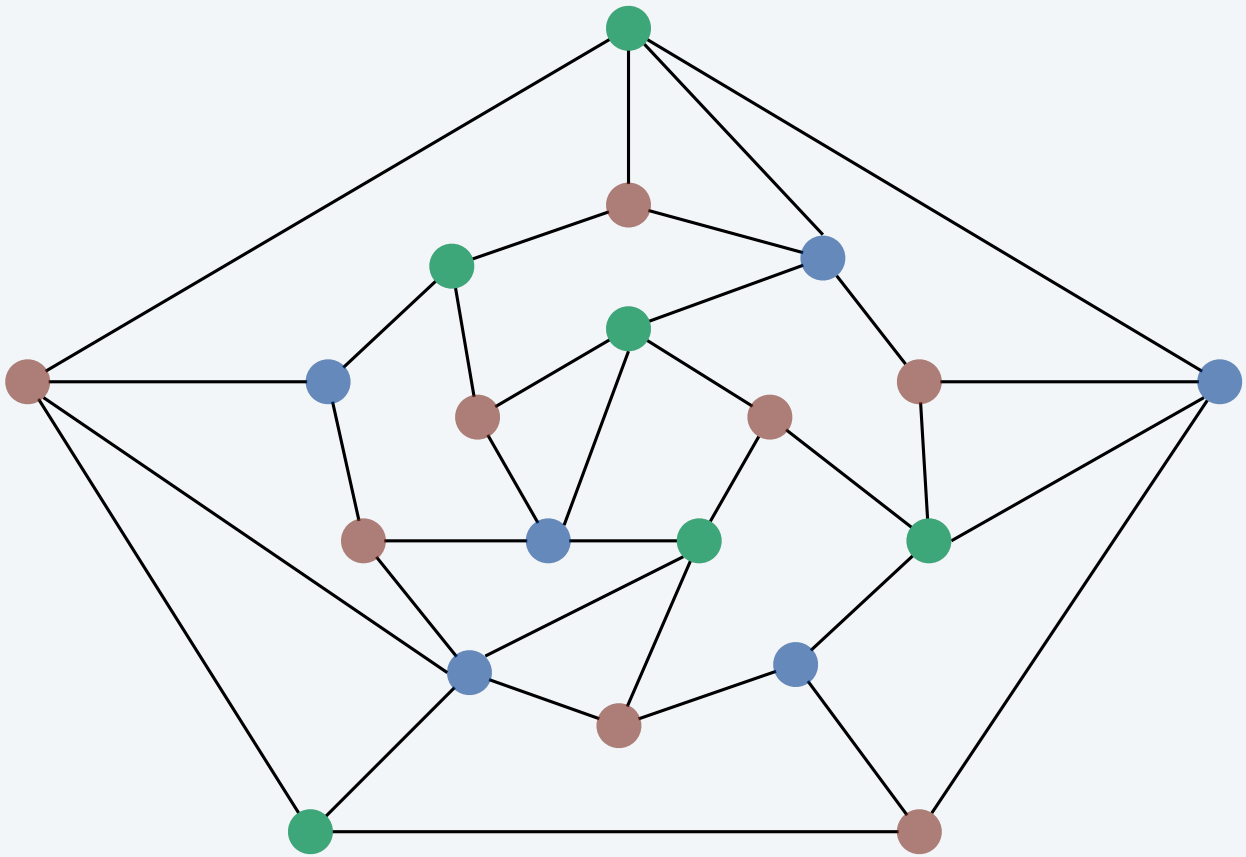
$$d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \notin E \end{cases}$$

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

- Note that we have *triangle inequality* $d(u, w) \leq d(u, v) + d(v, w)$.

NPC-problems: 3-colorability

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



yes instance

Application of 3-colorability: register allocation

- Register allocation. Assign program variables to machine register so that no more than k registers are used and no two program variables that are needed at the same time are assigned to the same register.
- Interference graph. Nodes are program variables names; edge between u and v if there exists an operation where both u and v are "live" at the same time.
- Observation. Can solve register allocation problem iff interference graph is k-colorable.
- Fact. 3-COLOR $\leq_P$ k-REGISTER-ALLOCATION for any constant $k \geq 3$.

NPC-problems: 3-colorability

Theorem

3-SAT $\leq_P$ *3-COLOR.*

Proof.

Given 3-SAT instance Φ, we construct an instance of 3-COLOR that is 3-colorable iff Φ is satisfiable. See Kleinberg-Tardos for details. □

- 3-COLOR is NP-complete.

NPC-problems: Subset sum

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

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

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

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

Theorem. 3-SAT $\leq_P$ SUBSET-SUM.

Pf. Given an instance Φ of 3-SAT, we construct an instance of SUBSET-SUM that has solution iff Φ is satisfiable.

- SUBSET-SUM is NP-complete.

NPC-problems: Partition

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

PARTITION. Given natural numbers $v_1, \ldots, v_m$, can they be partitioned into two subsets that add up to the same value $\frac{1}{2} \Sigma_i v_i$?

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.
  - $v_1 = w_1, v_2 = w_2, \ldots, v_n = w_n, \quad v_{n+1} = 2 \Sigma_i w_i - W, \quad v_{n+2} = \Sigma_i w_i + W$
- Lemma: there exists a subset that sums to W iff there exists a partition since elements v_{n+1} and v_{n+2} cannot be in the same partition. ▪

NPC-problems: Scheduling with release times

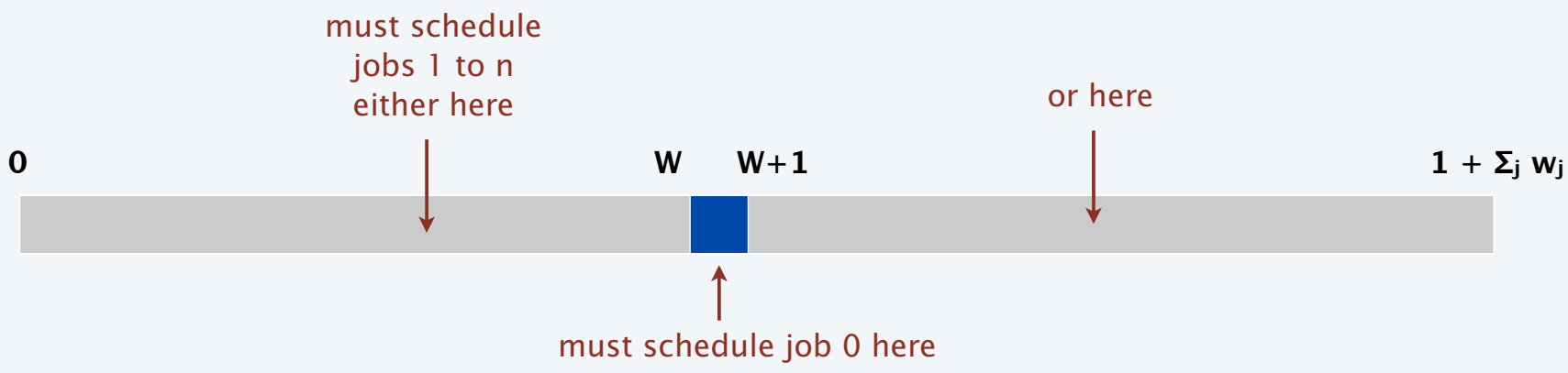
SCHEDULE. Given a set of n jobs with processing time t_j, release time r_j, and deadline d_j, is it possible to schedule all jobs on a single machine such that job j is processed with a contiguous slot of t_j time units in the interval $[r_j, d_j]$?

Theorem. SUBSET-SUM $\leq_P$ SCHEDULE.

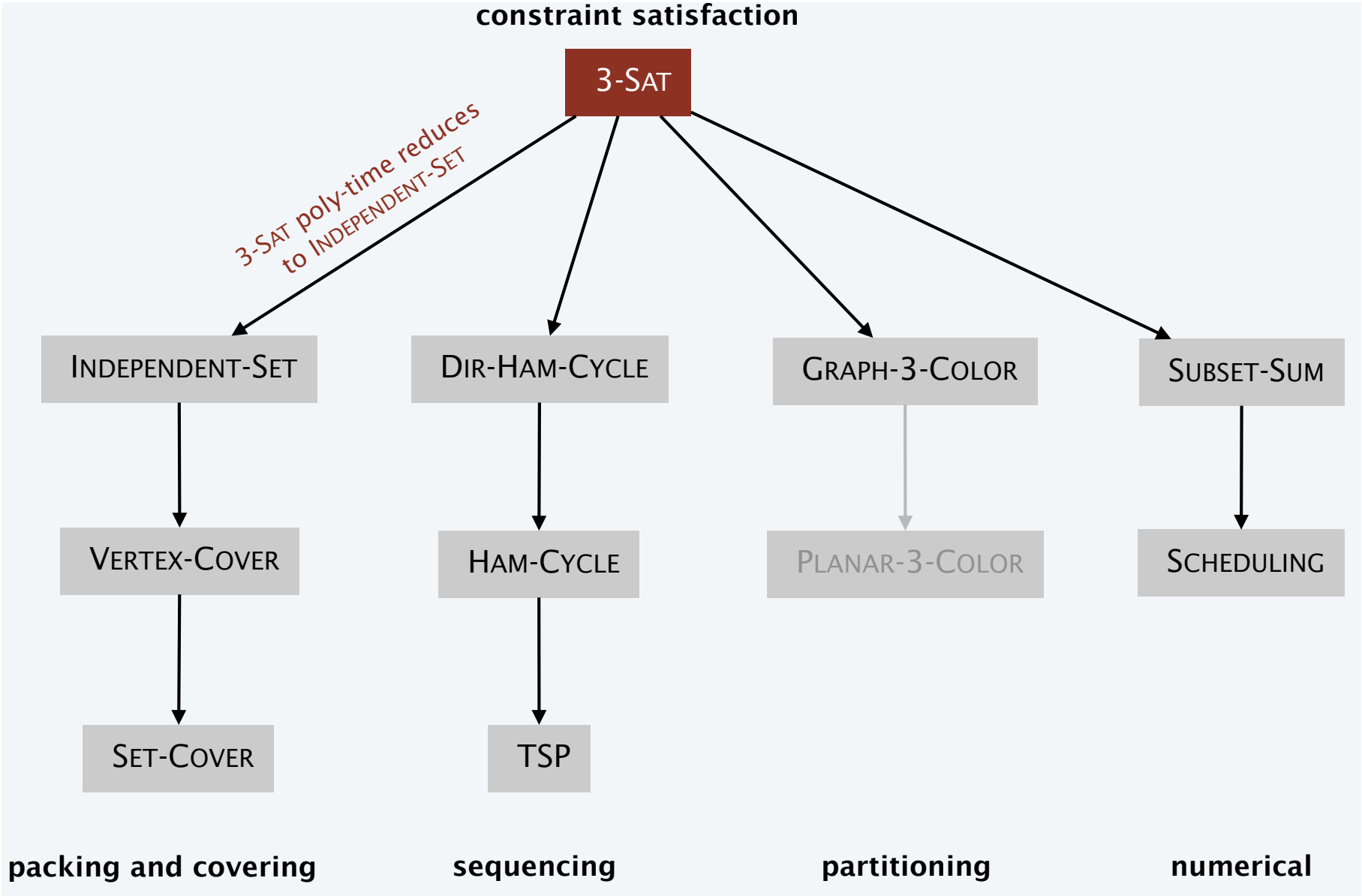
Pf. Given SUBSET-SUM instance $w_1, \ldots, w_n$ and target W, construct an instance of SCHEDULE that is feasible iff there exists a subset that sums to exactly W.

Construction.

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



NP-complete problems



- PLANAR-3-COLOR is just GRAPH-3-COLOR for planar graphs.