

NP-Completeness

Part 1

NP-Completeness

- So far we've seen a lot of good news!
 - Many of the problems we have seen could be solved quickly (i.e., in close to linear time, or at least a time that is some small polynomial function of the input size)
- NP-completeness is a form of bad news!
 - Evidence that many important problems can not be solved quickly.
- NP-complete problems really come up all the time!
 - 0-1 Knapsack, Travelling Salesman, Bin Packing, Scheduling

NP-Completeness

- Some problems are *intractable*:
as they grow large, we are unable to solve them in reasonable time
- What constitutes reasonable time? Standard working definition: *polynomial time*
 - On an input of size n the worst-case running time is $O(n^k)$ for some constant k
 - Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
 - Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$

Why should we care?

- Knowing that they are hard lets you use other methods to solve them...
 - **Use a heuristic:** come up with a method for solving a reasonable fraction of the common cases.
 - **Solve approximately:** come up with a solution that you can prove that is close to right.
 - **Use an exponential time solution:** if you really have to solve the problem exactly and stop worrying about finding a better solution.

Optimization vs Decision Problems

- **Decision problems**

- Given an input and a question regarding a problem, determine if the answer is yes or no

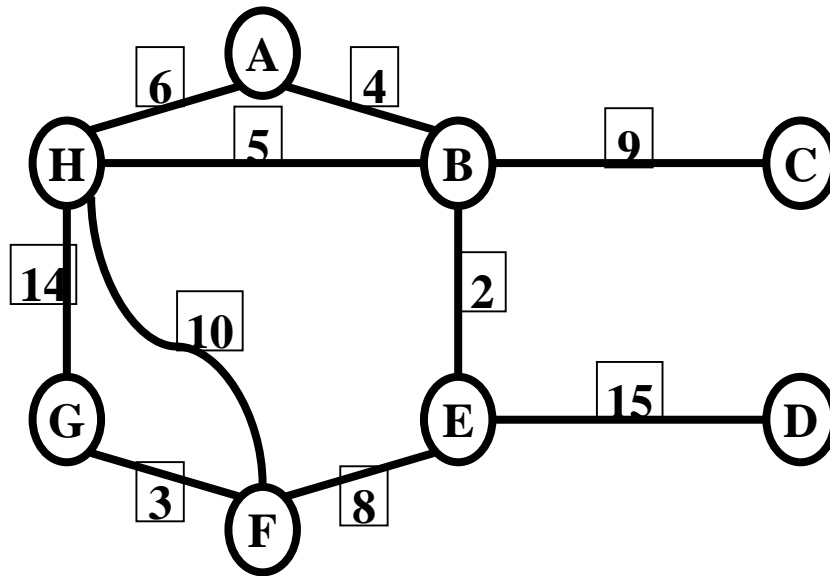
- **Optimization problems**

- Find a solution with the “best” value
- Optimization problems can be casted as decision problems that are easier to study

Optimization vs Decision Problems

Shortest Path given $G=(V,E)$ and $w(u,v)$ edge weights

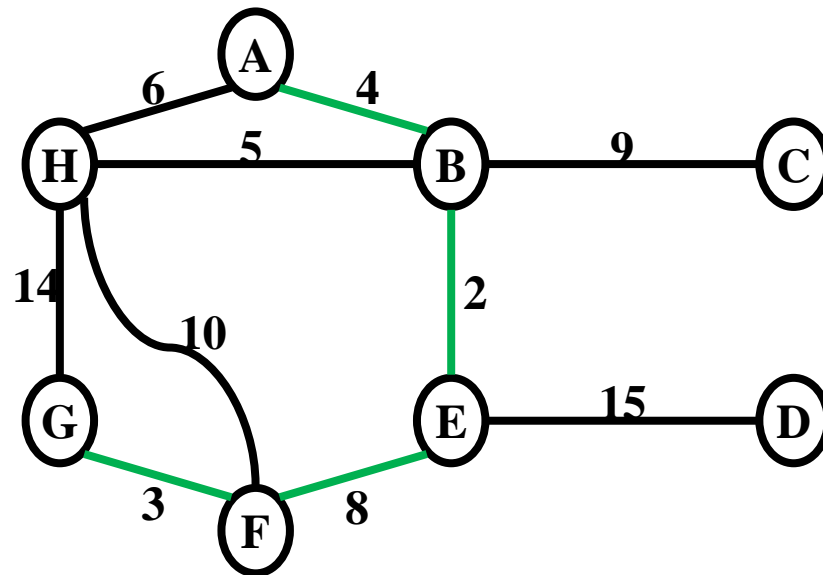
- **Optimization:** What is the minimum total weight of all paths between A and G?
- **Decision:** Does there exist a path between A and E with total weight at most 20?



Optimization vs Decision Problems

Shortest Path given $G=(V,E)$ and $w(u,v)$ edge weights

- **Optimization**: What is the minimum total weight of all paths between A and G? **17**
- **Decision**: Does there exist a path between A and E with total weight at most 20? **YES**



Optimization vs Decision Problems

Problem: Knapsack (items, weights, benefits, W)

- **Optimization:** What is the maximum total benefit of all sets of items that can fit in a knapsack with capacity W .
- **Decision:** Does there exist a set of items having a total benefit of at least k that can fit in the knapsack with capacity W

Algorithmic vs Problem Complexity

- The *algorithmic complexity* of a computation is some measure of how *difficult* it is to perform the computation (i.e., specific to an algorithm)
- The *complexity of a computational problem* or *task* is the complexity of the algorithm with the **lowest** order of growth of complexity for solving that problem or performing that task.
 - e.g. the problem of searching an ordered list has *at most* $\lg n$ time complexity.
- **Computational Complexity**: deals with classifying problems by how hard they are.

Class of “P” Problems

- **Class P** consists of (decision) problems that are solvable in polynomial time
- Polynomial-time algorithms
 - Worst-case running time is $O(n^k)$, for some constant k
- Examples of polynomial time:
 - $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
 - Searching and Sorting

Tractable/Intractable Problems

- Problems in P are also called **tractable**
- Problems **not** in P can be **or**
 - **intractable** - solved in reasonable time only for small inputs
 - **unsolvable** - can not be solved at all
- Are non-polynomial algorithms always worse than polynomial algorithms?
 - $n^{1,000,000}$ is *technically* tractable, but really impossible
 - $n^{\log \log \log n}$ is *technically* intractable, but easy

An Unsolvable Problem

- Turing discovered in the 1930's that there are problems **unsolvable** by *any* algorithm.
- The most famous of them is the **halting problem**
 - Given an arbitrary algorithm and its input, will that algorithm eventually halt, or will it continue forever in an “*infinite loop?*”
- This is an interesting topic but we will not be covering unsolvable problems in this class. To learn more you can take CS321

Intractable Problems

- Can be classified in various categories based on their degree of difficulty, e.g.,
 - NP
 - NP-complete
 - NP-hard
- Let's define NP algorithms and NP problems ...

Examples of Intractable Decision Problems

- **Hamiltonian Cycle (HAM-CYCLE).** Given a directed graph $G = (V, E)$, does there exist a simple cycle C that visits every vertex?
- **CIRCUIT-SAT.** Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?
- **Travelling Salesman (TSP).** Given a weighted graph $G=(V, E)$
 - Optimization Problem: Find a minimum weight Hamiltonian Cycle.
 - Decision Problem: Given a graph and an integer k , is there a Hamiltonian Cycle with a total weight at most k

Nondeterminism and NP Algorithms

Nondeterministic algorithm = two stage procedure:

1) Nondeterministic (“guessing”) stage:

generate randomly an arbitrary string that can be thought of as a candidate solution (“certificate”)

2) Deterministic (“verification”) stage:

take the certificate and the instance to the problem and returns YES if the certificate represents a solution

NP algorithms (Nondeterministic polynomial)

verification stage is polynomial

Class of “NP” Problems

- **Class NP** consists of problems that could be solved by Nondeterministic Polynomial algorithms
Or verifiable in polynomial time
- If we were given a “certificate” of a solution, we could verify/certify that the certificate is correct in time polynomial to the size of the input
- Warning: NP does **not** mean “non-polynomial”

Decision 0-1 Knapsack is in NP

Given a knapsack with capacity $W = 20$ is there a subset of items with total benefit at least \$25?

Easy to verify in poly-time that

$S = \{ 1, 3, 4, 5 \}$ is a certificate solution.

Total weight = $2 + 4 + 5 + 9 = 20$

Total benefit =

$$3 + 5 + 8 + 10 = 26 > 25$$

	Weight	Benefit
Item #	w_i	b_i
1	2	3
2	3	4
3	4	5
4	5	8
5	9	10

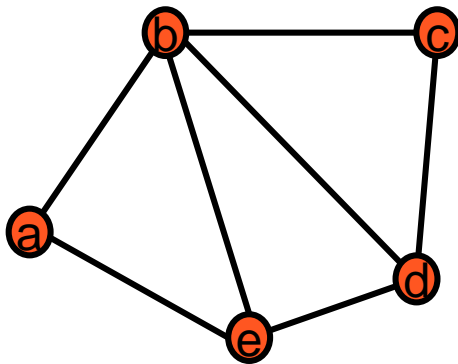
Hamiltonian Cycle is in NP

Given: a directed graph $G = (V, E)$, determine a simple cycle that contains each vertex in V

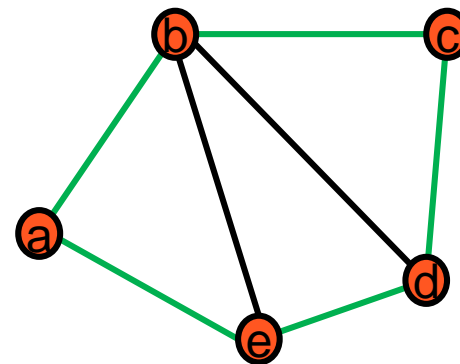
- Each vertex can only be visited once

Certificate:

- Sequence: $\langle a, e, d, c, b \rangle$



instance s



certificate t

Hamiltonian

3-SAT is in NP

Given a CNF formula Φ with three literals per clause, is there a satisfying assignment?

Certificate. An assignment of truth values to the n boolean variables.

Certifier. Check that each clause in Φ has at least one true literal.

instance s

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

certificate t

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

3-SAT is in NP

Given a CNF formula Φ with three literals per clause, is there a satisfying assignment?

Certificate. An assignment of truth values to the n boolean variables.

Certifier. Check that each clause in Φ has at least one true literal.

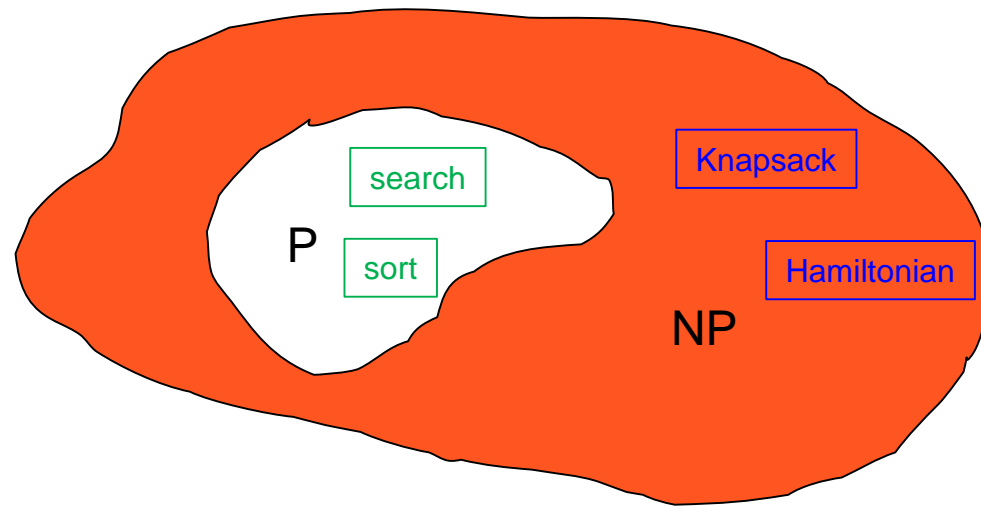
instance s

$$(\bar{1} \vee 1 \vee 0) \wedge (1 \vee \bar{1} \vee 0) \wedge (1 \vee 1 \vee 1) \wedge (\bar{1} \vee \bar{0} \vee \bar{1})$$

$$(1) \wedge (1) \wedge (1) \wedge (1)$$

P vs NP???

All problems that can be solved in polynomial time can be verified in polynomial time.



Any problem in P is also in NP: $P \subseteq NP$

P & NP-Complete Problems

- Shortest simple path

- Given a graph $G = (V, E)$ find a **shortest** path from a source to all other vertices
- Polynomial solution: $O(VE)$

- Longest simple path

- Given a graph $G = (V, E)$ find a **longest** path from a source to all other vertices
- NP-complete

P & NP-Complete Problems

- Euler tour

- $G = (V, E)$ a connected, directed graph find a cycle that traverses each edge of G exactly once (may visit a vertex multiple times)
- Polynomial solution $O(E)$

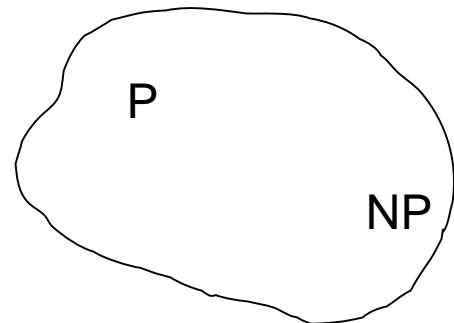
- Hamiltonian cycle

- $G = (V, E)$ a connected, directed graph find a cycle that visits each vertex of G exactly once
- NP-complete

Does $P = NP$?

The big (and **open question**)

is whether $NP \subseteq P$ or $P = NP$



– i.e., if it is always easy to check a solution, should it also be easy to find a solution?

- **If yes:** Efficient algorithms for KNAPSACK, TSP, FACTOR, SAT, ...
- **If no:** No efficient algorithms possible for KNAPSACK, TSP, SAT, ...
- **Consensus opinion on $P = NP$?** Probably no.

Most computer scientists believe that this is false but we do not have a proof ...

Why Prove NP-Completeness?

- Though nobody has proven that $\mathbf{P} \neq \mathbf{NP}$, if you prove a problem NP-Complete, most people accept that it is probably intractable
- Therefore it can be important to prove that a problem is NP-Complete
 - Don't need to come up with an efficient algorithm
 - Can instead work on *approximation algorithms*

-
- <https://www.youtube.com/watch?v=YX40hbAHx3s>

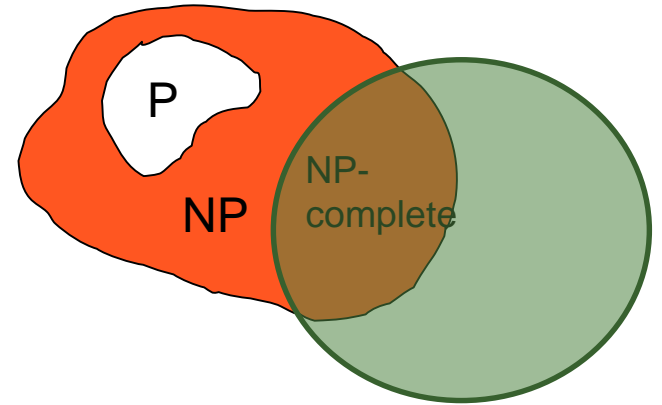
NP-Complete Problems

We will see that NP-Complete problems are the “hardest” problems in NP:

- If any *one* NP-Complete problem can be solved in polynomial time...
- ...then *every* NP-Complete problem can be solved in polynomial time...
- ...and in fact *every* problem in **NP** can be solved in polynomial time (which would show **P = NP**)
- Thus: solve Hamiltonian-cycle in $O(n^{100})$ time, you’ve proved that **P = NP**. Retire rich & famous.

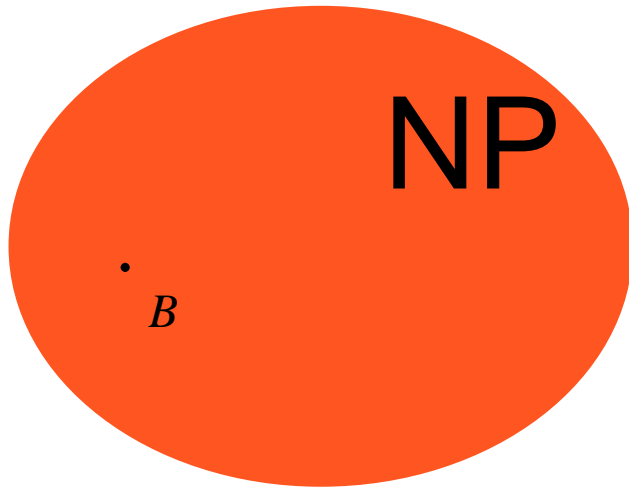
NP-Completeness (informally)

- **NP-complete** problems are defined as the hardest problems in NP
- Most practical problems turn out to be either P or NP-complete.

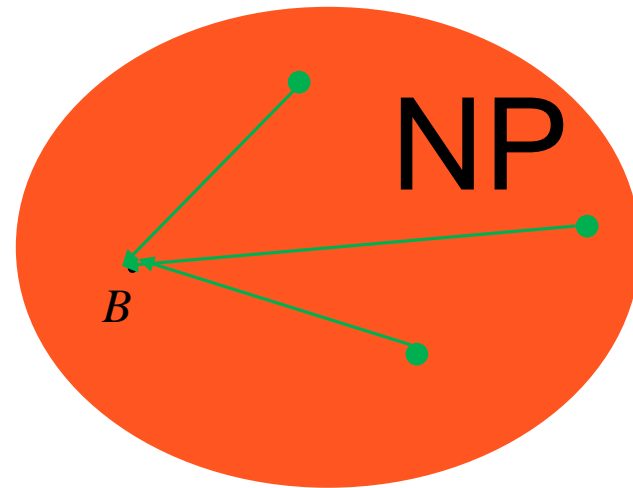


NP-Complete

A problem B is in NP-complete if:



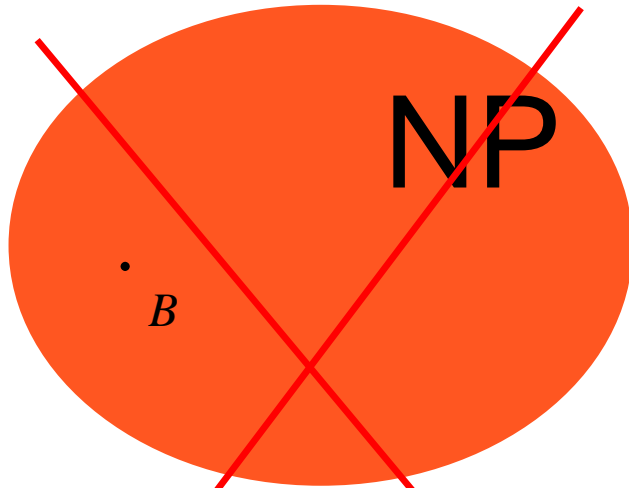
1. $B \in \mathbf{NP}$



2. There is a polynomial-time reduction from every problem $A \in \mathbf{NP}$ to B .

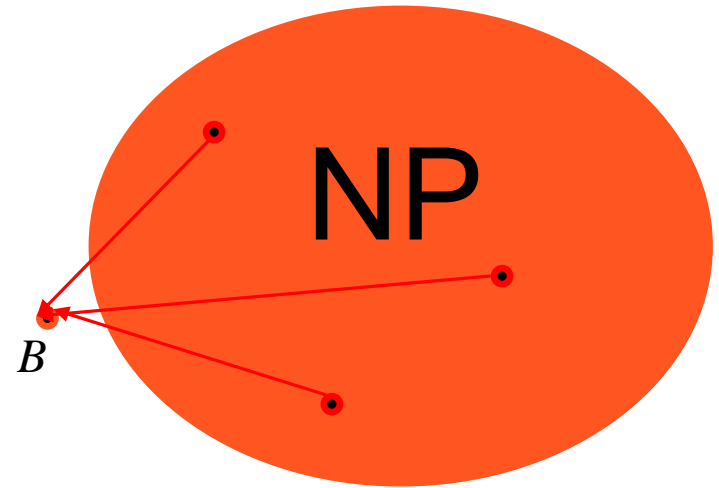
~~NP-Complete~~ Hard

A problem B is in NP-Hard if:



1. $B \in \mathbf{NP}$

Not *necessary* for NP-Hard



There is a polynomial-time reduction from every problem $A \in \mathbf{NP}$ to B .

Reductions

The crux of NP-Completeness is *reducibility* \leq_p

- Informally, a problem A can be reduced to another problem B if *any* instance of A can be “easily rephrased” as an instance of B, the solution to which provides a solution to the instance of A
 - *What do you suppose “easily” means?*
 - This rephrasing is called *transformation*
- Intuitively: If A reduces to B,
 - $A \leq_p B$
 - A is “no harder to solve” than B

Using Reductions

If A is *polynomial-time reducible* to B , we denote this $A \leq_p B$

Definition of NP-Complete:

- If A is NP-Complete, $A \in \mathbf{NP}$ and all problems X are reducible to A
- Formally: $X \leq_p A \ \forall X \in \mathbf{NP}$

If $A \leq_p B$ and A is NP-Complete, then B is NP-Hard

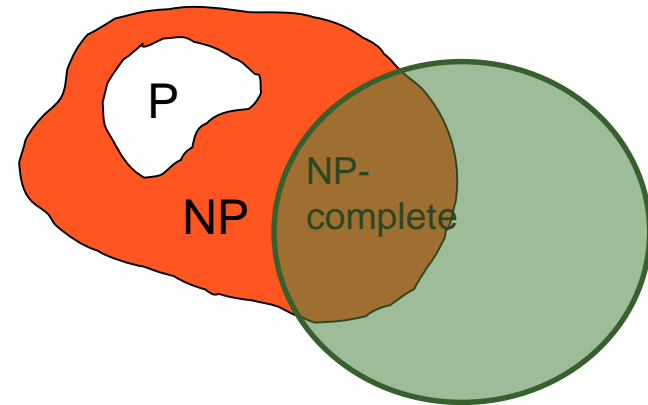
- If $B \in \mathbf{NP}$ too then B is NP-Complete

NP-Completeness (formally)

- A problem B is **NP-complete** if:

(1) $B \in \mathbf{NP}$

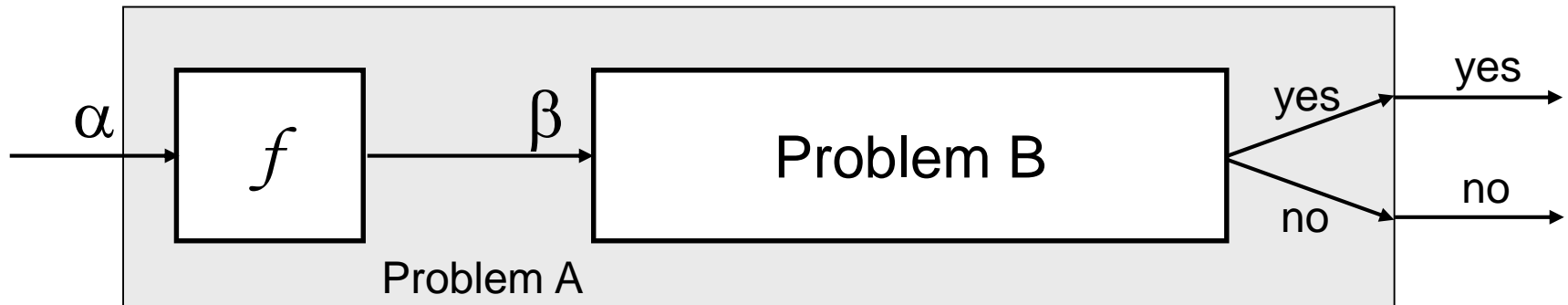
(2) $X \leq_p B$ for all $X \in \mathbf{NP}$



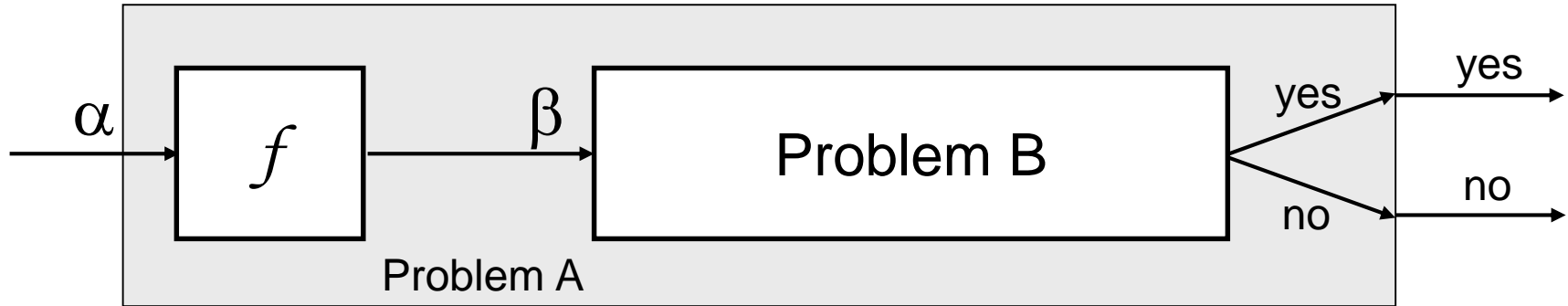
- If B satisfies only property (2) we say that B is **NP-hard**
- No polynomial time algorithm has been discovered for an **NP-Complete** problem
- No one has ever proven that no polynomial time algorithm can exist for any **NP-Complete** problem

Reductions

- Reduction is a way of saying that one problem is “**easier**” than another.
- We say that problem A is no harder than problem B, (i.e., we write “ **$A \leq_p B$** ”)
if we can solve A using the algorithm that solves B.
- **Idea:** transform the inputs of A to inputs of B

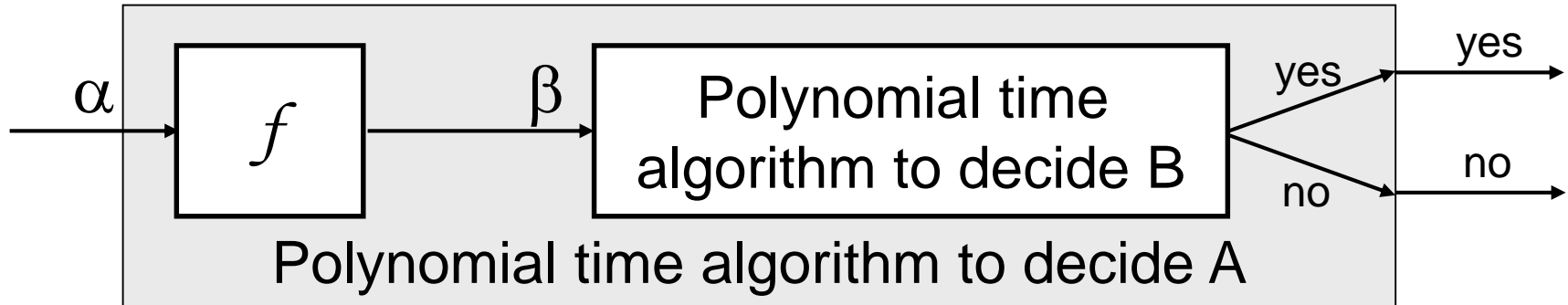


Implications of Reduction



- If $A \leq_p B$ and $B \in P$, then $A \in P$
- if $A \leq_p B$ and $A \notin P$, then $B \notin P$

Proving Polynomial Time



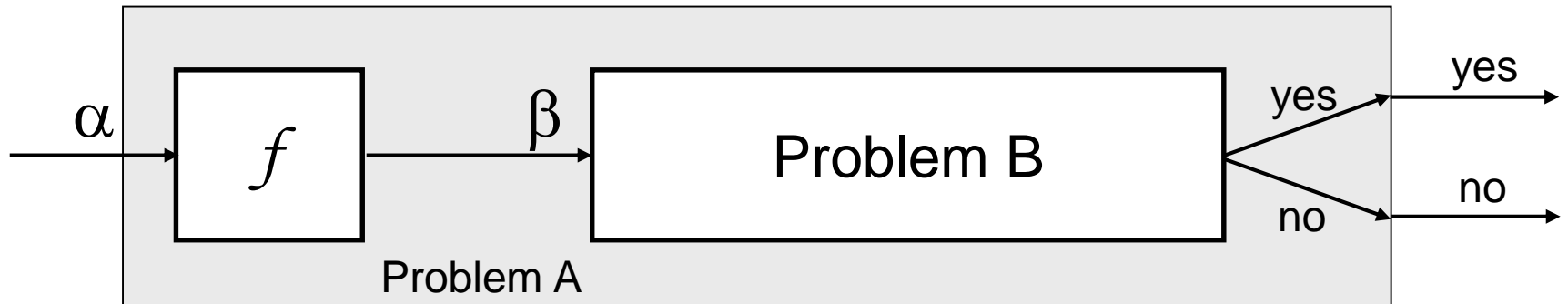
1. Use a **polynomial time** reduction algorithm to transform A into B
2. Run a known **polynomial time** algorithm for B
3. Use the answer for B as the answer for A

Reducibility

- An example:

- A: Given a set of Booleans, (x_1, x_2, \dots, x_n) , is at least one TRUE?
- B: Given a set of integers, (y_1, y_2, \dots, y_n) , is their sum positive?
- Transformation: $(x_1, x_2, \dots, x_n) = (y_1, y_2, \dots, y_n)$ where $y_i = 1$ if $x_i = \text{TRUE}$, $y_i = 0$ if $x_i = \text{FALSE}$

Reductions



Proving NP-Completeness

What steps do we have to take to prove a problem B is NP-Complete?

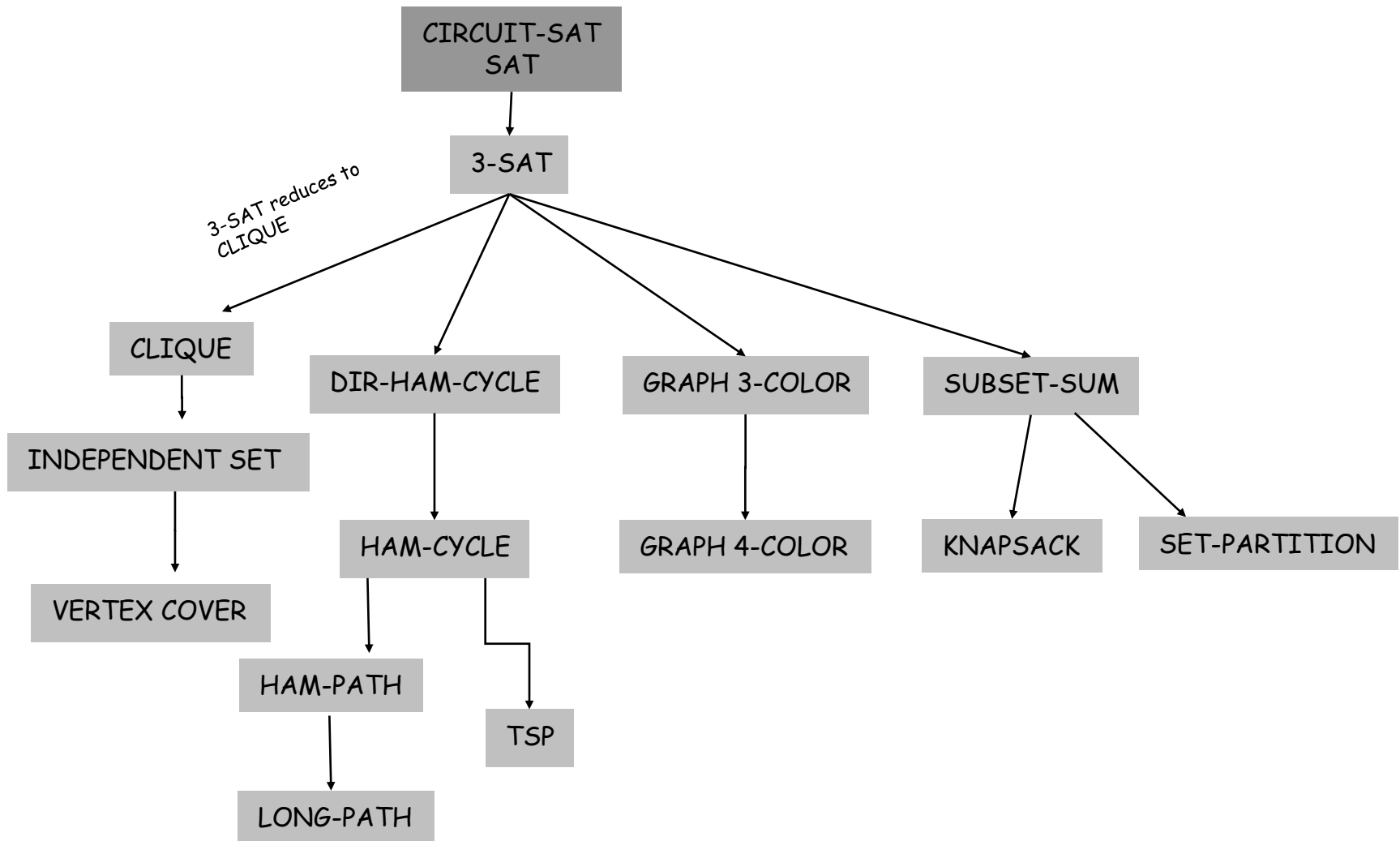
1. Pick a known NP-Complete problem A. Reduce A to B
 - Describe a polynomial time transformation/reduction that maps instances of A to instances of B, s.t. “yes” for B = “yes” for A
 - Prove the transformation works
 - Prove it runs in polynomial time

By proving step 1 you have proved that problem B is NP-Hard
2. Prove $B \in \mathbf{NP}$
 - Show that a solution to B can be verified in polynomial time

If you can prove steps 1 and 2 you have proven that B is NP-complete.

NP-Completeness

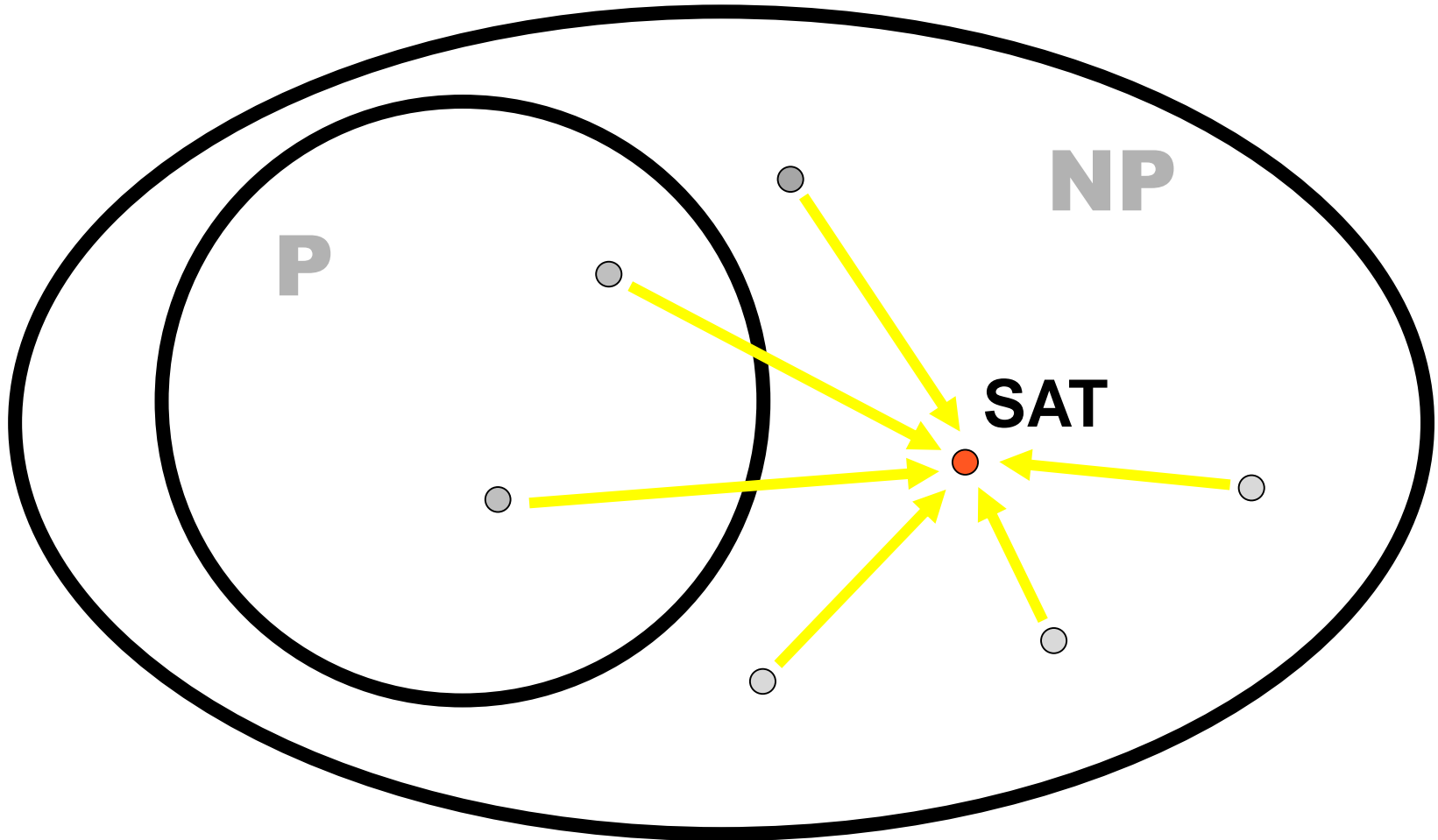
All problems below are NP-complete and polynomial reduce to one another!



Theorem (Cook-Levin): SAT is NP-complete

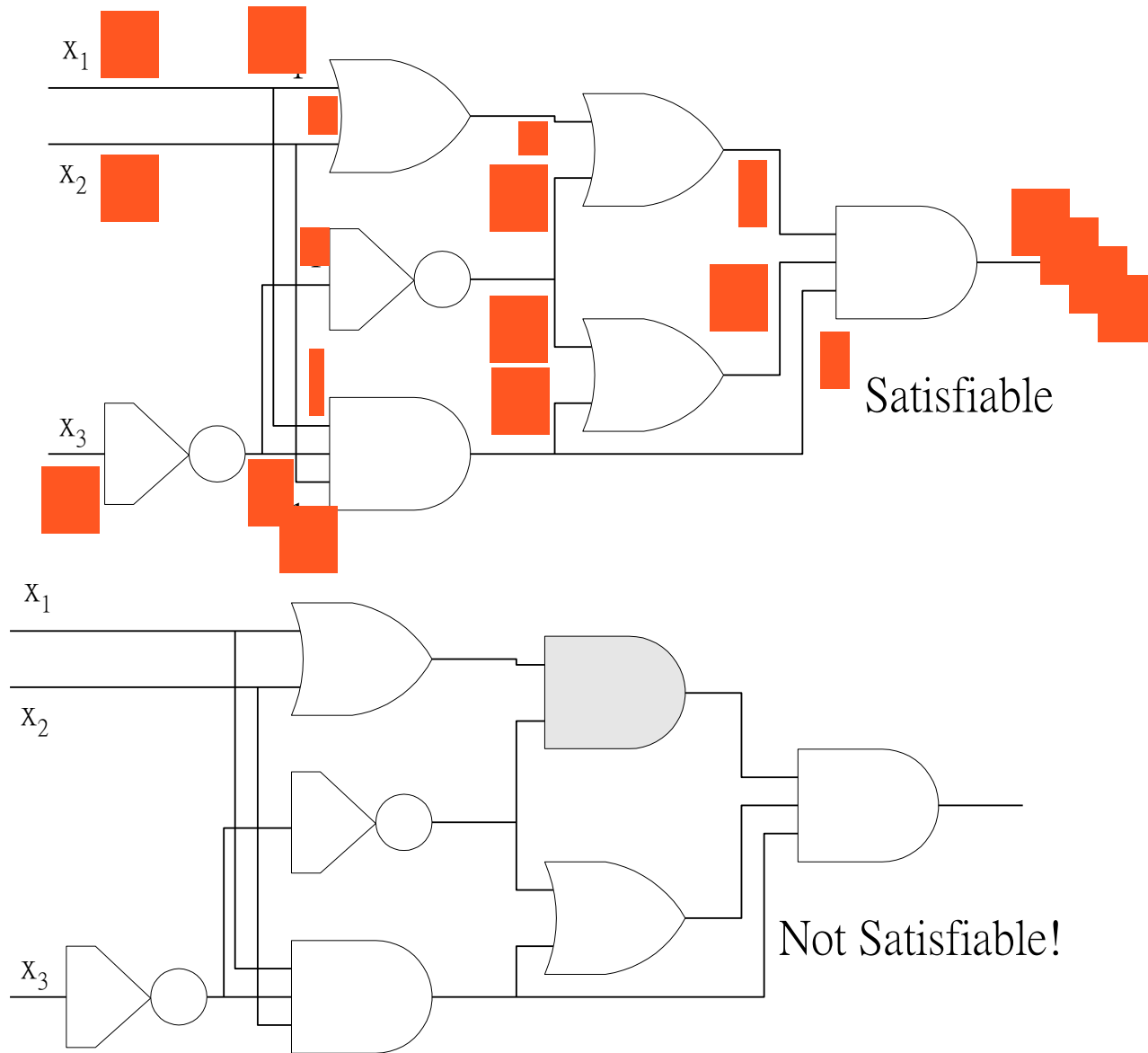
Corollary: $\text{SAT} \in \text{P}$ if and only if $\text{P} = \text{NP}$

Any thing in $NP \leq_p SAT$

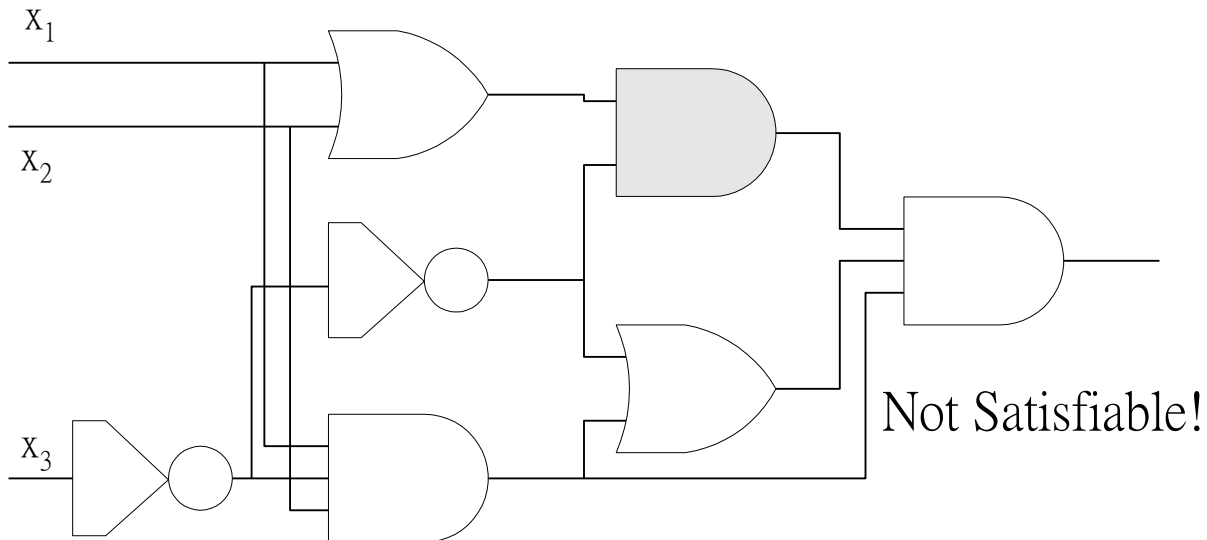
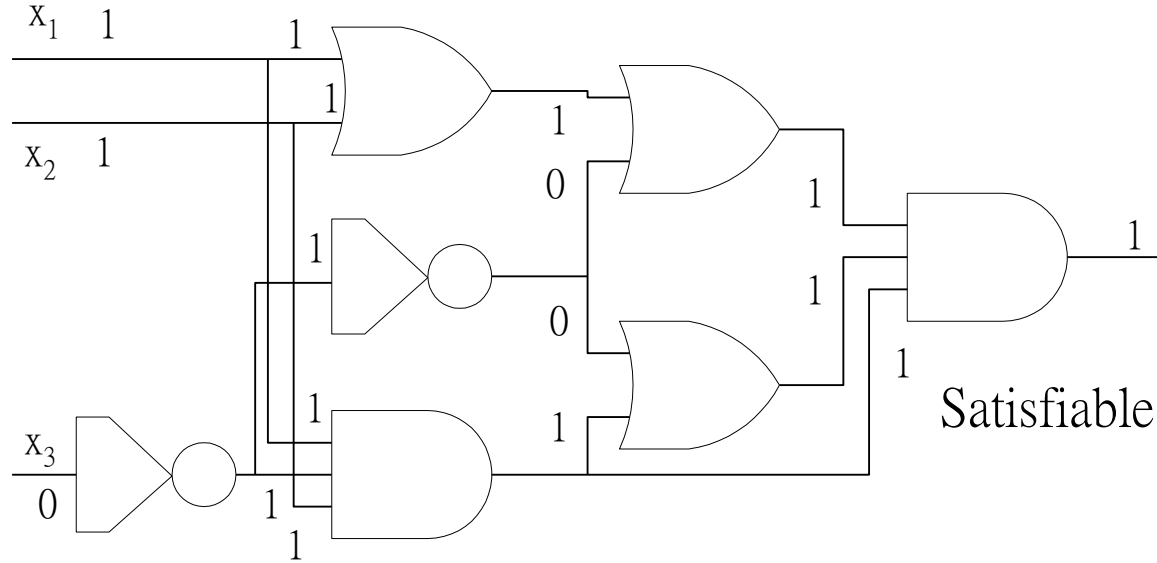


You can think of \rightarrow as “easier than”.
SAT is the hardest problem in NP.

Circuit satisfiability is in NP



Circuit satisfiability



Satisfiability Problem (SAT)

- **Satisfiability problem:** given a logical expression Φ , find an assignment of values (F, T) to variables x_i that causes Φ to evaluate to T

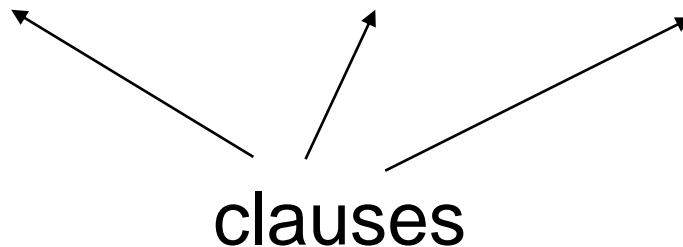
$$\Phi = x_1 \vee \neg x_2 \wedge x_3 \vee \neg x_4$$

- SAT was the first problem shown to be NP-complete!

CNF Satisfiability

- CNF is a special case of SAT
- Φ is in “Conjunctive Normal Form” (CNF)
 - “AND” of expressions (i.e., clauses)
 - Each clause contains only “OR”s of the variables and their complements

$$\Phi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$$



3-CNF Satisfiability (3-SAT)

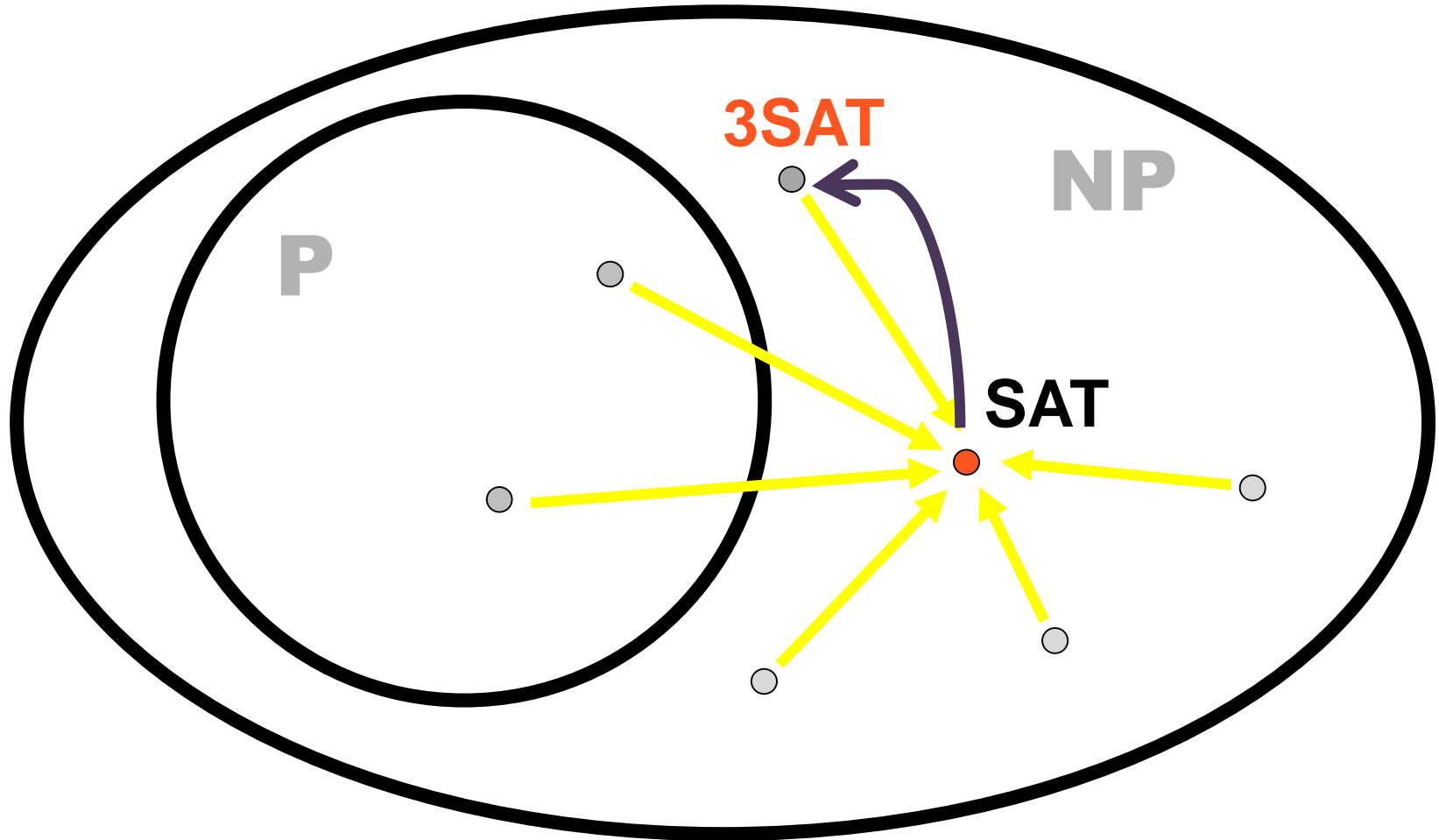
A subcase of CNF problem:

- Contains three clauses

$$\Phi = (x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

- 3-CNF (3-SAT) is NP-Complete

What we want to prove?



How to prove?

We can convert (in polynomial time) a given SAT instance S into a 3SAT instance S' such that

- If S is satisfiable, then S' is satisfiable.
- If S' is satisfiable, then S is satisfiable.

Key Observation

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$ is **satisfiable**

if and only if

$(x_1 \vee x_2 \vee z_1) \wedge (\neg z_1 \vee x_3 \vee z_2) \wedge$
 $(\neg z_2 \vee x_4 \vee z_3) \wedge (\neg z_3 \vee x_4 \vee x_5)$
is **satisfiable**.

Polynomial Time Reduction

Clause in SAT

$$x_1$$

$$x_1 \vee \neg x_2$$

$$x_1 \vee x_2 \vee x_3$$

$$x_1 \vee x_2 \vee x_3 \vee x_4$$

$$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$$

Clauses in 3SAT

$$x_1 \vee x_1 \vee x_1$$

$$x_1 \vee x_1 \vee \neg x_2$$

$$x_1 \vee x_2 \vee x_3$$

$$(x_1 \vee x_2 \vee z_1) \wedge (\neg z_1 \vee x_3 \vee x_4)$$

$$(x_1 \vee x_2 \vee z_1) \wedge (\neg z_1 \vee x_3 \vee z_2) \wedge$$
$$(\neg z_2 \vee x_4 \vee z_3) \wedge (\neg z_3 \vee x_4 \vee x_5)$$

Clique

Clique Problem:

- Undirected graph $G = (V, E)$
- **Clique:** a subset of vertices in V all connected to each other by edges in E (i.e., forming a complete graph)
- **Size of a clique:** number of vertices it contains

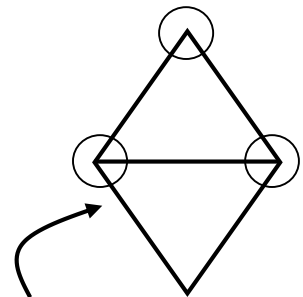
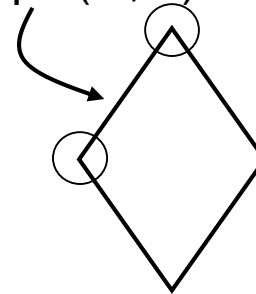
Optimization problem:

- Find a clique of maximum size

Decision problem:

- Does G have a clique of size k ?

Clique($G, 2$) = YES
Clique($G, 3$) = NO

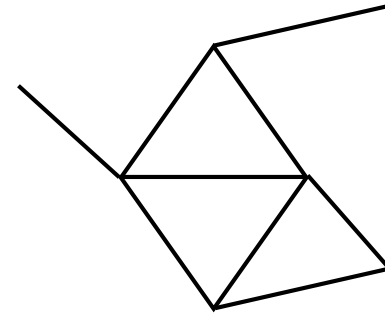


Clique($G, 3$) = YES
Clique($G, 4$) = NO

Clique Verifier

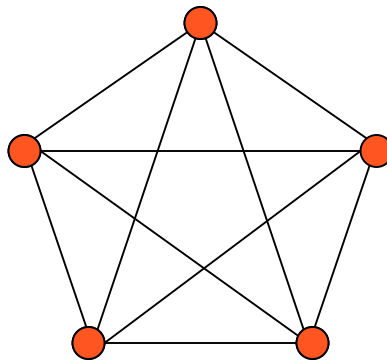
Given: an undirected graph $G = (V, E)$

- **Problem:** Does G have a clique of size k ?
- **Certificate:**
 - A set of k nodes
- **Verifier:**
 - Verify that for all pairs of vertices in this set there exists an edge in E



Example: Clique

- $\text{CLIQUE} = \{ \langle G, k \rangle \mid G \text{ is a graph with a clique of size } k \}$
- A clique is a subset of vertices that are all connected
- Why is CLIQUE in NP?



3-CNF \leq_p Clique

- **Idea:**

- Construct a graph G such that Φ is satisfiable only if G has a clique of size k

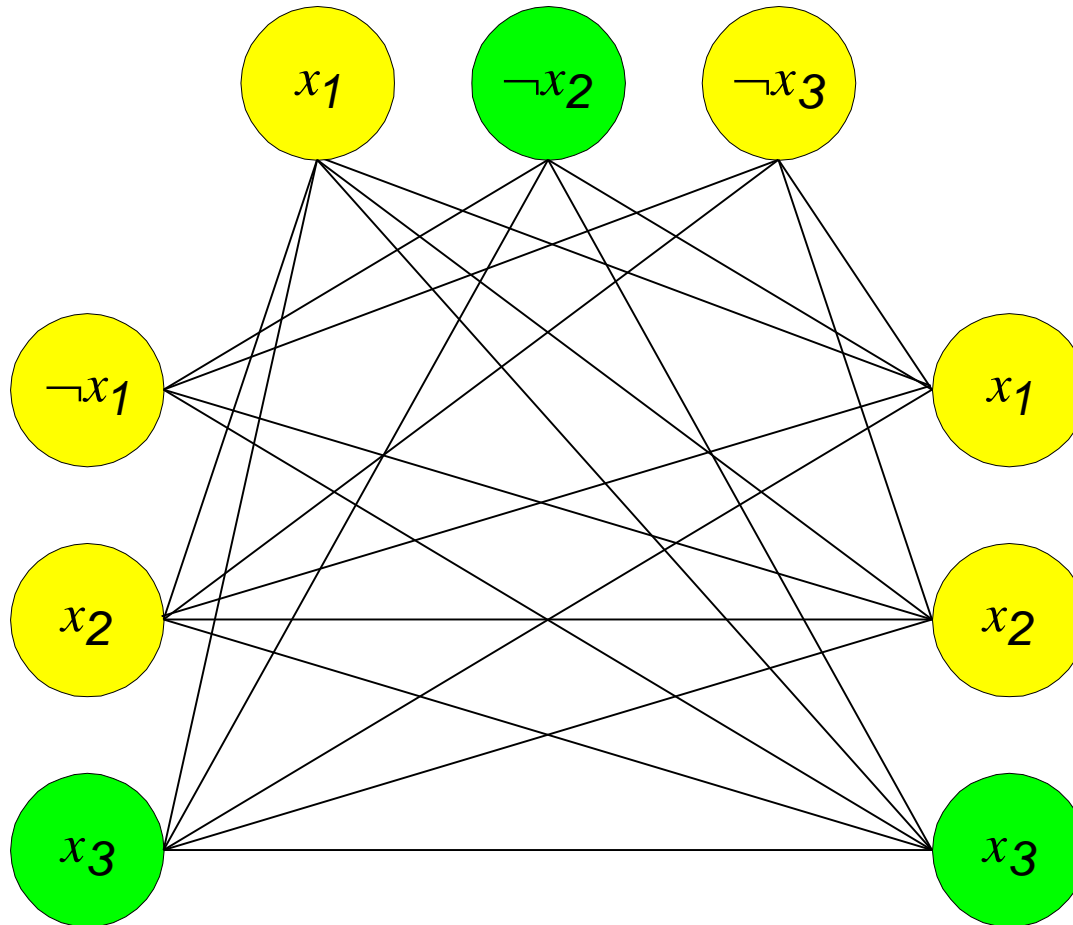
Reduce 3-SAT to Clique

- Pick an instance of 3-SAT, Φ , and transform into $\langle G, k \rangle$ an instance of Clique
- If Φ has m clauses, we create a graph with m clusters of 3 nodes each and set $k = m$.
- Each cluster corresponds to a clause.
- Each node in a cluster is labeled with a literal from the clause.
- We do not connect any nodes in the same cluster
- We connect nodes in different clusters whenever they are not contradictory
- Any k -clique in this graph corresponds to a satisfying assignment

Example

$$(\mathbf{x}_1 \vee \neg \mathbf{x}_2 \vee \neg \mathbf{x}_3) \wedge (\neg \mathbf{x}_1 \vee \mathbf{x}_2 \vee \mathbf{x}_3) \wedge (\mathbf{x}_1 \vee \mathbf{x}_2 \vee \mathbf{x}_3)$$

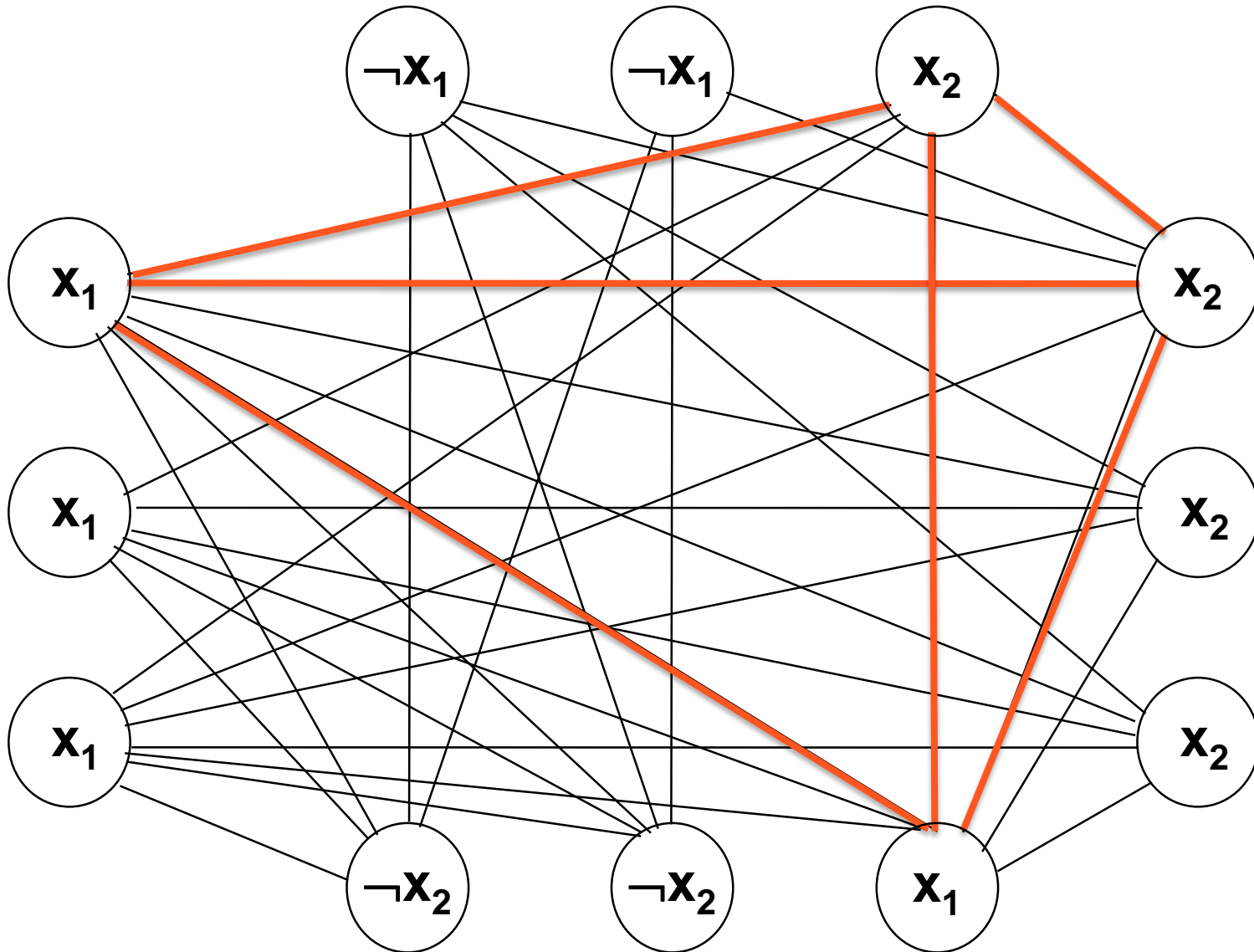
$$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$$



$$C_2 = \neg x_1 \vee x_2 \vee x_3$$

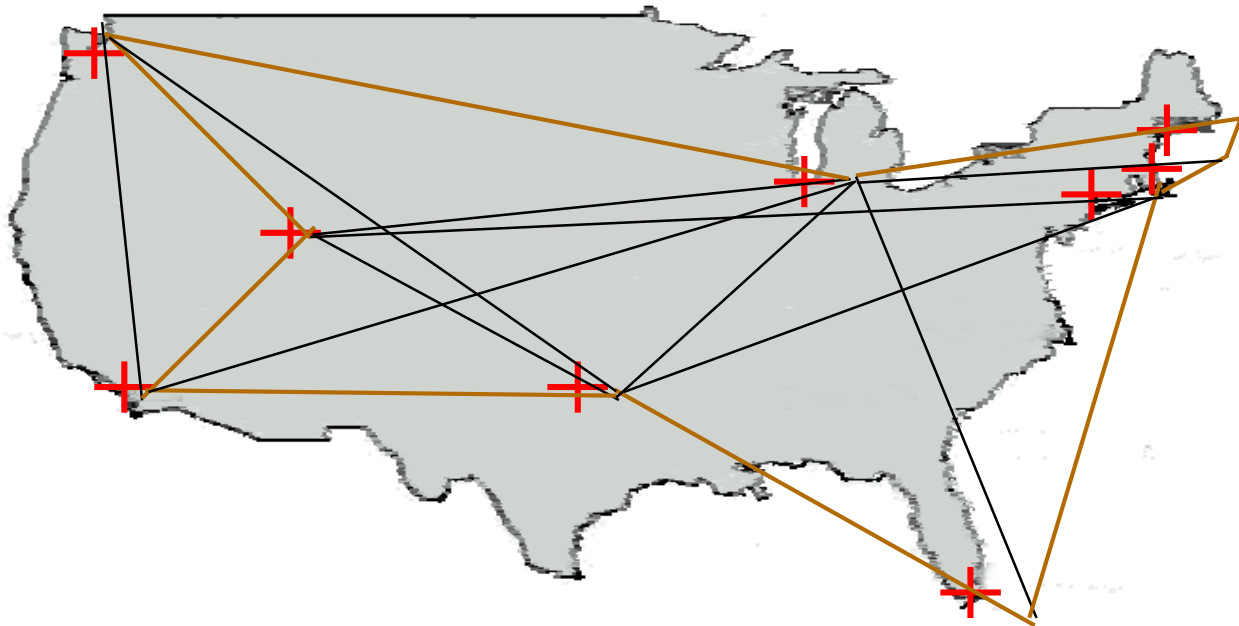
$$C_3 = x_1 \vee x_2 \vee x_3$$

$$(\mathbf{x}_1 \vee \mathbf{x}_1 \vee \mathbf{x}_1) \wedge (\neg \mathbf{x}_1 \vee \neg \mathbf{x}_1 \vee \mathbf{x}_2) \wedge \\ (\mathbf{x}_2 \vee \mathbf{x}_2 \vee \mathbf{x}_2) \wedge (\neg \mathbf{x}_2 \vee \neg \mathbf{x}_2 \vee \mathbf{x}_1)$$



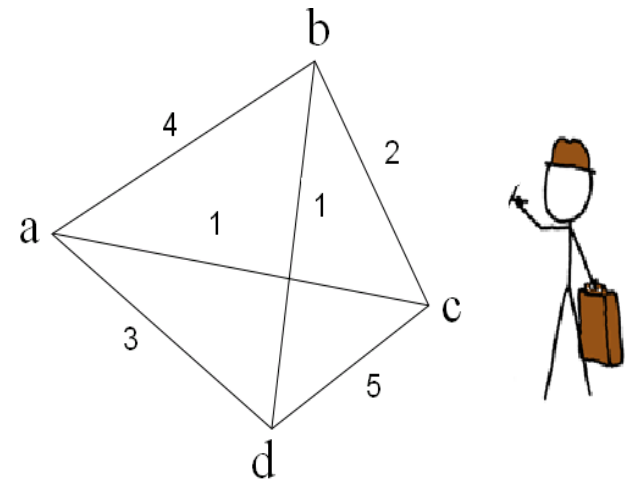
Traveling Salesman Problem

- A traveling salesperson needs to visit n cities
- Is there a route of at most d length? (decision problem)
 - Optimization-version asks to find a shortest cycle visiting all vertices once in a weighted graph



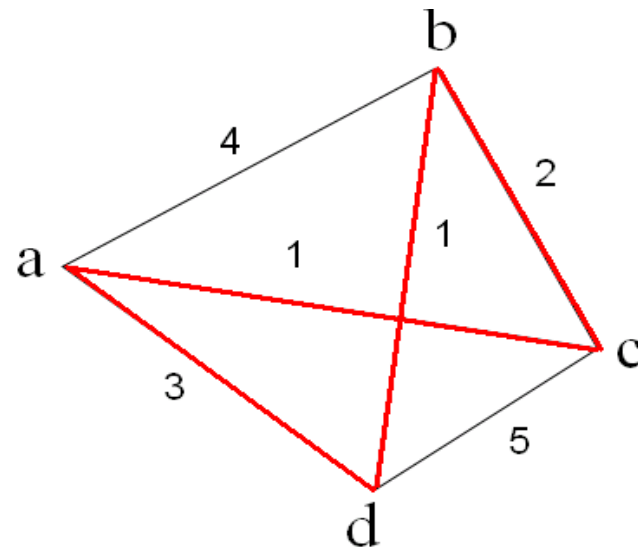
Traveling Salesman Problem

- In the traveling salesman problem, a salesman must visit n cities.
- Salesman wishes to make a tour visiting each city exactly once and finishing at the city he started.
- There is an integer cost $c(i,j)$ to travel from city i to city j .
- For example, the salesman must travel to a, b, c, d locations.
- Travel costs are given



Optimization TSP

- The salesman wishes to make the tour whose total cost is minimum.
- The total cost is sum of the individual costs along the edges of the tour
- In the example the minimum cost tour is a-c-b-d
- The cost of this tour is $1+2+1+3 = 7$



Decision TSP

The formal language:

$TSP = \{ \langle G, c, k \rangle : G=(V, E) \text{ is a complete graph, } c \text{ is a function from } (V \times V) \rightarrow \mathbb{N}, k \in \mathbb{N} \text{ and } G \text{ has a traveling salesman tour with cost at most } k \}$

$G = (\{a, b, c, d\}, \{(a,b), \dots (c,d)\})$ $c(a,b) = 4, c(a,c) = 1, \dots c(c,d) = 5$

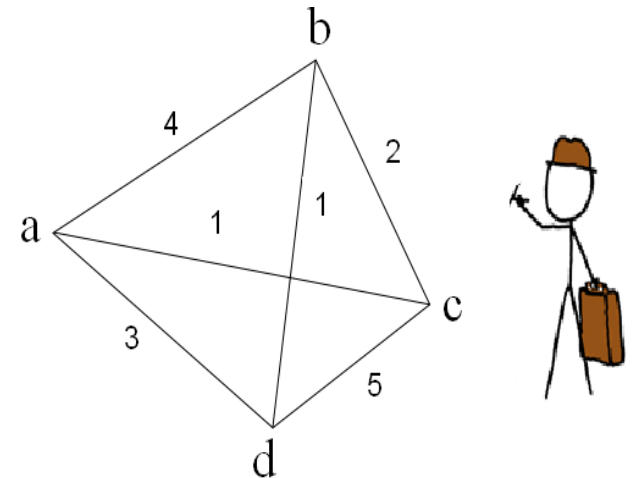
Suppose $k = 15$

Can we verify the solution certificate

(a,d,c,b) - yes

$c(a,d) + c(d,c) + c(c,b) + c(b,a) =$

$3 + 5 + 2 + 4 = 14 < 15$



NP-Completeness Proof Method

To show that B is NP-Complete:

1. Show that B is in NP.

Give a polynomial time algorithm for verifying a solution.

2. Show that $A \leq_p B$ for some $A \in \text{NP-Complete}$

Pick an instance, A, of your favorite NP-Complete problem

Show a polynomial algorithm to transform A into an instance of B

Step 2 alone shows that a problem is NP-Hard

Prove TSP-Decision is NP-complete

1) Show that TSP belongs to NP.

Given an instance of the problem the certificate is the sequence of n vertices (cities) in the tour.

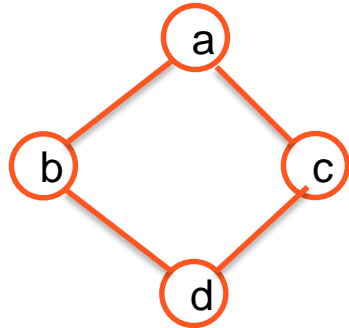
The certifier (verification algorithm) checks that

- this sequence contains each vertex exactly once,
- sums up the edge costs and checks whether the sum is at most **k** .

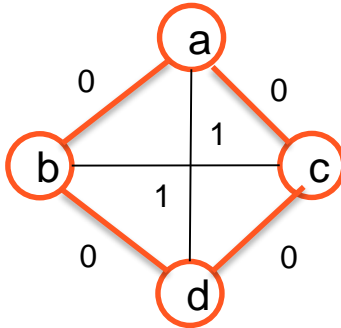
This process can be done in polynomial time.

Therefore TSP-Decision is in NP

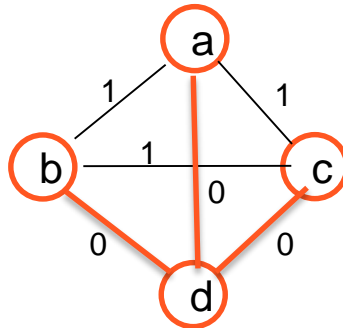
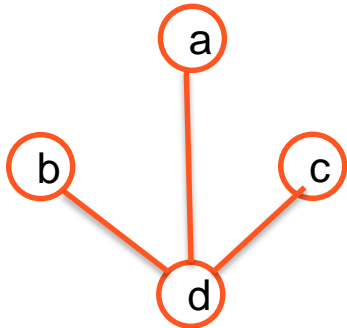
Ham-Cycle G



TSP-Decision G'



TSP-Decision
 $\langle G', c, 0 \rangle$
Yes



TSP-Decision
 $\langle G', c, 0 \rangle$
No

Prove TSP-Decision is NP-complete

2) Prove that TSP is NP-hard. We can show that

$$\text{Ham-cycle} \leq_p \text{TSP}.$$

where $\text{Ham-cycle} \in \text{NP-Complete}$

Let $G=(V,E)$ be an instance of Ham-cycle. We construct an instance of TSP as follows

- Form the complete graph $G' = (V, E')$ where $E' = \{ (i,j) : i, j \in V \text{ and } i \neq j \}$ and
- Define the cost function c by $c(i,j) = \{ 0 \text{ if } (i,j) \in E, 1 \text{ if } (i,j) \notin E \}$

The instance of TSP is then $\langle G', c, 0 \rangle$ which is easily formed in polynomial time.

By proving 2) TSP-Decision is NP-Hard. Since 1) held too then we have shown that TSP-Decision is NP-Complete

We now show that graph G has a Hamiltonian cycle if and only if graph G' has a tour of cost at most 0.

Suppose the graph G has a Hamiltonian cycle h .

Each edge in h belongs to E and thus has a cost 0 in G'

Thus h is a tour in G' with cost 0

Conversely suppose that graph G' has a tour h' of cost at most 0.

Since the cost of edges in E' are 0 and 1, the cost of tour h' is exactly 0 and each edge on the tour must have cost 0.

Thus h' contains only edges in E .

Hence we conclude that h' is a Hamiltonian cycle in graph G .

By proving 2) TSP-Decision is NP-Hard. Since 1) held too then we have shown that TSP-Decision is NP-Complete

SUBSET-SUM

Instance: A set of numbers denoted S and a target number t .

Problem: To decide if there exists a subset $S' \subseteq S$,
s.t $\sum_{y \in S'} y = t$.

Examples of SUBSET-SUM

$\langle \{2,4,8\}, 10 \rangle \in \text{SUBSET - SUM} \dots$ because $2+8=10$

$\langle \{2,4,8\}, 11 \rangle \notin \text{SUBSET - SUM}$

... because 11 cannot be made out of $\{2,4,8\}$

$$\text{SUBSET - SUM} = \{ \langle S, t \rangle \mid S = \{x_1, \dots, x_k\}$$

there is a subset $R \subseteq S$

such that $\sum_{y \in R} y = t \}$

SUBSET-SUM is NP-Complete

Proof:

1. Show SUBSET-SUM is in NP.
2. Show $3\text{SAT} \leq_p \text{SUBSET-SUM}$.

SUBSET-SUM is in NP

Given a set S and target t :

- Verify that $S' \subseteq S$ is a solution
- The answer is YES iff $\sum_{y \in S'} y = t$.

The length of the certificate: $O(n)$ ($n = |S|$)

Time complexity: Is the time to add the numbers in S' which is $O(n)$.

Reducing 3SAT to SubSet Sum

Proof idea:

- Choosing the subset numbers from the set S corresponds to choosing the assignments of the variables in the 3SAT formula.
- The different digits of the sum correspond to the different clauses of the formula.
- If the target t is reached, a valid and satisfying assignment is found.

Subset Sum

3CNF formula:

$$(x_1 \vee x_2 \vee x_3) \wedge$$

$$(\bar{x}_1 \vee x_2 \vee x_4) \wedge$$

$$(\bar{x}_2 \vee \bar{x}_2 \vee \bar{x}_3) \wedge$$

$$(x_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

clause:

1	2	3	4
---	---	---	---

$+x_1$
$-x_1$
$+x_2$
$-x_2$
$+x_3$
$-x_3$
$+x_4$
$-x_4$

1	0	0	0	1	0	0	1
1	0	0	0	0	1	0	0
	1	0	0	1	1	0	0
	1	0	0	0	0	1	0
		1	0	1	0	0	0
		1	0	0	0	1	1
			1	0	1	0	0
			1	0	0	0	1
				1	0	0	0
				1	0	0	0
					1	0	0
						1	0
							1
							1
1	1	1	1	3	3	3	3

dummies

Make the number table,
and the 'target sum' t

Reducing 3SAT to SubSet Sum

- Let $\phi \in 3\text{CNF}$ with k clauses and ℓ variables x_1, \dots, x_ℓ .
- Create a Subset-Sum instance $\langle S_\phi, t \rangle$ by:
 $2\ell + 2k$ elements of
$$S_\phi = \{y_1, z_1, \dots, y_\ell, z_\ell, g_1, h_1, \dots, g_k, h_k\}$$
 - y_j indicates positive x_j literals in clauses
 - z_j indicates negated x_j literals in clauses
 - g_j and h_j are dummies
 - and

Subset Sum

$$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

Note 1: The “1111” in the target forces a proper assignment of the x_i variables.

Note 2: The target “3333” is only possible if each clause is satisfied. (The dummies can add maximally 2 extra.)

$+x_1$	1	0	0	0	1	0	0	1
$-x_1$	1	0	0	0	0	1	0	0
$+x_2$		1	0	0	1	1	0	0
$-x_2$		1	0	0	0	0	1	0
$+x_3$			1	0	1	0	0	0
$-x_3$			1	0	0	0	1	1
$+x_4$				1	0	1	0	0
$-x_4$				1	0	0	0	1
					1	0	0	0
					1	0	0	0
						1	0	0
						1	0	0
							1	0
							1	0
								1
								1
	1	1	1	1	3	3	3	3

Subset Sum

$$(\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

1	0	0	0	1	0	0	1
	1	0	0	0	0	1	0
		1	0	0	0	1	1
			1	0	1	0	0
				1	0	0	0
				1	0	0	0
					1	0	0
					1	0	0
						1	0
							1
1	1	1	1	3	3	3	3

$+x_1$
$-x_1$
$+x_2$
$-x_2$
$+x_3$
$-x_3$
$+x_4$
$-x_4$

1	0	0	0	1	0	0	1
1	0	0	0	0	1	0	0
	1	0	0	1	1	0	0
	1	0	0	0	0	1	0
		1	0	1	0	0	0
		1	0	0	0	1	1
			1	0	1	0	0
			1	0	0	0	1
				1	0	0	0
				1	0	0	0
					1	0	0
					1	0	0
						1	0
						1	0
							1
							1
1	1	1	1	3	3	3	3

$x_1, \bar{x}_2, \bar{x}_3, x_4$ is a satisfying assignment

Subset Sum

$$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

1	0	0	0	0	1	0	0
	1	0	0	0	0	1	0
		1	0	0	0	1	1
			1	0	1	0	0
				1	0	0	0
				1	0	0	0
					?	?	?
1	1	1	1	2	?	?	?

$+x_1$
$-x_1$
$+x_2$
$-x_2$
$+x_3$
$-x_3$
$+x_4$
$-x_4$

1	0	0	0	1	0	0	1
1	0	0	0	0	1	0	0
	1	0	0	1	1	0	0
	1	0	0	0	0	1	0
		1	0	1	0	0	0
		1	0	0	0	1	1
			1	0	1	0	0
			1	0	0	0	1
				1	0	0	0
				1	0	0	0
					1	0	0
					1	0	0
						1	0
						1	0
							1
							1
1	1	1	1	3	3	3	3

$\bar{x}_1, \bar{x}_2, \bar{x}_3, x_4$ is not a satisfying assignment

Proof $3SAT \leq_p \text{Subset Sum}$

- For every 3CNF ϕ , take target $t=1\dots 13\dots 3$ and the corresponding set S_ϕ .
- If $\phi \in 3SAT$, then the satisfying assignment defines a subset that reaches the target.
- Also, the target can only be obtained via a set that gives a satisfying assignment for ϕ .

$\phi \in 3SAT$ if and only if $\langle S_\phi, 1\dots 13\dots 3 \rangle \in \text{SubsetSum}$

0-1 KNAPSACK

Prove the following knapsack problem to be NP complete

Decision version

n objects, each with a weight $w_i > 0$ and a benefit $b_i > 0$
capacity of knapsack : W . Can you fill the knapsack so
that the sum of benefits is at least K ?

For all item i in the solution set S .

$$\sum b_i \geq K \quad \text{and} \quad \sum w_i \leq W$$

KNAPSACK is NP-Complete

- 1) Show NP – Verify a solution in polynomial time
- 2) Show NP-Hard. Reduce SUBSET-SUM to KNAPSACK

KNAPSACK is NP-Complete

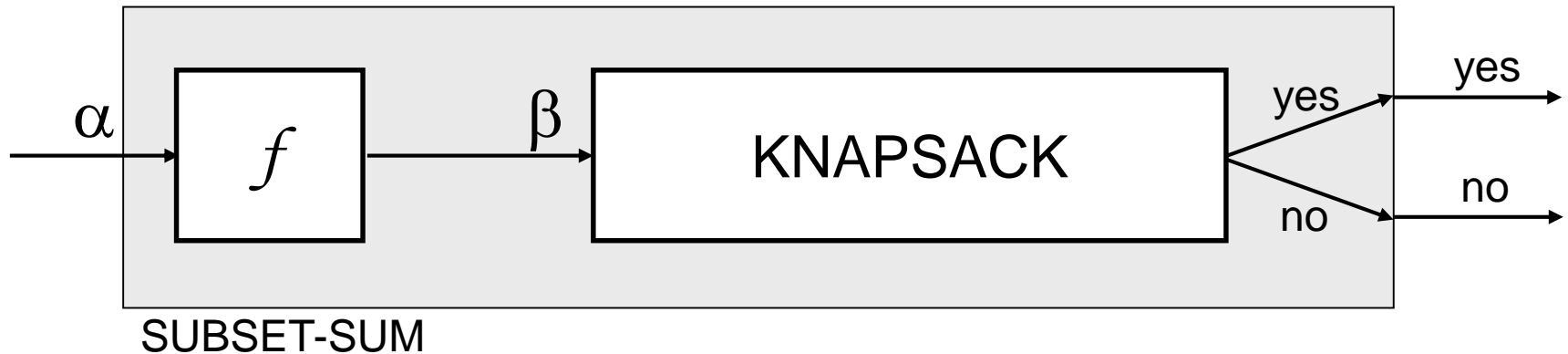
1) Show NP – Verify a solution in polynomial time

Given a certificate solution $X = \{ x_1, \dots, x_n \}$ can we verify in poly time.

- Sum the weights of $x \in X$ must be $\leq W$. Time $O(n)$
- Sum the benefits of $x \in X$ must be $\geq K$. Time $O(n)$

Reduction

- Show NP-Hard. SUBSET-SUM \leq_p KNAPSACK. How can you use Knapsack to solve Subset-Sum



2) Show NP-Hard. $SUBSET-SUM \leq_p KNAPSACK$. How can you use Knapsack to solve Subset-Sum

Reduction from *SUBSET-SUM* to Decision-*KNAPSACK*:

$SUBSET-SUM = \{ \langle S, t \rangle \mid S = \{y_1, \dots, y_k\} \text{ and for some Subset } T = \{y_j, \dots, y_l\} \subseteq S, \sum y_i = t \}$

Set $b_i = y_i$ and $w_i = y_i$

Set $W = t$ and $K = t$

Then for any subset $T \subseteq S$

$$\sum_{i \in T} y_i = t \text{ if and only if } \sum_{i \in T} b_i = \sum_{i \in T} y_i \geq t \text{ and } \sum_{i \in T} w_i = \sum_{i \in T} y_i \leq t$$

Example: Reduce SUBSET-SUM to KNAPSACK.

SUBSET-SUM = $\langle \{12, 6, 8, 13, 20\}, 26 \rangle$

Set $b_i = w_i = x_i$

Set $W = k = 26$

Knapsack has capacity 26. Is there a subset of items that will fit in the knapsack and have a total benefit of at least 26?

item	weight	benefit
1	12	12
2	6	6
3	8	8
4	13	13
5	20	20

$$\sum_{i \in T} b_i = \sum_{i \in T} x_i \geq 26$$

$$\sum_{i \in T} w_i = \sum_{i \in T} x_i \leq 26$$

Yes $T = \{\text{item2}, \text{item 5}\}$. This corresponds to a subset sum $T = \{6, 20\}$ sum 26

SUBSET-SUM to PARTITION

- $\text{PARTITION} = \{ x_1, x_2, \dots, x_k \mid \text{we can split the integers into two sets which sum to half} \}$
- $\text{SUBSET-SUM} = \{ \langle x_1, x_2, \dots, x_k, t \rangle \mid \text{there exists a subset which sums to } t \}$
- 1) If I can solve SUBSET-SUM, how can I use that to solve an instance of PARTITION?
- 2) If I can solve PARTITION, how can I use that to solve an instance of SUBSET-SUM?

Polynomial Reductions

- 1) Partition REDUCES to Subset-Sum
 - $\text{Partition} \leq_p \text{Subset-Sum}$
- 2) Subset-Sum REDUCES to Partition
 - $\text{Subset-Sum} \leq_p \text{Partition}$
- Therefore they are equivalently hard

Bin Packing Problem

- Given a set of items $S = \{x_1 \dots x_n\}$ each with some weight w_i , pack maximum number of items into a collection of finite number of bins each with some capacity B_i using minimum number of bins.
- Knapsack problem is a particular case of Bin-packing when the number of bins is 1 and its capacity is K