Design and Analysis of Algorithms

CS3230

Lecture 10

Reductions & Intractability

Warut Suksompong

Today: Reductions

- Reductions between computational problems is a fundamental idea in algorithm design

- Viewed another way, reductions also give a way to compare the **hardness** of two problems.

What is a reduction?

Consider two problems A and B. A can be solved as follows:

Input: An instance α of A

1. Convert α to an instance β of B
2. Solve β and obtain a solution
3. Based on the solution of β, obtain the solution of α

What is a reduction?

Consider two problems A and B. A can be solved as follows:

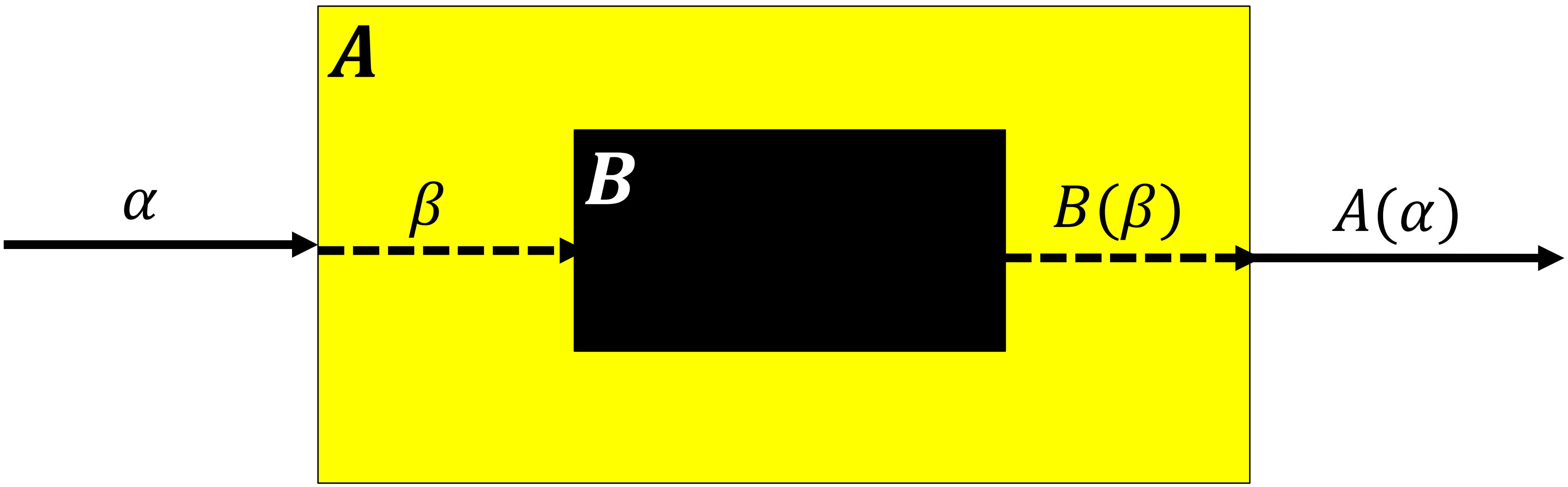
Another word for "input"

Input: An instance α of A

1. Convert α to an instance β of B
2. Solve β and obtain a solution
3. Based on the solution of β, obtain the solution of α

Then, we say ***A* reduces to *B***.

What is a reduction?



Matrix multiplication and squaring

MAT-MULTI

Input:

- Two $N \times N$ matrices A and B

Output:

- A x B

MAT-SQR

Input:

- One $N \times N$ matrix C

Output:

- C^2

Matrix multiplication and squaring

Claim: MAT-SQR reduces to MAT-MULTI.

Proof: Given input matrix C for MAT-SQR, let $A = C$ and $B = C$ be the inputs for MAT-MULTI. Clearly, $AB = C^2$.

Matrix multiplication and squaring

Claim: MAT-MULTI reduces to MAT-SQR.

Proof: Given input matrices A and B:

Construct:

$$C = \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix}$$

Call MAT-SQR to get

$$C^2 = \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix}\begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix} = \begin{bmatrix} AB & 0 \\ 0 & BA \end{bmatrix}$$

Example Problem

Consider the following two problems:

0-SUM

Input:

- An array A of length n

Output:

- $i, j \in \{1, \dots, n\}$ such that $A[i] + A[j] = 0$

T-SUM

Input:

- An array B of length n and number T

Output:

- $i, j \in \{1, \dots, n\}$ such that $B[i] + B[j] = T$

Show that T-SUM reduces to 0-SUM.

Solution

- **Careful** about which way you do the reduction!!
- Given array B, define array A such that $A[i] = B[i] - T/2$.
- If i, j satisfy $A[i] + A[j] = 0$, then $B[i] + B[j] = T$.

$\boldsymbol{p(n)}$-time Reduction

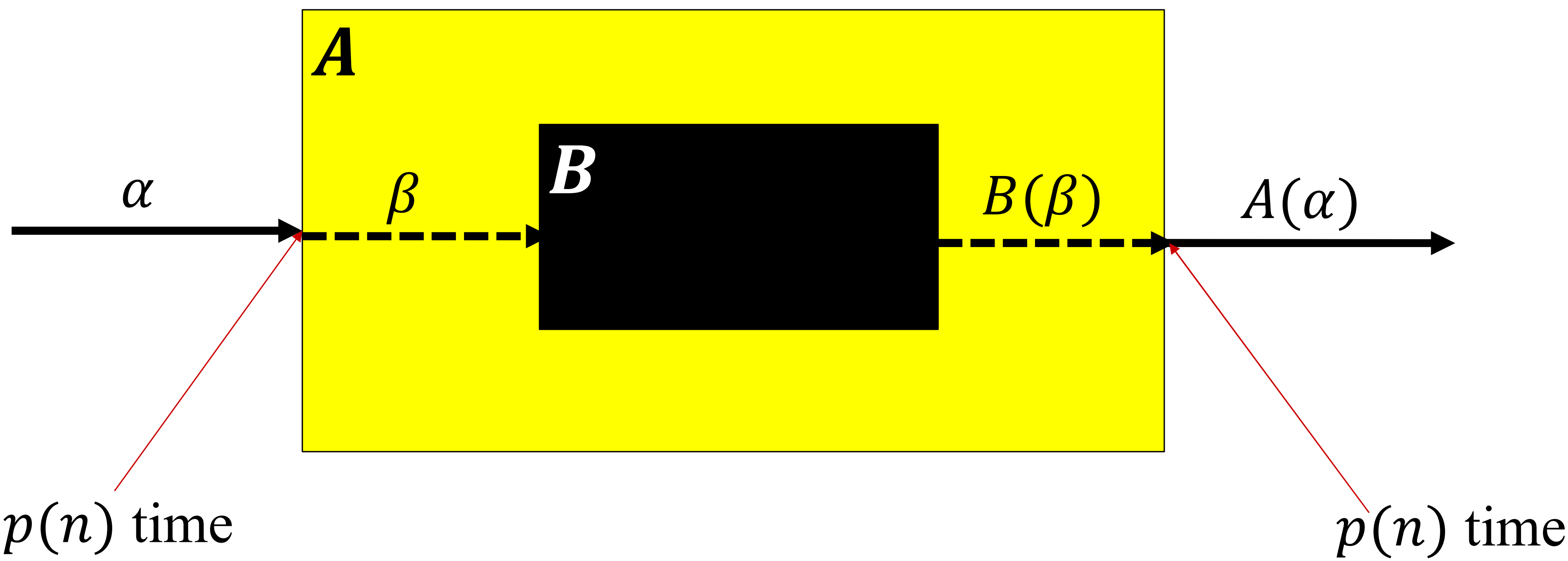
Consider two problems A and B.

If for any instance α of problem A **of size $\boldsymbol{n}$**:

- An instance β for problem B can be constructed in $p(n)$ time
- A solution to problem A for input α can be recovered from a solution to problem B for input β in time $p(n)$

we say that there is a **$\boldsymbol{p(n)}$-time reduction from $\boldsymbol{A}$ to $\boldsymbol{B}$**.

$p(n)$-time Reduction



Example Question

The reduction from MAT-MULTI to MAT-SQR is an:

(1) $O(n)$-time reduction
(2) $O(n^2)$-time reduction
(3) $O(n \log n)$-time reduction

Example Question: Solution

$O(n)$-time reduction.

Important: n is the size of the input, here N^2.

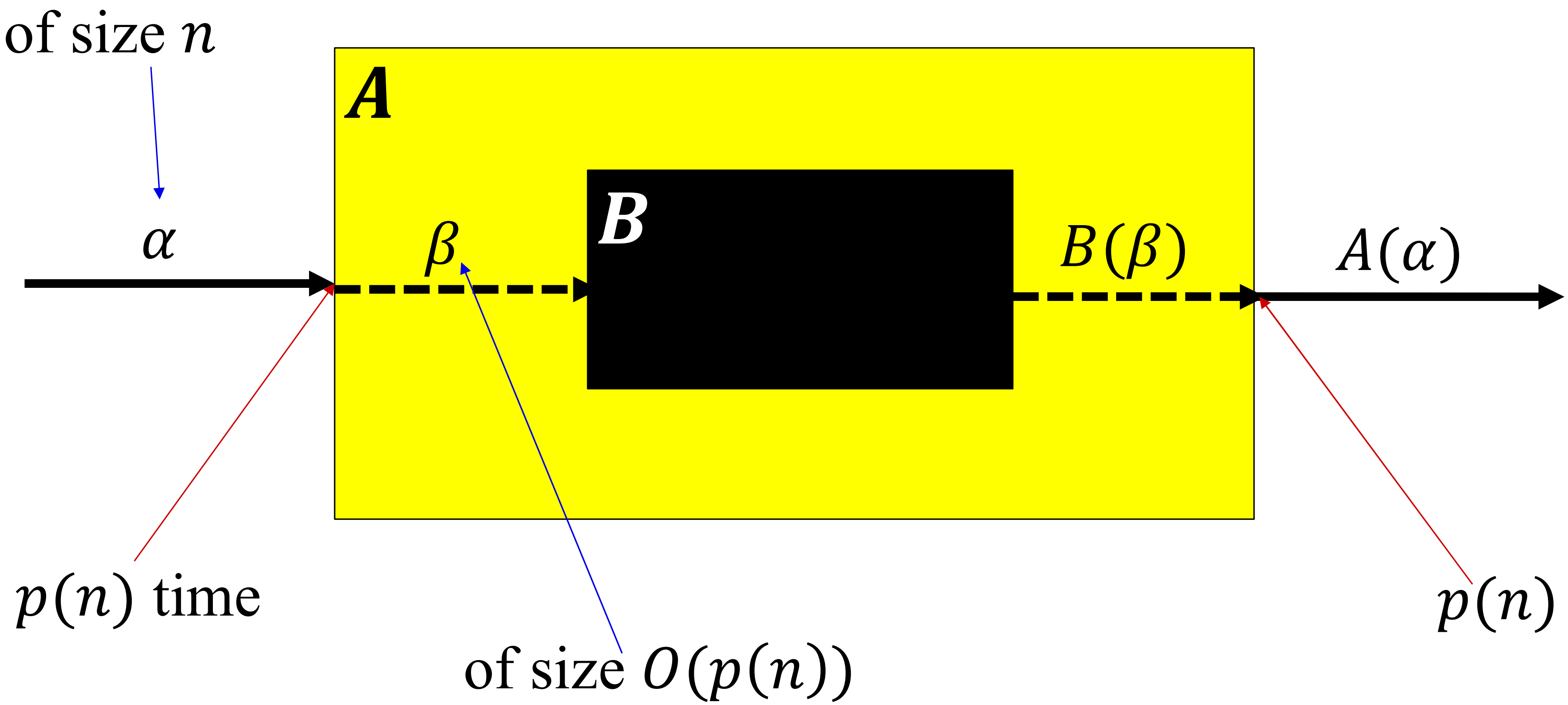
Running Time Composition

Claim: If there is a $p(n)$-time reduction from problem A to problem B, and there exists a $T(n)$-time algorithm to solve problem B on instances of size n, then there is a

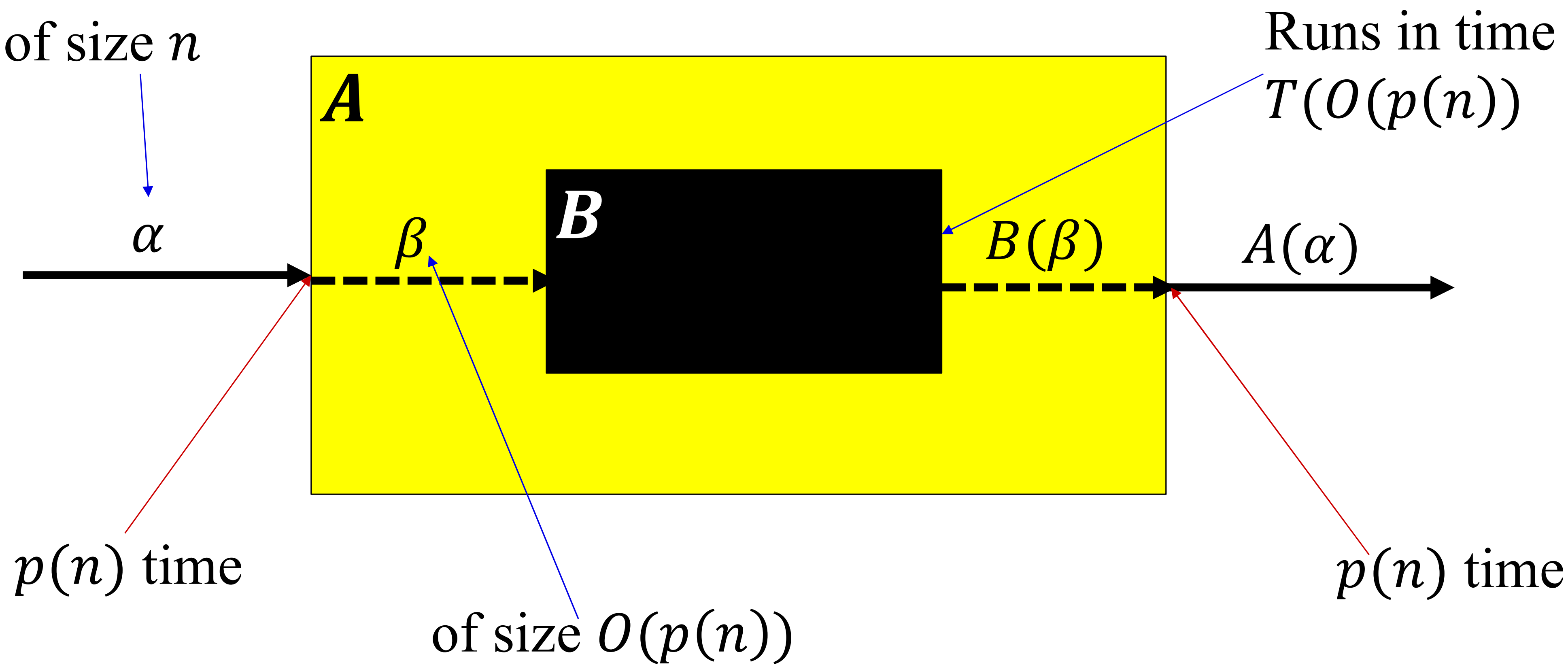
$$T\left(O\big(p(n)\big) + O(p(n))\right)$$

time algorithm to solve problem A on instances of size n.

Running Time Composition



Running Time Composition



Polynomial-Time Reduction

Definition:

$$A \leq_P B$$

if there is a $p(n)$-time reduction from A to B for some polynomial function $p(n) = O(n^c)$ for some constant c.

Poly-time Reduction

$$A \leq_P B$$

If B has a polynomial time algorithm, then so does A!

Poly-time Reduction

$$A \leq_P B$$

If B is "easily solvable", then so is A!

Why Poly-Time?

- Notion is broad and robust even if computing model/hardware is changed "reasonably"

- Usually, poly-time algorithms for real-life problems have runtime $O(n)$ or $O(n^2)$ or $O(n^3)$, not $O(n^{100})$.

A note on encoding

- For polynomial time, we mean that the runtime is polynomial in the **length of the encoding of the problem instance.**

- For many problems, can use a “standard” encoding.
  - Binary encoding of integers
  - For mathematical objects (graphs, matrices, etc.): list of parameters enclosed in braces, separated by commas

Example Question

Are the algorithms we saw for KNAPSACK and FRACTIONAL KNAPSACK in the last two lectures polynomial time?

- Yes for both
- Yes for KNAPSACK, no for FRACTIONAL KNAPSACK
- No for KNAPSACK, yes for FRACTIONAL KNAPSACK
- No for both

Example Question: Solution

No for KNAPSACK, yes for FRACTIONAL KNAPSACK

The input for both problems is a list $(v_1, w_1), \dots, (v_n, w_n), W$. The input size is $O(n \log M + \log W)$ where M is an upper bound on the v_i's and w_i's.

Running time for KNAPSACK is $O(nW \log M)$. Running time for FRACTIONAL KNAPSACK is $O(n \log n \log W \log M)$.

Here, we are being extra careful about M, the size of v_i's and w_i's. Normally, for array inputs, we do not have to consider this.

Pseudo-polynomial algorithms

An algorithm that runs in time polynomial in the numeric value of the input but is exponential in the length of the input is called a **pseudo-polynomial** time algorithm.

Example: The dynamic programming algorithm for knapsack.

Recap

- Reductions are a basic tool in algorithm design: using an algorithm for one problem to solve another.
- If you have a polynomial-time reduction from A to B and you also have a polynomial-time algorithm for B, then you get a polynomial-time algorithm for A.

Poly-time Reduction

$$A \leq_P B$$

If B is "easily solvable", then so is A!

Poly-time Reduction

$$A \le_P B$$

If A is "hard", then so is B!

Poly-time Reduction

$$A \leq_P B$$

Intuition: A is a special case of B.

If A is "hard", then so is B!

Example Question

Suppose that $A \leq_P B$. Which of the following can we infer?

a) If A can be solved in poly time, so can B.
b) A can be solved in poly time iff B can be solved in poly time.
c) If A cannot be solved in poly time, then neither can B.
d) If B cannot be solved in poly time, then neither can A.

Example Question: Solution

Only (C)

If B can be solved in polynomial time, so can A.

If A cannot be solved in polynomial time, neither can B.

Intractability

Goal in this lecture and the next will be to **compare** the hardness of basic computational problems.

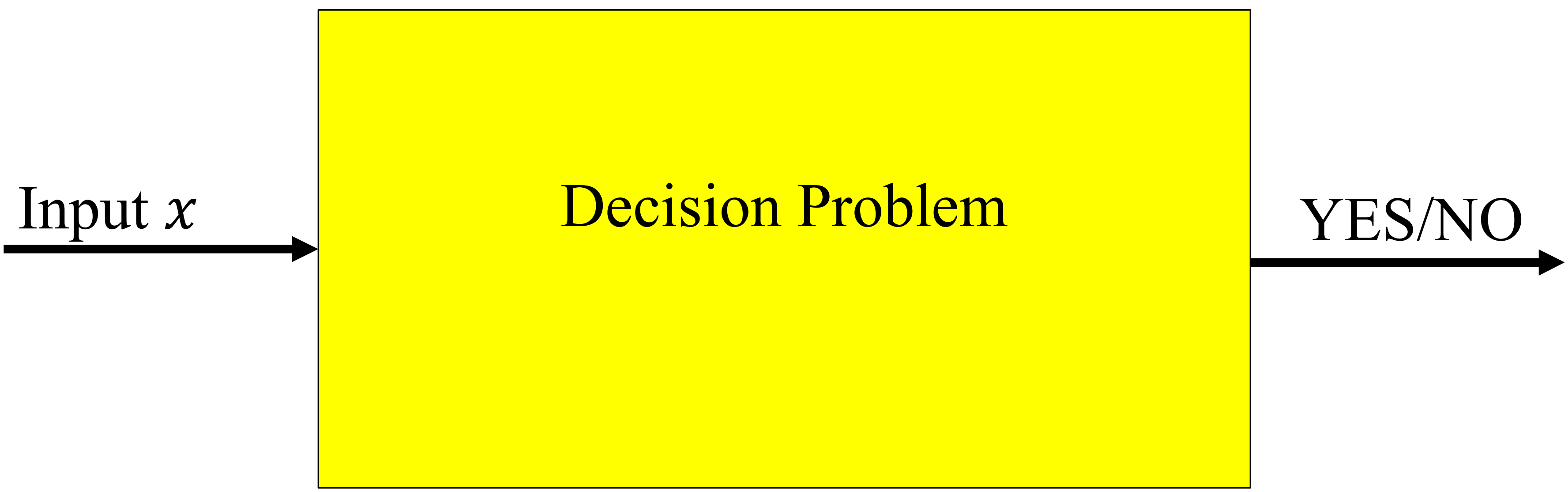
Intractability

Goal in this lecture and the next will be to **compare** the hardness of basic computational problems.

– Need a framework to talk about all problems using the same language!

Decision Problems

A **decision problem** is a function that maps an instance space *I* to the solution set {YES, NO}.



Decision vs Optimization

- **Decision Problem**: Given a directed graph G with two given vertices u and v, is there a path from u to v of length $\leq k$?

- **Optimization Problem**: Given a directed graph G with two given vertices u and v, what is the length of the shortest path from u to v?

Decision vs Optimization

Given an optimization problem, we can convert it into a decision problem:

> Given an instance of the optimization problem and a number k, ask for a solution with value $\leq k$?

Examples: a minimum spanning tree with weight $\leq k$, a longest common subsequence with length $> k$, a knapsack solution with value $> k$, etc.

Decision reduces to optimization

- The decision problem is no harder than the optimization problem
  - Given the value of the optimal solution, simply check whether it is $\leq k$.

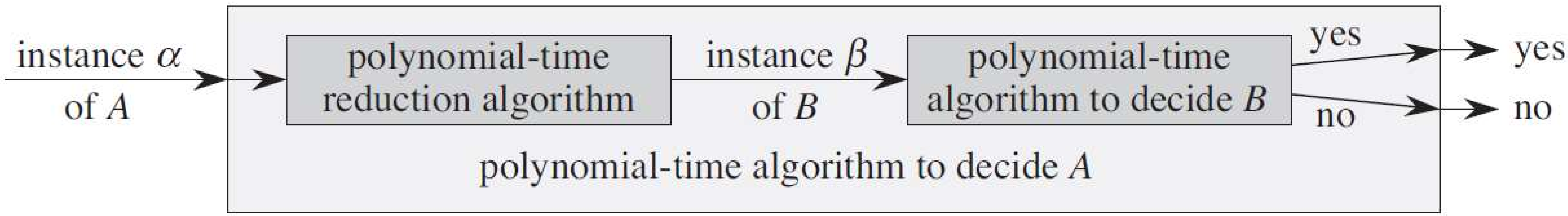
- So, if we cannot solve the decision problem quickly, we cannot solve the optimization problem quickly! For hardness, enough to study decision problems.

Reductions between Decision Problems

Given two decision problems A and B, a **polynomial-time reduction** from A to B, denoted $A \leq_P B$, is a transformation from instances α of A to instances β of B such that:

1. α is a YES-instance for A **<u>if and only if</u>** β is a YES-instance for B.
2. The transformation takes polynomial time in the size of α.

Reductions



Suffices to show:

- Reduction runs in polynomial time
- If α is a YES-instance of A, β is a YES-instance of B
- If β is a YES-instance of B, α is a YES-instance of A

Amazing Power of Reductions

We will show a web of reductions between many different fundamental decision problems: some about graphs, some about sets, some about numbers, some about circuits!

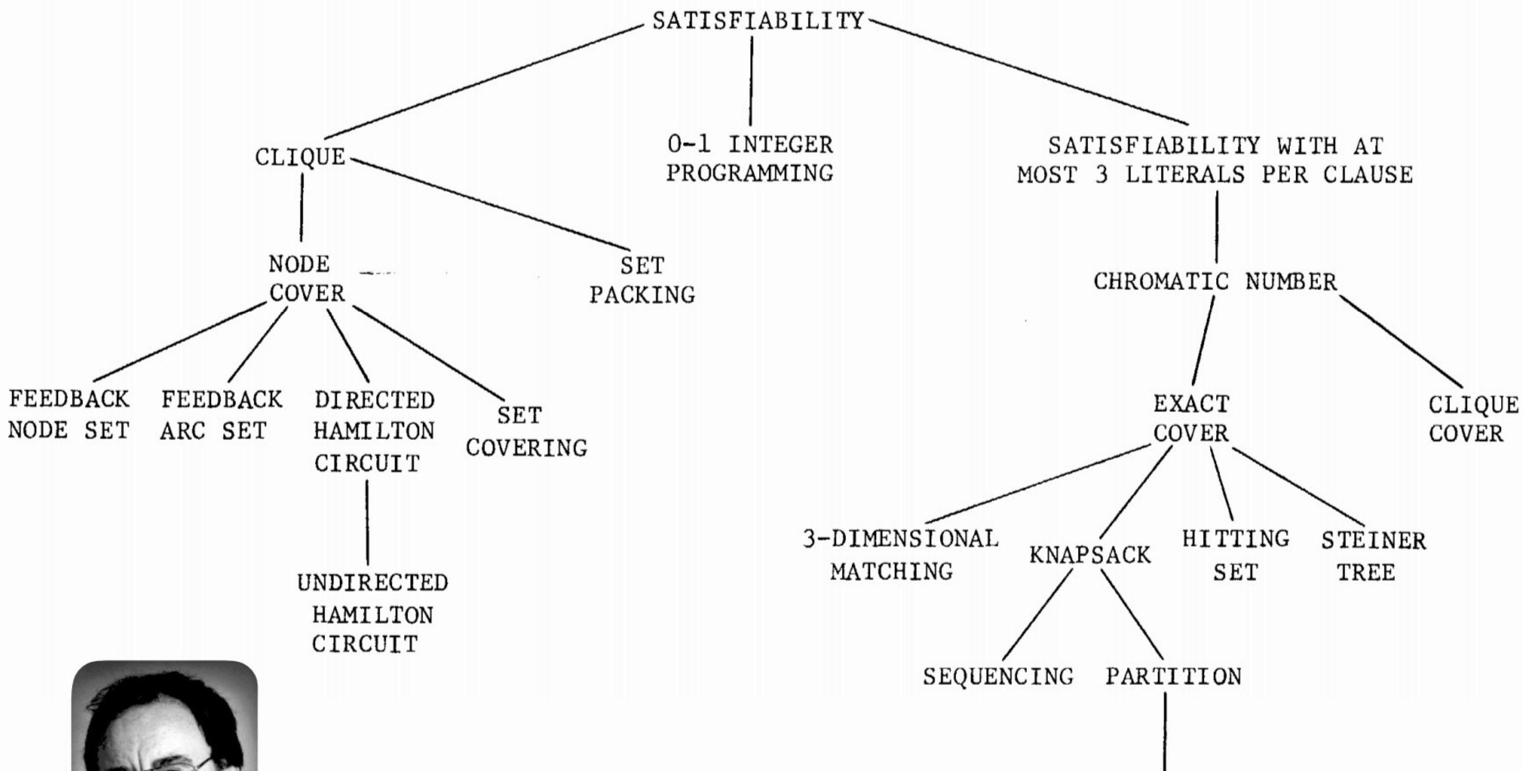
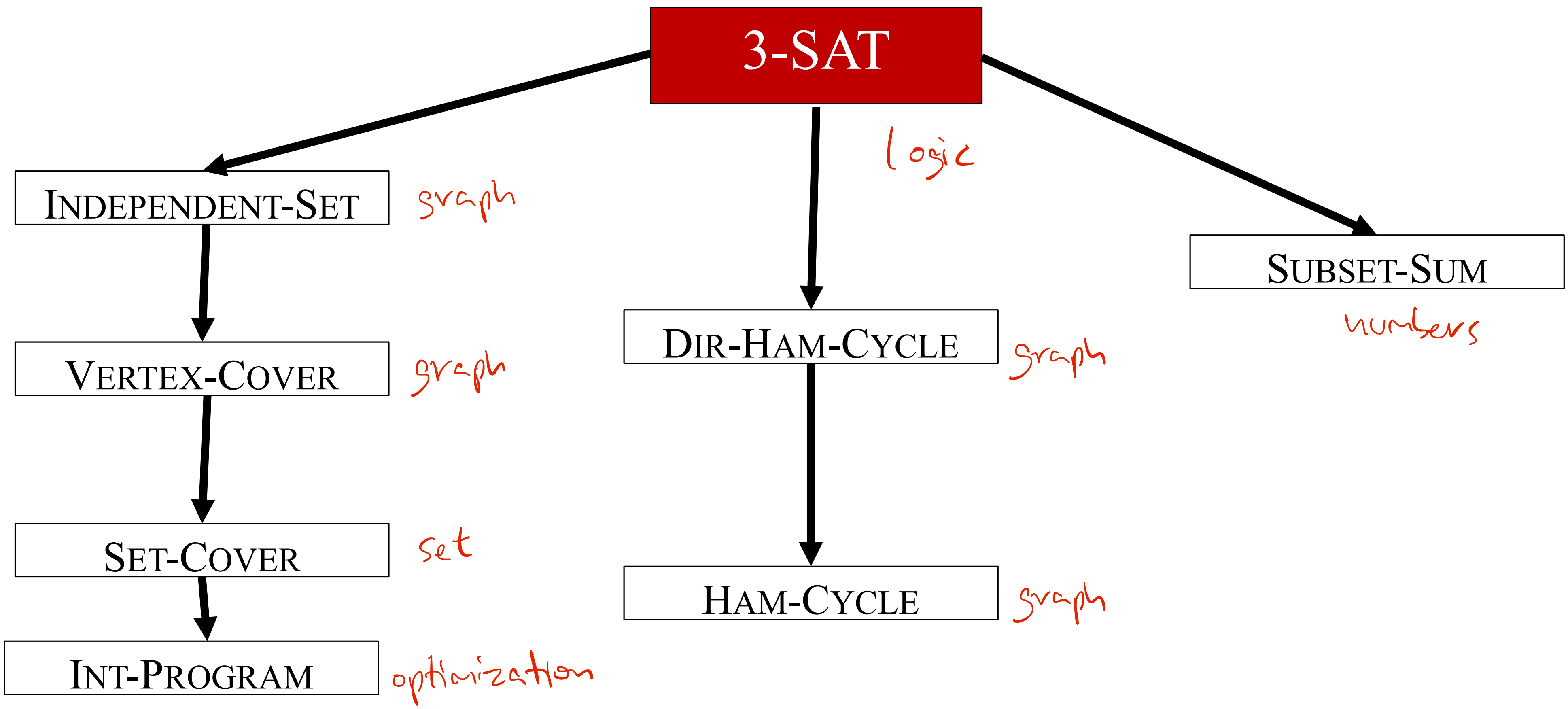


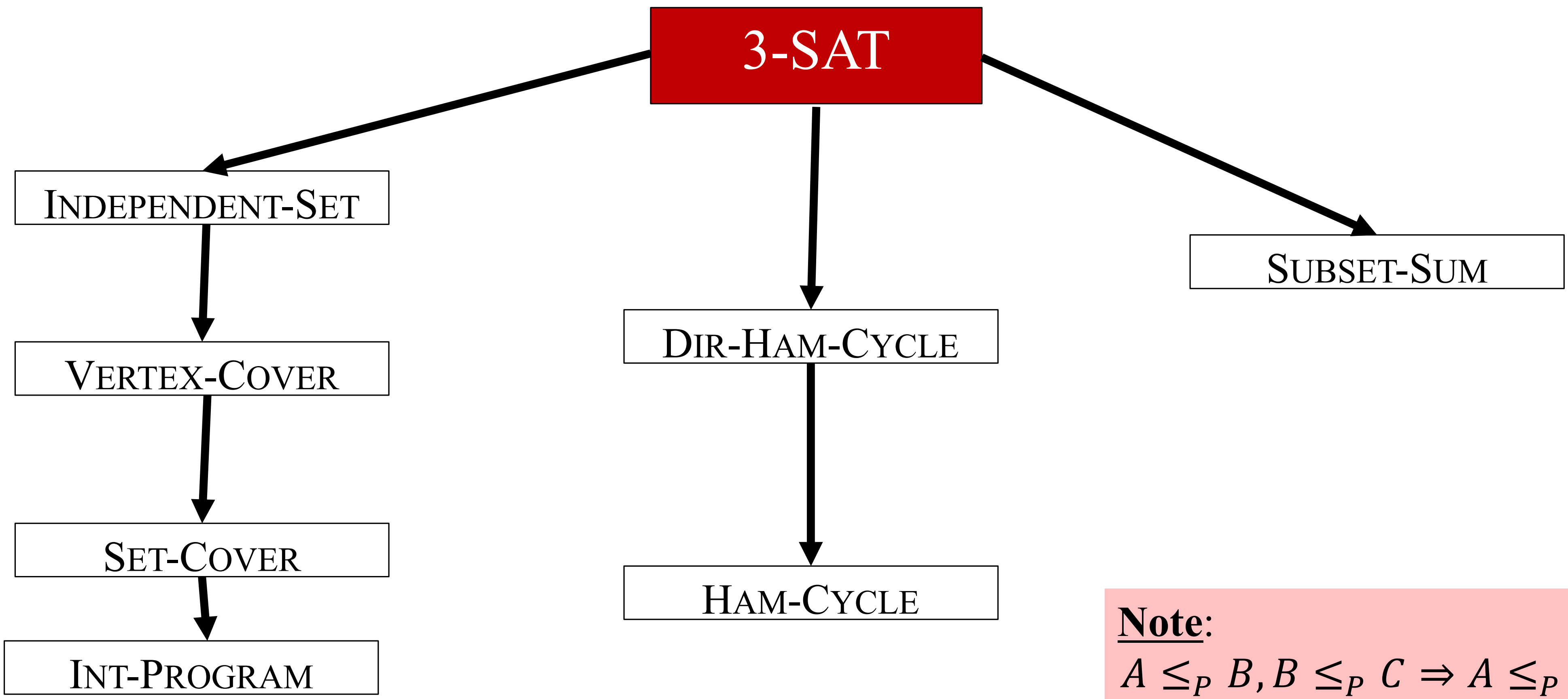
FIGURE 1 - Complete Problems

Dick Karp (1972)
1985 Turing Award

Reductions

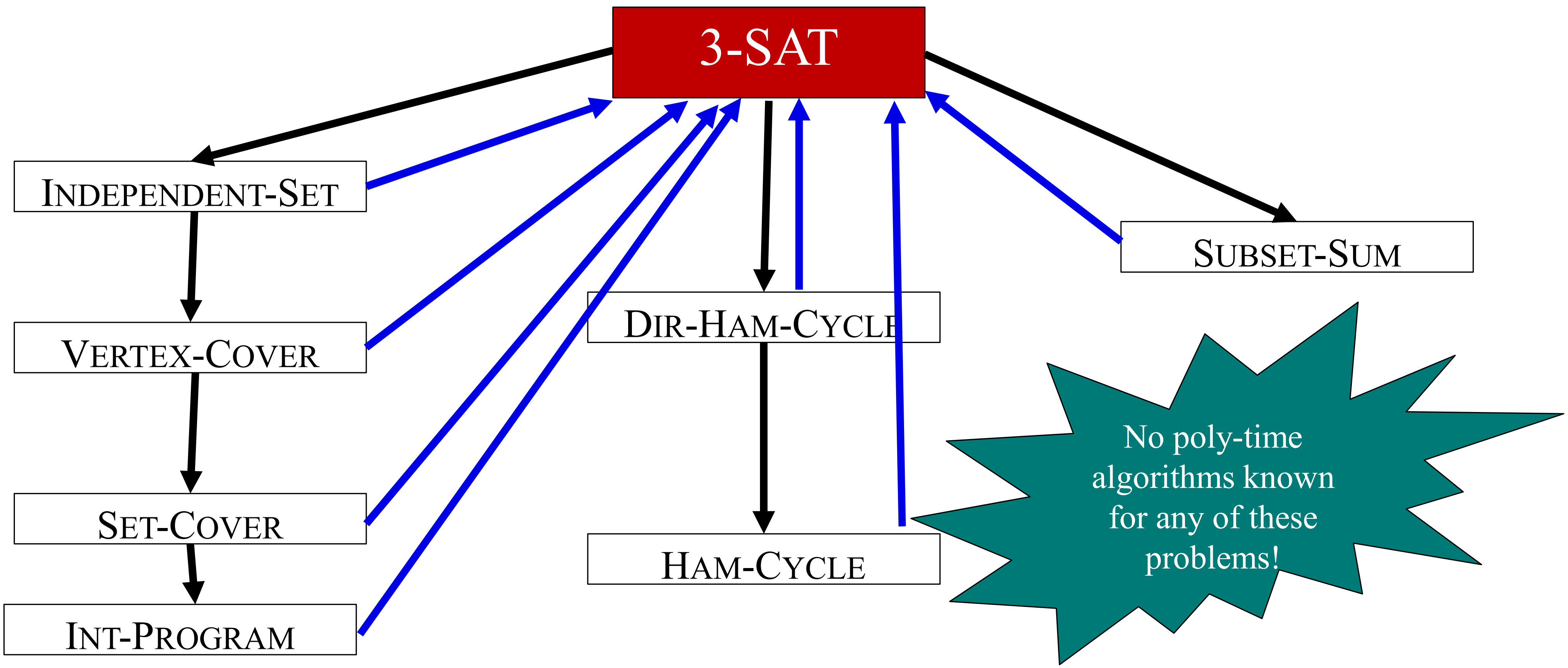

3-SAT
logic
INDEPENDENT-SET
graph
VERTEX-COVER
graph
SET-COVER
set
INT-PROGRAM
optimization
DIR-HAM-CYCLE
graph
HAM-CYCLE
graph
SUBSET-SUM
numbers

Reductions

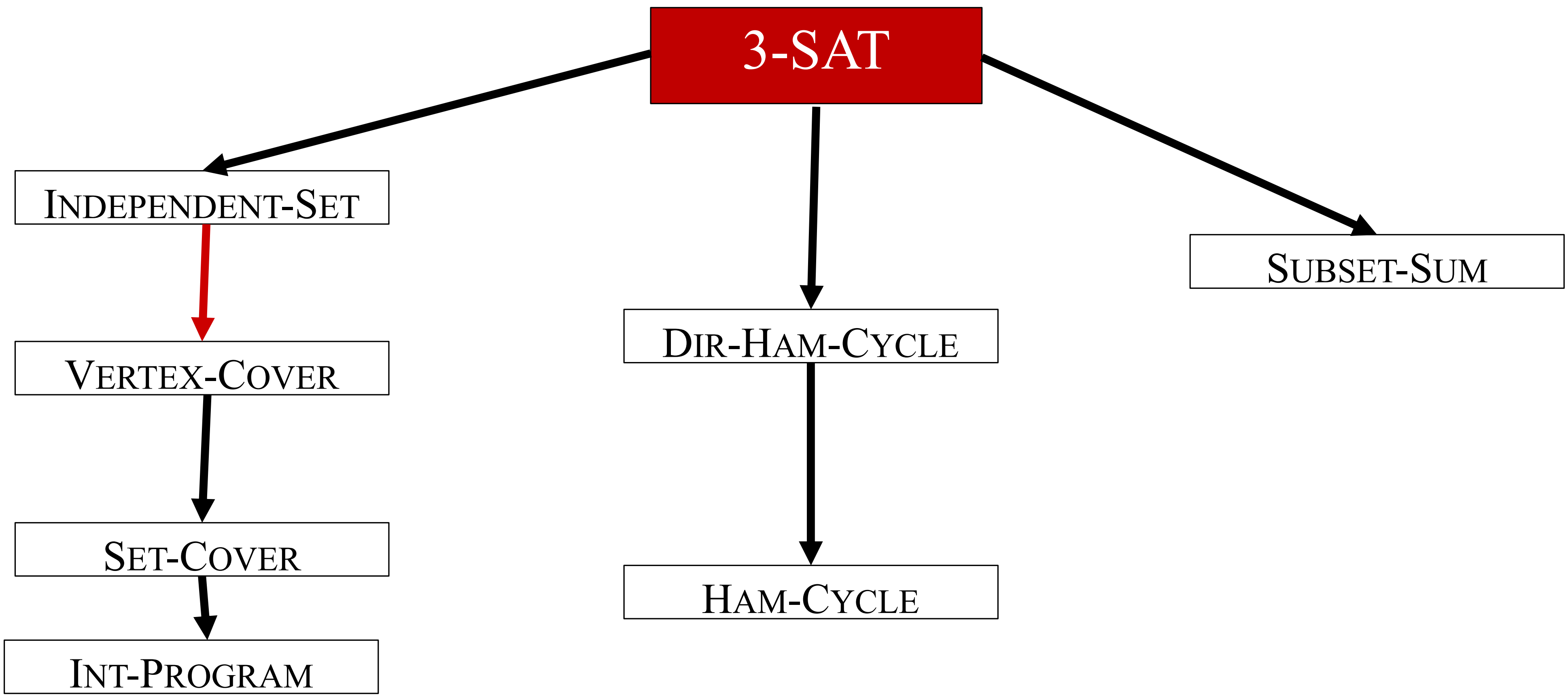


Note:
$A \leq_P B, B \leq_P C \Rightarrow A \leq_P C$

Reductions

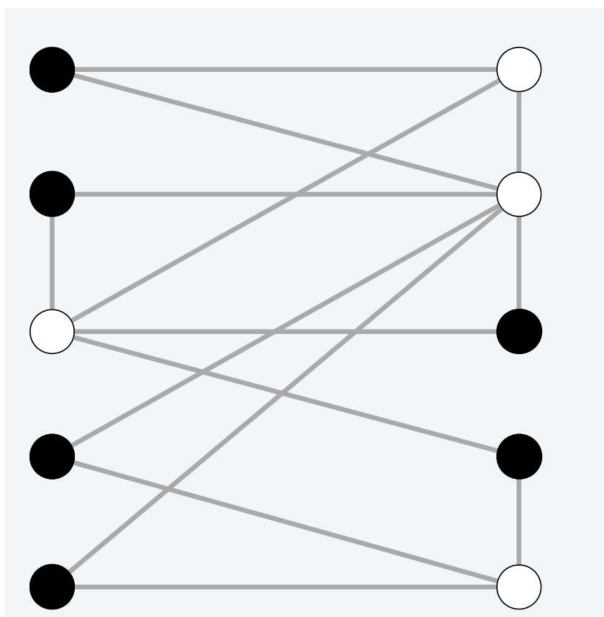

3-SAT
INDEPENDENT-SET
VERTEX-COVER
SET-COVER
INT-PROGRAM
DIR-HAM-CYCLE
HAM-CYCLE
SUBSET-SUM
No poly-time algorithms known for any of these problems!

Reductions


3-SAT
INDEPENDENT-SET
VERTEX-COVER
SET-COVER
INT-PROGRAM
DIR-HAM-CYCLE
HAM-CYCLE
SUBSET-SUM

INDEPENDENT-SET

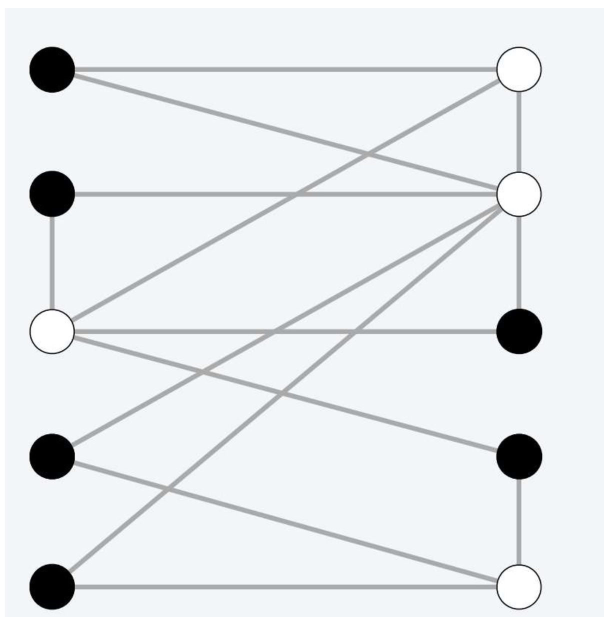
Given a graph $G = (V, E)$ and an integer k, is there a subset of k (or more) vertices such that no two are adjacent?



● Independent set of size 6

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 of size 4

INDEPENDENT-SET $\leq_P$ VERTEX-COVER

Reduction: To check whether G has an independent set of size k, we check whether G has a vertex cover of size $n - k$. (Here, n = number of vertices in G)

Proof:

- Clearly, reduction runs in polynomial time

INDEPENDENT-SET $\leq_P$ VERTEX-COVER

Reduction: To check whether G has an independent set of size k, we check whether G has a vertex cover of size $n - k$.

Proof:

- Suppose (G, k) is a YES-instance of INDEPENDENT-SET. So, there is a subset S of size $\geq k$ that is an independent set.
- **Claim**: $V - S$ is a vertex cover, of size $\leq n - k$. Why?
- Let $(u, v) \in E$. Then, either $u \notin S$ or $v \notin S$. So, either u or v in $V - S$. Done!

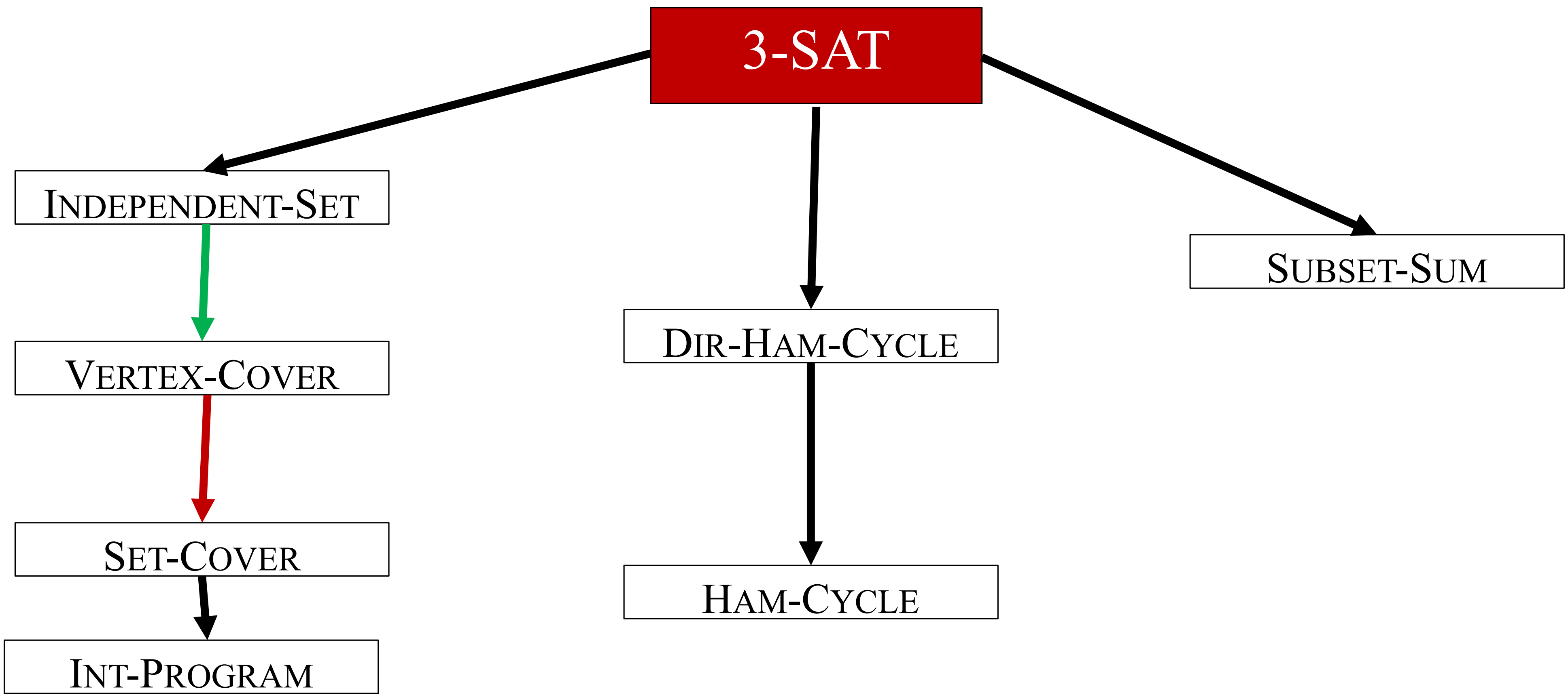
INDEPENDENT-SET $\leq_P$ VERTEX-COVER

Reduction: To check whether G has an independent set of size k, we check whether G has a vertex cover of size $n - k$.

Proof:

- Suppose $(G, n - k)$ is a YES-instance of VERTEX-COVER. So, there is a subset S of size $\leq n - k$ that is a vertex cover.
- **Claim**: $V - S$ is an independent set, of size $\geq k$. Why?
- Let $(u, v) \in E$ with both u and v in $V - S$. But then, S does not cover (u, v), a contradiction!

Reductions



3-SAT
INDEPENDENT-SET
VERTEX-COVER
SET-COVER
INT-PROGRAM
DIR-HAM-CYCLE
HAM-CYCLE
SUBSET-SUM

SET-COVER

Given integers k and n, and a collection $\mathcal{S}$ of subsets of $\{1, \ldots, n\}$, are there $\leq k$ of these subsets whose union equals $\{1, \ldots, n\}$?

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

S_2 and S_6 form a set cover of size 2

VERTEX-COVER $\leq_P$ SET-COVER

Reduction:

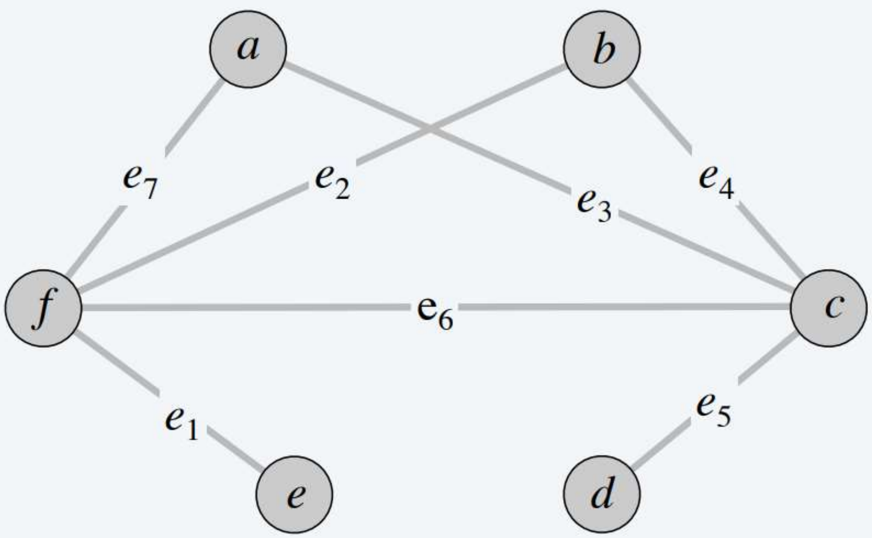
- Given (G, k) instance of VERTEX-COVER, we generate an instance $(n, k', \mathcal{S})$ of Set-Cover.
- Set $n = |E(G)|$, and $k' = k$.
- Order the edges of G arbitrarily: $e_1, \ldots, e_n$. For each $v \in V(G)$:

$$S_v = \{i \colon e_i \text{ incident on } v\}$$

  $\mathcal{S}$ is the collection of all such subsets S_v.

VERTEX-COVER $\leq_P$ SET-COVER

NUS National University of Singapore



vertex cover instance
(k = 2)

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

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

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

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

set cover instance
(k = 2)

For every vertex-cover instance, the reduction converts it into a set-cover instance

But some set-cover instance may never be reached by this reduction.

VERTEX-COVER $\leq_P$ SET-COVER

Reduction:

- Order the edges of G arbitrarily: $e_1, \dots, e_n$. For each $v \in V(G)$:

$$S_v = \{i : e_i \text{ incident on } v\}$$

$\mathcal{S}$ is the collection of all such subsets S_v.

Clearly, reduction runs in polynomial time.

VERTEX-COVER $\leq_P$ SET-COVER

Reduction:

- Order the edges of G arbitrarily: $e_1, \ldots, e_n$. For each $v \in V(G)$:

$$S_v = \{i: e_i \text{ incident on } v\}$$

$\mathcal{S}$ is the collection of all such subsets S_v.

Suppose (G, k) is a YES-instance of VERTEX-COVER. Let U be the subset of size $\leq k$ forming the vertex cover. Then, by definition, the union of S_u's over all $u \in U$ is $\{1, \ldots, n\}$.

VERTEX-COVER $\leq_P$ SET-COVER

Reduction:

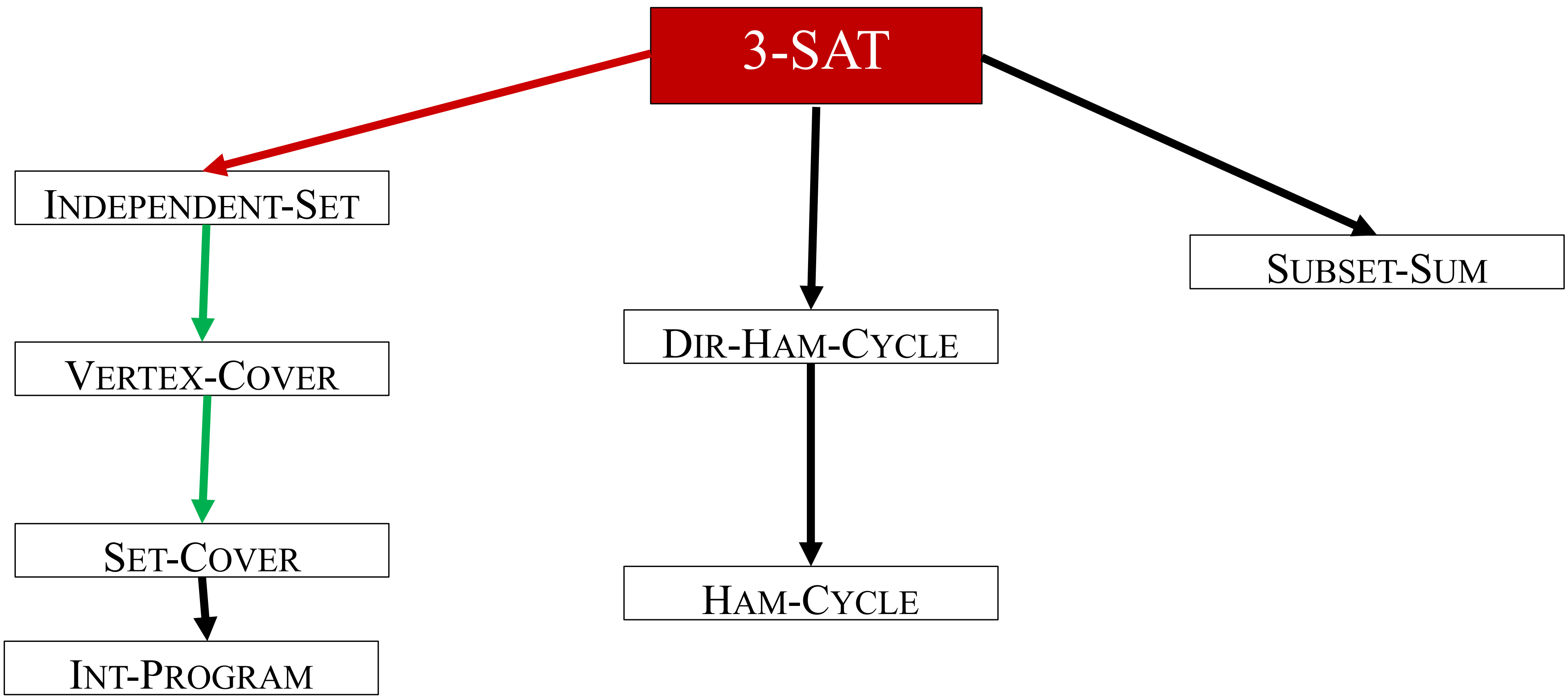
- Order the edges of G arbitrarily: $e_1, \ldots, e_n$. For each $v \in V(G)$:

$$S_v = \{i : e_i \text{ incident on } v\}$$

$\mathcal{S}$ is the collection of all such subsets S_v.

Suppose $(n, k, \mathcal{S})$ is a YES-instance of SET-COVER. Let the cover correspond to the sets $S_{v_1}, \ldots, S_{v_t}$ for $t \leq k$. Then, the vertices $v_1, \ldots, v_t$ form a vertex cover in G.

Reductions


3-SAT
INDEPENDENT-SET
VERTEX-COVER
SET-COVER
INT-PROGRAM
DIR-HAM-CYCLE
HAM-CYCLE
SUBSET-SUM

Satisfiability

- **Literal**: A Boolean variable or its negation. $x_i, \overline{x_i}$

- **Clause**: A disjunction (OR) of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$

- **Conjunctive Normal Form (CNF)**: a formula Φ that is a conjunction (AND) of clauses $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

- **SAT**: Given a CNF formula Φ, does it have a satisfying truth assignment?

3-SAT

SAT where each clause contains exactly 3 literals (not necessarily distinct)

can have any number of clauses

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

Satisfying assignment: $x_1 = \text{True}, x_2 = \text{True}, x_3 = \text{False}, x_4 = \text{True}$

Non-satisfying assignment: $\underbrace{x_1 = \text{True}, x_2 = \text{False}, x_3 = \text{False}, x_4 = \text{False}}_{\text{The first clause is not satisfied by this assignment}}$

Φ is a YES-instance if and only if it admits at least one satisfying assignment.

3-SAT $\leq_P$ INDEPENDENT-SET

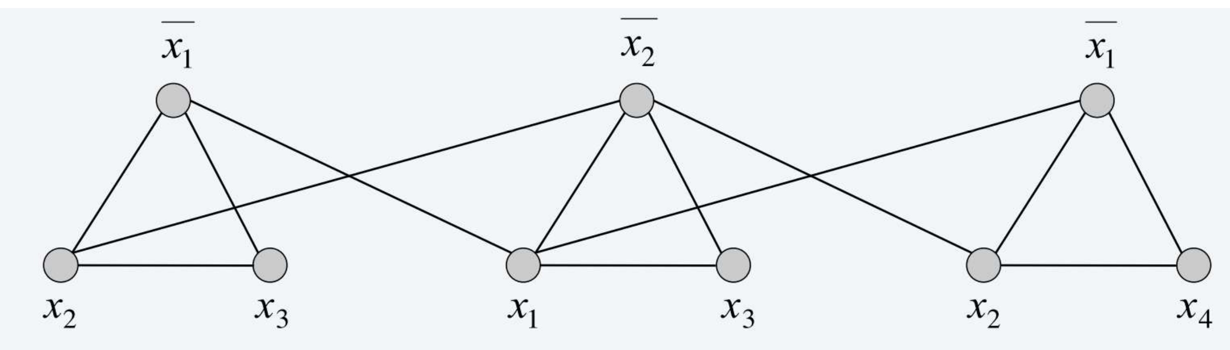
Given an instance Φ of 3-SAT, goal is to construct an instance (G, k) of INDEPENDENT-SET so that G has an independent set of size k iff Φ is satisfiable.

3-SAT $\leq_P$ INDEPENDENT-SET

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

<u>Reduction</u>

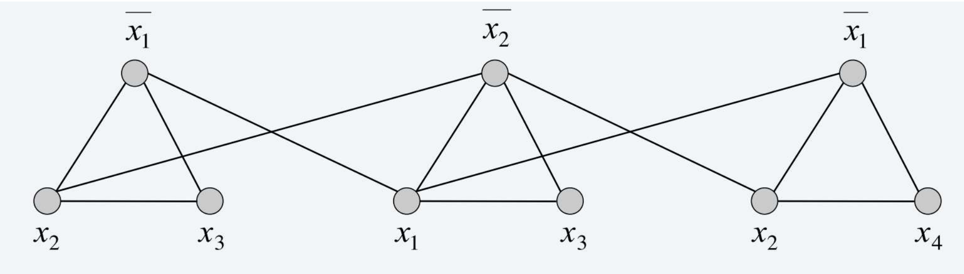
- G contains 3 vertices for each clause, one for each literal
- Connect 3 literals in clause in a triangle
- Connect literal to each of its negations
- Set $k =$ number of clauses



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

3-SAT $\leq_P$ INDEPENDENT-SET

Reduction
- G contains 3 vertices for each clause, one for each literal
- Connect 3 literals in clause in a triangle
- Connect literal to each of its negations
- Set k = number of clauses

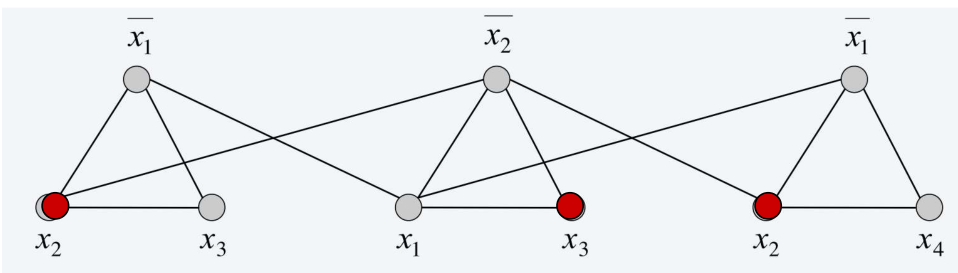


$$(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

Reduction clearly runs in linear time.

3-SAT $\leq_P$ INDEPENDENT-SET

Reduction
- G contains 3 vertices for each clause, one for each literal
- Connect 3 literals in clause in a triangle
- Connect literal to each of its negations
- Set $k =$ number of clauses



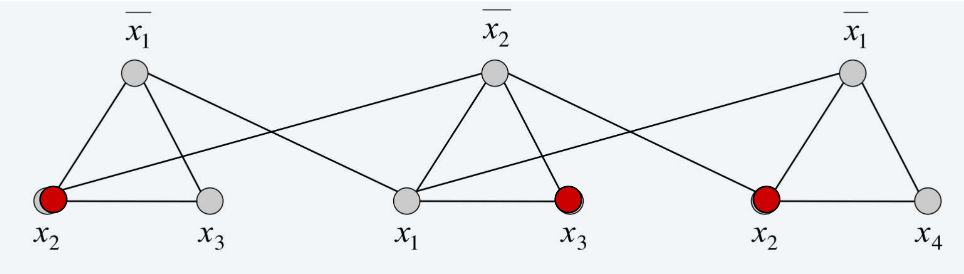
$$(\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)$$

Suppose Φ is a YES-instance. Take any satisfying assignment for Φ and select a true literal from each clause. Corresponding k vertices form an independent set in G.

3-SAT $\leq_P$ INDEPENDENT-SET

Reduction

- G contains 3 vertices for each clause, one for each literal
- Connect 3 literals in clause in a triangle
- Connect literal to each of its negations
- Set k = number of clauses



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

Suppose (G, k) is a YES-instance. Let S be the independent set of size k. Each of the k triangles must contain exactly one vertex in S. Set these literals to true, so all clauses satisfied.

NP-completeness

- Actually, there are hundreds of problems (**NP-complete**) that have reductions to and from the above problems. Put differently, if we have a poly-time algorithm for any one of these NP-complete problems, we have poly-time algorithms for all of them!

Acknowledgements

- The slides are modified from
  - the slides from Prof. Erik D. Demaine and Prof. Charles E. Leiserson
  - the slides from Prof. Lee Wee Sun
  - the slides from Prof. Kevin Wayne
  - the slides from Prof. Arnab Bhattacharyya