P, NP & NP-Complete

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.

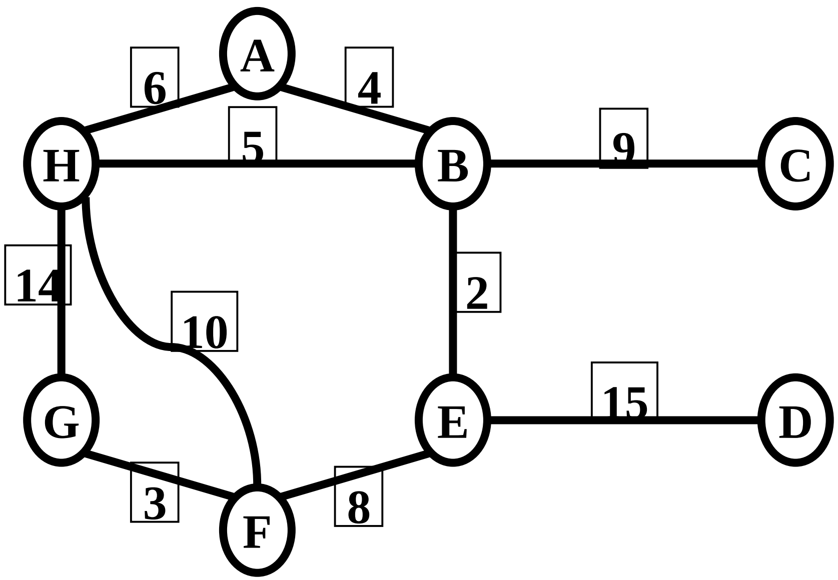
Optimization vs Decision Problems

- **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

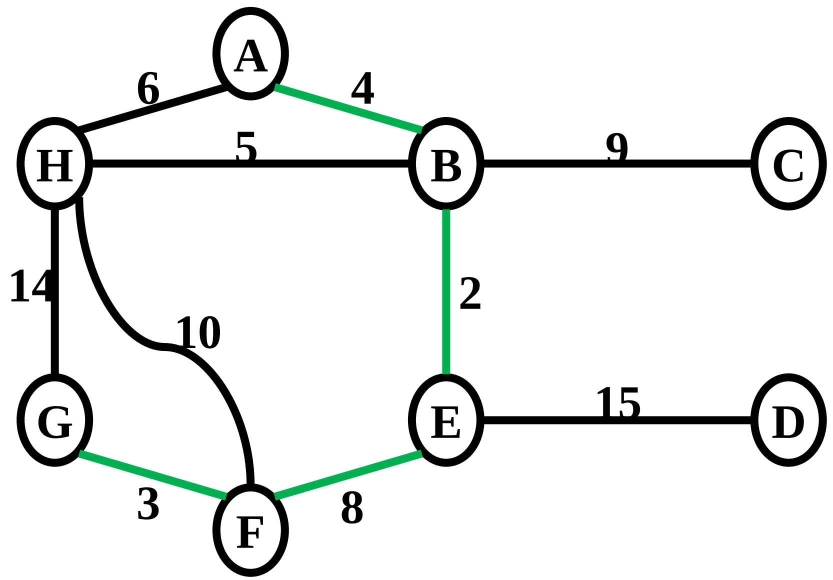
- Optimization: What is the minimum total weight of all paths between A and G?
- Decision: Does there exists a path between A and E with total weight at most 20?



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 exists 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 exists 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 lgn* 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 worst 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

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 Nondetermistic 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

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

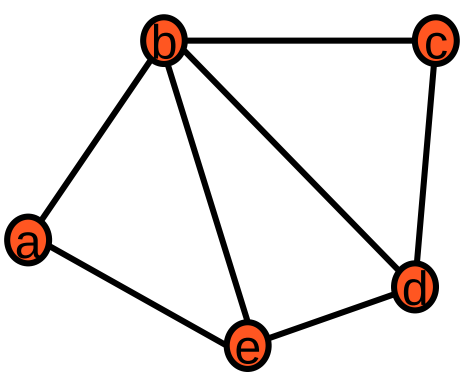
Hamiltonian Cycle is in NP

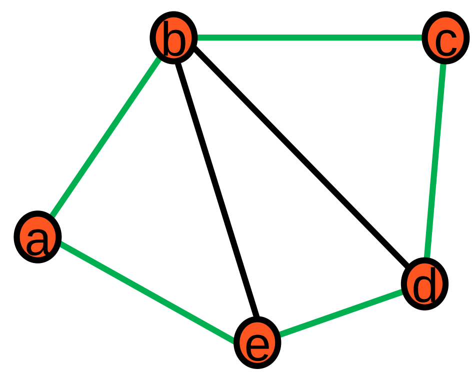
Given: a directed graph $G = (V, E)$, determine a simple cycle that contains each vertex in V

- Each vertex can only be visited once

Certificate:

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





certificate t

Hamiltonian

instance s

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

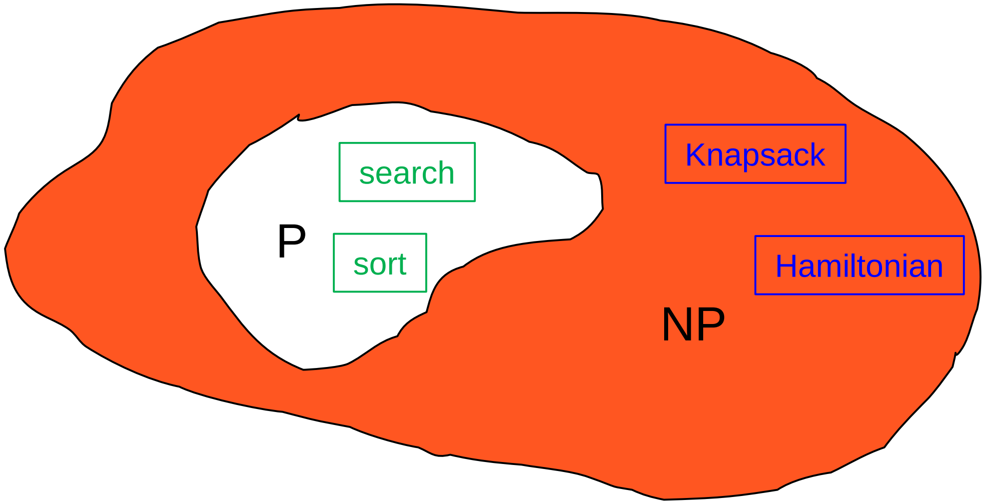
$$\left(\bar{1} \vee 1 \vee 0\right) \wedge \left(1 \vee \bar{1} \vee 0\right) \wedge \left(1 \vee 1 \vee 1\right) \wedge \left(\bar{1} \vee \bar{0} \vee \bar{1}\right)$$

$$\left(1\right) \wedge \left(1\right) \wedge \left(1\right) \wedge \left(1\right)$$

$$1$$

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

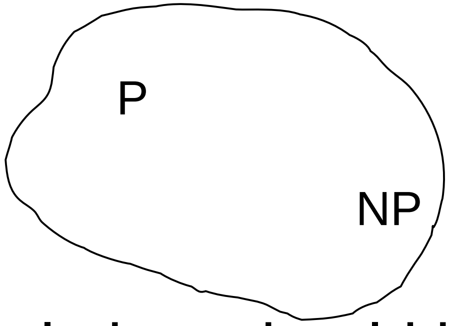
P & NP-Complete Problems

- Euler tour
  - G = (V, E) a connected, directed graph find a cycle that traverses <u>each edge</u> of G exactly once (may visit a vertex multiple times)
  - <u>Polynomial solution O(E)</u>
- Hamiltonian cycle
  - G = (V, E) a connected, directed graph find a cycle that visits <u>each vertex</u> of G exactly once
  - <u>NP-complete</u>

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 **P ≠ 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*

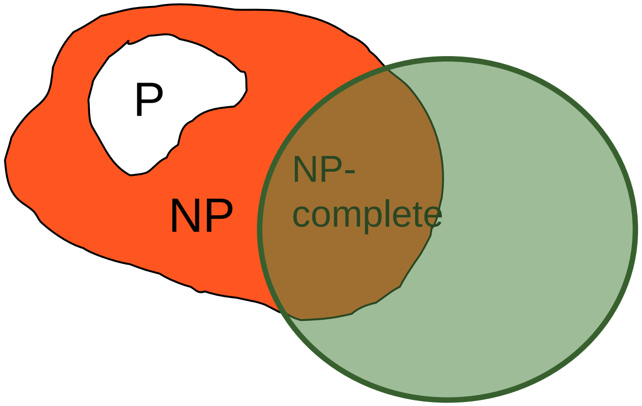
- https://www.youtube.com/watch?v=YX40hbAHx3s

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.

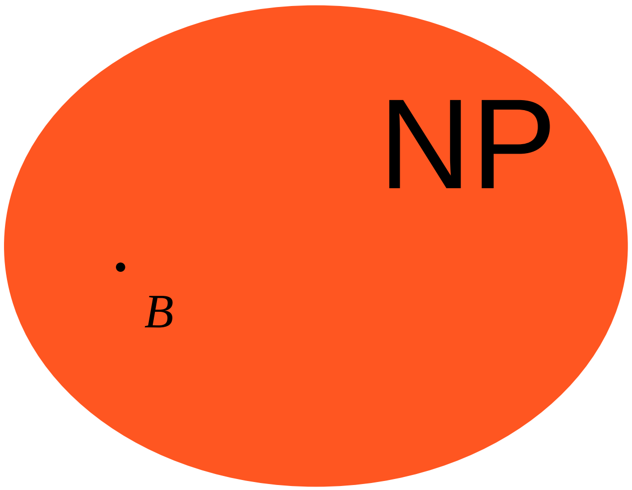
NP-Completeness (informally)



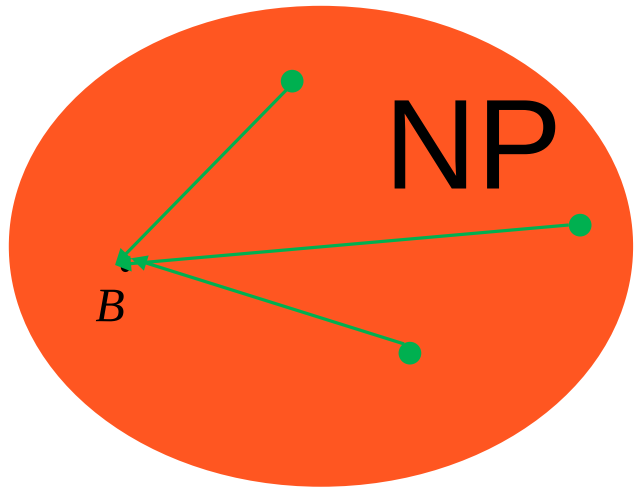
- **NP-complete** problems are defined as the hardest problems in NP
- Most practical problems turn out to be either P or NP-complete.

NP-Complete

A problem B is in NP-complete if:



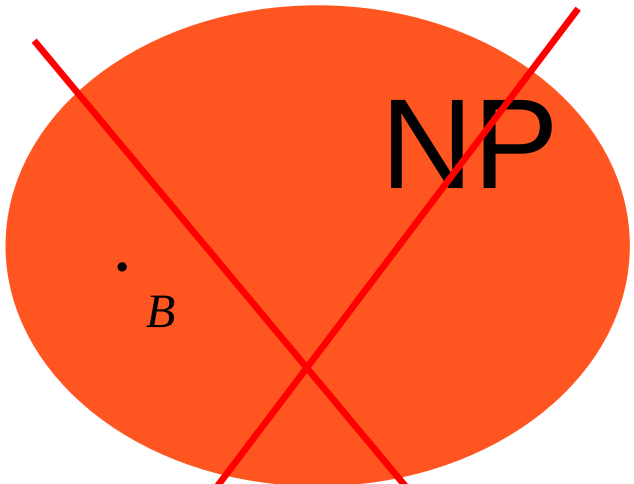
1. $B \in \mathbf{NP}$



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

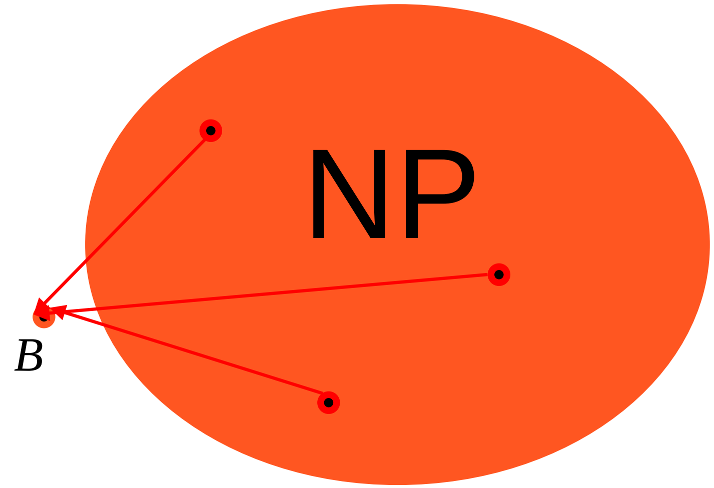
NP-~~Complete~~ Hard

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, $A \in$ **NP** and all problems X are reducible to A
- Formally: $X \leq_p A \ \forall X \in$ **NP**

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

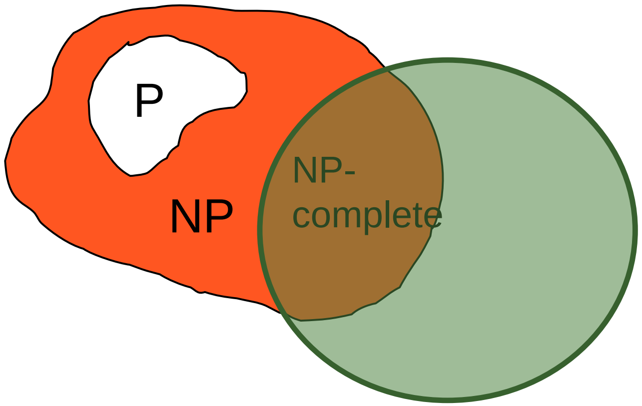
- If $B \in$ **NP** too then B is NP-Complete

NP-Completeness (formally)

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

  (1) B $\in$ **NP**

  (2) X $\leq_p$ B for all X $\in$ **NP**

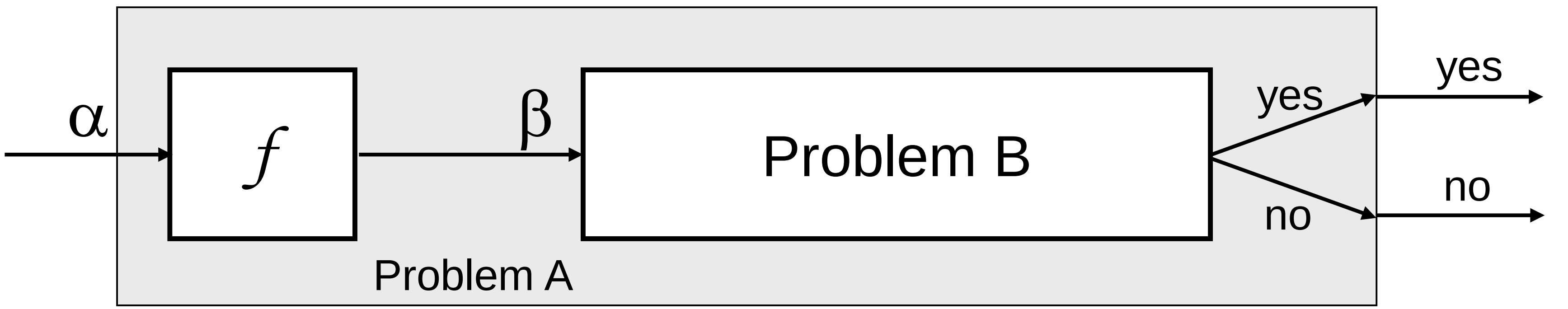


- If B satisfies only property (2) we say that B is **NP-hard**
- No polynomial time algorithm has been discovered for an **NP-Complete** problem
- No one has ever proven that no polynomial time algorithm can exist for any **NP-Complete** problem

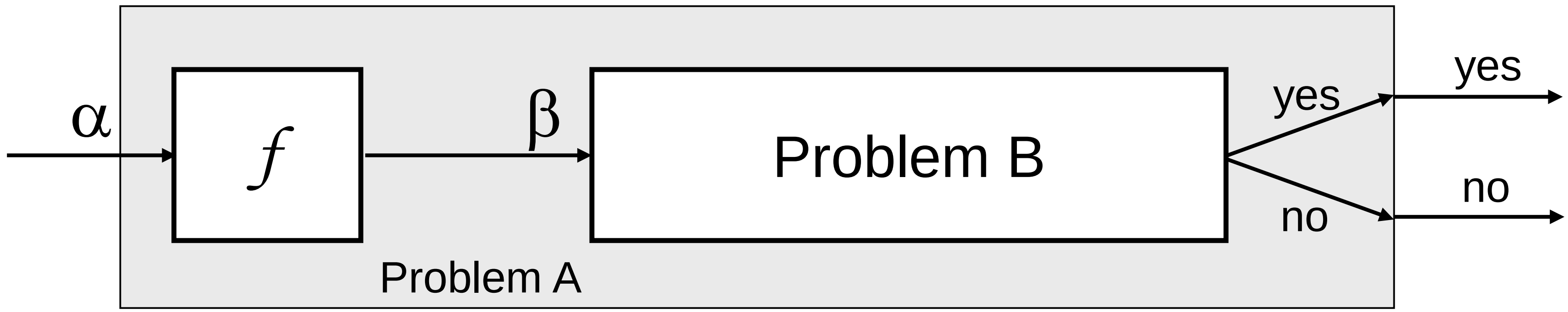
Reductions

- 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



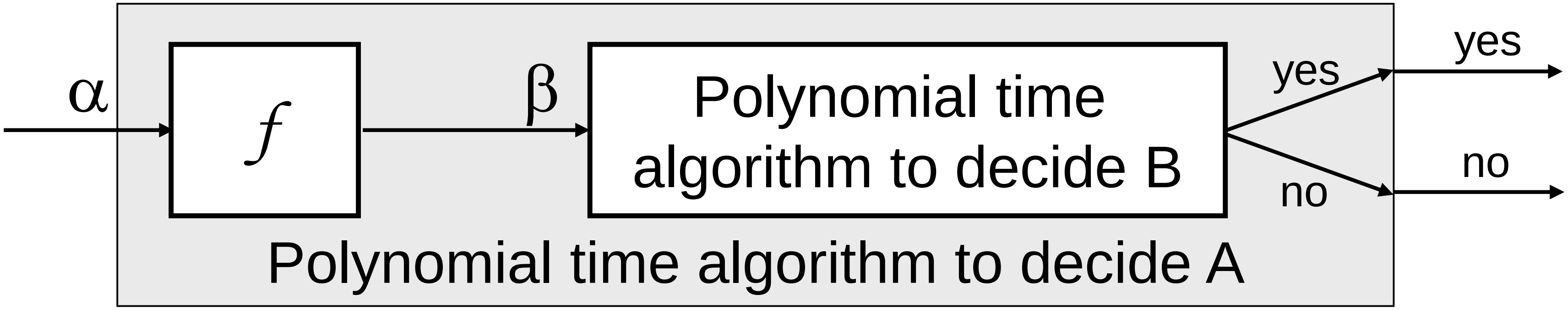
Implications of Reduction



- 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

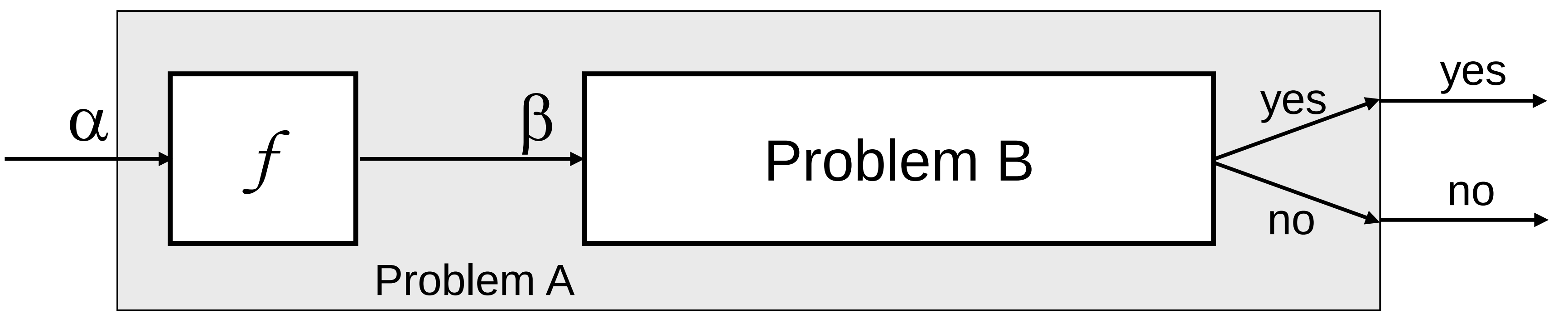
Proving Polynomial Time



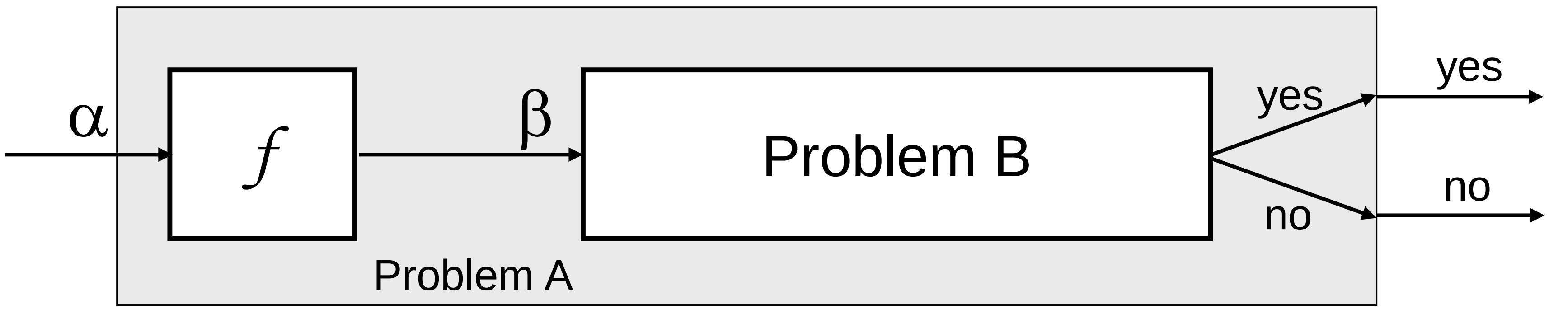
1. Use a **polynomial time** reduction algorithm to transform A into B
2. Run a known **polynomial time** algorithm for B
3. Use the answer for B as the answer for A

Reducibility - Example

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



Reductions



Proving NP-Completeness

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

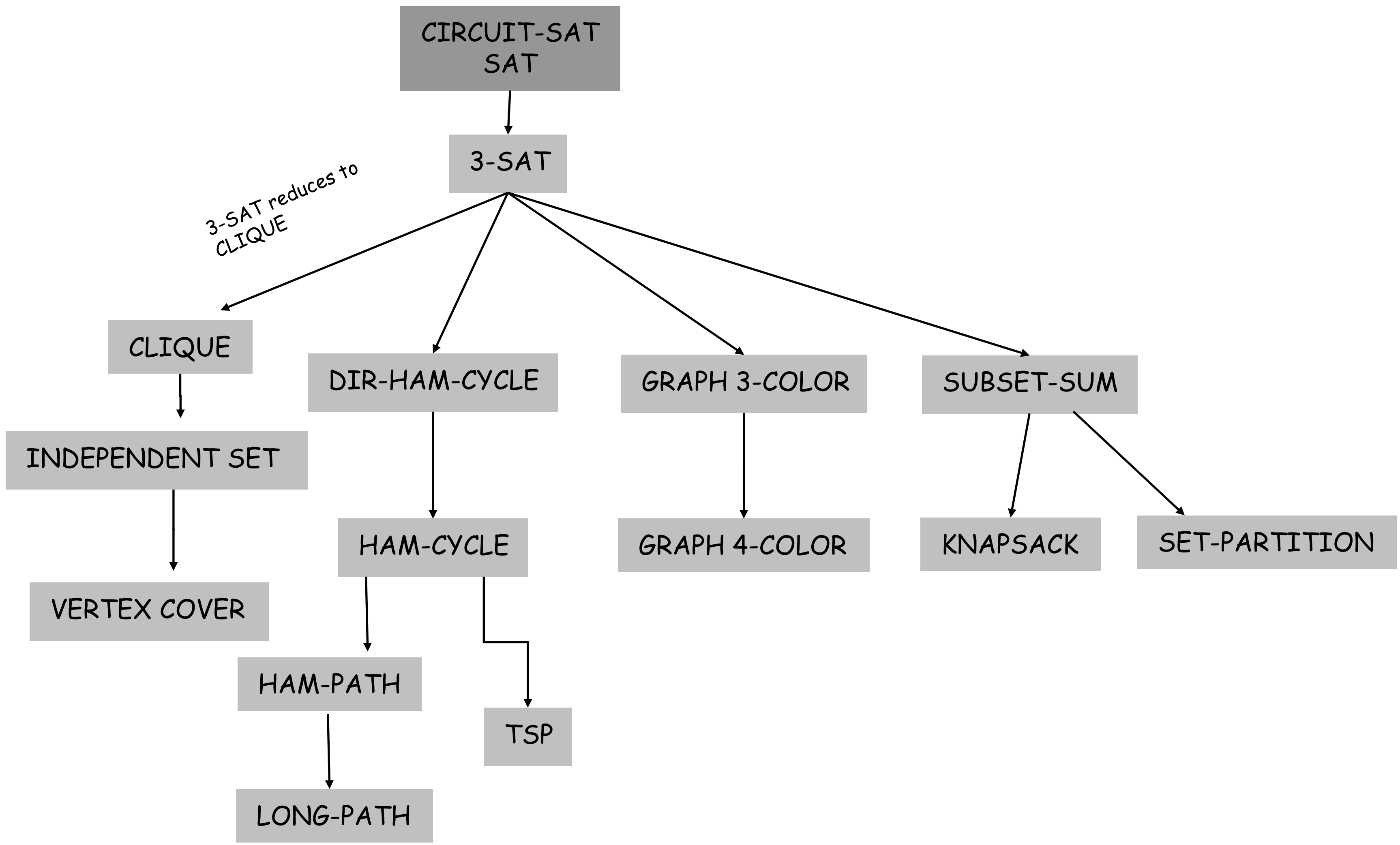
1. Pick a known NP-Complete problem A. Reduce A to B
   - Describe a polynomial time transformation/reduction that maps instances of A to instances of B, s.t. “yes” for B = “yes” for A
   - Prove the transformation works
   - Prove it runs in polynomial time

   By proving step 1 you have proved that problem B is NP-Hard
2. Prove B $\in$ **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: SAT $\in$ P if and only if P = NP

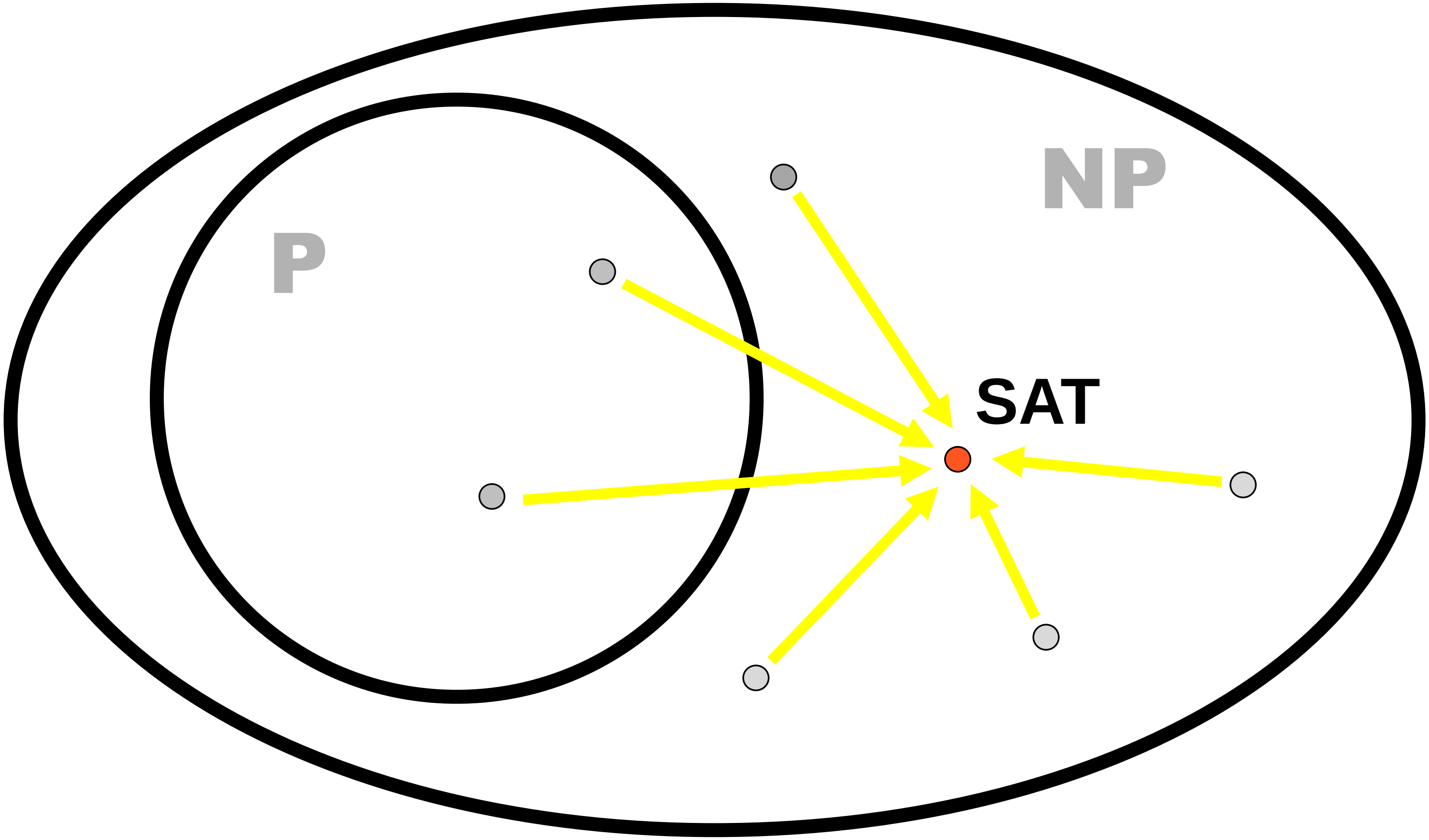
Satisfiability Problem (SAT)

- **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

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

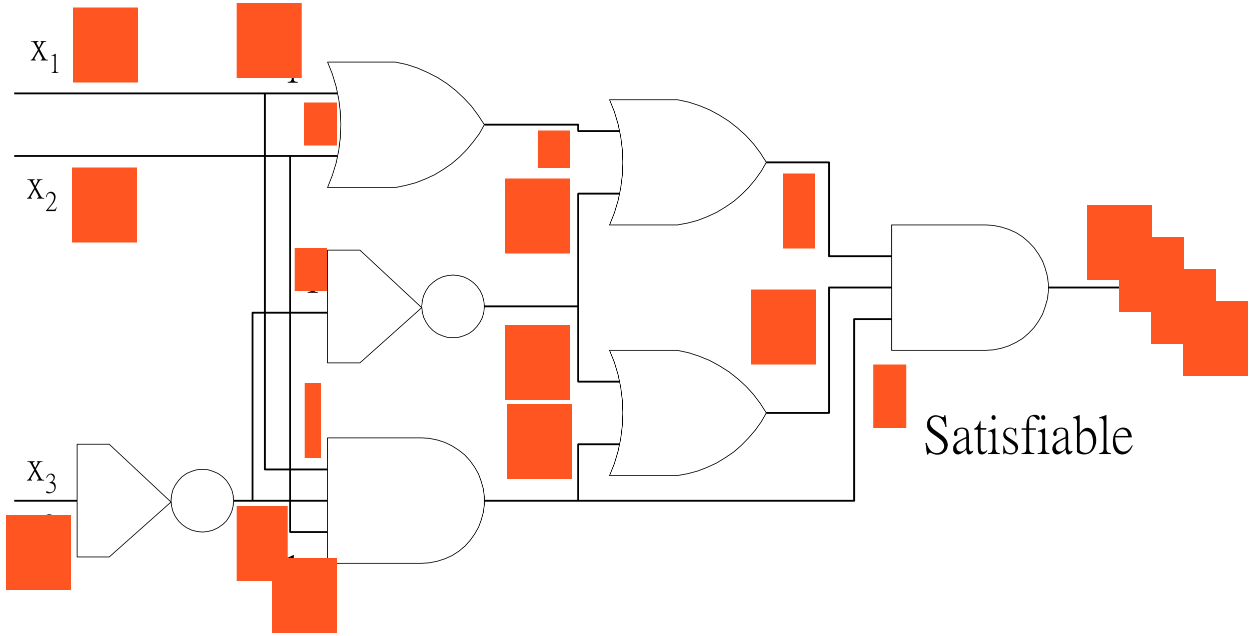
- SAT was the first problem shown to be NP-complete!

Any thing in NP $\leq_P$ SAT

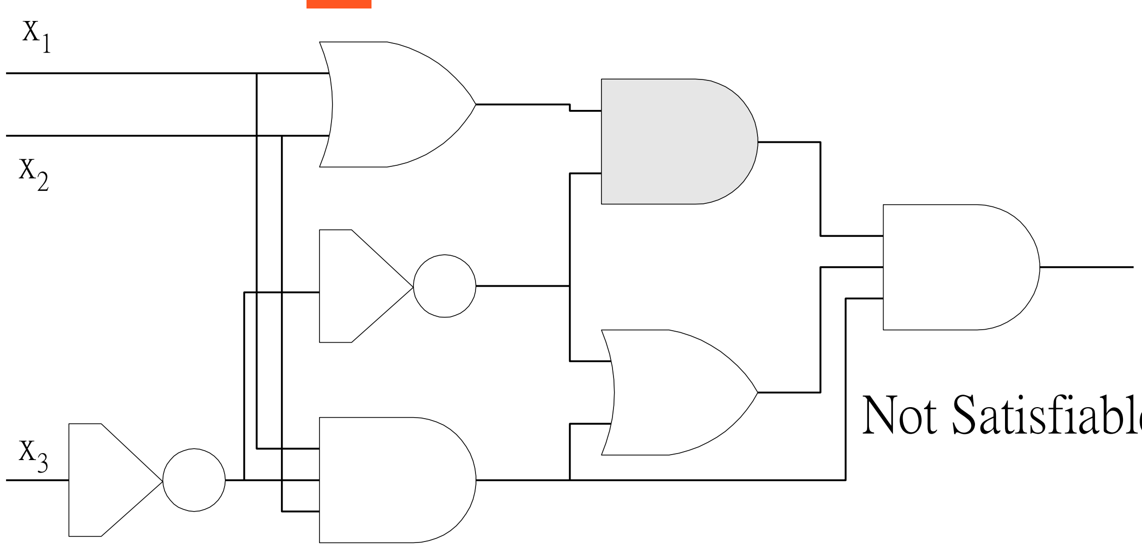


You can think of -> as “easier than”.
SAT is the hardest problem in NP.

Circuit satisfiability is in NP

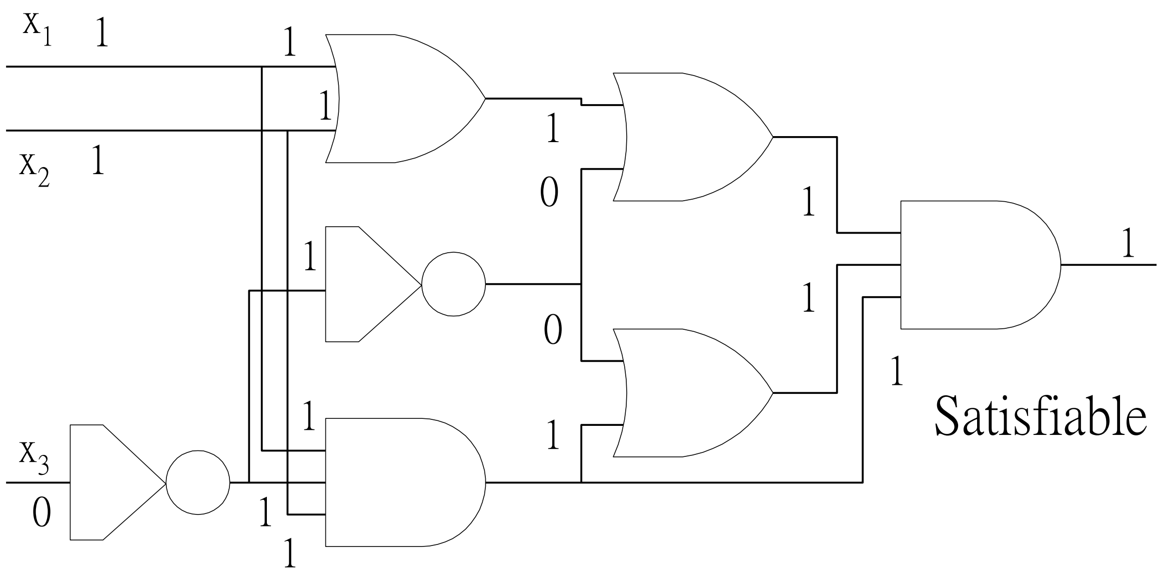


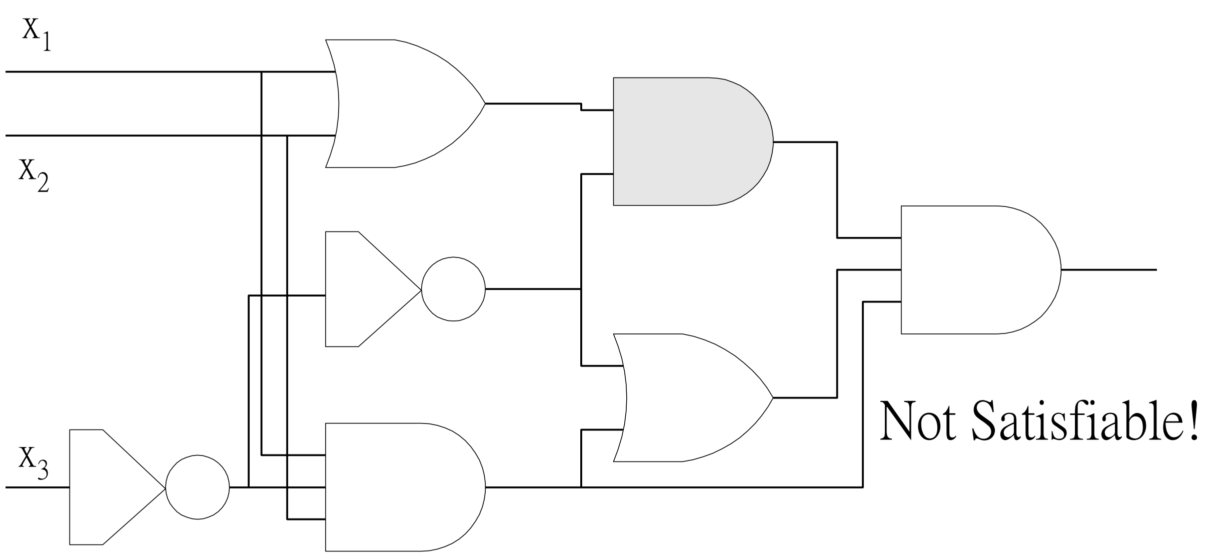
Satisfiable



Not Satisfiable!

Circuit satisfiability





CNF Satisfiability

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

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

clauses

3-SAT Satisfiability (3-CNF)

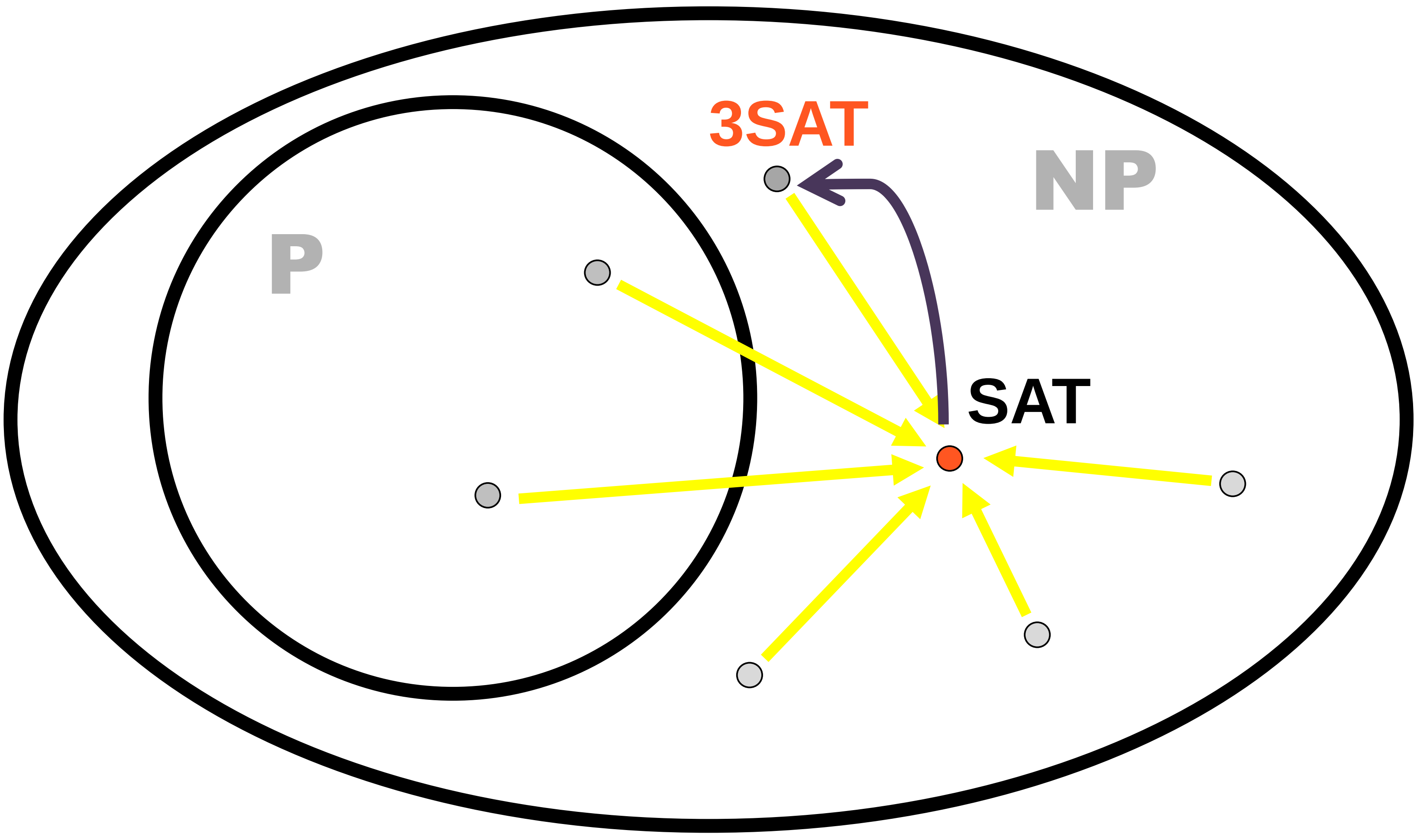
A subcase of CNF problem:

– 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 3-SAT 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	**Clauses in 3SAT**
x_1	$x_1 \lor x_1 \lor x_1$
$x_1 \lor \neg x_2$	$x_1 \lor x_1 \lor \neg x_2$
$x_1 \lor x_2 \lor x_3$	$x_1 \lor x_2 \lor x_3$
$x_1 \lor x_2 \lor x_3 \lor x_4$	$(x_1 \lor x_2 \lor z_1) \land (\neg z_1 \lor x_3 \lor x_4)$
$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$	$(x_1 \lor x_2 \lor z_1) \land (\neg z_1 \lor x_3 \lor z_2) \land$ $(\neg z_2 \lor x_4 \lor z_3) \land (\neg z_3 \lor x_4 \lor x_5)$

Clique

Clique Problem:

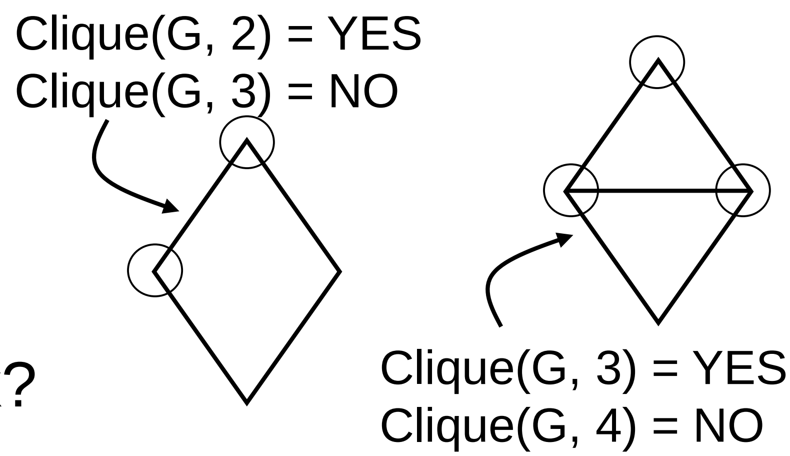
- Undirected graph G = (V, E)
- **Clique:** a subset of vertices in V all connected to each other by edges in E (i.e., forming a complete graph)
- **Size of a clique:** number of vertices it contains

Optimization problem:

- Find a clique of maximum size

Decision problem:

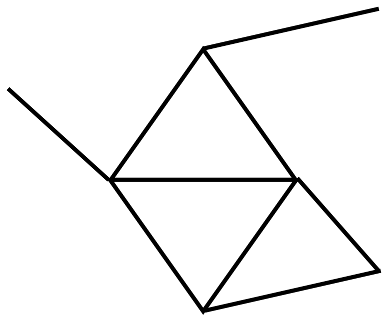
- Does G have a clique of size k?



Clique Verifier

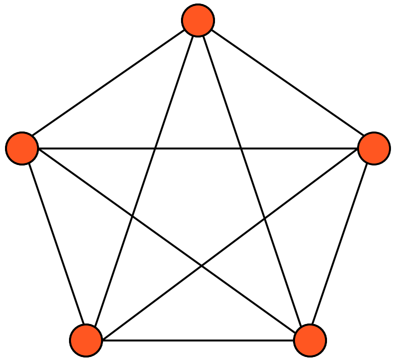
Given: an undirected graph G = (V, E)

- **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

- CLIQUE = { <G,k> | G 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-SAT $\leq_p$ Clique

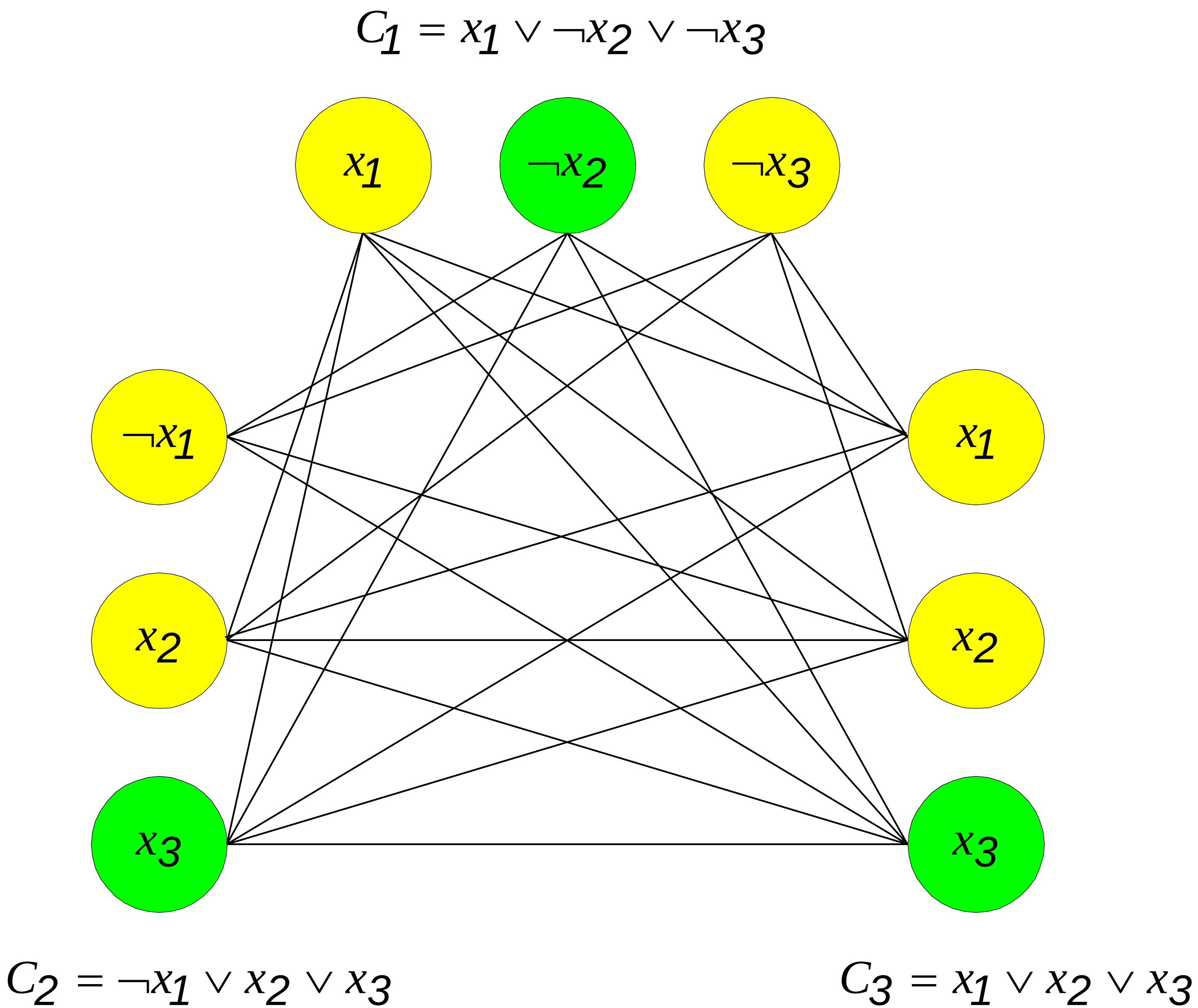
- **Idea:**
  - Construct a graph G such that Φ is satisfiable only if G has a clique of size k

Reduce 3-SAT to Clique

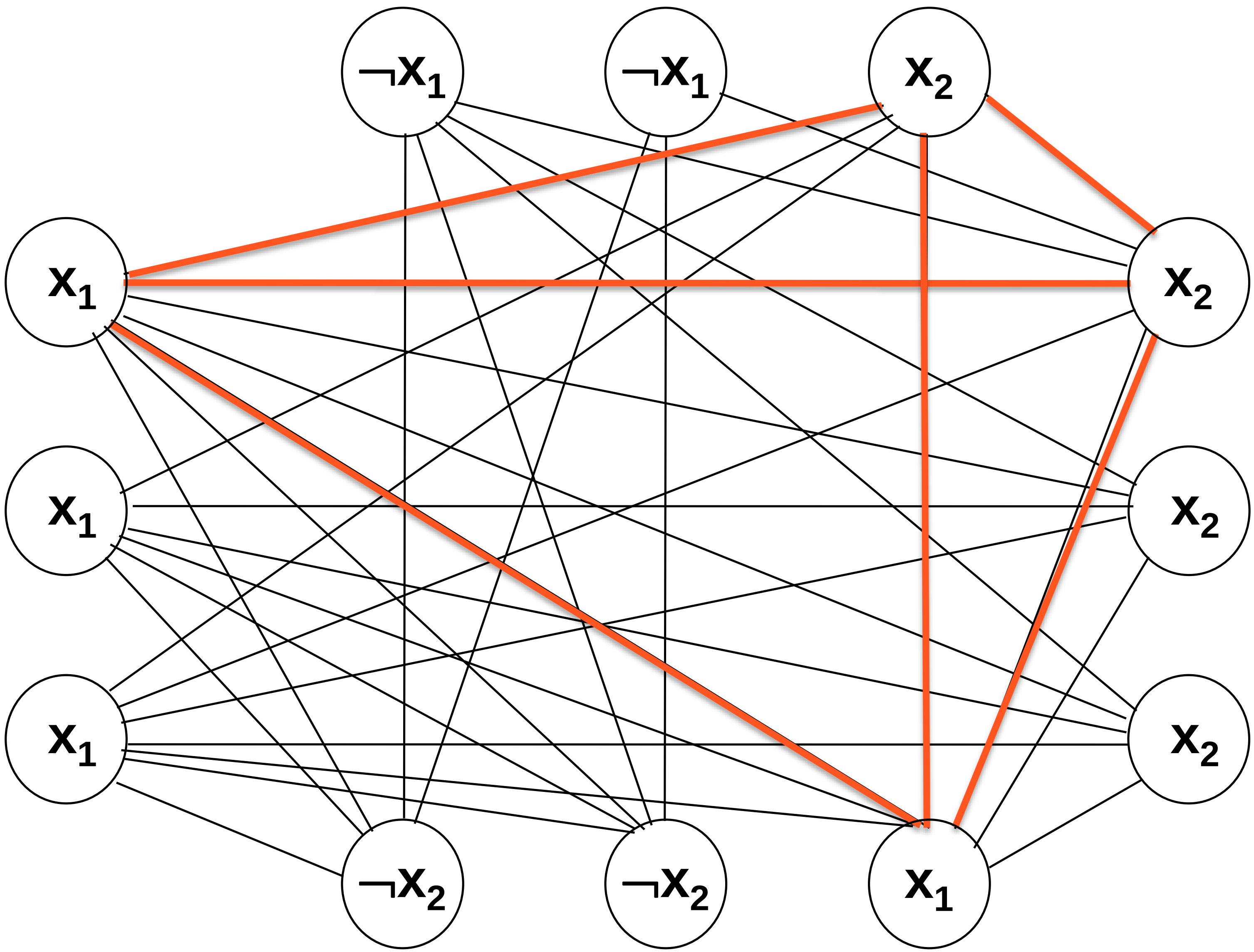
- Pick an instance of 3-SAT, Φ, and transform into <G,k> an instance of Clique
- If Φ has m clauses, we create a graph with m clusters of 3 nodes each and set $k = m$.
- Each cluster corresponds to a clause.
- Each node in a cluster is labeled with a literal from the clause.
- We do not connect any nodes 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

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



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



Hamiltonian Cycle is in NP-Complete

- Given a graph G, does G contain a Hamiltonian cycle?
- HAM-CYCLE = { G | G is a graph with a Hamitonian Cycle}
- A Hamiltonian cycle is a cycle passing every vertex exactly once.

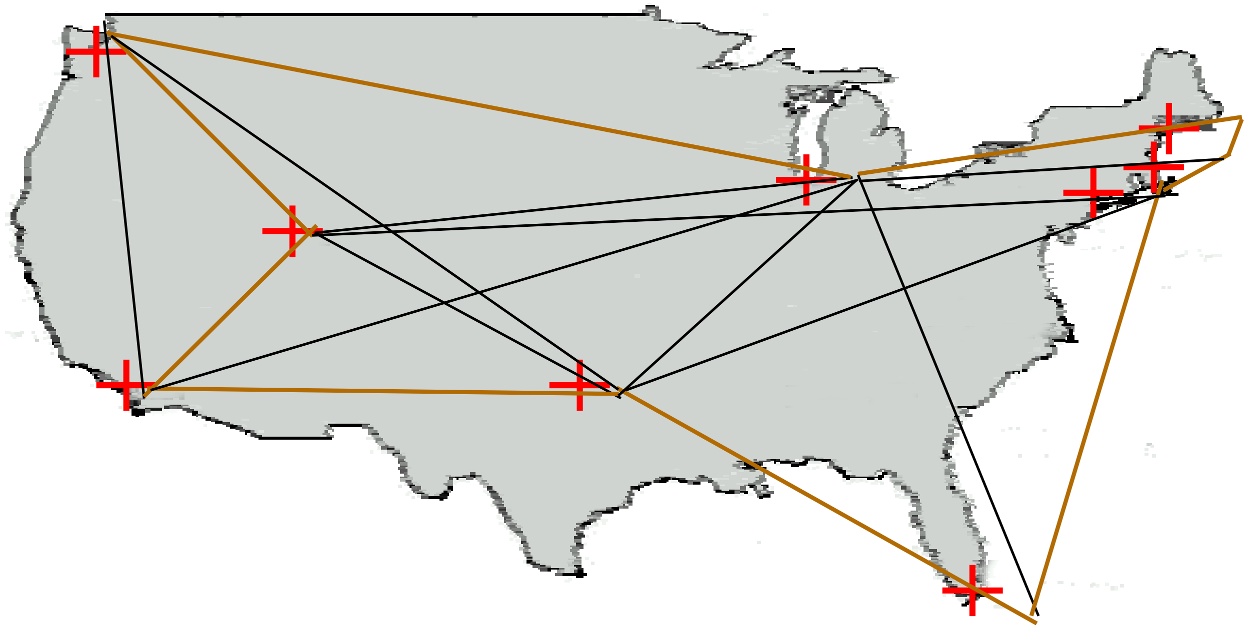
Hamiltonian Cycle is in NP-Complete

- Why is Ham-Cycle in NP?
  - Given a cycle it can be verified in polynomial time.

- Ham-Cycle is in NP-Complete
  - Proof in Jeff Erickson's Algorithms Section 12.11
  - Vertex Cover $\leq_P$ Ham-Cycle
  - 3-SAT $\leq_P$ Ham-Cycle

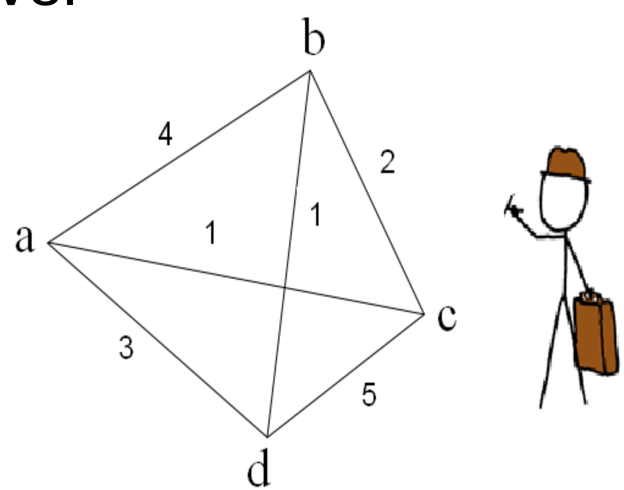
Traveling Salesman Problem

- A traveling salesman 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



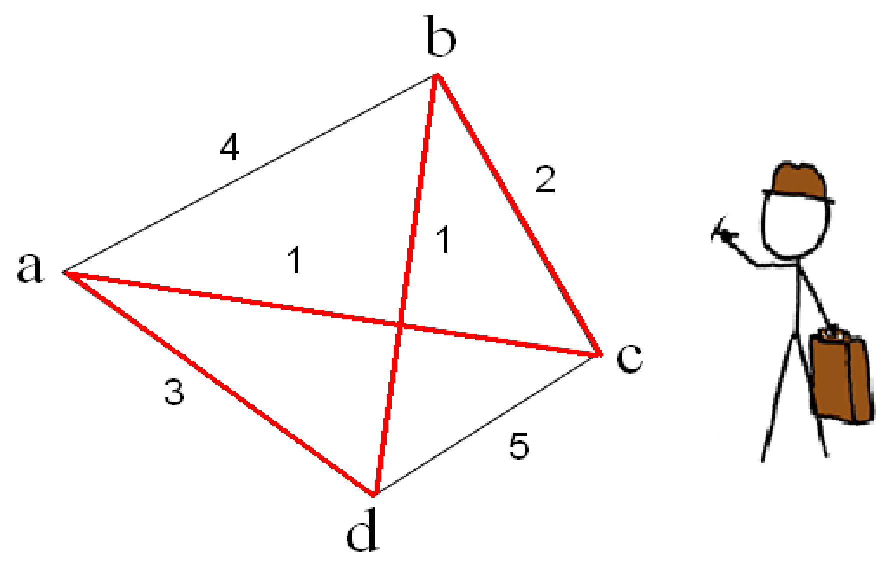
Traveling Salesman Problem

- 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 = { $<G,c,k>$: $G=(V,E)$ is a complete graph,

c is a function from $(V \times V) \rightarrow N$, $k \in$ N and

G has a traveling salesman tour with cost at most k}

G = ({a, b, c, d},{(a,b),..(c,d)}) c(a,b) = 4, c(a,c) = 1,…c(c,d) = 5

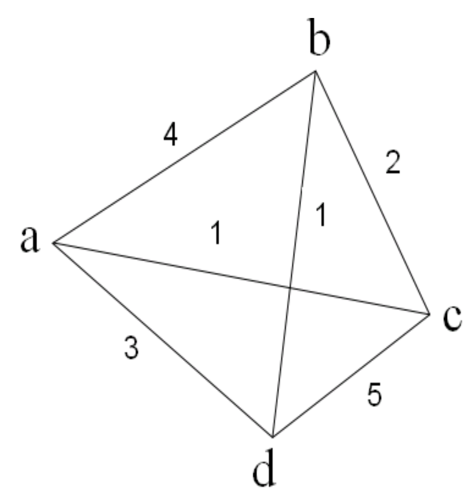
Suppose k = 15

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 $A \in$ 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.

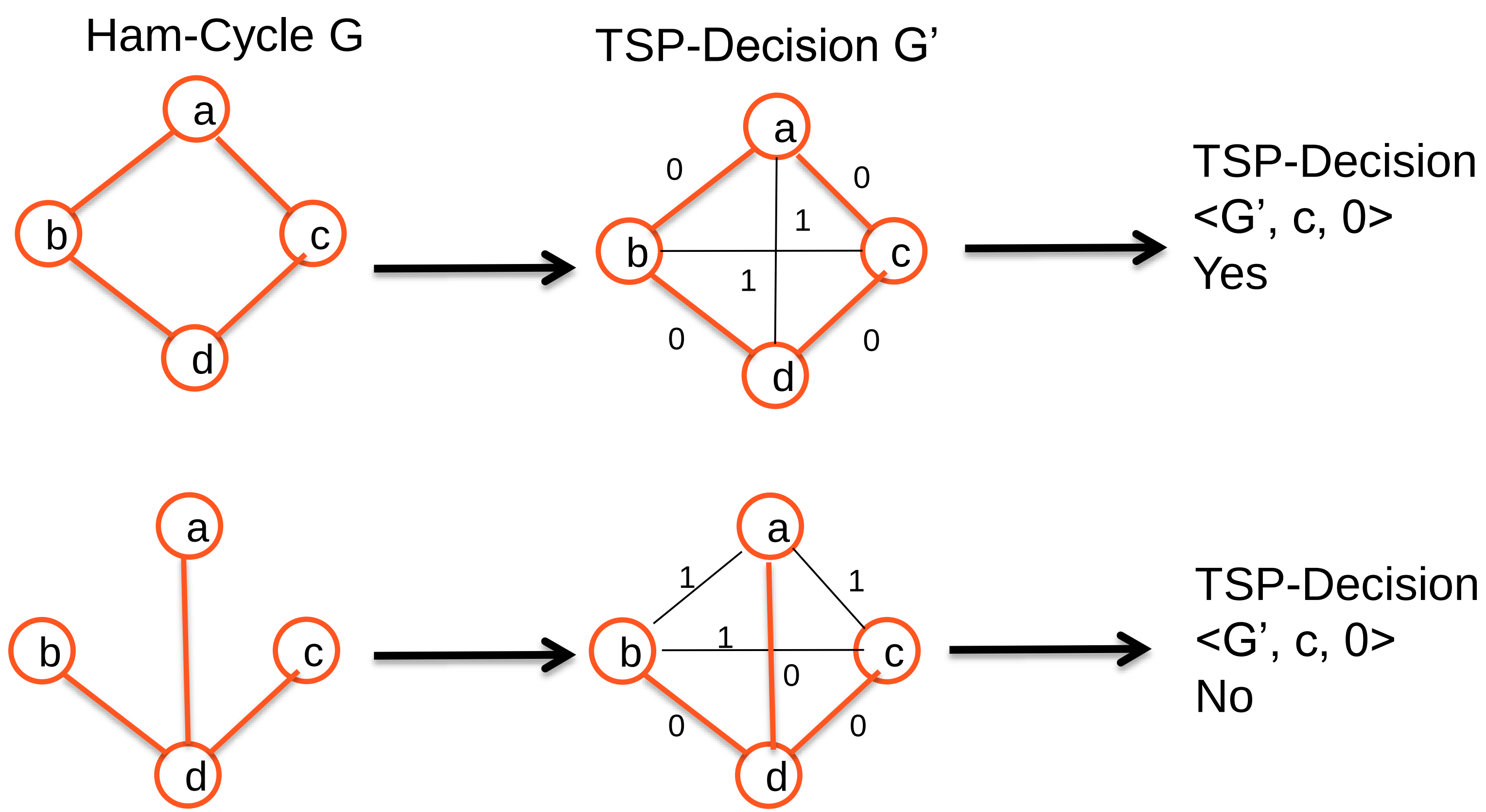
Given an instance of the problem the certificate is the sequence of *n* vertices (cities) in the tour.

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’
a
b
c
d
0
0
1
1
0
0
TSP-Decision
<G’, c, 0>
Yes
a
b
c
d
1
1
1
0
0
0
TSP-Decision
<G’, c, 0>
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 Ham-cycle $\in$ 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.

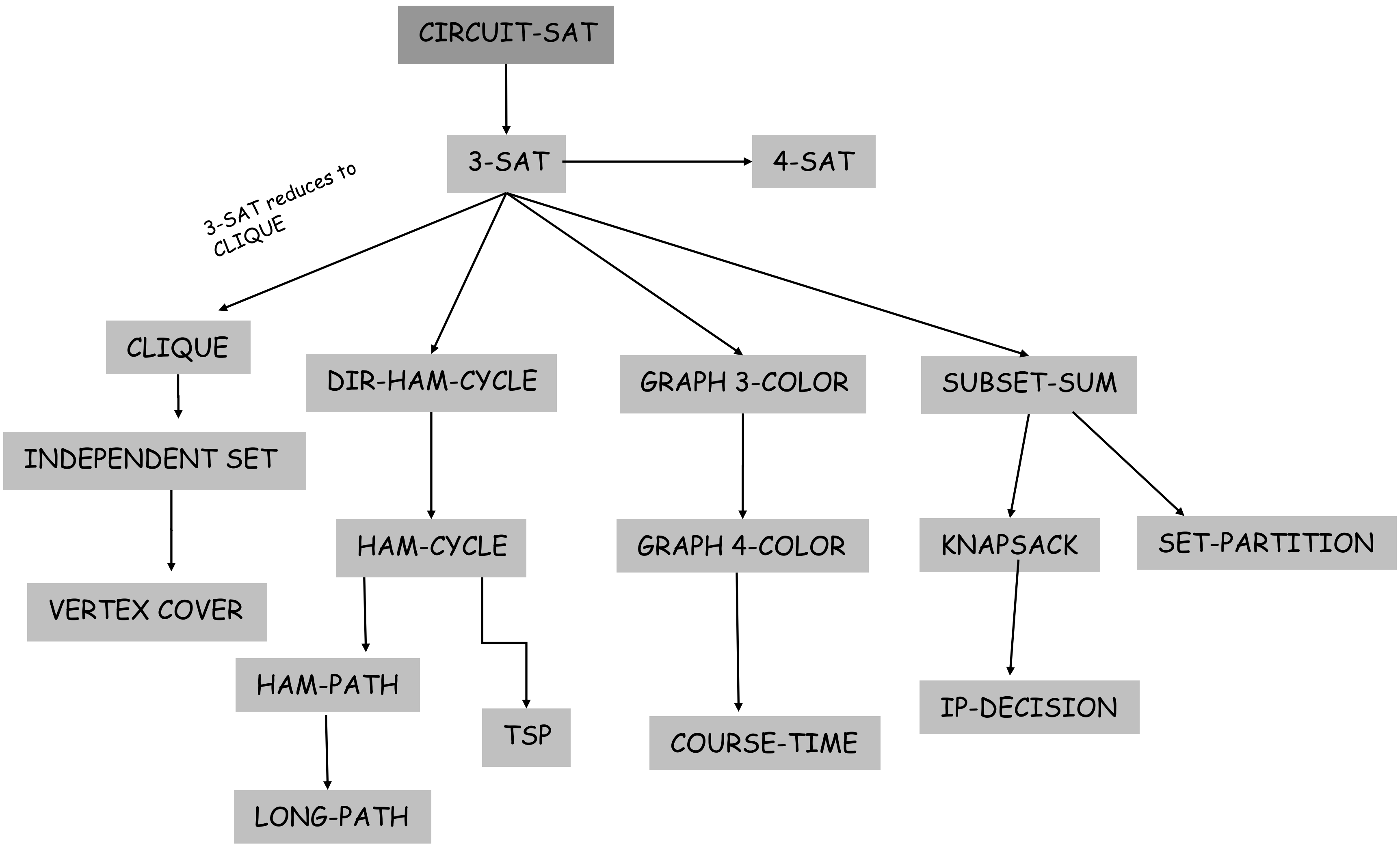
Thus ***h′*** contains only edges in ***E***.

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

NP-Completeness

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



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 $\Sigma_{y \in S'} y = t$.

Examples of SUBSET-SUM

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

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

… because 11 cannot be made out of {2,4,8}

$$\text{SUBSET-SUM} = \{\langle S, t \rangle \mid S = \{x_1, \ldots, x_k\}$$
$$\text{there is a subset } R \subseteq S$$
$$\text{such that } \sum_{y \in R} y = t\}$$

SUBSET-SUM is NP-Complete

Proof:

1. Show SUBSET-SUM is in NP.

2. Show 3SAT$\leq_p$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 $\Sigma_{y \in S'} y = t$.

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:

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

clause:

					1	2	3	4
$+x_1$	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
dummies					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

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

Reducing 3SAT to SubSet Sum

- Let $\phi \in$3CNF with k clauses and ℓ variables $x_1,\ldots,x_\ell$.
- Create a Subset-Sum instance $<S_\phi,t>$ by: $2\ell+2k$ elements of $S_\phi = \{y_1,z_1,\ldots,y_\ell,z_\ell,g_1,h_1,\ldots,g_k,h_k\}$
  - 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

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

1	0	0	0	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

+

$+x_1$	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

$x_1, \bar{x}_2, \bar{x}_3, x_4$ 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	?	?	?

+

$+x_1$	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

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

Proof 3SAT $\leq_P$ Subset Sum

- For every 3CNF ϕ, take target t=1…13…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.

$$\boxed{\phi \in 3SAT \text{ if and only if } \left\langle \mathrm{S}_\phi, 1...13...3 \right\rangle \in \mathrm{SubsetSum}}$$

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.

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

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

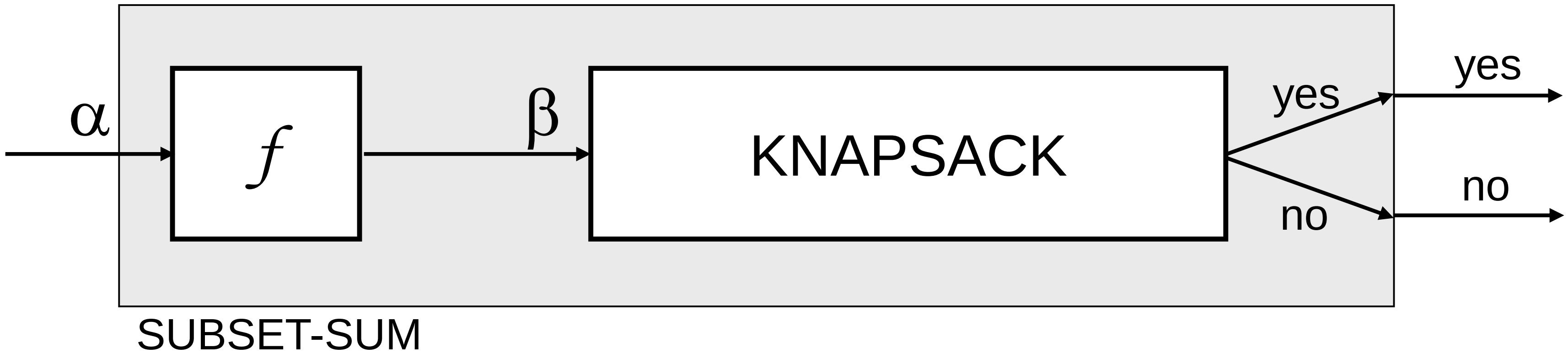
1) Show NP – Verify a solution in polynomial time

Given a certificate solution $X = \{ x_1, \ldots, x_n \}$ 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 = { <*S*, *t*> | *S* = *{y_1, …, y_k} and for some* Subset T = {y_j, …, y_l } $\subseteq$ S, Σy_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 = < {12, 6, 8, 13, 20}, 26 >

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

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

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

SUBSET-SUM to PARTITION

- SET - PARTITION = { $x_1, x_2, \ldots x_k$ | we can split the integers into two sets which sum to half }
- SUBSET-SUM = { $\langle x_1, x_2, \ldots x_k, t \rangle$ | there exists a subset which sums to t }

- If I can solve SET- PARTITION, how can I use that to solve an instance of SUBSET-SUM?

Prove that SET-PARTITION is in NP-complete

SET-PARTITION. Given a set S can we partition S into two sets X and $\overline{X}$ = S-X (both sets are nonempty) such that the sum of the elements in X equals the sum of the elements in S-X. That is

$$\sum_{y \in X} y = \sum_{y \in X-S} y$$

To show that SET-PARTITION is NP-Complete, we need to show:

(1) That SET-PARTITION ∈ NP. Given a partition of set S we can verify in polynomial time that the two subsets X and S-X have equal sums by adding the values in each set. This clearly takes time O(n) where n is the number of elements in S.

(2) That some NP-Complete problem A can be reduced to SET-PARTITION in polynomial time and the original problem A has a yes solution if and only if SET-PARTITION has a yes solution.

We will select A to be SUBSET-SUM which has been proven to be in NP-Complete. SUBSET-SUM is defined as follows: Given a set S of integers and a target number t, find a subset $Y \subseteq S$ such that the members of Y add up to exactly t.

We will need to show a polynomial time reduction from SUBSET-SUM to SET-PARTITION, SUBSET-SUM $\leq_p$ SET-PARTITION

Let s be the sum of numbers in S. Feed S' = S ∪ {s − 2t} into SET-PARTITION and answer YES if and only if SET-PARTITION answers YES.

This reduction takes polynomial time since all we did was add a single element, s − 2t to S. To calculate s-2t we must computer the sum of all numbers in S which takes O(n) time.

Note: sum(S') = sum(S) + (s-2t) = s + (s-2t) = 2s − 2t = 2(s – t) .

We must show that <S, t> ∈ SUBSET-SUM iff <S’> ∈ SET-PARTITION. In other words There exists a subset of S that sums to t if and only if there exists a set partition of S’ with equal sums.

1) If Y is a solution to <S,t> SUBSET-SUM then S’ has a SET-PARTITION.

If there exists a subset Y of numbers in S that sum to t then the remaining numbers in S-Y sum to s − t. Now S’ = (S-Y) ∪ Y ∪ {s-2t} such that we can partition S’ into two sets (S-Y) and Y ∪ {s-2t} with each partition summing to s − t. Therefore there is a solution to SET-PARTITION.

2) If there exists a partition of S' then there exists a solution to <S, t> ∈ SUBSET-SUM.

Recall sum(S') = 2(s – t) and S' = S ∪{s-2t} .

If there exists a partition of S' into two sets X and S'-X such that the sum over each set is s−t then one of these sets say X must contain the number s−2t. By removing it, we get a set X' = X- {s-2t} with sum(X') = (s – t) - (s-2t) = t, and since S' = S ∪ {s − 2t} all of the elements in X' are in S. Therefore there exists a subset X' of S that sums to t.

Since, SET-PARTITION ∈ NP and SUBSET-SUM $\leq_p$ SET-PARTITION, SET-PARTITION is in NP-Complete.

Recall 3-SAT

A special of CNF problem: Each clause contains three boolean literals

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

3-SAT is NP-Complete

- 3-SAT is in NP
- SAT $\leq_p$ 3-SAT

Is 4-SAT NP-Complete?

Instance: A collection of clause *C* where each clause contains exactly *4* literals, Boolean literals x.

Question: Is there a truth assignment to *x* so that each clause is satisfied?

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

NP-Completeness Proof Method

To show that 4-SAT is NP-Complete:

1. Show that 4-SAT is in NP.

   Give a polynomial time algorithm for verifying a solution.

2. Show that 3-SAT $\leq_P$ 4-SAT

   Give a polynomial algorithm F to transform 3-SAT into an instance of 4-SAT such that:

   – For any instance X of 3-SAT if X is True then F(X) is true for 4-SAT and

   – For any instance F(Y) of 4-SAT if F(Y) is True then Y is true for 3-SAT

Step 2 alone shows that a problem is NP-Hard

1. Show that 4-SAT is in NP.

Give a polynomial time algorithm for verifying a 4-SAT solution. This algorithm takes a 4-SAT instance and proposed truth assignments as input and evaluates the 4-SAT instance. If the 4-SAT instance evaluates to true the algorithm outputs yes; otherwise the algorithm outputs no. Thus runs in polynomial time.

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

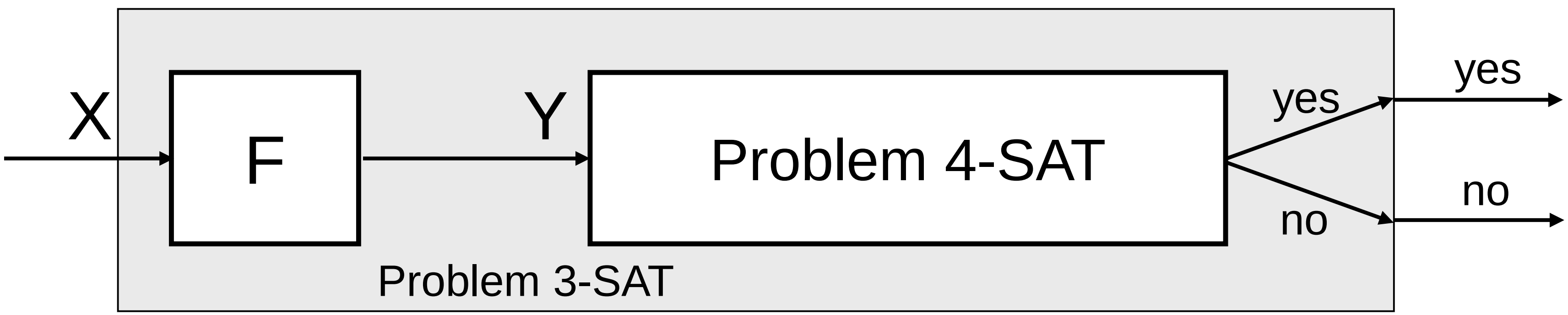
Candidate solution $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, $x_4 = 1$

yes

2. Show that 3-SAT $\leq_P$ 4-SAT

Give a polynomial algorithm F to transform 3-SAT into an instance of 4-SAT such that:

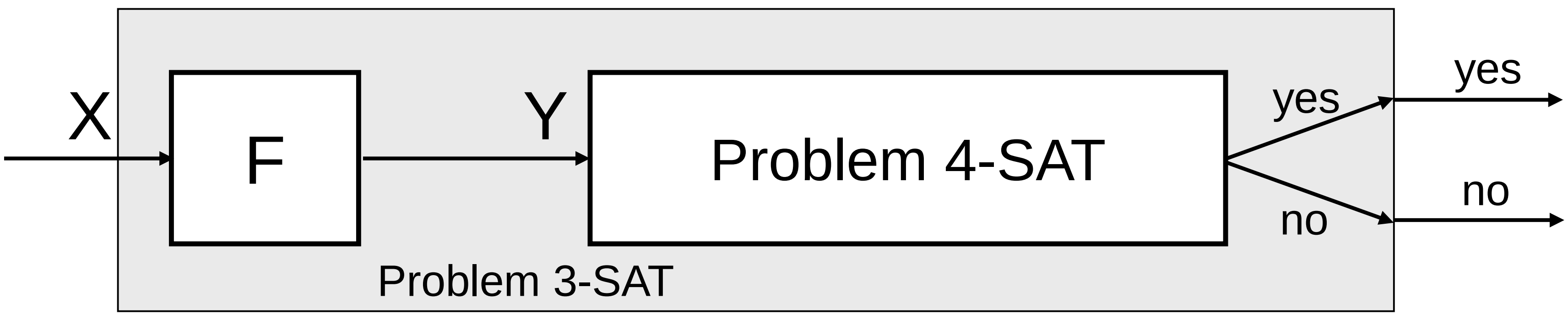
- For any instance X of 3-SAT if X is True then F(X) is true for 4-SAT and
- For any instance F(Y) of 4-SAT if F(Y) is True then Y is true for 3-SAT



2. Show that 3-SAT $\leq_P$ 4-SAT

Give a polynomial algorithm F to transform 3-SAT into an instance of 4-SAT such that:

- For any instance X of 3-SAT if X is True then F(X) is true for 4-SAT and
- For any instance F(Y) of 4-SAT if F(Y) is True then Y is true for 3-SAT



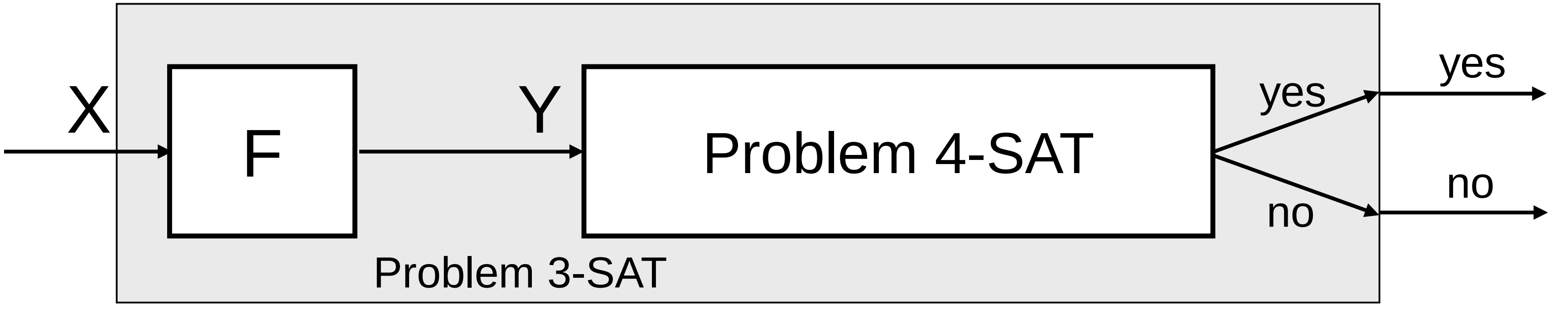
$$X = \left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right)$$

$$Y = \left(\overline{x_1} \vee x_2 \vee x_3 \vee h\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3 \vee h\right) ???$$

2. Show that 3-SAT $\leq_P$ 4-SAT

Give a polynomial algorithm F to transform 3-SAT into an instance of 4-SAT such that:

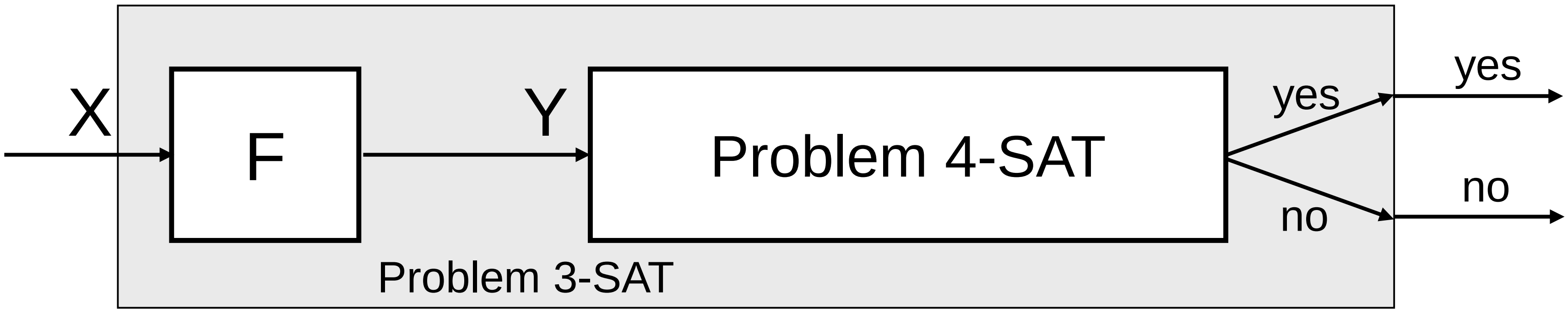
– For any instance X of 3-SAT if X is True then F(X) is true for 4-SAT and
– For any instance F(Y) of 4-SAT if F(Y) is True then Y is true for 3-SAT



$$X = \left(\overline{x_2} \vee \overline{x_2} \vee \overline{x_2}\right) \wedge \left(x_2 \vee x_2 \vee x_2\right)$$

$$Y = \left(\overline{x_2} \vee \overline{x_2} \vee \overline{x_2} \vee h\right) \wedge \left(x_2 \vee x_2 \vee x_2 \vee h\right) ???$$

2. Show that 3-SAT $\leq_P$ 4-SAT



$$X = \left(\overline{x_2} \vee \overline{x_2} \vee \overline{x_2}\right) \wedge \left(x_2 \vee x_2 \vee x_2\right)$$

$$Y = \left(\overline{x_2} \vee \overline{x_2} \vee \overline{x_2} \vee h\right) \wedge \left(x_2 \vee x_2 \vee x_2 \vee h\right) \wedge \left(x_2 \vee x_2 \vee x_2 \vee \overline{h}\right) \wedge \left(\overline{x_2} \vee \overline{x_2} \vee \overline{x_2} \vee \overline{h}\right)$$

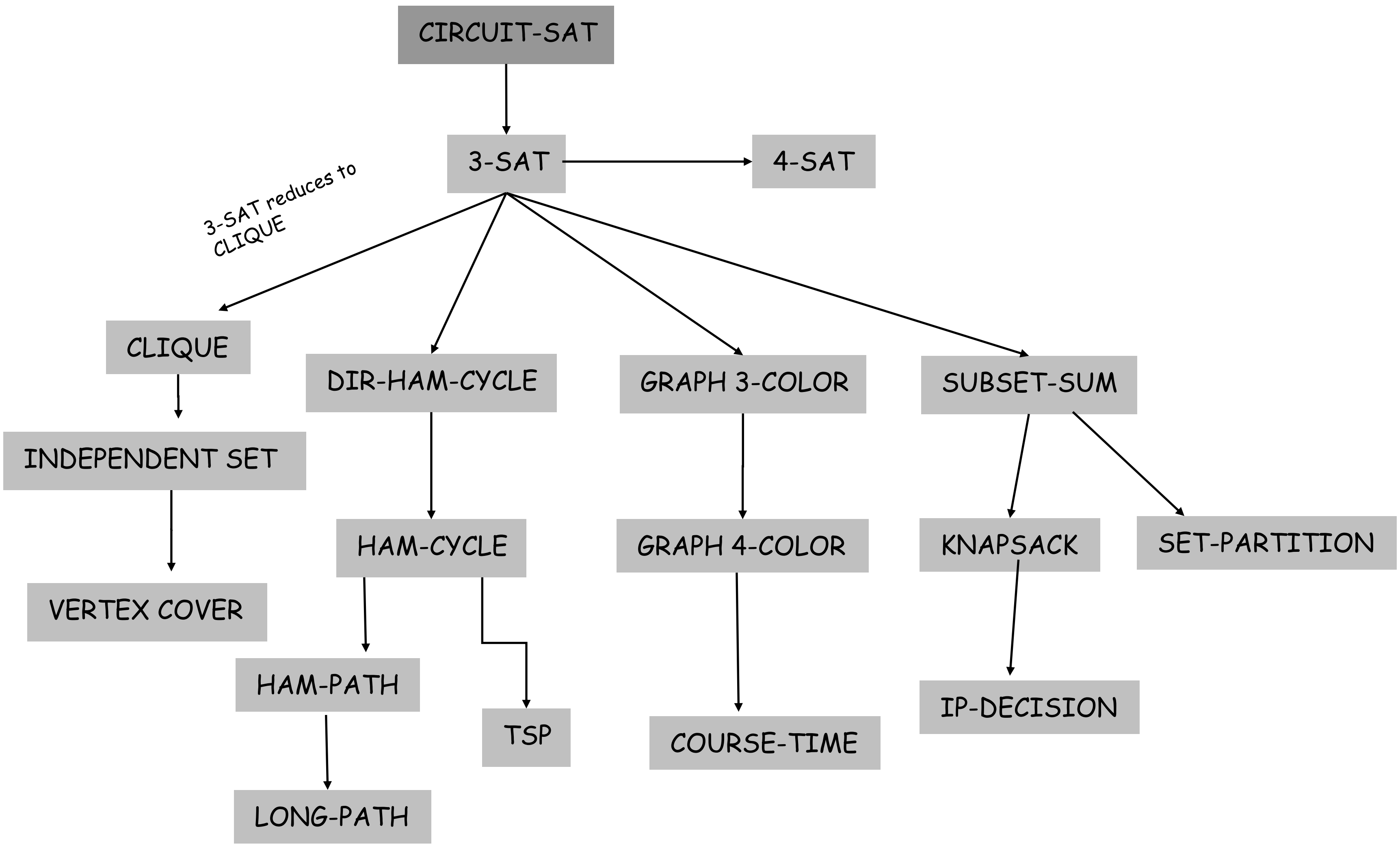
To prove that 4-SAT is NP-hard, we reduce 3-SAT to 4-SAT as follows. Let ϕ denote an instance of 3-SAT. We convert ϕ to a 4-SAT instance ϕ' by turning each clause $(x \vee y \vee z)$ in ϕ to $(x \vee y \vee z \vee h) \wedge (x \vee y \vee z \vee \neg h)$, where h is a new variable. Clearly this is polynomial-time doable.

$\Rightarrow$ If a given clause $(x \vee y \vee z)$ is satisfied by a truth assignment, then $(x \vee y \vee z \vee h) \wedge (x \vee y \vee z \vee \neg h)$ is satisfied by the same truth assignment with h arbitrarily set. Thus if ϕ is satisfiable, ϕ' is satisfiable.

$\Leftarrow$ Suppose ϕ' is satisfied by a truth assignment T. Then $(x \vee y \vee z \vee h) \wedge (x \vee y \vee z \vee \neg h)$ must be true under T. As h and $\neg h$ assume different truth values, $x \vee y \vee z$ must be true under T as well. Thus ϕ is satisfiable.

NP-Completeness

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



COURSE-TIME is NP-Complete

The COURSE-TIME assignment problem is as follows.

- **Input:** A set of m students S, a set of n classes C, a positive integer K and, for each student $x \in$ S, a list L of courses that student *x* wants to take.
- **Question:** Is it possible to schedule courses into only K time slots such that each student to take all of his or her desired courses, without any time conflicts?

Prove that the COURSE-TIME assignment problem is NP-complete. (**Hint:** Use that fact that K-COLOR is in NP-Complete.)

COURSE-TIME is NP-Complete

1. Show that COURSE-TIME is in NP.

   Give a polynomial time algorithm for verifying a solution.

2. Show that K-COLOR $\leq_P$ COURSE-TIME

Step 2 alone shows that a problem is NP-Hard

COURSE-TIME is NP-Complete

1. Show that COURSE-TIME is in NP.

Give a polynomial time algorithm for verifying a solution.

A certificate is an assignment of courses to time slots. To check the certificate, check that each student's list of courses has no time conflicts, and that only the K number of time slots have been used.

Number of students = m

Number of courses n = max size of student list L

Time to check conflicts O(m*n)

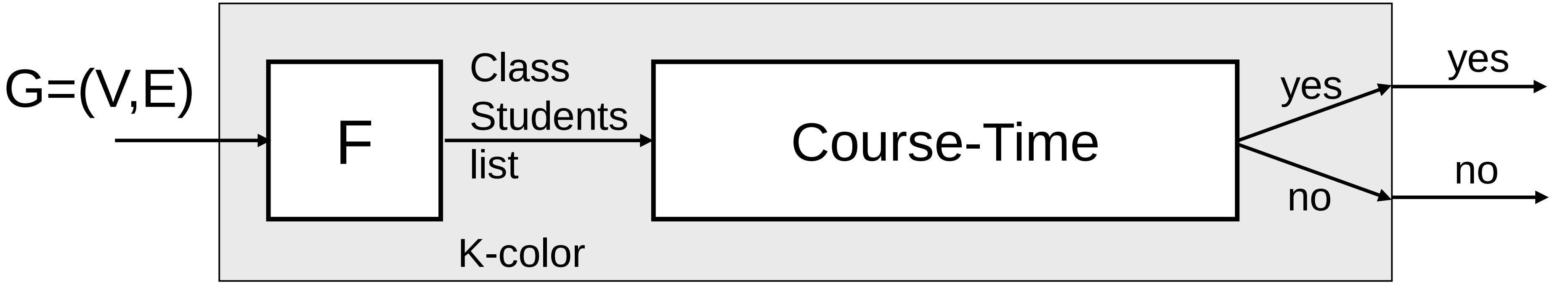
Time to check K time slots = O(n)

COURSE-TIME is NP-Complete

2. Show that K-COLOR $\leq_P$ COURSE-TIME

The K-COLOR problem is as follows.

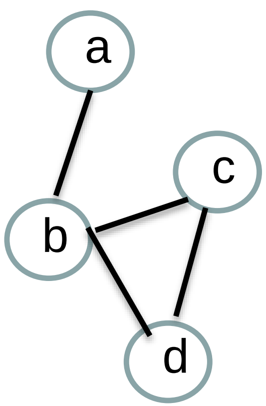
- **Input:** An undirected graph *G* and a positive integer *K*.
- **Question:** Is it possible to color the vertices of *G* using no more than *K* colors (coloring each vertex just one color) so that no two adjacent vertices have the same color?



K-COLOR $\leq_P$ COURSE-TIME

The K-COLOR problem reduces to the course assignment problem via reduction defined as follows. Given graph *G* and positive integer *K*, produce a course assignment problem whose courses are the vertices of *G*, and with *K* available time slots. Include, in the problem, a student for each edge of *G*. The student associated with edge (*u*,*v*) wants to take courses *u* and *v*, and nothing more.

Instance of 3-COLOR

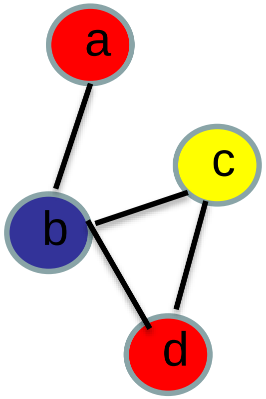


Instance of COURSE-TIME

Students S = { ab, bc, bd, cd }
Class lists L student
ab = {a, b}
bc = {b, c}
bd = {b, d}
cd = {c, d}
Courses = { a, b, c, d}
K = 3 time slots

K-COLOR $\leq_P$ COURSE-TIME

Instance of 3-COLOR



Instance of COURSE-TIME

Students S = { ab, bc, bd, cd }
Class lists L student
ab = {a, b}
bc = {b, c}
bd = {b, d}
cd = {c, d}
Courses = { a, b, c, d}
K = 3 time slots

a = 9am = red
b = 10am = blue
c = 11am = yellow
d = 9am = red

a. Any K-COLOR solution gives a solution to COURSE-TIME.
Suppose that *G* can be colored with *K* colors. Get a *K*-coloring of *G*. Assign times slots to the courses by following the coloring. If vertex *v* is colored by color *m*, then use time slot *m* for course *v*. Since the coloring does not color any two adjacent vertices the same color, there can be no time conflicts.

b. Any COURSE-TIME solution gives a solution to K-COLOR.

Suppose that the courses can be assigned time slots so that there are no conflicts. Then assign colors to *G* in the same way, using the *m*-th color for vertex *v* if course *v* received the *m*-th time slot. Since there is a student for each edge, and none of the students have time slot conflicts, this coloring must avoid coloring two adjacent vertices the same color.

Remark. *Be sure that your reduction goes the right direction. You need to show how to solve the graph coloring problem, assuming that you already have a solution to the course assignment problem, not the other way around.*

Bin Packing–Dec. is NP-complete

Bin Packing problem: Given n items of sizes a_1, $a_2,\ldots, a_n$ ($0 < a_i \leq 1$), pack these items in at most k bins of size 1.

1. Bin packing in in NP
   - To verify a solution
     - Add the weights of the items in each bin.
     - Each bin must contain < 1unit.
     - Check that each item is in a bin
     - There are at most k bins used.
   - This can be done in O(n).

2. SET-PARTITION reduces to Bin Packing

Bin Packing–Dec. is NP-complete

SET-PARTITION: Given a set of numbers X = { $x_1, x_2, \ldots x_k$ }. Is there a subset of X, B, such that the sum of the elements in B is equal to the sum of the elements in S-B.

Bin Packing : Given n items of sizes $a_1, a_2, \ldots, a_n$ ($0 < a_i \leq 1$), pack these items in at most k bins of size 1.

2. SET-PARTITION $\leq_p$ Bin Packing

Let sum = $\sum_{i=1}^{k} x_i$. Define S = { $s_1, s_2, \ldots s_k$ }

where $s_i = \frac{2x_i}{sum}$ for i = 1, ..k. Then if { $s_1, s_2, \ldots s_k$ } can be packed into 2 bins, X can be partitioned into 2 sets.

Thus Bin Packing is NP-Complete