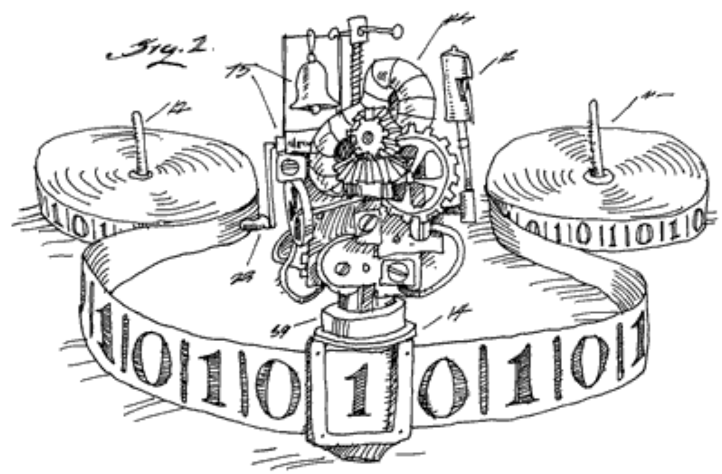
EECS 376: Foundations of Computer Science

Lecture 17 - More on NP-Completeness



NP-Completeness via reductions

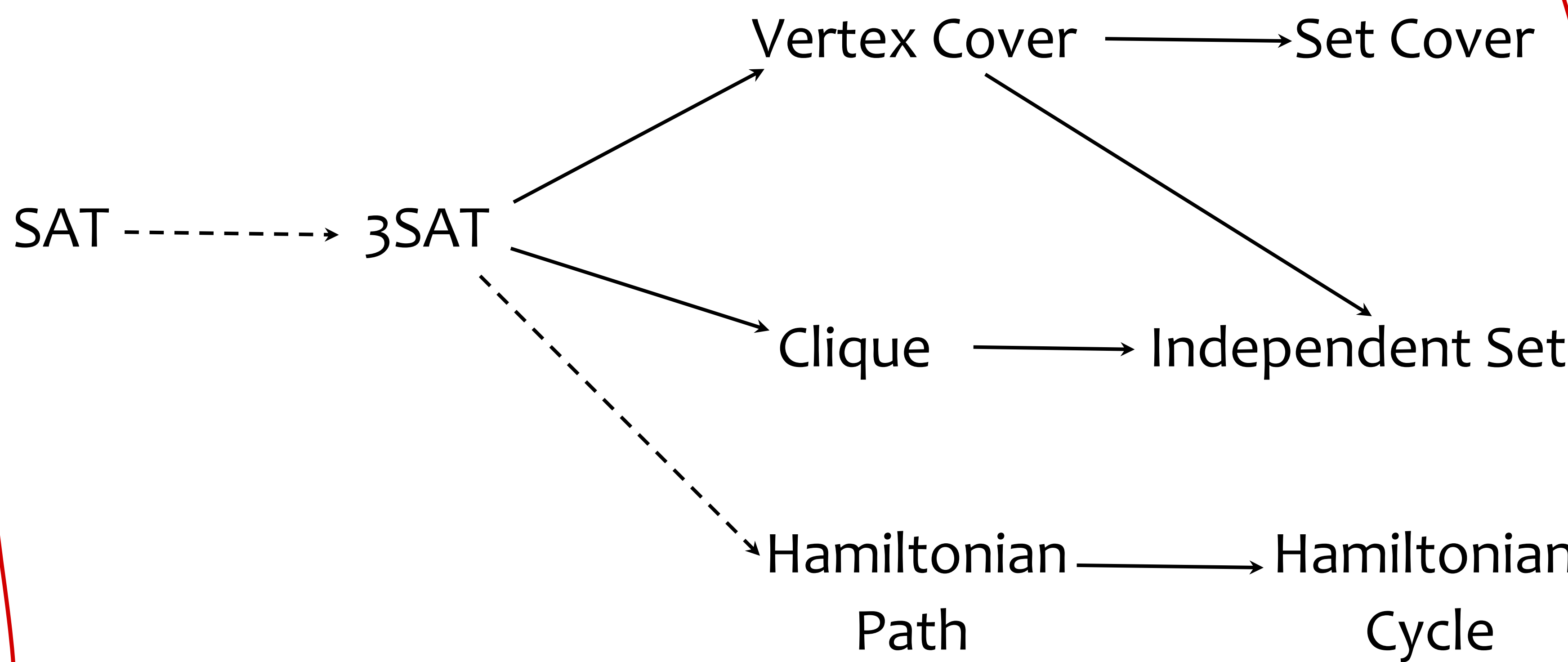
To show that a problem **B** is **NP**-Complete:

1 * Prove **B** is in **NP.**
  * Write a verifier V for B, show that it is correct and efficient.

2 * Prove **B** is NP-hard
  * Pick some *known* **NP**-hard problem **A.**
  * Show **A** $\leq_p$ **B:**
    1. Show a mapping f from instances of **A** to instances of **B**
    2. x is a yes-instance for **A** $\Leftrightarrow f(x)$ is a yes-instance of **B** **(both directions!)**
    3. $f(x)$ runs in poly($|x|$) time

A Web of NP-Hard Problems

(all of these are also in NP, and therefore NP-Complete)



Set Cover is NP-complete

Will only show that Set Cover is NP-hard.
Proving Set Cover is in NP is straightforward.

Set Cover (SC) Problem

- Given a set of elements {1, 2, …, n} (called the universe) and a collection S of m subsets whose union equals the universe, the set cover problem is to identify the smallest sub-collection of S whose union equals the universe.

- For example, consider the universe U = {1, 2, 3, 4, 5} and the collection of sets S = { {1, 2, 3}, {2, 4}, {3, 4}, {4, 5} }. The union of S is U. However, we can cover all elements with only two sets: { {1, 2, 3}, {4, 5} }. Therefore, the solution to the set cover problem is size 2.

Set Cover (SC) Problem

(Contractor Problem)

Problem Setup:

- n workers, each worker has a set of skills
- Goal: hire a team of workers that together have every skill.

Formally:

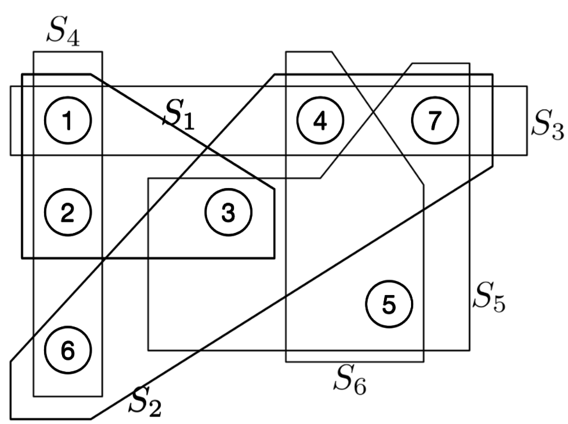
- Given a collection U of elements (skills) and
- n subsets $S_1, \dots, S_n \subseteq U$ (skills of each worker),
- a ***set cover*** is a group of S_i's whose union is U.

Set cover decision problem:

- Given collection U, subsets $S_1, \dots, S_n \subseteq U$, and a budget k,
- does there exist a set cover of size k or less?

U = 1,2,…,7

S1 = {1,2,3}
S2 = {3,4,6,7}
S3 = {1,4,7}
S4 = {1,2,6}
S5 = {3,5,7}
S6 = {4,5}



, 3

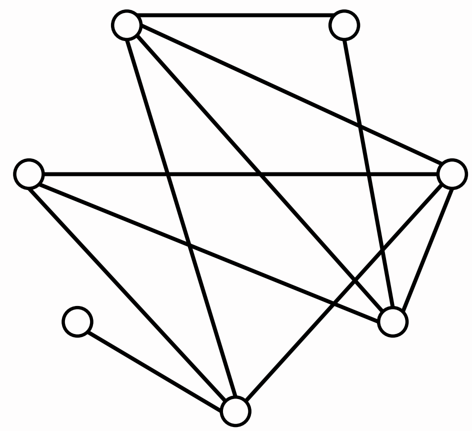
We will show

VC $\leq_p$ SC

Example

Given an arbitrary VC instance

VC: can we circle ≤k vertices so that every edge has at least one circled endpoint?



, k

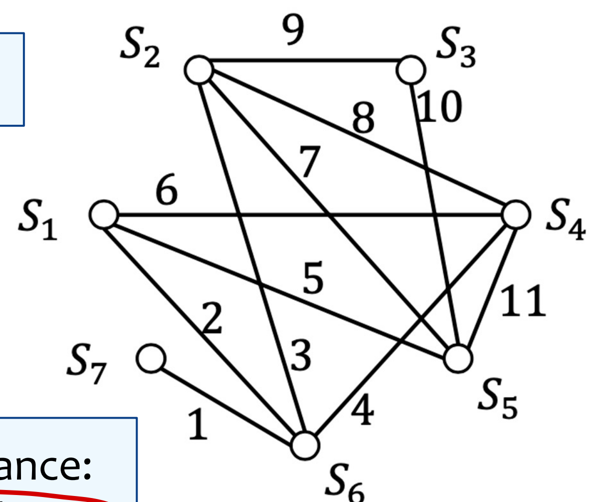
Construct a (carefully crafted) instance of SC

edges from VC instance

$$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$$

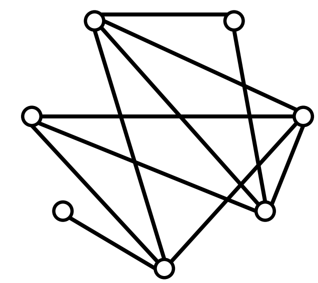
$$S_1 = \{2, 5, 6\}, S_2 = \{3, 7, 8, 9\},$$
$$S_3 = \{9, 10\}, S_4 = \{4, 6, 8, 11\},$$
$$S_5 = \{5, 7, 10, 11\}, S_6 = \{1, 2, 3, 4\},$$
$$S_7 = \{1\}$$

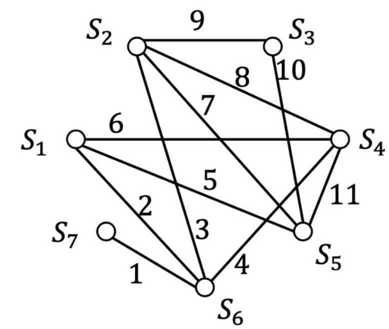
for each vertex v from VC instance:
a set containing all incident edges



, k

Details





Step 1: describe the mapping

- Given an input $G = (V, E)$ for vertex cover and budget k
- Let's formally describe the set cover instance.
- What is U? what are the sets? What is the budget?
  - $U = E$
  - For each vertex v, create $S_v = \{e \mid v \in e\}$.
  - Budget is k too.

Step 2: prove correctness

- "Yes"-vertex cover instance ⇒ "Yes"-set cover instance
  - Suppose there is a vertex cover $C \subseteq V$ of size k, there is a set cover of size k. How?
  - Consider $\{S_v \mid v \in C\}$. This is a set cover of U.
- "Yes"-set cover instance ⇒ "Yes"-vertex cover instance
  - Suppose there is a set cover $S_{v_1}, \dots, S_{v_k}$ of U,
  - then $\{v_1, \dots, v_k\}$ is a vertex cover of G. Why?

(显然)

Step 3: poly-time mapping

- How fast is the reduction?
- $O(|E| + |V|)$.

Vertex Cover is just a special case of Set Cover. How?
Each element is in at most 2 sets.

Hamiltonian Cycle is NP-complete

Will only show NP-hardness, again.

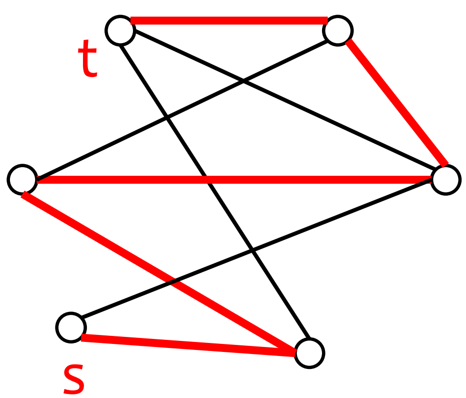
Hamiltonian Path and Hamiltonian Cycle

A ***(s,t)-Hamiltonian Path*** in an undirected graph is a path from s to t that visits every vertex **exactly** once.

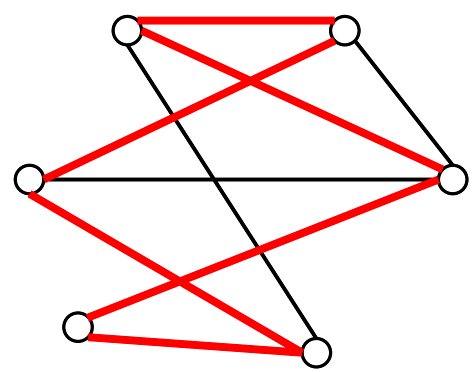
- Decision Problem: Given a graph and s, t, is there a (s,t)-Hamiltonian Path?

A ***Hamiltonian Cycle*** in an undirected graph is a cycle that visits every vertex **exactly** once.

- Decision Problem: Given a graph, does it have a Hamiltonian Cycle?



A Hamiltonian Path



A Hamiltonian Cycle

Hamiltonian Path and Hamiltonian Cycle

A ***(s,t)-Hamiltonian Path*** in an undirected graph is a path from s to t that visits every vertex **exactly** once.

- Decision Problem: Given a graph and s, t, is there a (s,t)-Hamiltonian Path?

A ***Hamiltonian Cycle*** in an undirected graph is a cycle that visits every vertex **exactly** once.

- Decision Problem: Given a graph, does it have a Hamiltonian Cycle?

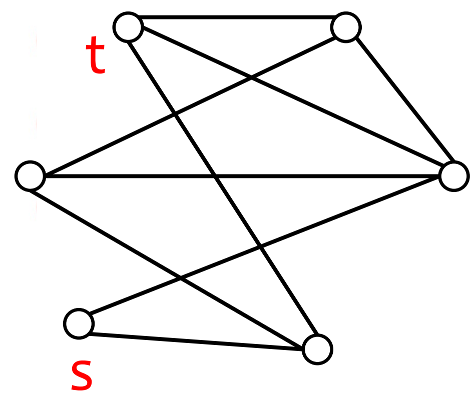
Hamiltonian Path (HP) is NP-Complete
(we won't prove)

Hamiltonian Cycle (HC) is NP-Complete

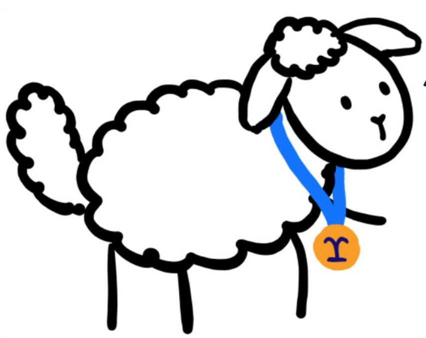
We will prove HC is NP-hard by showing HP $\leq_p$ HC

Some not-even-wrong idea…

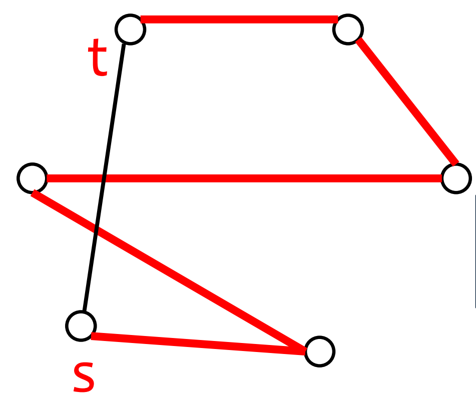
Given an arbitrary HP instance



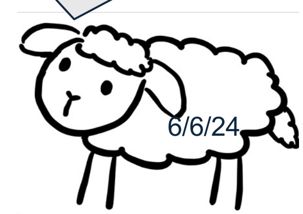
Construct a (carefully crafted) instance of HC



Include every edge of the Hamiltonian Path plus the edge (s,t) to form a Hamiltonian Cycle!

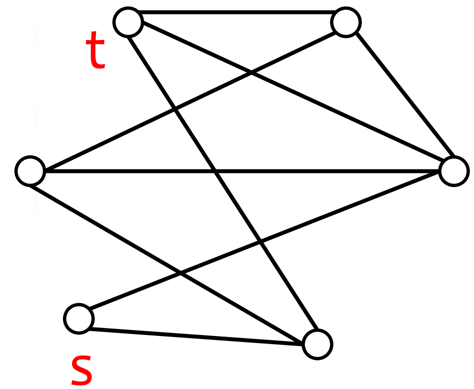


Something is fishy…

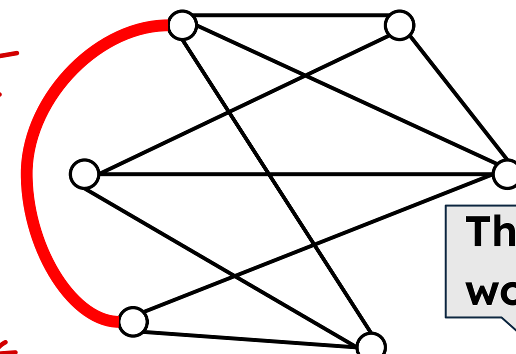


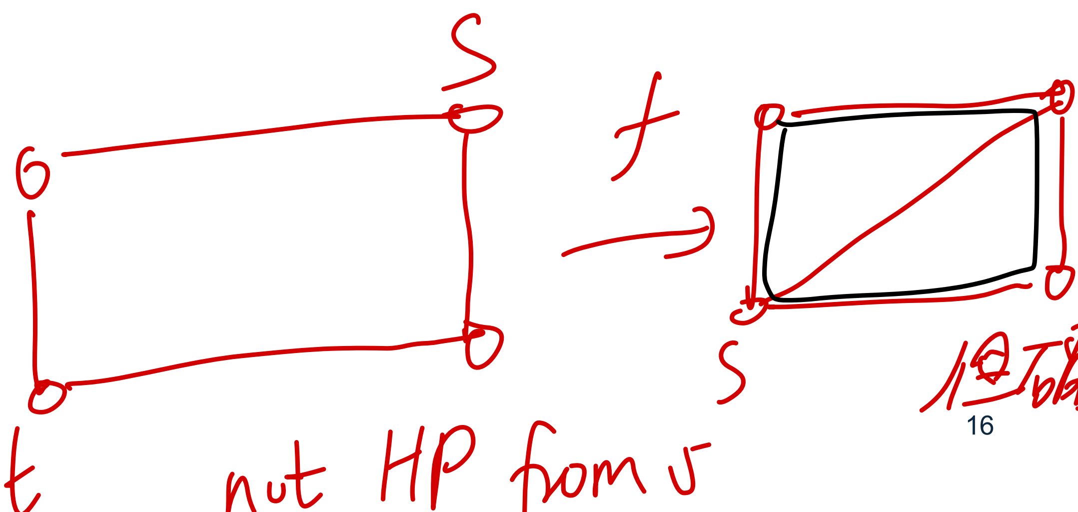
Some wrong-but-nice attempt…

Given an arbitrary HP instance



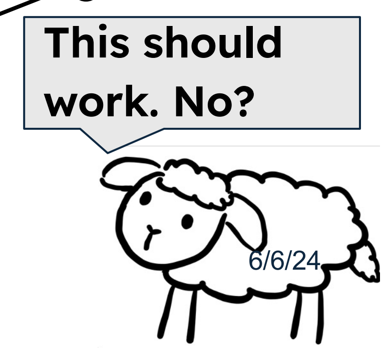
Construct a (carefully crafted) instance of HC





not HP from s to t

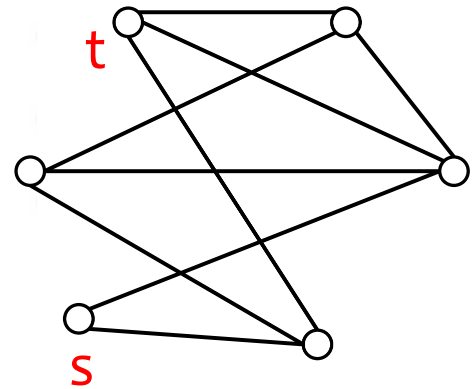
但确有HC



原因：没有保证 f(G) 的HC是从s或t出发最后回来的

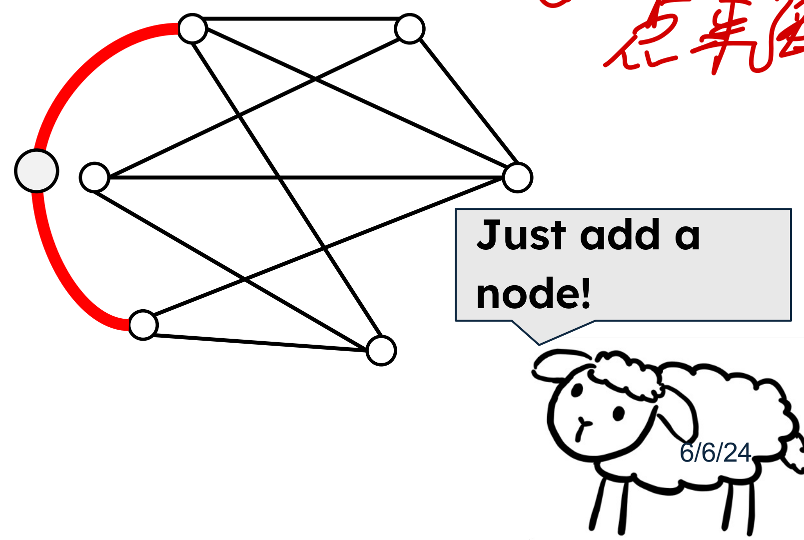
Correct idea

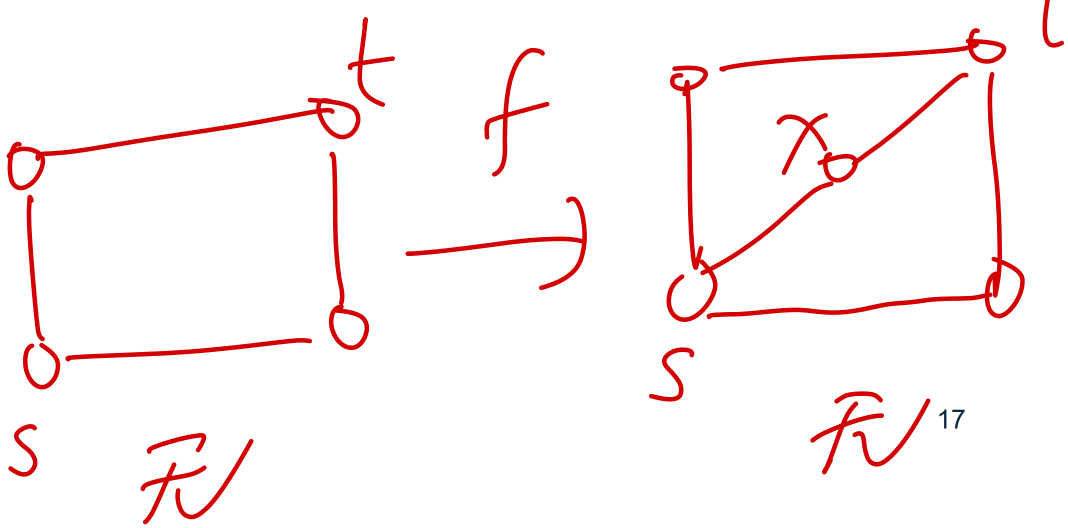
Given an arbitrary HP instance



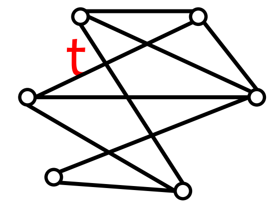
Construct a (carefully crafted) instance of HC

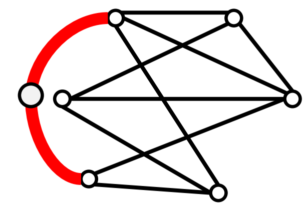
因而加个中间节点来强迫





Details





Step 1: describe the mapping

- Given an instance $G = (V, E)$ for HP
- An instance G' for HC is obtained by
    - Adding a path (s, x, t) into G

Step 2: prove correctness

- "Yes"-HP instance $\Rightarrow$ "Yes"-HC instance
    - Suppose there is a (s, t)-HP P in G, how to construct an HC in G'?
    - Just add (s, x, t) into P to get a HC.
- "Yes"-HC instance $\Rightarrow$ "Yes"-HP instance
    - Suppose there is an HC C in G', how to construct an HP in G.
    - Observe that $(s, x, t) \subseteq C$. So $P = C \setminus (s, x, t)$ is an (s, t)-HP.

Step 3: poly-time mapping. Clearly, linear time.

Aside

- An **Eulerian Cycle** which is a cycle that visits every **edge** exactly once
  - We can check if it exists in linear time!
  - Euler's Theorem:

  "A graph has an Eulerian cycle iff every vertex has an even degree."

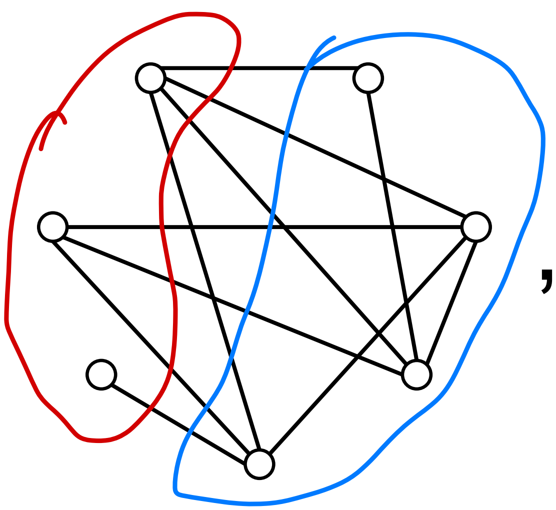
- People tried to show a similar characterization for Hamiltonian cycles but failed.
- Now, we have an explanation for that.
  - If it admits efficient characterization, then P = NP (which should not happen)

Independent Set is NP-complete

Will only show NP-hardness, again.

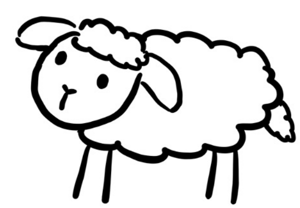
Independent Set (IS) Problem

- Given a graph, an ***independent set*** is a set **S** of vertices so that there is *no edge* between any pair of vertices in **S**
- **Independent Set decision problem:**
  - Given a graph **G** and a budget **k**,
  - does **G** have an independent set of size **k** or more?



, 3

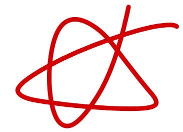
IS is the "opposite" of Clique. We can use this observation to build a reduction Clique ≤p IS. See if you can find it…



Observation: For any graph G=(V,E),

S is a vertex cover if and only if $V \setminus S$ is an independent set

因为任意一个 $e \in E$,



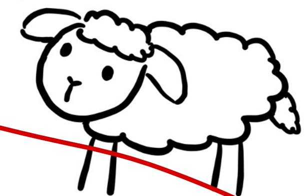
如果它一个 node $\in S$, 那么另一个 node 一定 $\notin S$

From this, how would you show that Independent-Set (IS) is NP-hard?

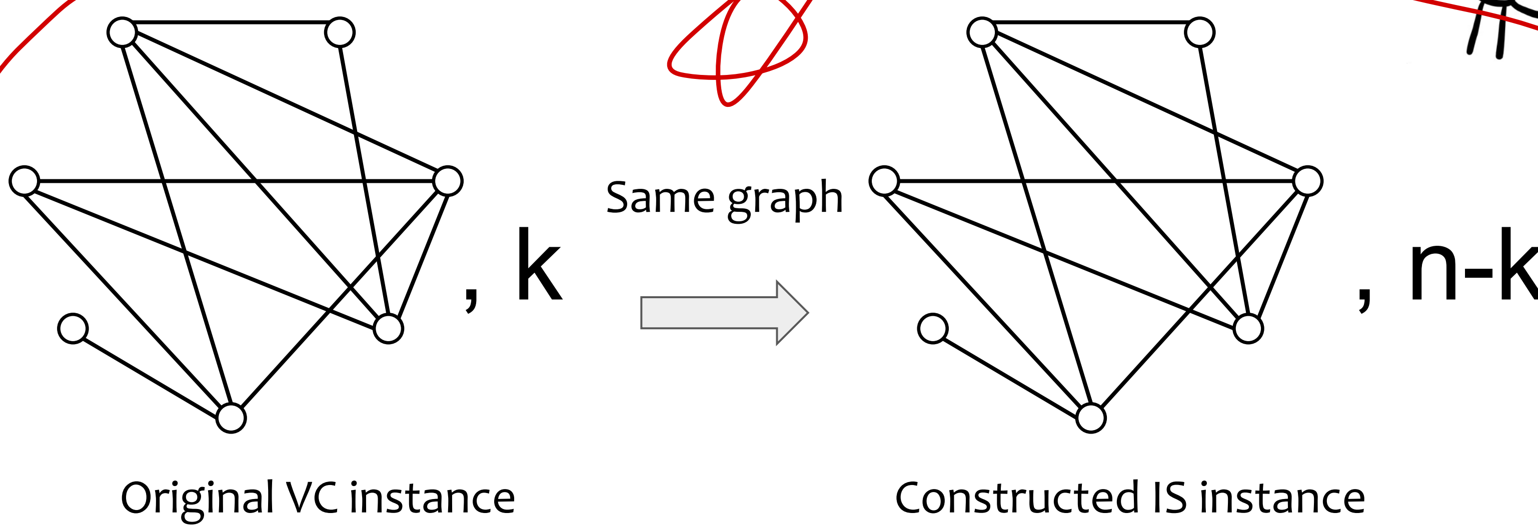
$\Rightarrow$ 一定 $\in V \setminus S$

- Show that VC $\leq_p$ IS
- Given an instance $G = (V, E)$ with budget k for VC,
- How would you construct an instance for IS? What is the budget?
  - Same graph G
  - Budget: $n - k$
- **Exercise**: show correctness.

IS is NP-hard: VC ≤p IS



This reduction illustrates the principle that the budget can change.



SUBSET-SUM Problem

Definition:

$$\mathit{SUBSET\text{-}SUM} = \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t\}.$$

For example, $\langle\{4, 11, 16, 21, 27\}, 25\rangle \in \mathit{SUBSET\text{-}SUM}$ because $4 + 21 = 25$. Note that $\{x_1, \ldots, x_k\}$ and $\{y_1, \ldots, y_l\}$ are considered to be ***multisets*** and so allow repetition of elements.

$\cancel{\emptyset}$

3SAT ≤p SUBSET-SUM

Theorem:

SUBSET-SUM is in NP.

SUBSET-SUM is NP-complete.

Proof:

$x_1, \ldots, x_l$

PROOF We already know that *SUBSET-SUM* $\in$ NP, so we now show that *3SAT* $\leq_P$ *SUBSET-SUM*. $\emptyset$

$C_1, \ldots, C_l$

Let ϕ be a Boolean formula with variables $x_1, \ldots, x_l$ and clauses $c_1, \ldots, c_k$. The reduction converts ϕ to an instance of the *SUBSET-SUM* problem $\langle S, t\rangle$, wherein the elements of S and the number t are the rows in the table in Figure 7.57, expressed in ordinary decimal notation. The rows above the double line are labeled

$$y_1, z_1, y_2, z_2, \ldots, y_l, z_l \quad \text{and} \quad g_1, h_1, g_2, h_2, \ldots, g_k, h_k$$

and constitute the elements of S. The row below the double line is t.

Thus, S contains one pair of numbers, y_i, z_i, for each variable x_i in ϕ. The decimal representation of these numbers is in two parts, as indicated in the table. The left-hand part comprises a 1 followed by $l - i$ 0s. The right-hand part contains one digit for each clause, where the digit of y_i in column c_j is 1 if clause c_j contains literal x_i, and the digit of z_i in column c_j is 1 if clause c_j contains literal $\overline{x_i}$. Digits not specified to be 1 are 0.

The table is partially filled in to illustrate sample clauses, c_1, c_2, and c_k:

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

Additionally, S contains one pair of numbers, g_j, h_j, for each clause c_j. These two numbers are equal and consist of a 1 followed by $k - j$ 0s.

Finally, the target number t, the bottom row of the table, consist
followed by k 3s.

	1	2	3	4	$\cdots$	l	c_1	c_2	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_l						1	0	0	$\cdots$	0
z_l						1	0	0	$\cdots$	0
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$									$\ddots$	$\vdots$
g_k										1
h_k										1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

SUBSET-SUM Problem and Knapsack Problem

- The Subset-Sum problem can be seen as a specific instance of the Knapsack problem where each item's value is equal to its weight, and we would like to find if there's a combination of items that exactly fills the knapsack to its capacity (the target sum).

SUBSET-SUM ≤p KNAPSACK

Theorem:

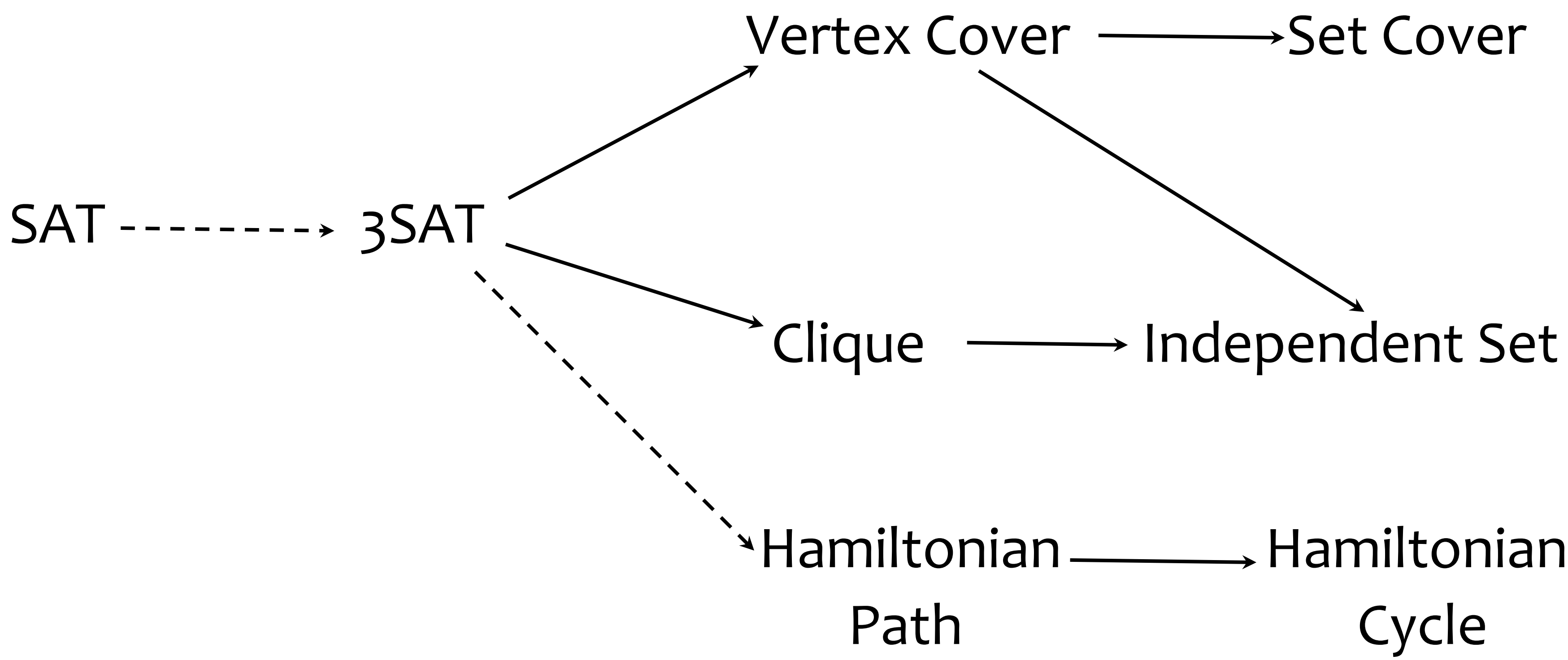
KNAPSACK problem is NP-complete.

Wrap Up

Will only show NP-hardness, again.

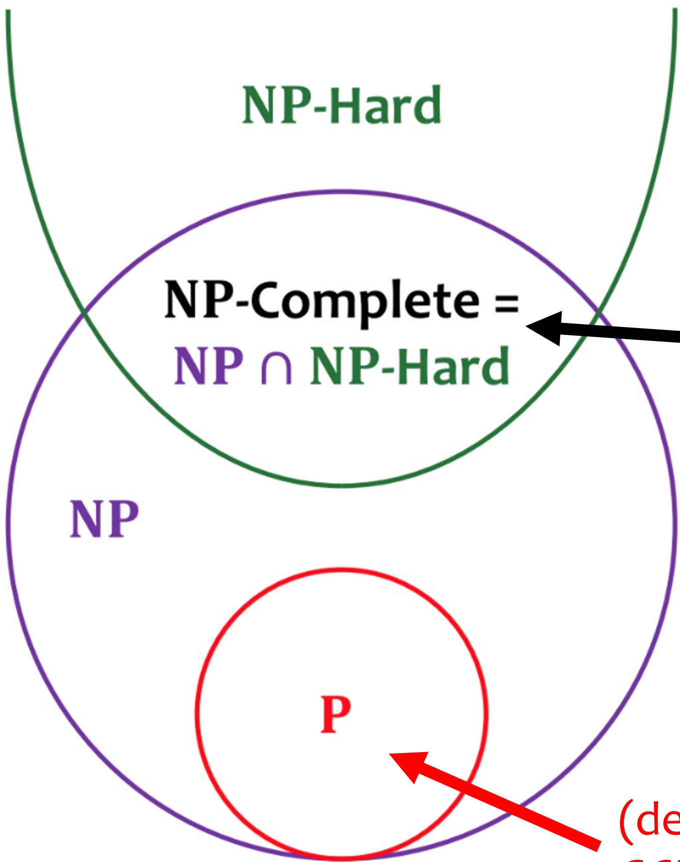
A Web of NP-Hard Problems

(all of these are also in NP, and therefore NP-Complete)



Classification of Problems: Efficient vs Inefficient

Assuming $P \neq NP$, we have classified problems into two classes



SAT (Cook-Levin)
3SAT
Vertex Cover, Clique, Knapsack, Set Cover, Independent Set, and many more…

(decision version of)
GCD, Multiplication, LIS, LCS, shortest path…

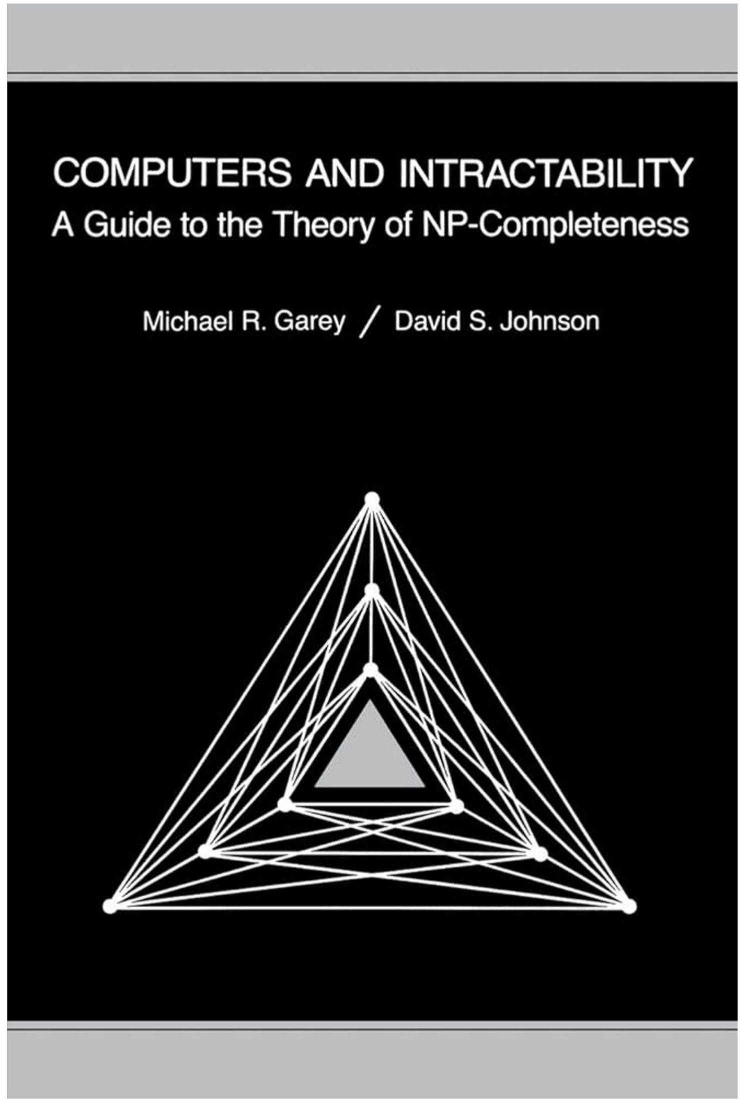
NP-Completeness is Everywhere

- **Constraint Satisfaction:** SAT, 3SAT
- **Routing:** Longest Path, Hamiltonian Path, Traveling Salesperson
- **Covering Problems:** Vertex Cover, Set Cover
- **Coloring Problem:** 3-Coloring a Graph
- **Scheduling Problems**
- **Social Networks:** Clique, Maximum Cut
- **Arithmetic Problems:** Subset Sum, Knapsack
- **Games:** Sudoku, Battleship, Super Mario, Pokémon

... ONE ALGORITHM WOULD SOLVE THEM ALL!

"About 20 diverse scientific disciplines were unsuccessfully struggling with some of their internal questions and came to recognize their intrinsic complexity when realizing that these questions are, in some form, NP-complete"

- Theory of Computing: a Scientific Perspective (Oded Goldreich, Avi Wigderson 1996)



Contains hundreds of NP-complete problems (1979)

- the most cited reference in the CS literature (> 80000 now)
- also contains the following comic…