NP COMPLETE PROBLEMS

INTRODUCTION AND DISCUSSIONS

THREE BASIC CONCEPTS

- **Decision Problems**: output yes/no
- **Nondeterministic algorithm**: certificate additional input
- **Polynomial transformation**: one problem input to another type problem input in poly-time

Decision Problems: Output is "True/False," "Yes/No"

- **Example (sorting-decision problem):**
  - **Input: a list of numbers L**
  - Output, an answer: is L sorted?

- **Complexity of Decision Problem?**
  - **Sorting problem P: lower bound Ω(n log n), but**
  - Sorting-decision problem P_D : O(n)

ADVANTAGE OF DECISION PROBLEMS

- Every problem *P* has a corresponding related decision problem P_D,
  - Has a lesser or equal complexity
  - Logical output to work with makes it formal
- Hence, complexity theory (or, any computer science formal theory) uses only decision problems
- But, the results are valid for all types of problems

EXAMPLE DECISION PROBLEM

- Example (0-1 Knapsack-decision problem):
  - Input: a list of objects with (weight, profit), a knapsack upper bound by weight M, and a profit lower bound *t*
  - Output: Does there exist a ks_profit ≥ *t* by choosing some objects with ks_wt ≤ *M*?

- Complexity?
- Which one is lower – original *01KS* or $01KS_D$?

- *$01KS_D$ algorithm: Backtrack just like before for 01KS but may terminate as soon as a valid ks_profit ≥ t found, need not go over all branches.*

DECISION PROBLEMS

- Theory developed in terms of yes/no problems
  - Independent set
    - *Given a graph G and an integer K, does G have an independent set of size at least K*
  - Vertex cover
    - *Given a graph G and an integer K, does the graph have a vertex cover of size at most K.*

USE OF DECISION PROBLEMS

- Algorithm A(P_D) for a *decision problem* P_D may be used to solve *original problem* P, and vice versa

- Example of sorting, using sorting-decision algorithm:
  - Sort (input *L)*:
    - For each permutation *p* of input list *L*
    - if AlgoSorting$_D$ (*p*) == *True* then return *p*;

  - Sorting$_D$ *(*input *L): If any comparison within the algorithm Sorting(L) leads to an actual exchange of elements, then return "False" (L was not sorted)*

- Similarly Alg(01KS$_D$) may be used to solve 0-1KS: *Hint: binary search over ks_profit bound variable t*

NON-DETERMINISTIC VERSION OF A PROBLEM

- Presume a *suggested* solution is input: "certificate"

- Problem is simplified: Check only the certificate,
  Is the certificate correct answer? – return "True/False"

- If an algorithm can *check certificate* in polynomial time
- => The problem is *Non-deterministically Polynomial*,
- *in NP-class*

WHAT IS NON-DETERMINISTICALLY POLYNOMIAL!!

Imagine you're a treasure hunter, and you've found a map that leads to a treasure buried on a vast island. The challenge is that the map doesn't point to one specific location but rather gives clues that could match many places on the island. Your task is to find the exact spot where the treasure is buried.

WHAT IS NON-DETERMINISTICALLY POLYNOMIAL!! APPROACH -1

Starting at one end of the island and digging at every single spot that matches the clues on the map, one by one, until you find the treasure. If the island is huge and the clues are vague, **searching every possible location could take a very long time**.

This method is thorough and guarantees that you'll eventually find the treasure, but it's incredibly slow and inefficient, much like solving a complex problem without any shortcuts or efficient algorithms.

WHAT IS NON-DETERMINISTICALLY POLYNOMIAL!! APPROACH -II

Now, imagine if you had a **magical intuition** that could instantly "guess" a promising spot where the treasure might be buried, without having to check every location one by one. You don't know why, but you feel strongly about this particular spot.

The magic here isn't in digging (which still takes effort) but in **somehow knowing where to dig**. Once you guess a spot, you still have to verify it by digging there to see if the treasure is indeed buried there.

The "verification" process (digging in this case) takes some time but is much quicker than checking the entire island blindly.

WHAT IS NON-DETERMINISTICALLY POLYNOMIAL!!

- In computational terms, if you're given a solution to a problem (like the "right spot" on the island), you can verify whether it's correct relatively quickly and efficiently, even though finding that solution in the first place (guessing the right spot) might seem almost magically difficult.

- So, when we say a problem is in **NP (Nondeterministic Polynomial time**), we mean that
  - if we were somehow able to come up with a potential solution out of thin air, we could verify its correctness in polynomial time, which is a fancy way of saying "reasonably quickly" based on the size of the input.
  - However, the class doesn't necessarily provide a method for quickly finding or guessing that solution in the first place.

CERTIFICATE EXAMPLES

- Independent set of size K
  - The Independent Set
- Satifisfiable formula
  - Truth assignment to the variables
- Hamiltonian Circuit Problem
  - A cycle including all of the vertices
- K-coloring a graph
  - Assignment of colors to the vertices

NON-DETERMINISTIC PROBLEMS: EXAMPLE

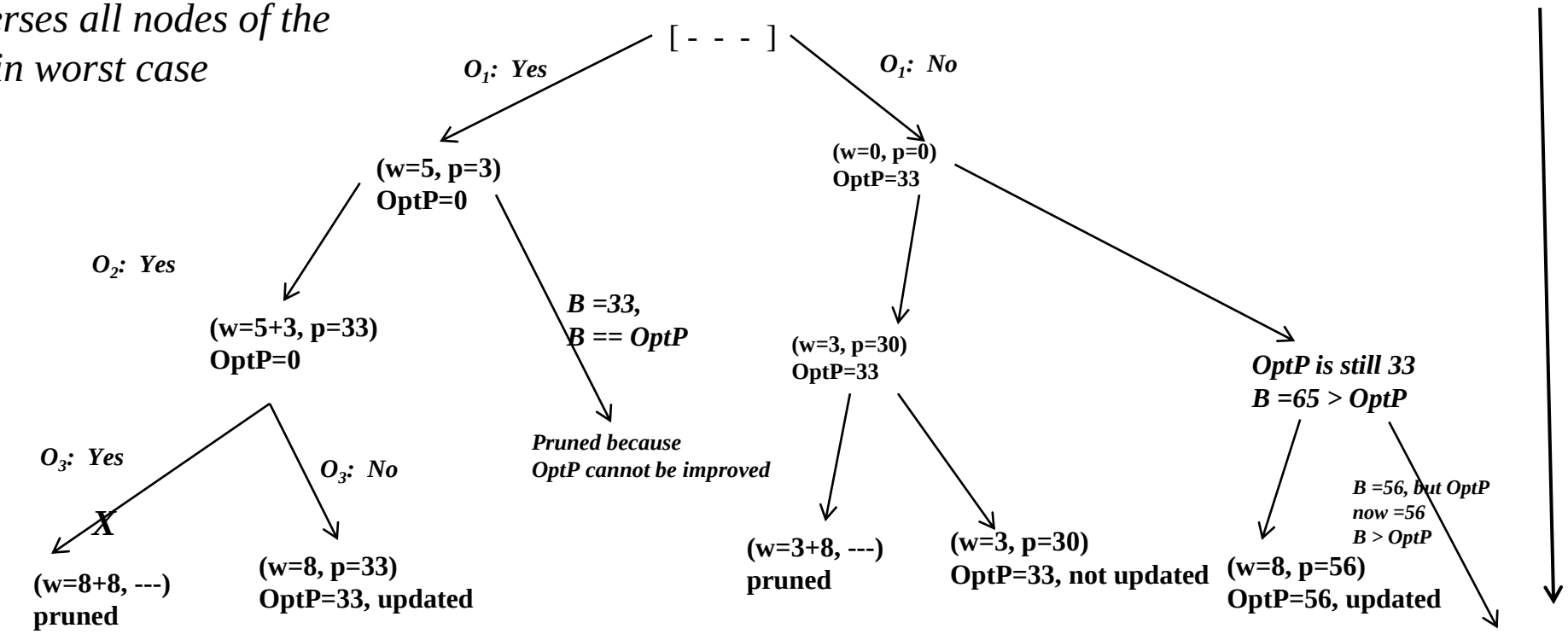
- Sorting is an NP-class problem:
  - Input: (1) a list of numbers *L*, and (2) a certificate *c* (another list)
  - Output: Is *c* a sort of *L?*
  - Algorithm? …
  - Can you find a polynomial-time algorithm?
  - If your answer is yes: Sorting_D problem is in *NP-class*

- *Is Sorting also a P-class problem?*

NON-DETERMINISIM IS NOT A MISTERY

Non-deterministic algorithm: Given the certificate, it traverses only one path from root to a leaf: bounded by depth of the tree

Deterministic algorithm: Traverses all nodes of the tree in worst case



Deterministic Machine versus Non-deterministic Machine

ALGORITHMS VS. LOWER BOUNDS

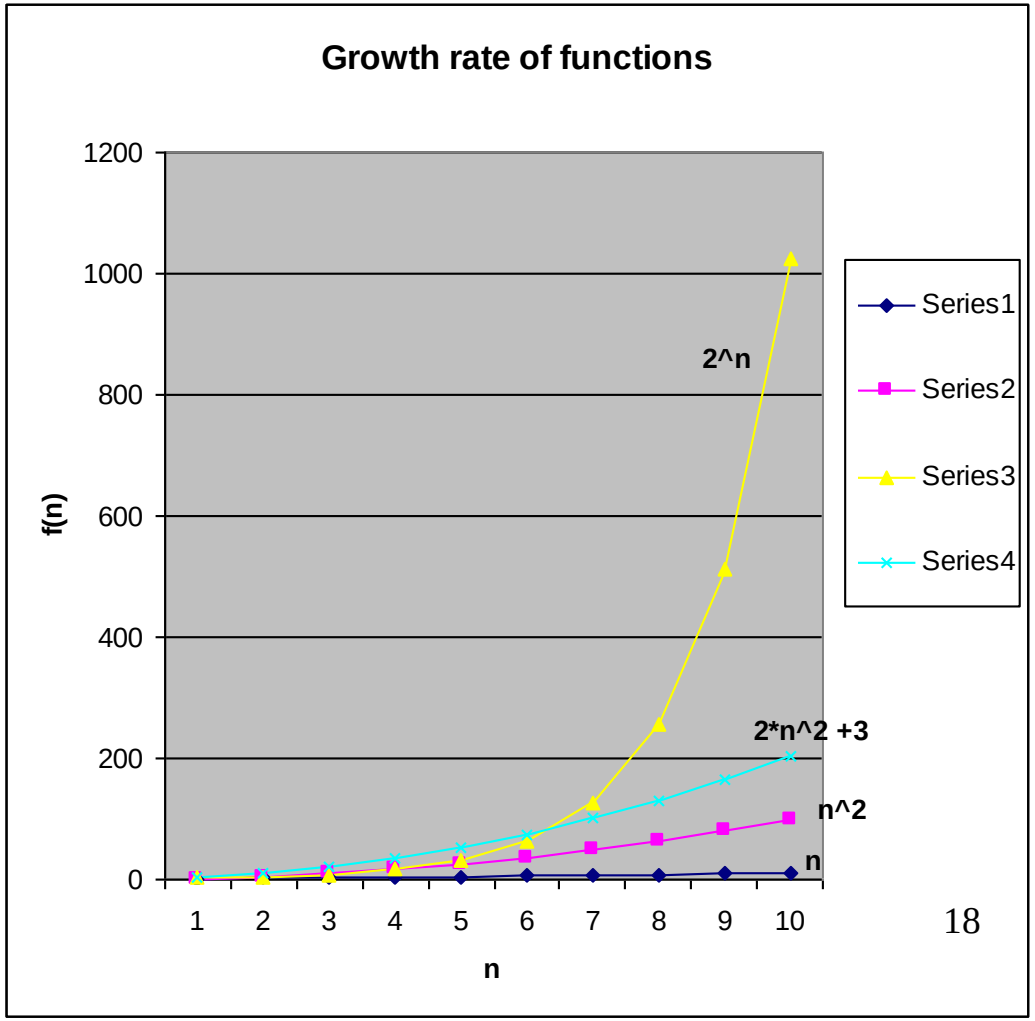
- Algorithmic Theory
  - What we can compute
    - I can solve problem X with resources R
  - Correctness Proof
- Lower bounds
  - How do we show that something can’t be done?

POLYNOMIAL TIME

- P Class of problems that can be solved in polynomial time
  - Corresponds with problems that can be solved efficiently in practice
  - Right class to work with “theoretically”

CLASSIFICATION OF PROBLEMS BY COMPLEXITY

- Polynomial algorithms are “more efficient” than Exponential algorithms, or *correctly speaking:*
  - P-algorithms has slower growth rate than that of Exp-algorithms, with respect to input size *n*
  - $O(n^k) < O(a^n)$, *a>1* and *k≥0* are constants



P-CLASS PROBLEMS AND BEYOND …

- ***P-class of problems***: which have asymptotic Polynomial-time algorithms $O(n^k)$, for $k \geq 0$

- There is no "*Exp-class*" problems: No problem *S* may be designated as in Exp-class just because no polynomial algorithm for *S* has been found *yet*!

- Even if you have an exponential algorithm $O(a^n)$, for $a > 1$, that does not mean a polynomial algorithm will also not exist!

- We are talking about provable lower bound Ω

WHAT IS NP?

- Problems solvable in non-deterministic polynomial time . . .
- Problems where “yes” instances have polynomial time checkable certificates

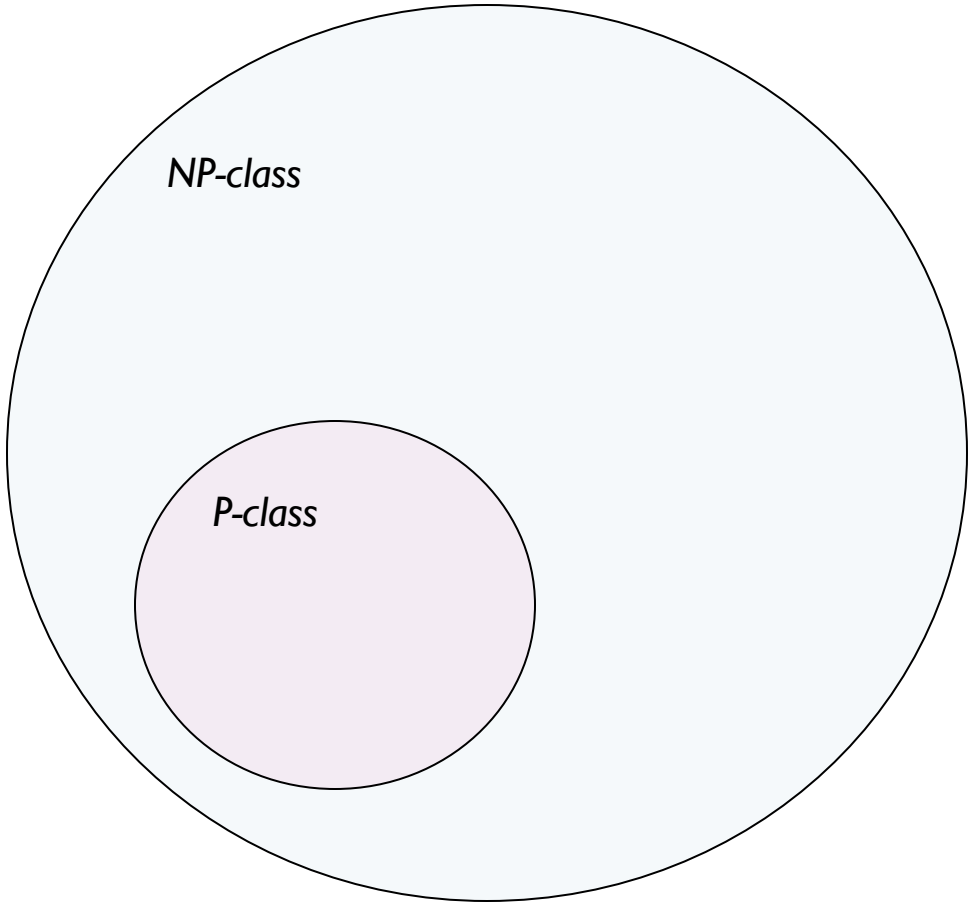
P VERSUS NP

- Say, problem *X* is in *P-class* (*has a polynomial algorithm*)

- Is *X* in *NP-class* as well?
  - *Does X have non-deterministic Polynomial algorithm also*?
  - *Given a certificate for X, can you check it in polynomial-time*?

P VERSUS NP

Which one subset of which?
Polynomial class, or Non-det polynomial class?



P VERSUS NP
$P \subseteq NP$ TRUE, TRIVIAL

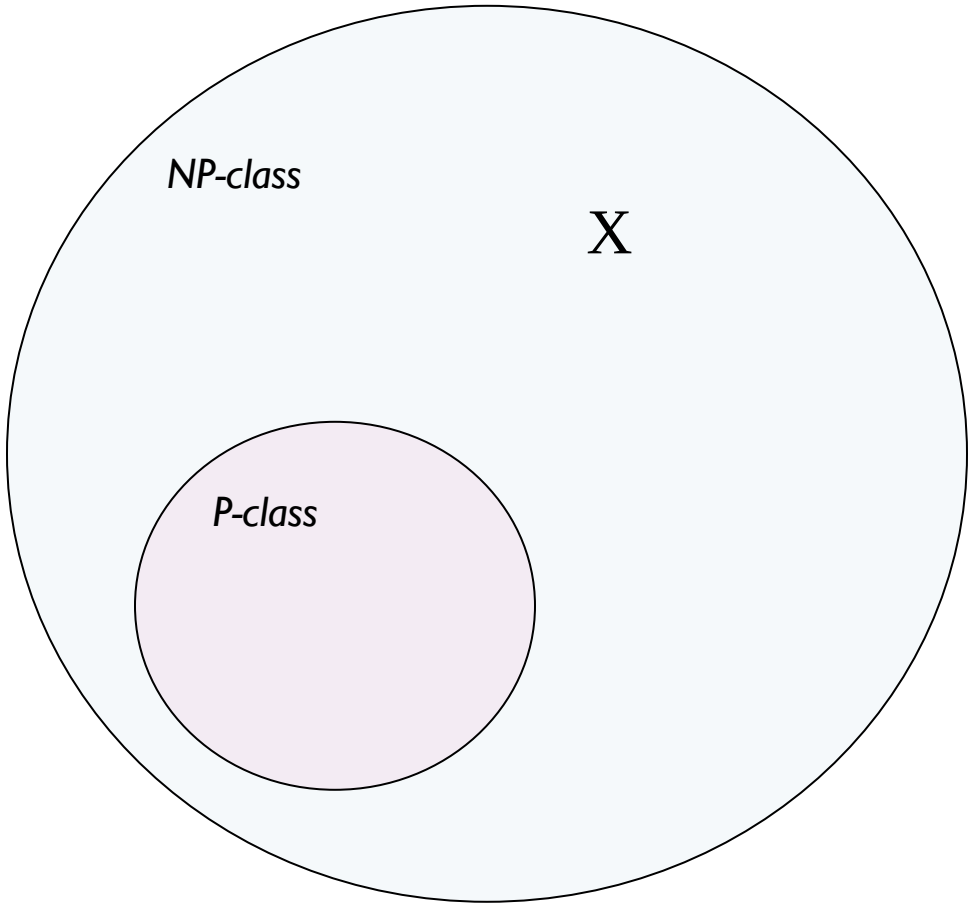
P VERSUS NP

- $P \subseteq NP$ is True.
- But is $NP \subseteq P$ also true? Meaning: $NP \equiv P$?
- *A million dollar question:*
  http://www.claymath.org/millennium-problems
- *Jury is out there:*
  - $NP \equiv P$ *(i.e.,* $NP \subseteq P$ *and* $P \subseteq NP$*),* XOR, $NP \neq P$?

P VERSUS NP

How would you prove: *NP* $\subseteq$ *P NOT TRUE?*

OR, THAT THE TRUTH IS P $\subset$ *NP, A PROPER SUBSET*



P VERSUS NP LEADS TO COMPLEXITY HIERARCHY

- *Jury is out there:*
  - *Prove, NP $\subseteq$ P, and so, NP $\equiv$ P?*
  - *Or, show P $\neq$ NP?*

- *Did not happen yet!*

- But, can we find a "hardest" problem *H* in NP class?

- And, then maybe look for a Poly-algorithm for that *H*?
- Or, find exponential lower bound for *H*?

- Looking for "hardest" needs some concepts on *Polynomial Transformation (another new concept!!)*

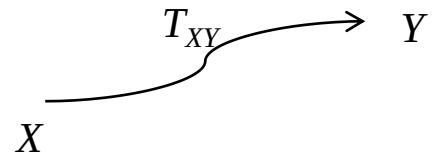
POLYNOMIAL PROBLEM-TRANSFORMATION

- How do you formally express a problem?
- Problem: (Input, Output) unambiguously expressed
- Problem instance: Specific values for Input,
  - Remember: Output is T/F in decision problems

POLYNOMIAL PROBLEM-TRANSFORMATION

- Problem *X can be transformed* to another problem *Y*

- Problem Transformation: an algorithm
  - T_{XY} (input of X) := (input of Y)
  - Such that, corresponding output of *X* and *Y* are same (T/F)
  - T_{XY} is an algorithm, like any other ones we have seen



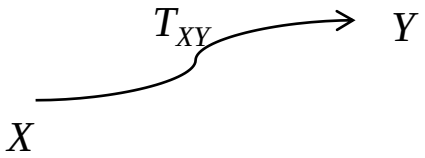
POLYNOMIAL PROBLEM-TRANSFORMATION

- Problem Transformation: an algorithm T_{XY}
  - T_{XY}(input of X) => (input of Y)
  - Such that, output T/F for (input of *X)* = output for T_{XY}(input X)

- If there exists a T_{XY},

  and its time-complexity is Polynomial,

- then *X is polynomially transformed to Y*



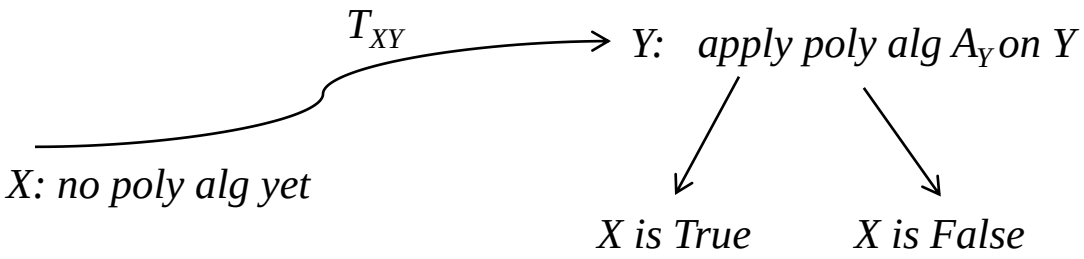
SIGNIFICANCE OF POLYNOMIAL TRANSFORMATION

- If there exists a T_{XY}, and time-complexity of it is Polynomial, then *X is polynomially transformable to Y*

- Suppose, *Y* has a <u>polynomial</u> algorithm A_y

  but, *NO* polynomial algorithm exists for *X,*

  YET!



- However, now you can solve *X* by
  - $A_y(T_{XY}(X))$ → *T/F*
  - *What is the time-complexity of this dual algorithm?*

A CATCH IN POLYNOMIAL TRANSFORMATION

- $A_Y(T_{XY}(X))$ → *T/F*
  - *What is the time-complexity of this dual algorithm?*

- *Note:* $|(Y \leftarrow output(T_{XY}(X))|$ is a polynomial function of $|X|$,
  - *where | | indicates problem size*

- $A_y(Y)$ takes polynomial-time with respect to |Y|
- *But,* |Y| is a polynomial of |X|, as said above

- *Hence:* $A_y(Y)$ above takes *polynomial of polynomial-time* over $|X|$,
- Or, polynomial-time with respect to $|X|$

SIGNIFICANCE OF POLYNOMIAL TRANSFORMATION

- $A_y(T_{XY}(X))$ → *T/F*
  - Time-complexity of this dual algorithm is polynomial

- *So, X* is <u>now</u> in P-class
  - *Because Y was in P-class and polynomial transformation T_{XY} is found*
  - *So, X has an indirect polynomial algorithm now*

- *<u>Note, this is not true in the opposite direction</u>*

SIGNIFICANCE OF POLYNOMIAL TRANSFORMATION

- $A_y(T_{XY}(X))$ => *T/F*
  - *If A_y is polynomial, then the time-complexity of this dual algorithm is also polynomial*

- If *Y* is in P-class, so is *X*
- *Y* is at least as "hard" as *X*

- Note, opposite direction is not necessarily true:
  - If, source problem *X* is already in P-class (has poly alg),
  - Then, polynomial transformation T_{XY} does not say anything about the classification of target problem *Y*

POLYNOMIAL TIME REDUCTIONS

- Y is Polynomial Time Reducible to X
  - Solve problem Y with a polynomial number of computation steps and a polynomial number of calls to a black box that solves X
  - Notations: $Y <_P X$

LEMMA

Suppose $Y \leq_P X$. If X can be solved in polynomial time, then Y can be solved in polynomial time.

LEMMA

Suppose $Y <_P X$. If Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.

NP-COMPLETENESS

- A problem X is NP-complete if
  - X is in NP
  - For every Y in NP, $Y <_P X$

- X is a "hardest" problem in NP

- If X is NP-Complete, Z is in NP and $X <_P Z$
  - Then Z is NP-Complete

COOK'S THEOREM

The Circuit Satisfiability Problem is NP-Complete

HISTORY

- Jack Edmonds
  - Identified NP
- Steve Cook
  - Cook's Theorem – NP-Completeness
- Dick Karp
  - Identified "standard" collection of NP-Complete Problems
- Leonid Levin
  - Independent discovery of NP-Completeness in USSR

POPULATING THE NP-COMPLETENESS UNIVERSE

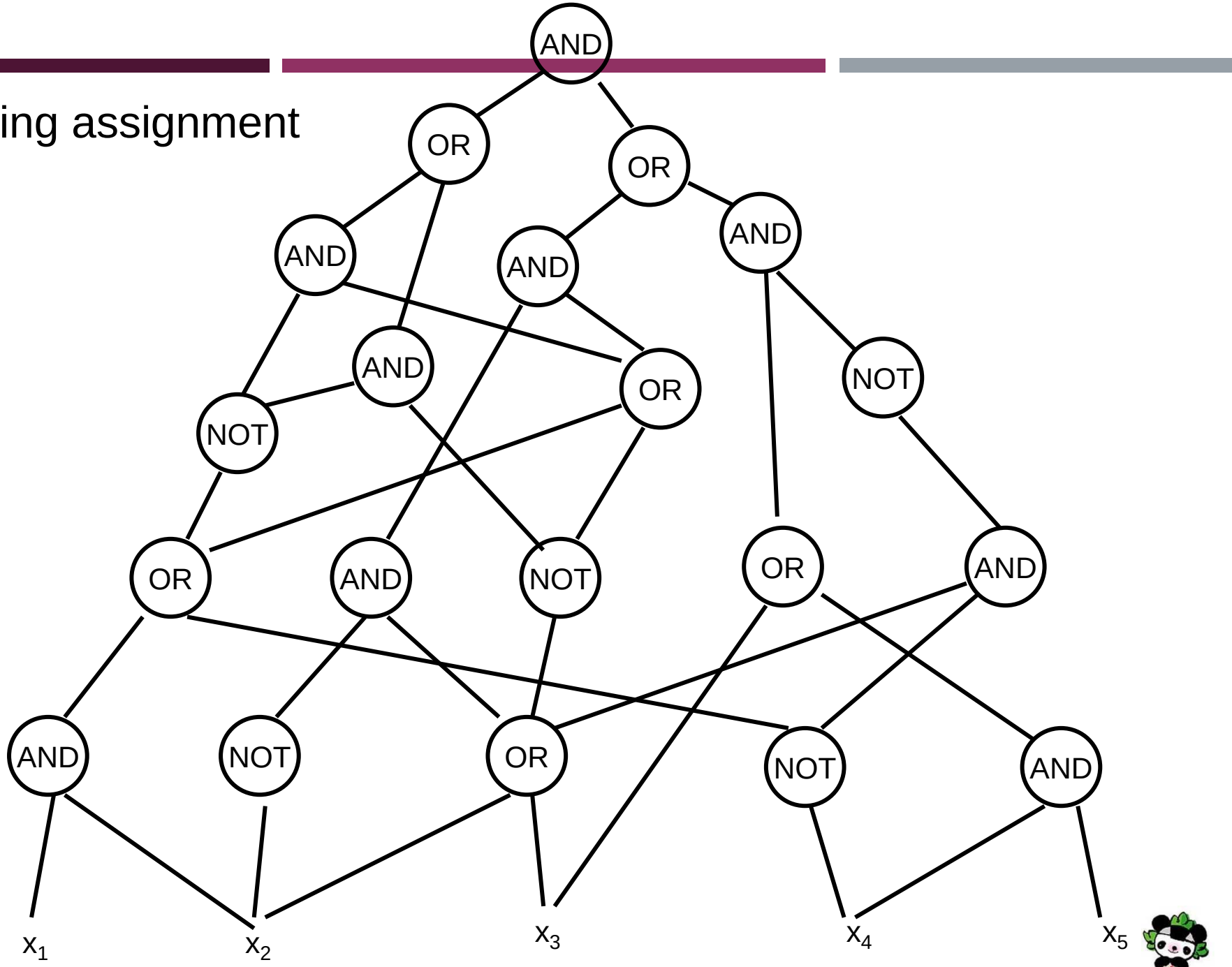
- Circuit Sat $<_P$ 3-SAT
- 3-SAT $<_P$ Independent Set
- 3-SAT $<_P$ Vertex Cover
- Independent Set $<_P$ Clique
- 3-SAT $<_P$ Hamiltonian Circuit
- Hamiltonian Circuit $<_P$ Traveling Salesman
- 3-SAT $<_P$ Integer Linear Programming
- 3-SAT $<_P$ Graph Coloring
- 3-SAT $<_P$ Subset Sum
- Subset Sum $<_P$ Scheduling with Release times and deadlines

COOK'S THEOREM

- The Circuit Satisfiability Problem is NP-Complete

- Circuit Satisfiability
  - Given a boolean circuit, determine if there is an assignment of boolean values to the input to make the output true

Find a satisfying assignment



PROOF OF COOK'S THEOREM

- Reduce an arbitrary problem Y in NP to X
- Let A be a non-deterministic polynomial time algorithm for Y
- Convert A to a circuit, so that Y is a Yes instance iff and only if the circuit is satisfiable

SATISFIABILITY

Given a boolean formula, does there exist a truth assignment to the variables to make the expression true

DEFINITIONS

- Boolean variable: $x_1, \ldots, x_n$
- Term: x_i or its negation $!x_i$
- Clause: disjunction of terms
  - t_1 or t_2 or … t_j
- Problem:
  - Given a collection of clauses $C_1, \ldots, C_k$, does there exist a truth assignment that makes all the clauses true
  - $(x_1$ or $!x_2)$, $(!x_1$ or $!x_3)$, $(x_2$ or $!x_3)$

3-SAT

- Each clause has exactly 3 terms
- Variables $x_1, \ldots, x_n$
- Clauses $C_1, \ldots, C_k$
  - $C_j = (t_{j1}$ or t_{j2} or $t_{j3})$

- Fact: *Every instance of SAT can be converted in polynomial time to an equivalent instance of 3-SAT*

FIND A SATISFYING TRUTH ASSIGNMENT

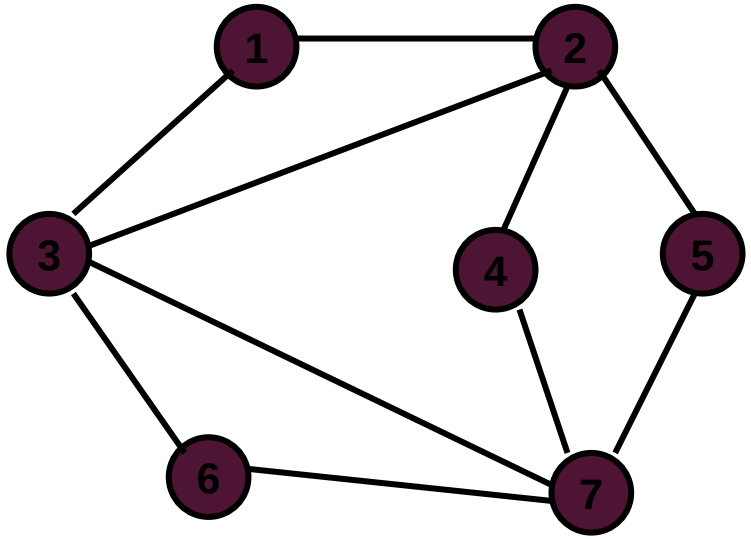
(x || y || z) && (!x || !y || !z) && (!x || y) && (x || !y) && (y || !z) && (!y || z)

THEOREM: CIRCUITSAT $<_P$ 3-SAT

THEOREM: 3-SAT $<_P$ INDSET

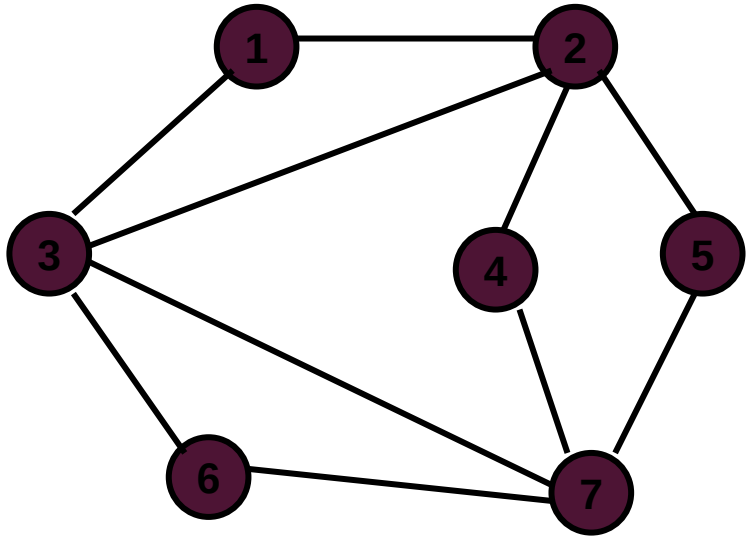
SAMPLE PROBLEMS

- Independent Set
  - Graph G = (V, E), a subset S of the vertices is independent if there are no edges between vertices in S



VERTEX COVER

- Vertex Cover
  - Graph G = (V, E), a subset S of the vertices is a vertex cover if every edge in E has at least one endpoint in S



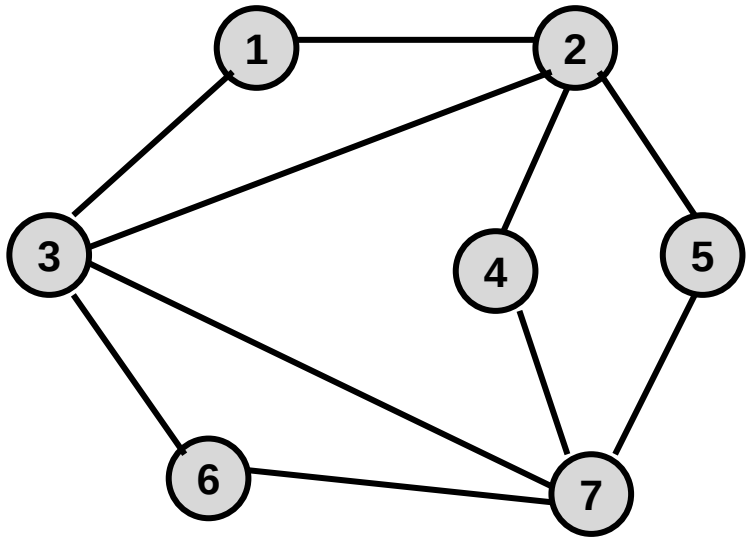
IS $<_P$ VC

Lemma: A set S is independent iff V-S is a vertex cover

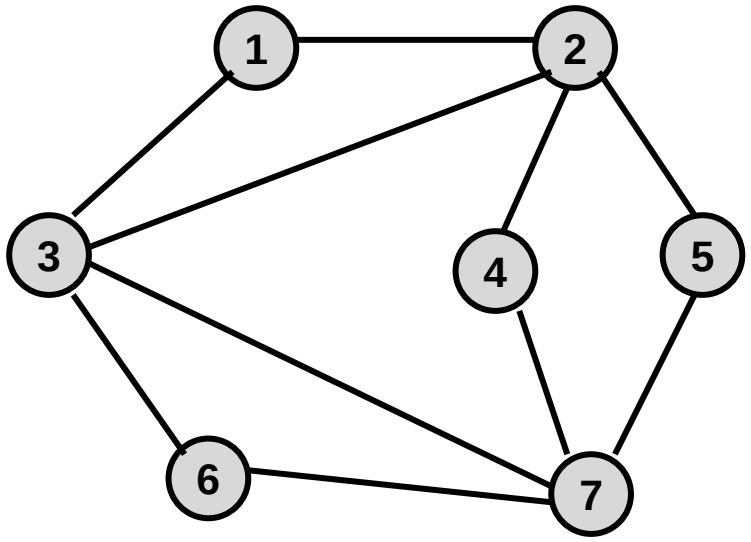
To reduce IS to VC, we show that we can determine if a graph has an independent set of size K by testing for a Vertex cover of size n - K

IS $<_P$ VC

Find a maximum independent set S

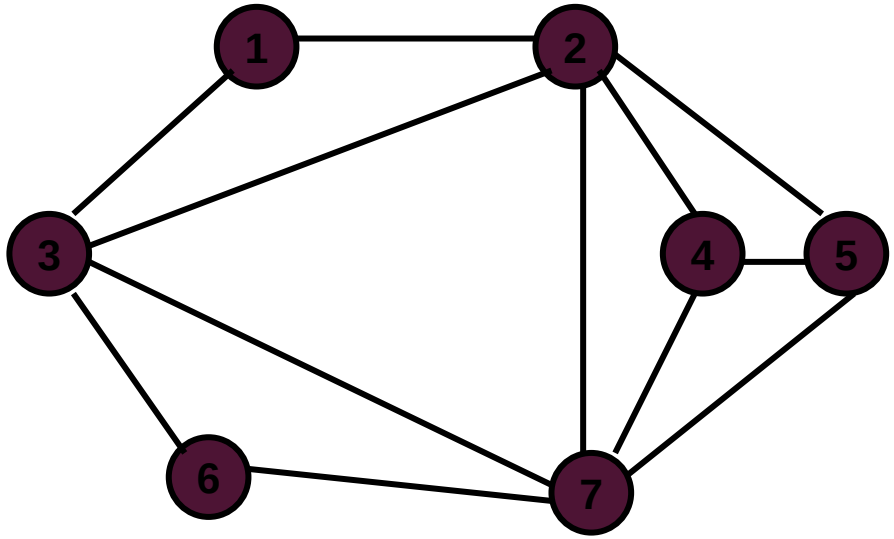


Show that V-S is a vertex cover



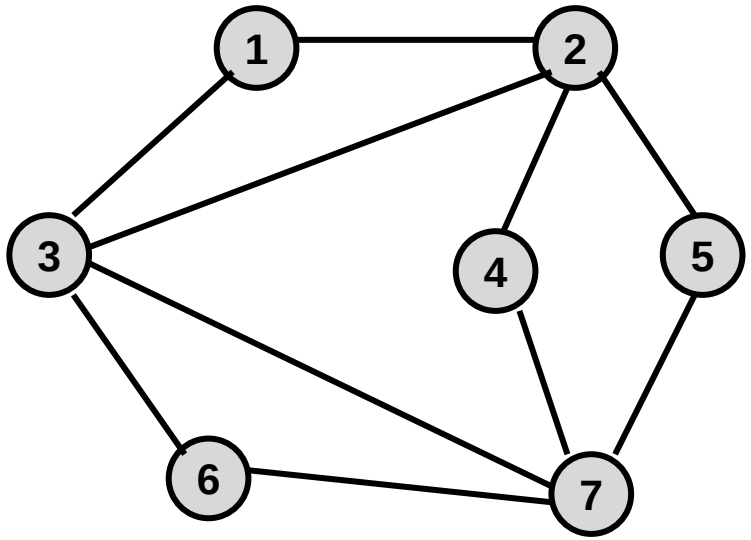
CLIQUE

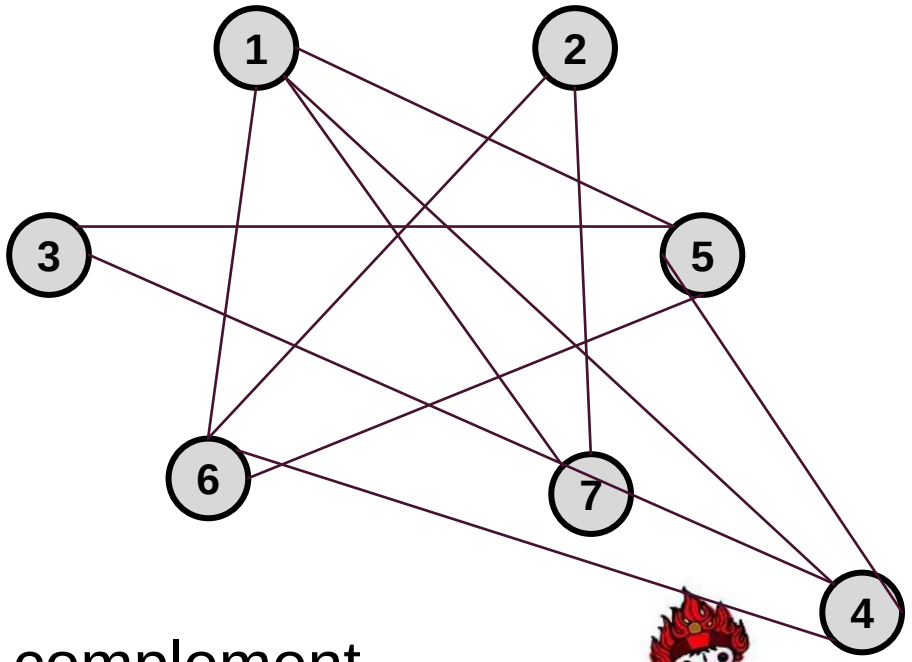
- Clique
  - Graph G = (V, E), a subset S of the vertices is a clique if there is an edge between every pair of vertices in S



COMPLEMENT OF A GRAPH

- Defn: G'=(V,E') is the complement of G=(V,E) if (u,v) is in E' iff (u,v) is not in E





Construct the complement

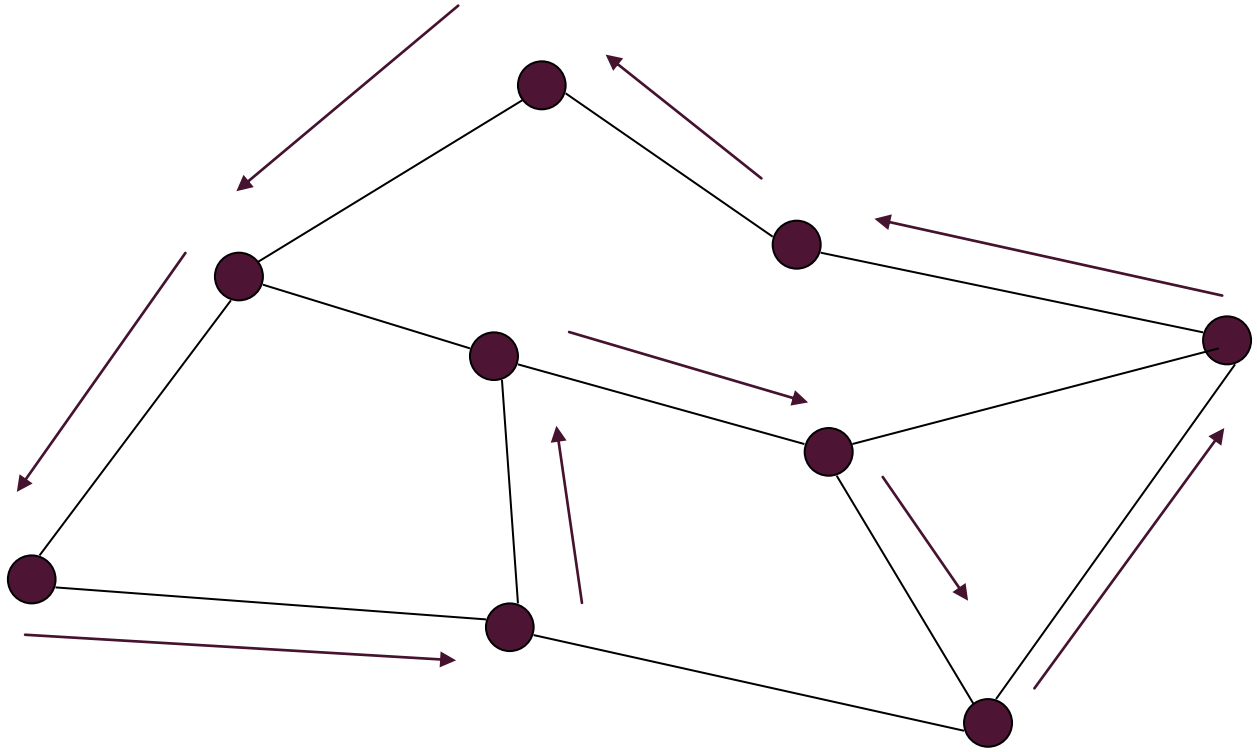
IS $<_P$ CLIQUE

- Lemma: S is Independent in G iff S is a Clique in the complement of G

- To reduce IS to Clique, we compute the complement of the graph. The complement has a clique of size K iff the original graph has an independent set of size K

HAMILTONIAN CIRCUIT PROBLEM

- Hamiltonian Circuit – a simple cycle including all the vertices of the graph

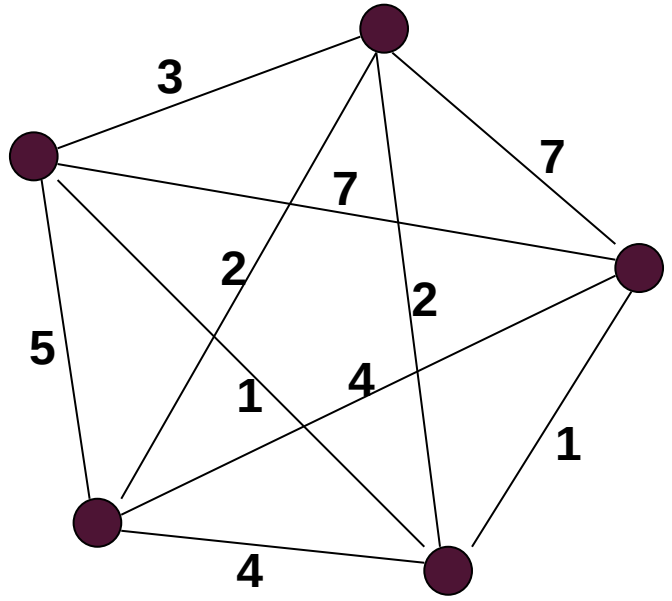


THM: HAMILTONIAN CIRCUIT IS NP COMPLETE

- Reduction from 3-SAT

TRAVELING SALESMAN PROBLEM

- Given a complete graph with edge weights, determine the shortest tour that includes all of the vertices (visit each vertex exactly once, and get back to the starting point)



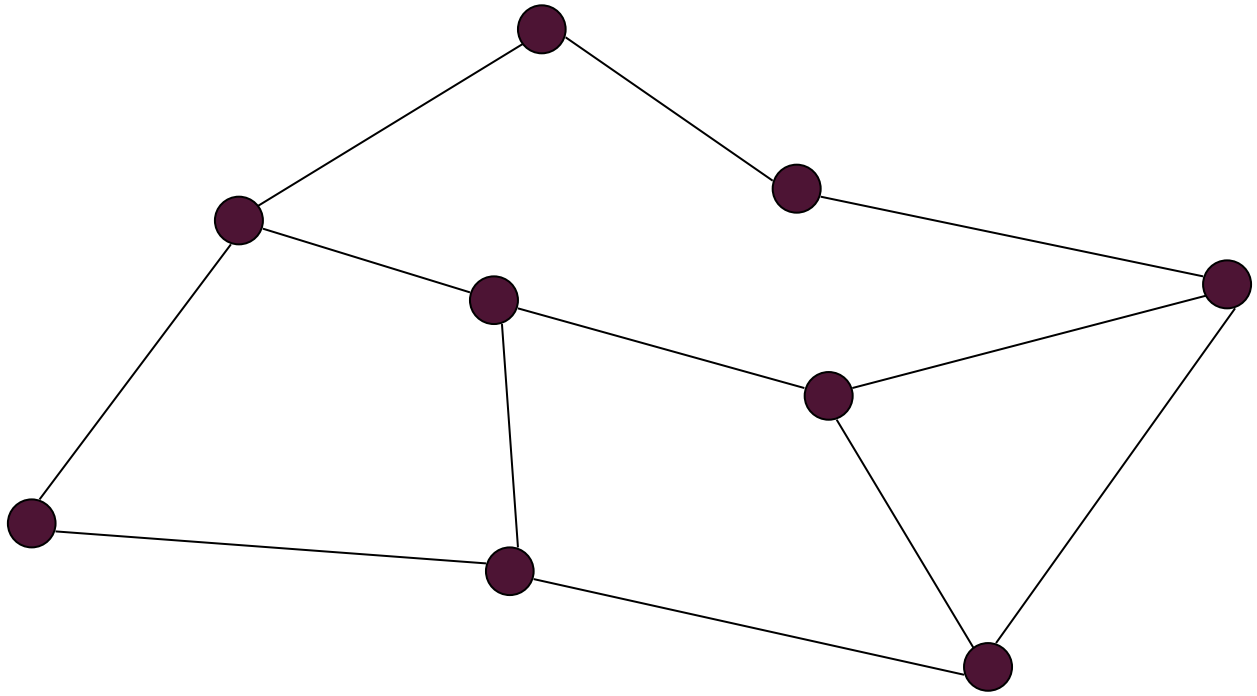
Find the minimum cost tour

NP-COMPLETENESS REDUCTIONS

If X is NP-Complete, Y is in NP,

and $X <_P Y$, then Y is NP-Complete

HAMILTONIAN CIRCUIT, HAMILTONIAN PATH



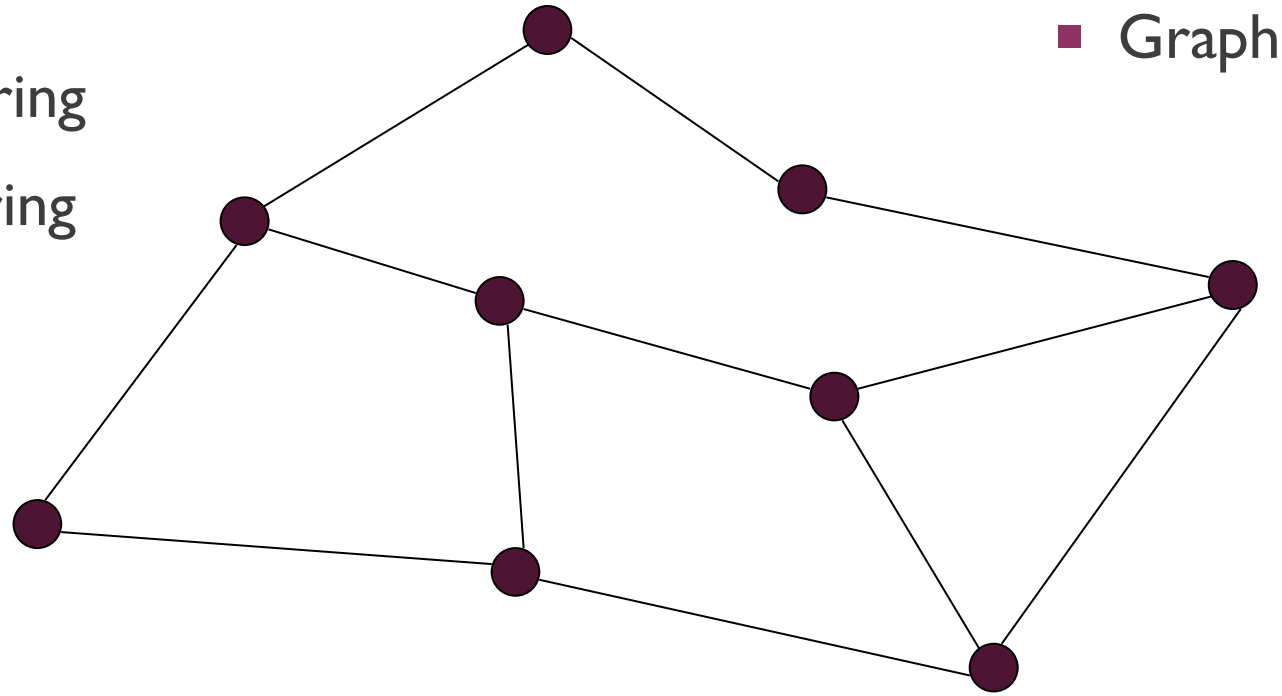
How do you show that Hamiltonian Path is NP-Complete?

PROBLEM DEFINITION

- Given a graph G, does G have an independent set?
- Given a graph G, does G have an independent set of size 7?
- Given a graph G, and an integer K, does G have an independent set of size K?

GRAPH COLORING

- NP-Complete
  - Graph K-coloring
  - Graph 3-coloring
- Polynomial
  - Graph 2-Coloring



NUMBER PROBLEMS

- Subset sum problem
  - Given natural numbers $w_1, \ldots, w_n$ and a target number W, is there a subset that adds up to exactly W?

- Subset sum problem is NP-Complete
- Subset Sum problem can be solved in O(nW) time

SUBSET SUM PROBLEM

- The reduction to show Subset Sum is NP-complete involves numbers with n digits
- In that case, the O(nW) algorithm is an exponential time and space algorithm

WHAT IS NP?

- Problems where ‘yes’ instances can be efficiently verified
  - Hamiltonian Circuit
  - 3-Coloring
  - 3-SAT

- Succinct certificate property

WHAT ABOUT 'NEGATIVE INSTANCES'

- How do you show that a graph does not have a Hamiltonian Circuit
- How do you show that a formula is not satisfiable?

WHAT WE DON'T KNOW

- P vs. NP