

NP Completeness

CLO Survey

Please complete the CLO Survey for ABET using the following link:

https://waynestate.az1.qualtrics.com/jfe/form/SV_0ikaFGGCNXUcMFT

Section: 001

CRN: 27503

Semester: Winter 2020

Where did you take this course? Select “Main Campus”


Run-time analysis

We've spent a lot of time in this class putting algorithms into specific run-time categories:

- $O(\log n)$
- $O(n)$
- $O(n \log n)$
- $O(n^2)$
- $O(n \log \log n)$
- $O(n^{1.67})$
- ...

When I say an algorithm is $O(f(n))$, what does that mean?

Tractable vs. intractable problems

trac·ta·ble  (trăk'tə-bəl)

adj.

1. Easily managed or controlled; governable.
 2. Easily handled or worked; malleable.
-

What is a “tractable” problem?

Tractable vs. intractable problems


trac·ta·ble  (trăk'tə-bəl)

adj.

1. Easily managed or controlled; governable.
 2. Easily handled or worked; malleable.
-

Tractable problems can be solved in $O(f(n))$ where $f(n)$ is a polynomial

Tractable vs. intractable problems

trac·ta·ble  (trăk'tə-bəl)
adj.

1. Easily managed or controlled; governable.
 2. Easily handled or worked; malleable.
-

What about...

$O(n^{100})$?

$O(n^{\log \log \log \log n})$?

Tractable vs. intractable problems

trac·ta·ble  (trăk'tə-bəl)


adj.

1. Easily managed or controlled; governable.
 2. Easily handled or worked; malleable.
-

Technically $O(n^{100})$ is tractable by our definition

Why don't we worry about problems like this?

Tractable vs. intractable problems


trac·ta·ble  (trāk'tə-bəl)
adj.

1. Easily managed or controlled; governable.
 2. Easily handled or worked; malleable.
-

Technically $O(n^{100})$ is tractable by our definition

- Few practical problems result in solutions like this
- Once a polynomial time algorithm exists, more efficient algorithms are usually found
- Polynomial algorithms are amenable to parallel computation


Solvable vs. unsolvable problems

solv·a·ble  (sɒl'və-bəl, sɒl'-)
adj.

Possible to solve: *solvable problems; a solvable riddle.*

What is a “solvable” problem?

Solvable vs. unsolvable problems

solv·a·ble  (sɒl'və-bəl, sɒl'-)
adj.

Possible to solve: *solvable problems; a solvable riddle.*

A problem is solvable if given enough
(i.e. finite) time you could solve it

Sorting

Given n integers, sort them from smallest to largest.

Tractable/intractable?

Solvable/unsolvable?

Sorting

Given n integers, sort them from smallest to largest.

Solvable and tractable:

Mergesort: $\Theta(n \log n)$

Enumerating all subsets

Given a set of n items, enumerate all possible subsets.

Tractable/intractable?

Solvable/unsolvable?

Fractional-Knapsack Problem

Tractable/intractable?

Solvable/unsolvable?

Fractional-Knapsack Problem

Tractable

Solvable

Polynomial time optimal
Solution exists.

0/1-Knapsack Problem

Tractable/intractable?

Solvable/unsolvable?

0/1-Knapsack Problem

Tractable/intractable?

Solvable: Recall dynamic prog solution
 $O(nW)$

0/1-Knapsack Problem

Intractable: $O(nW) = O(n 2^{\lg W})$ which is exponential in input size \rightarrow Pseudopolynomial

Solvable: Recall dynamic prog solution
 $O(nW)$

Enumerating all subsets

Given a set of n items, enumerate all possible subsets.

Solvable, but intractable: $\Theta(2^n)$ subsets

For large n this will take a very, very long time

Halting problem

Given an arbitrary algorithm/program and a particular input, will the program terminate?

Tractable/intractable?

Solvable/unsolvable?

Halting problem

Given an arbitrary algorithm/program and a particular input, will the program terminate?

Unsolvable ☹️

Integer solution?

Given a polynomial equation, are there *integer* values of the variables such that the equation is true?

$$x^3yz + 2y^4z^2 - 7xy^5z = 6$$

Tractable/intractable?

Solvable/unsolvable?

Integer solution?

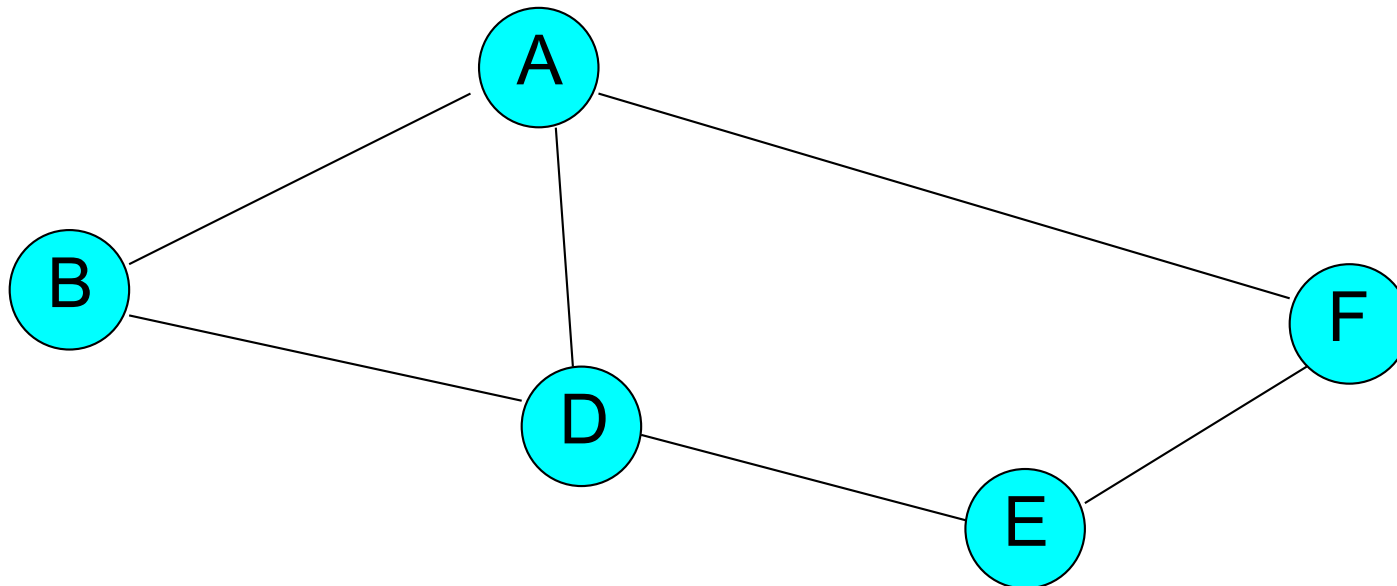
Given a polynomial equation, are there *integer* values of the variables such that the equation is true?

$$x^3yz + 2y^4z^2 - 7xy^5z = 6$$

Unsolvable ☹

Hamiltonian cycle

Given an undirected graph $G=(V, E)$, a hamiltonian cycle is a cycle that visits every vertex V exactly once



Hamiltonian cycle

Given an undirected graph, does it contain a hamiltonian cycle?

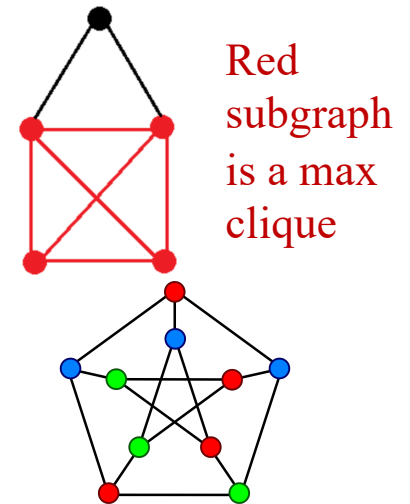
Solvable: Enumerate all possible paths (i.e. include an edge or don't) check if it's a hamiltonian cycle

Problem Types

- **Optimization problem:** find a solution that maximizes or minimizes some objective function
- **Decision problem:** answer yes/no to a question
- Many problems have decision and optimization versions.

Decision version example:

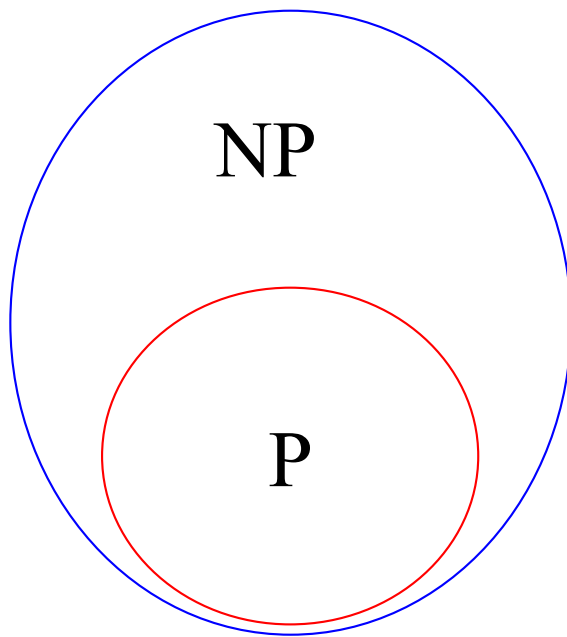
- **Traveling salesman problem:** find Hamiltonian cycle of minimum length.
Decision: does a Hamiltonian cycle of length $\leq m$ exist?
- **Partition problem:** Is it possible to partition a set of n integers into two disjoint subsets with the same sum?
- **Clique** (a complete subgraph): **finding** the largest clique in a graph.
Decision: in graph G , does a clique of size k exist?
- **Graph coloring:** Minimize the number of colors to color the vertices (two neighbors cannot have the same color)
Decision: Can you color with k colors?



Complexity Classes: P and NP

P: class of decision problems, each solvable in polynomial time.

NP (nondeterministic polynomial): Verifiable in polynomial time.



Big-O allowed us to group algorithms by run-time

Today, we're talking about sets of problems grouped by how easy they are to solve

NP problems

Why might we care about NP problems?

- If we can't verify the solution in polynomial time then an algorithm cannot exist that determines the solution in this time (why not?)
- All algorithms with polynomial time solutions are in NP

The NP problems that are currently not solvable in polynomial time *could in theory be solved in polynomial time*

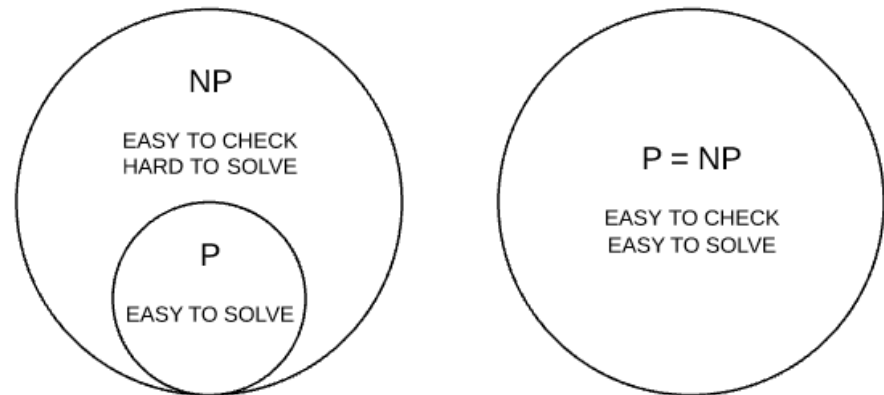
More Details about P and NP

P: class of decision problems, each solvable in polynomial time.

NP (nondeterministic polynomial):

- class of decision problems whose proposed solutions can be **verified** in polynomial time.
- solvable in polynomial time on a *nondeterministic Turing Machine* (can have states with several alternative successor states, for the same input).
- Example: Decision versions of TSP, knapsack, graph coloring, partition.

➤ $P \subseteq NP$ (widely believed
 $P \neq NP$ but no proof exists)



$P = NP$ or $P \neq NP$? Millennium Prize Problem (\$1000,000)

Example: CNF Satisfiability (SAT)

CNF-SAT: Is a boolean expression in its conjunctive normal form (CNF) satisfiable, i.e., are there values of its variables that makes it true?

This problem is in *NP*. **Nondeterministic algorithm:**

- Guess truth assignment
- Substitute the values into the CNF formula to see if it evaluates to true

Example: $(A \mid \neg B \mid \neg C) \& (A \mid B) \& (\neg B \mid \neg D \mid E) \& (\neg D \mid \neg E)$

Truth assignments:

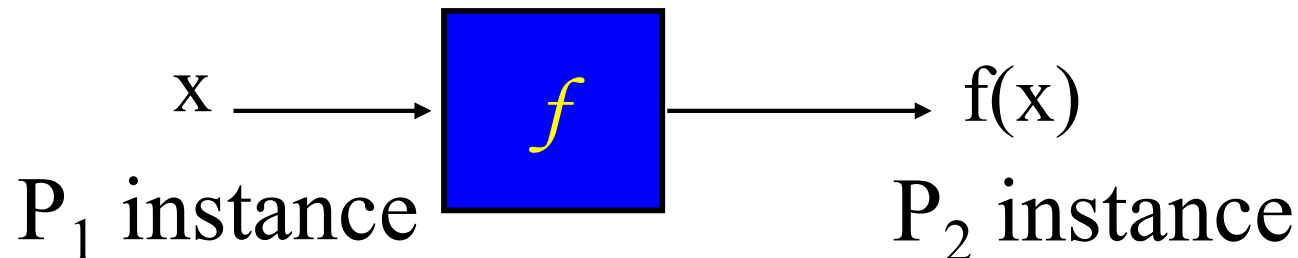
<u>A</u>	<u>B</u>	<u>C</u>	<u>D</u>	<u>E</u>
0	0	0	0	0
		.	.	.
1	1	1	1	1

Verification time: $O(n)$

Reduction function

Given two problems P_1 and P_2 a *reduction function*, $f(x)$, is a function that transforms a problem instance x of type P_1 to a problem instance of type P_2

such that: a solution to x exists for P_1 iff a solution for $f(x)$ exists for P_2

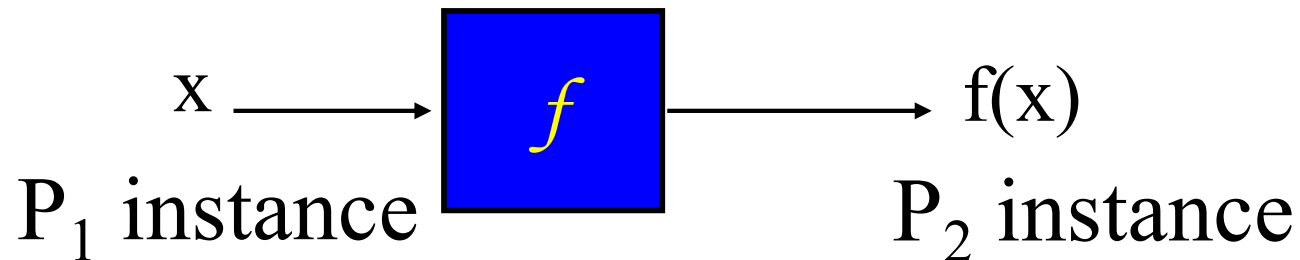


Reduction function

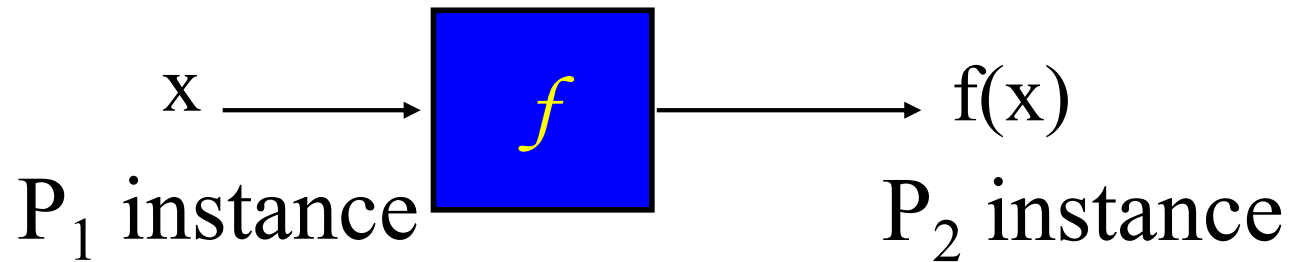
Where have we seen reductions before?

- All pairs shortest path *through a particular vertex* reduced to single source shortest path

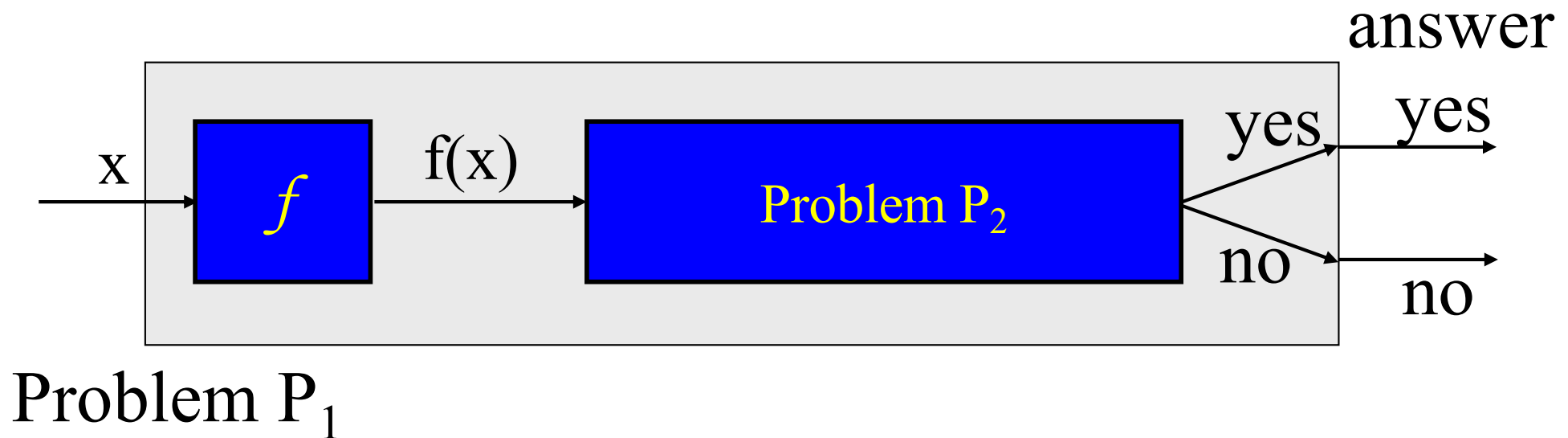
Why are they useful?



Reduction function



Allow us to solve P_1 problems if we have a solver for P_2



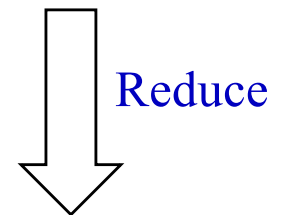
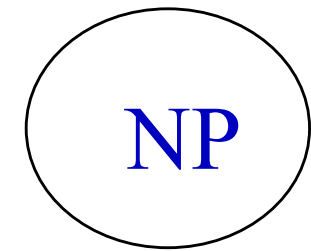
NP-Hard and NP-Complete

- **NP-Hard**

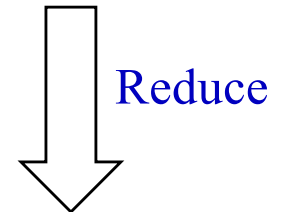
- A problem (which may or may not be a decision problem) is called **NP-hard** if it is at least as hard as every problem in **NP**.
- A problem is NP-hard when every problem in NP can be reduced in **polynomial time** to it.
- To prove problem A is **NP-hard**, reduce (in polynomial time) a known **NP-hard** problem to A.

- **NP-Complete**

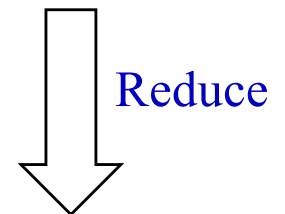
- A Problem that is both in NP and is NP-hard
- Example: Decision CNF-SAT, TSP, Clique, Subsetsum, graph coloring



An NP-Hard problem



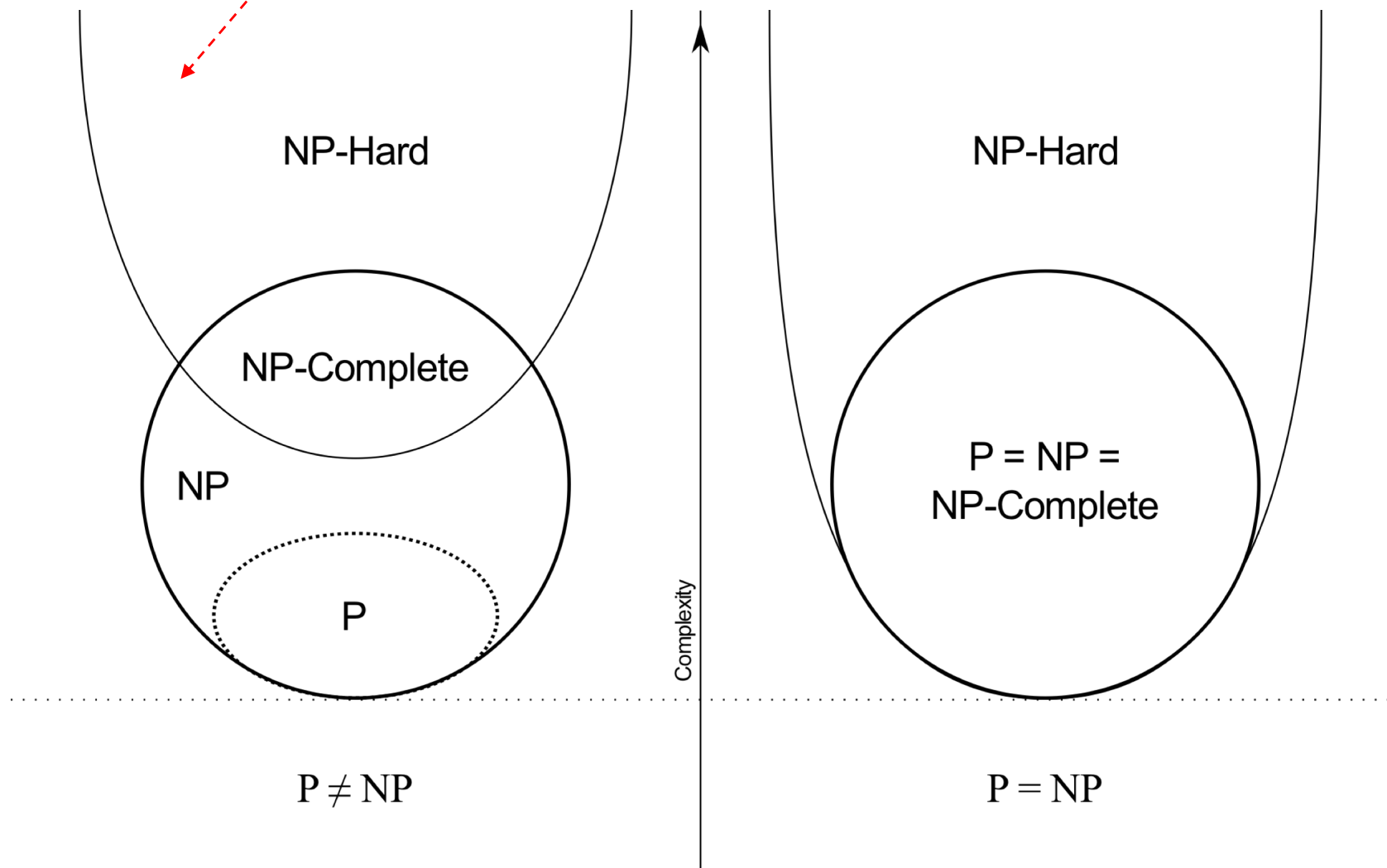
A (NP-Hard)



Another problem (NP-Hard)

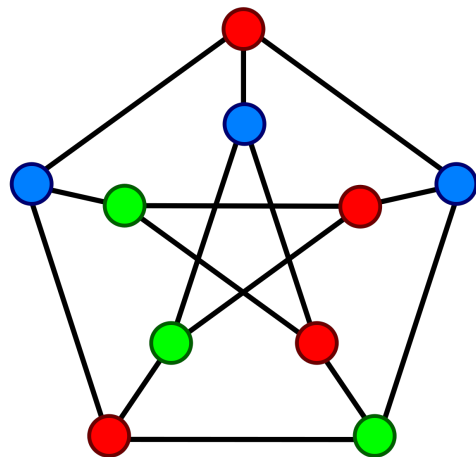
P, NP, NP-Hard, NP-Complete

Halting problem is NP-hard but not NP Complete. **Halting problem** is the problem of determining, from a description of an arbitrary computer program and an input, whether the program will finish running or continue to run forever.

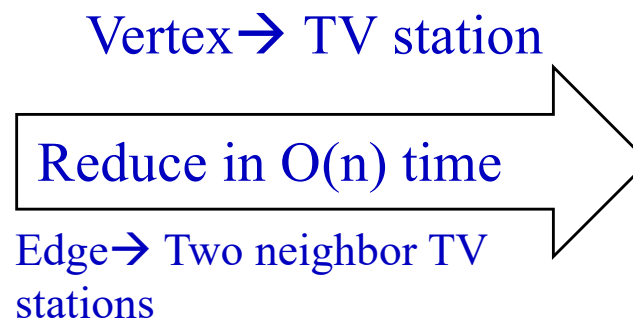


How to Prove NP -Completeness?

- From scratch (difficult – Cook's Theorem)
- **Generalization (examples?)**
- Reduction
 - Show this problem is in NP
 - Prove NP -Hard: Show a known NP -complete problem can be polynomially transformed to this.



Graph
coloring

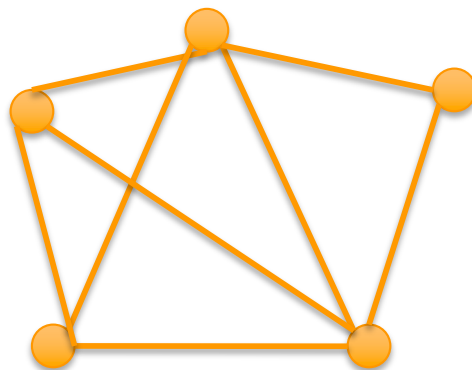


Channel assignment:
Minimize total # of
channels to be assigned to
all TV stations (two
neighboring stations cannot
have the same channel)

CLIQUE

A *clique* in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices that are fully connected, i.e. every vertex in V' is connected to every other vertex in V'

CLIQUE problem: Does G contain a clique of size k ?



Is there a clique of size 4 in this graph?

Proving NP-completeness

Given a problem NEW to show it is NP-Complete

1. Show that NEW is in NP
 - a. Provide a verifier
 - b. Show that the verifier runs in polynomial time
2. Show that all NP-complete problems are reducible to NEW in polynomial time
 - a. Describe a reduction function f from a known NP-Complete problem to NEW
 - b. Show that f runs in polynomial time
 - c. Show that a solution exists to the NP-Complete problem IFF a solution exists *to the NEW problem generate by f*

Proving NP-completeness

Show that a solution exists to the NP-Complete problem IFF a solution exists *to the NEW problem generate by f*

- Assume we have an NP-Complete problem instance that has a solution, show that the NEW problem instance generated by f has a solution
- Assume we have a problem instance of NEW *generated by f* that has a solution, show that we can derive a solution to the NP-Complete problem instance

Other ways of proving the IFF, but this is often the easiest

HALF-CLIQUE

Given a graph G , does the graph contain a clique containing exactly half the vertices?

Is HALF-CLIQUE an NP-complete problem?

Is Half-Clique NP-Complete?

1. Show that NEW is in NP
 - a. Provide a verifier
 - b. Show that the verifier runs in polynomial time
 2. Show that all NP-complete problems are reducible to NEW in polynomial time
 - a. Describe a reduction function f from a known NP-Complete problem to NEW
 - b. Show that f runs in polynomial time
 - c. Show that a solution exists to the NP-Complete problem IFF a solution exists *to the NEW problem generate by f*
-

Given a graph G , does the graph contain a clique containing exactly half the vertices?

HALF-CLIQUE

1. Show that HALF-CLIQUE is in NP
 - a. Provide a verifier
 - b. Show that the verifier runs in polynomial time
-

Verifier: A solution consists of the set of vertices in V'

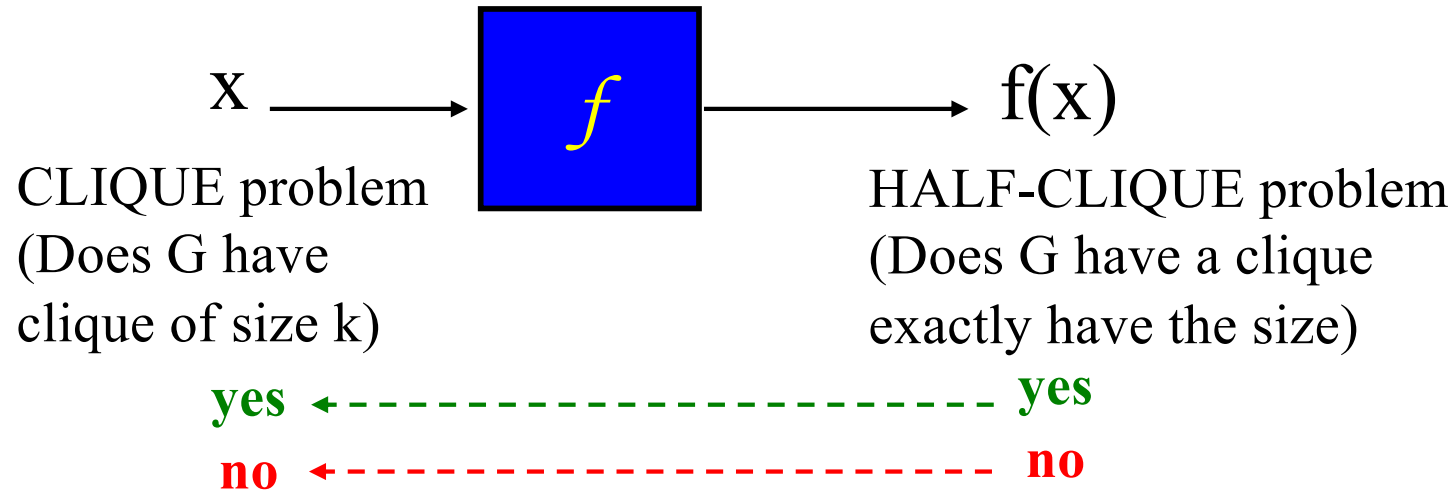
- check that $|V'| = |V|/2$
- for all pairs of $u, v \in V'$
 - there exists an edge $(u,v) \in E$
- Check for edge existence in $O(V)$
- $O(V^2)$ checks
- $O(V^3)$ overall, which is polynomial

HALF-CLIQUE

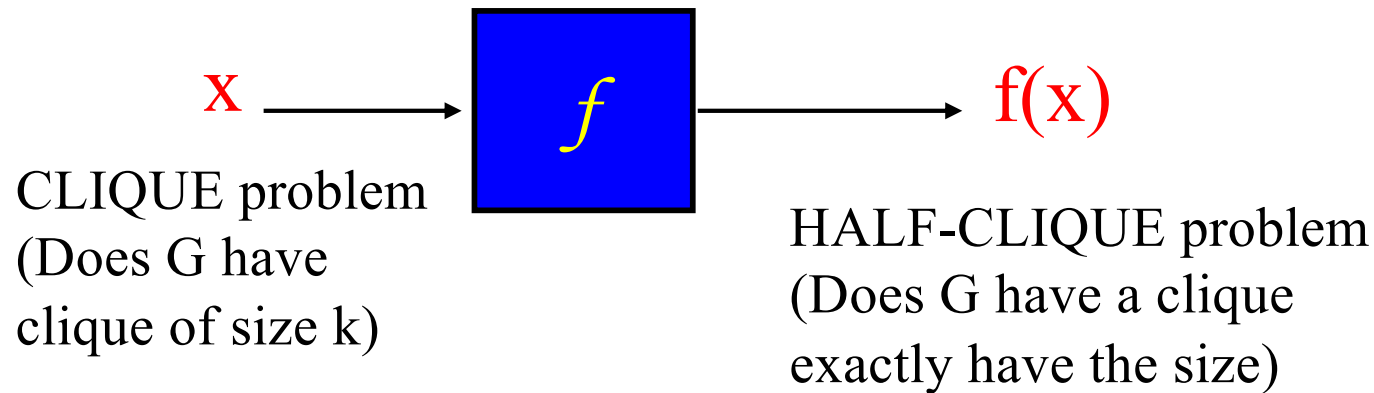
- 1.
2. Show that all NP-complete problems are reducible to HALF-CLIQUE in polynomial time
 - a. Describe a reduction function f from a known NP-Complete problem to HALF-CLIQUE
 - b. Show that f runs in polynomial time
 - c. Show that a solution exists to the NP-Complete problem IFF a solution exists *to the HALF-CLIQUE problem generate by f*

Reduce CLIQUE to HALF-CLIQUE:

Given a problem instance of CLIQUE, turn it into a problem instance of HALF-CLIQUE



HALF-CLIQUE



Three cases:

1. $k = |V|/2$
2. $k < |V|/2$
3. $k > |V|/2$

In reduction, we shall convert each scenario to half-clique of a different graph.

HALF-CLIQUE

Reduce CLIQUE to HALF-CLIQUE:

Given an instance of CLIQUE, turn it into an instance of HALF-CLIQUE

$f(G, k)$

```
1  if  $\lceil |V| \rceil / 2 = k$   
2      return  $G$ 
```

It's already a half-clique problem

```
3  elseif  $k < \lceil |V| \rceil / 2$   
4      return  $G$  plus  $(|V| - 2k)$  nodes which are fully connected  
      and are connected to every node in  $V$   
5  else  
6      return  $G$  plus  $2k - |V|$  nodes which have no edges
```

HALF-CLIQUE

Reduce CLIQUE to HALF-CLIQUE:

Given an instance of CLIQUE, turn it into an instance of HALF-CLIQUE

We're looking for a clique that is smaller than half, so add an artificial clique to the graph and connect it up to all vertices

$f(G, k)$

1 **if** $\lceil |V| \rceil / 2 = k$

2 **return** G

3 **elseif** $k < \lceil |V| \rceil / 2$

4 **return** G plus $(|V| - 2k)$ nodes which are fully connected
and are connected to every node in V

5 **else**

6 **return** G plus $2k - |V|$ nodes which have no edges

HALF-CLIQUE

Reduce CLIQUE to HALF-CLIQUE:

Given an instance of CLIQUE, turn it into an instance of HALF-CLIQUE

We're looking for a clique that is bigger than half, so add vertices until $k = |V|/2$

```
f(G, k)  
1  if  $\lceil |V| \rceil / 2 = k$   
2      return G  
3  elseif  $k < \lceil |V| \rceil / 2$   
4      return G plus  $(|V| - 2k)$  nodes which are fully connected  
      and are connected to every node in V  
5  else  
6      return G plus  $2k - |V|$  nodes which have no edges
```

HALF-CLIQUE

Reduce CLIQUE to HALF-CLIQUE:

Given an instance of CLIQUE, turn it into an instance of HALF-CLIQUE

```
 $f(G, k)$   
1  if  $\lceil |V| \rceil / 2 = k$   
2      return  $G$   
3  elseif  $k < \lceil |V| \rceil / 2$   
4      return  $G$  plus  $(|V| - 2k)$  nodes which are fully connected  
      and are connected to every node in  $V$   
5  else  
6      return  $G$  plus  $2k - |V|$  nodes which have no edges
```

Runtime: From the construction we can see that it is polynomial time

Reduction proof

Given a graph G that has a CLIQUE of size k , show that $f(G,k)$ has a solution to HALF-CLIQUE

If $k = |V|/2$:

- the graph is unmodified
- $f(G,k)$ has a clique that is half the size

Reduction proof

Given a graph G that has a CLIQUE of size k , show that $f(G,k)$ has a solution to HALF-CLIQUE

If $k < |V|/2$:

- we added a clique of $|V| - 2k$ fully connected nodes
- there are $|V| + |V| - 2k = 2(|V| - k)$ nodes in $f(G)$
- there is a clique in the original graph of size k
- plus our added clique of $|V| - 2k$
- $k + |V| - 2k = |V| - k$, which is half the size of $f(G)$

Reduction proof

Given a graph G that has a CLIQUE of size k , show that $f(G,k)$ has a solution to HALF-CLIQUE

If $k > |V|/2$:

- we added $2k - |V|$ unconnected vertices
- $f(G)$ contains $|V| + 2k - |V| = 2k$ vertices
- Since the original graph had a clique of size k vertices, the new graph will have a half-clique

Reduction proof

Given a graph $f(G)$ that has a CLIQUE half the elements, show that G has a clique of size k

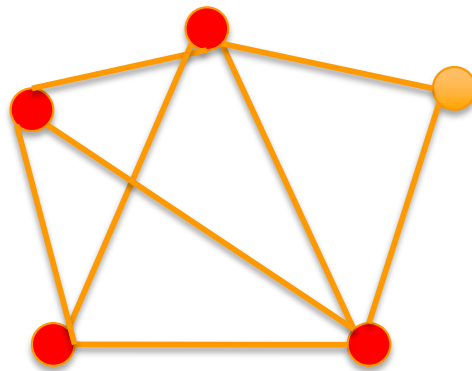
Key: $f(G)$ was constructed by your reduction function

Use a similar argument to what we used in the other direction

Is CLIQUE NP-Complete?

A *clique* in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices that are fully connected, i.e. every vertex in V' is connected to every other vertex in V'

CLIQUE problem: Does G contain a clique of size k ?

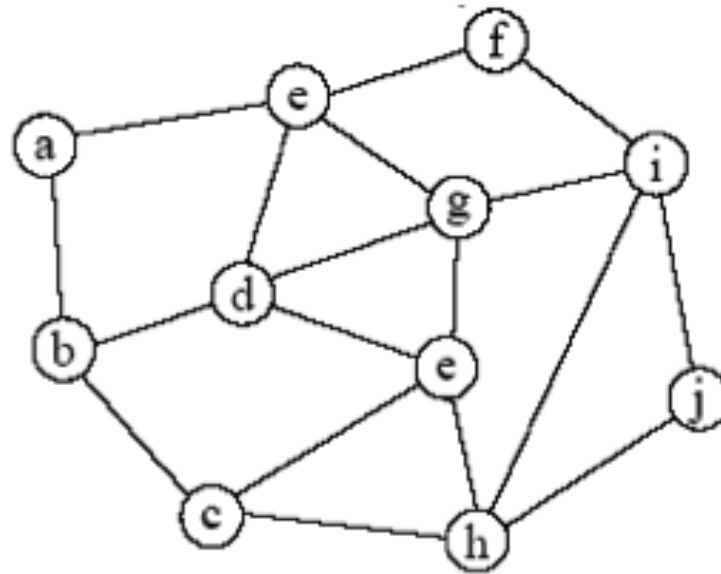


Is CLIQUE
NP-Complete?

Verification in polynomial time: straightforward.

Independent-Set

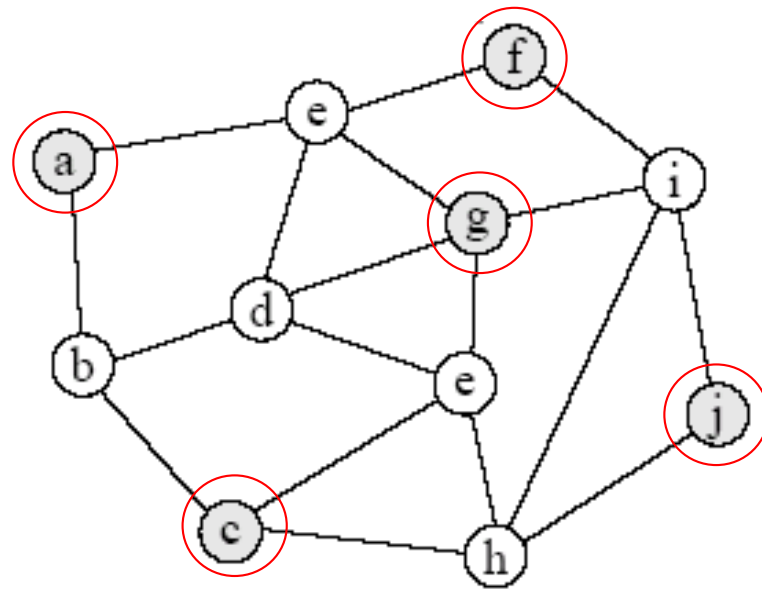
Given a graph $G = (V, E)$ is there a subset $V' \subseteq V$ of vertices of size $|V'| = k$ that are independent, i.e. for any pair of vertices $u, v \in V'$ there exists no edge between any of these vertices



Does the graph contain an independent set of size 5?

Independent-Set

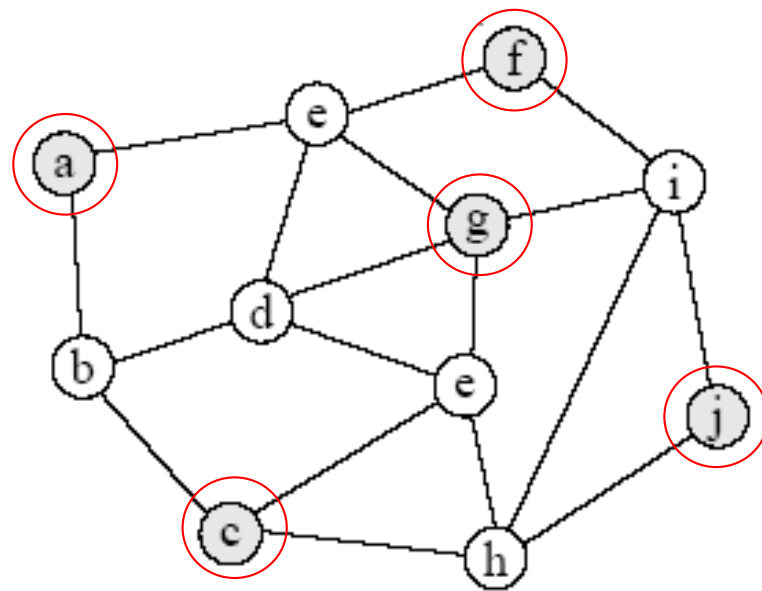
Given a graph $G = (V, E)$ is there a subset $V' \subseteq V$ of vertices of size $|V'| = k$ that are independent, i.e. for any pair of vertices $u, v \in V'$ there exists no edge between any of these vertices



Independent-Set is NP-Complete

Independent-Set

Given a graph $G = (V, E)$ is there a subset $V' \subseteq V$ of vertices of size $|V'| = k$ that are independent, i.e. for any pair of vertices $u, v \in V'$ there exists no edge between any of these vertices. Is there an independent set of size k ?



Reduce Independent-Set to CLIQUE

Independent-Set to Clique

Given a graph $G = (V, E)$ is there a subset $V' \subseteq V$ of vertices of size $|V'| = k$ that are independent, i.e. for any pair of vertices $u, v \in V'$ there exists no edge between any of these vertices

Both are selecting vertices

Independent set wants vertices where NONE are connected

Clique wants vertices where ALL are connected

How can we convert a NONE problem to an ALL problem?

Independent-Set to Clique

Given a graph $G = (V, E)$, the **complement of that graph** $G' = (V, E)$ is the graph constructed by removing all edges E and including all edges not in E .

For example, for adjacency matrix this is flipping all of the bits

```
f(G)
    return G'
```

Proof

Given a graph G that has an independent set of size k , show that $f(G)$ has a clique of size k

- By definition, the independent set has no edges between any vertices.
- These will all be edges in $f(G)$ and therefore they will form a clique of size k .

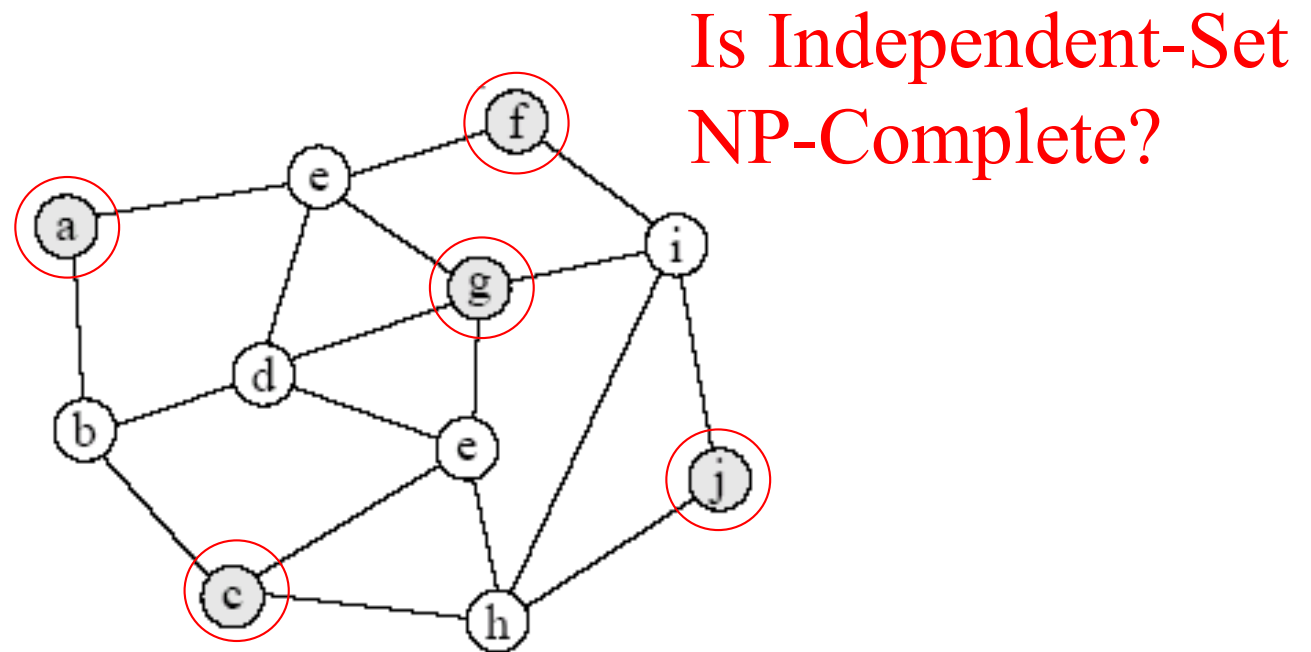
Proof

Given $f(G)$ that has clique of size k , show that G has an independent set of size k

- By definition, the clique will have an edge between every vertex
- None of these vertices will therefore be connected in G , so we have an independent set

Is Independent-Set NP-Complete?

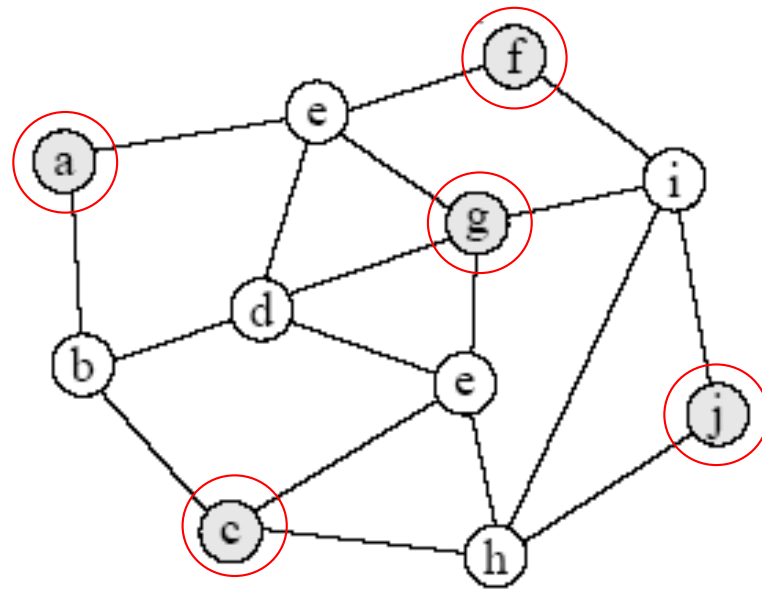
Given a graph $G = (V, E)$ is there a subset $V' \subseteq V$ of vertices of size $|V'| = k$ that are independent, i.e. for any pair of vertices $u, v \in V'$ there exists no edge between any of these vertices



Verification in polynomial time: straightforward.

Reduction

Given a graph $G = (V, E)$ is there a subset $V' \subseteq V$ of vertices of size $|V'| = k$ that are independent, i.e. for any pair of vertices $u, v \in V'$ there exists no edge between any of these vertices



Reduce 3-SAT to Independent-Set

SAT and 3 SAT-revisited

Problem: SAT

Instance: A CNF formula φ .

Question: Is there a truth assignment to the variable of φ such that φ evaluates to true?

Problem: 3SAT

Instance: A 3CNF formula φ .

Question: Is there a truth assignment to the variable of φ such that φ evaluates to true?

A formula φ is a **3CNF** :

A CNF formula such that every clause has *exactly* 3 literals.

(A) $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_1)$ is a **3CNF** formula, but $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is not.

3-SAT to Independent-Set

Given a 3-CNF formula, convert into a graph.

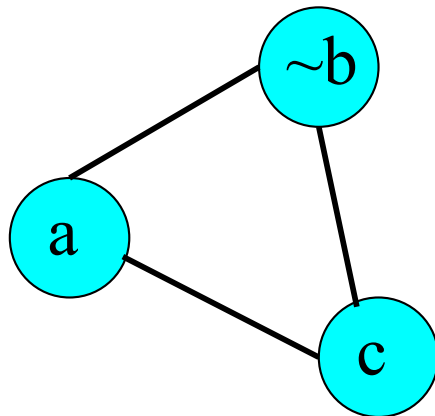
For the Boolean formula in 3-SAT to be satisfied, at least one of the literals in each clause must be true.

In addition, we must make sure that we enforce a literal and its complement must not both be true.

3-SAT to Independent-Set

Given a 3-CNF formula, convert into a graph.

For each clause, e.g. *(a OR not(b) OR C)* create a clique containing vertices representing these literals.

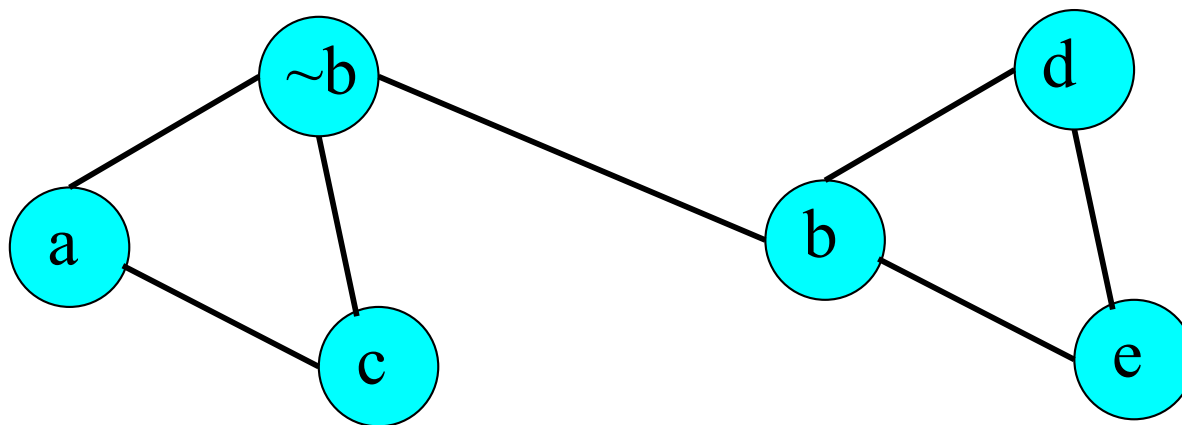


- for the Independent-Set problem to be satisfied it can only select one variable
- to make sure that all clauses are satisfied, we set **k = number of clauses**

3-SAT to Independent-Set

Given a 3-CNF formula, convert into a graph

To enforce that only one variable and its complement can be set we connect each vertex representing x to each vertex representing its complement $\sim x$



Proof

Given a 3-SAT problem with k clauses and a valid truth assignment, show that $f(3\text{-SAT})$ has an independent set of size k . (Assume you know the solution to the 3-SAT problem and show how to get the solution to the independent set problem)

Proof

Given a 3-SAT problem with k clauses and a valid truth assignment, show that $f(3\text{-SAT})$ has an independent set of size k . (Assume you know the solution to the 3-SAT problem and show how to get the solution to the independent set problem)

Since each clause is an OR of variables, at least one of the three must be true for the entire formula to be true. Therefore each 3-clique in the graph will have at least one node that can be selected.

Proof

Given a graph with an independent set S of k vertices, show there exists a truth assignment satisfying the boolean formula

- For any variable x_i , S cannot contain both x_i and $\neg x_i$ since they are connected by an edge
- For each vertex in S , we assign it a true value and all others false. Since S has only k vertices, it must have one vertex per clause.

More NP-Complete problems

SUBSET-SUM:

- Given a set S of positive integers, is there some subset $S' \subseteq S$ whose elements sum to t .

TRAVELING-SALESMAN:

- Given a weighted graph G , does the graph contain a hamiltonian cycle of length k or less?

VERTEX-COVER:

- Given a graph $G = (V, E)$, is there a subset $V' \subseteq V$ such that if $(u,v) \in E$ then $u \in V'$ or $v \in V'$?

Our known NP-Complete problems

We can reduce any of these problems to a new problem in an NP-completeness proof

- SAT, 3-SAT
- CLIQUE, HALF-CLIQUE
- INDEPENDENT-SET
- HAMILTONIAN-CYCLE
- TRAVELING-SALESMAN
- VERTEX-COVER
- SUBSET-SUM

Tackling Difficult Combinatorial Problems

There are two principal approaches to tackling difficult combinatorial problems (NP-hard problems):

- Use a strategy that guarantees to solve the problem exactly but doesn't guarantee to find a solution in polynomial time
- Use an approximation algorithm that can find an approximate (sub-optimal) solution in polynomial time

Exact Solution Strategies

- *exhaustive search* (brute force)
 - useful only for small instances
- *dynamic programming*
 - applicable to some problems (e.g., the knapsack problem)
- *backtracking*
 - eliminates some unnecessary cases from consideration
 - yields solutions in reasonable time for many instances but worst case is still exponential
- *branch-and-bound*
 - further refines the backtracking idea for optimization problems

Backtracking

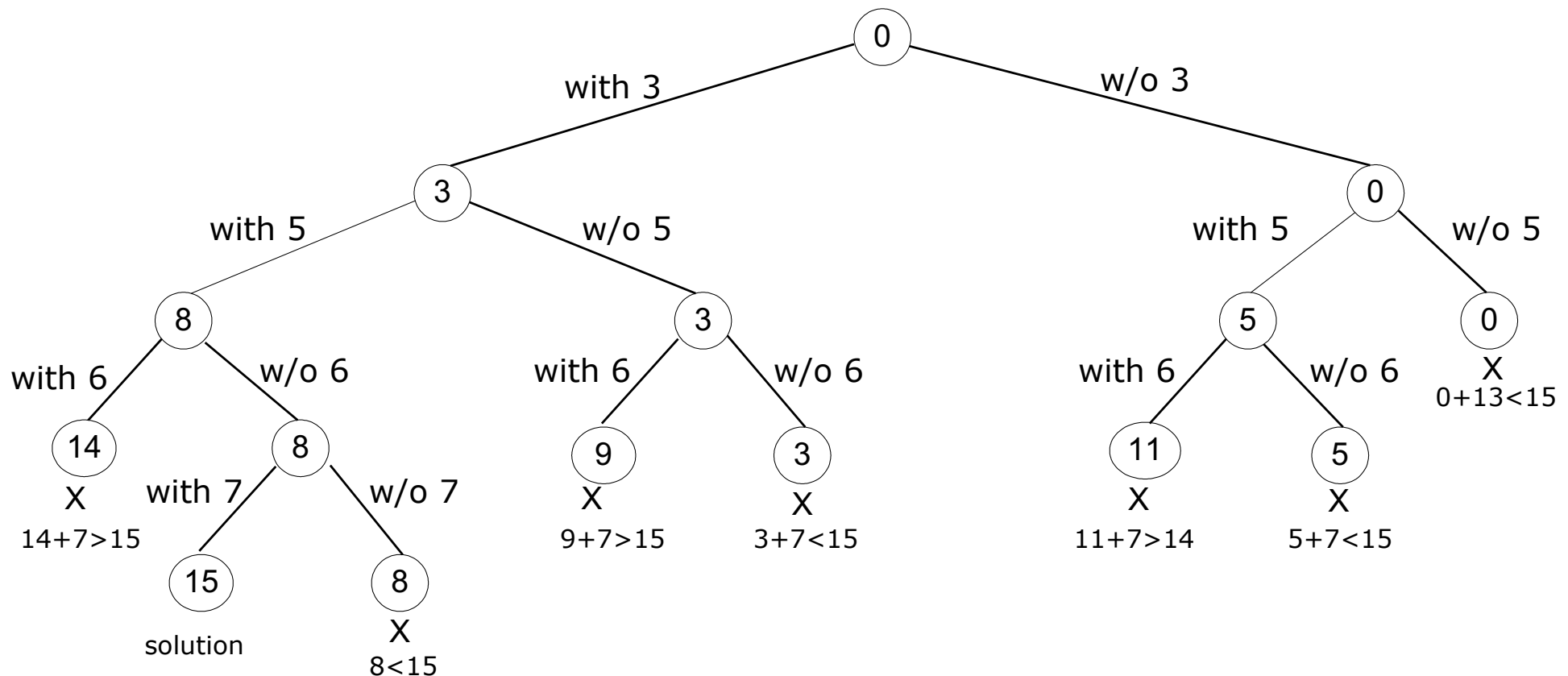
- Construct the state-space tree
 - nodes: partial solutions
 - edges: choices in extending partial solutions
- Explore the state space tree using depth-first search (DFS)
- “Prune” nonpromising nodes
 - dfs stops exploring subtrees rooted at nodes that cannot lead to a solution and backtracks to such a node’s parent to continue the search

Subset-Sum Problem

- Given a set $S = \{s_1, s_2, \dots, s_n\}$ of positive integers and an integer d .
- Find a subset of S which sums to d .
- Example:
 - $S = \{1, 2, 5, 6, 8\}$ and $d = 9$.
 - Two solutions:
 - $\{1, 2, 6\}$
 - $\{1, 8\}$

Example: Subset-Sum Problem

- $S = \{3, 5, 6, 7\}$ and $d = 15$



Branch-and-Bound

- An enhancement of backtracking
- Applicable to optimization problems
- For each node (partial solution) of a state-space tree, computes a bound on the value of the objective function for all descendants of the node (extensions of the partial solution)
- Uses the bound for:
 - ruling out certain nodes as “nonpromising” to prune the tree
 - if a node’s bound is not better than the best solution seen so far
 - guiding the search through state-space

Approximation Approach

- Apply a fast (i.e., a polynomial-time) approximation algorithm to get a solution that is not necessarily, but hopefully close to, optimal

- **Approximation ratio** r :

$r = \text{APPR} / \text{OPT}$ for minimization problems

$r = \text{OPT} / \text{APPR}$ for maximization problems

where APPR and OPT are values of the objective function for the approximate solution and actual optimal solution, resp.

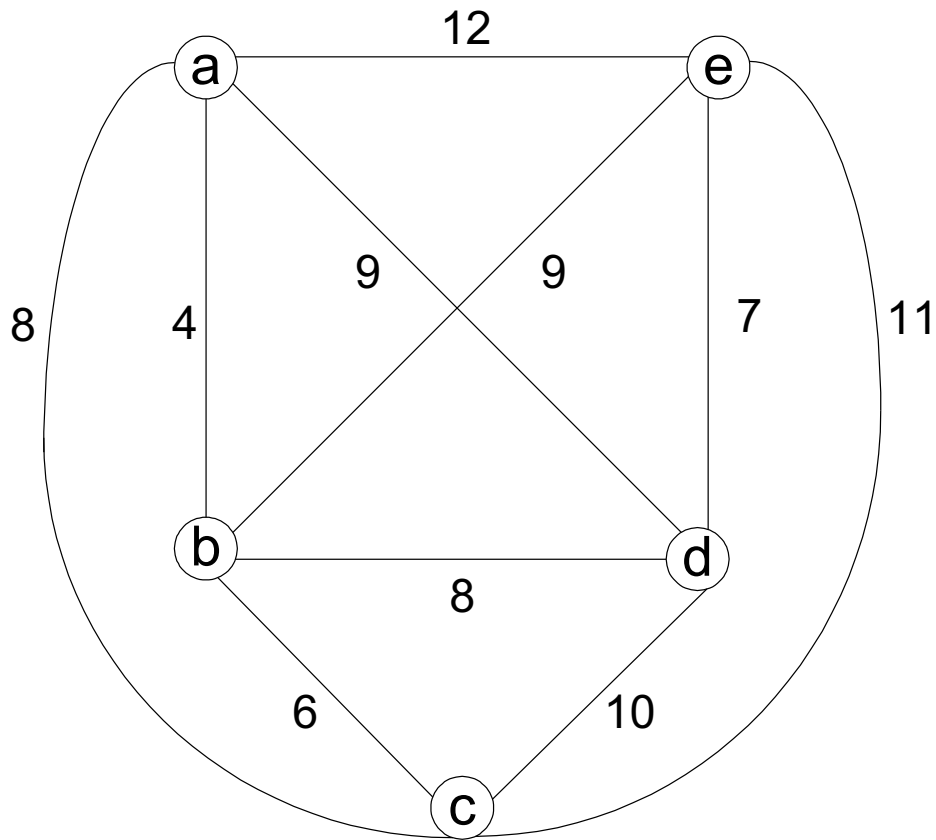
MST-based Algorithm for TSP

Stage 1: Construct a minimum spanning tree (MST) of the graph (e.g., by Prim's or Kruskal's algorithm).

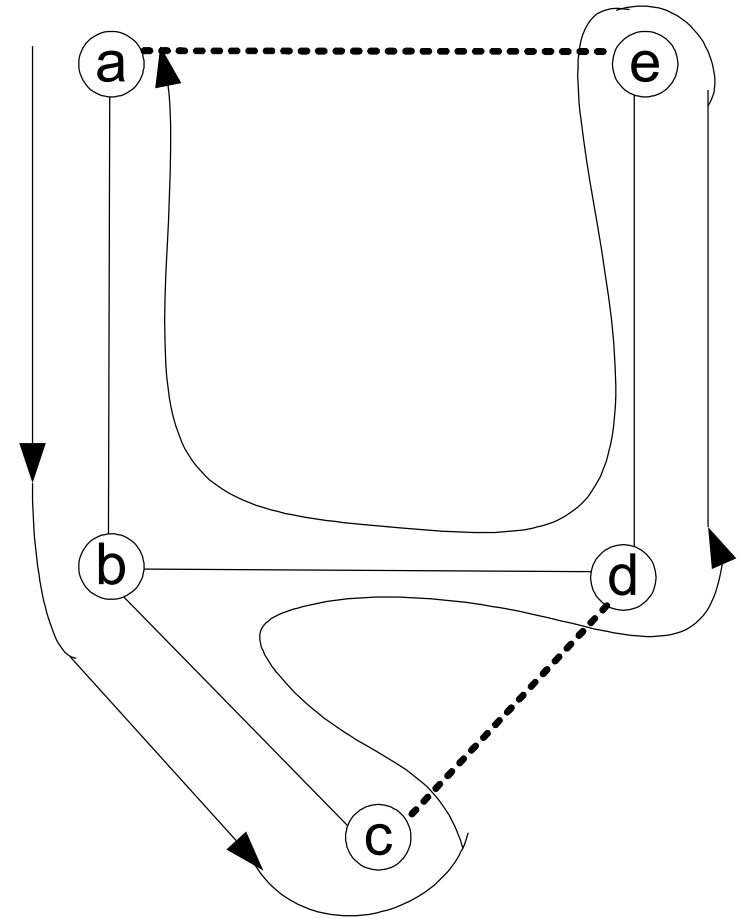
Stage 2: Starting at an arbitrary vertex, create a path that goes twice around the tree and returns to the same vertex

Stage 3: Create a tour from the circuit constructed in Stage 2 by making shortcuts to avoid visiting intermediate vertices more than once

Example



Walk: $a - b - c - b - d - e - d - b - a$



Tour: $a - b - c - d - e - a$

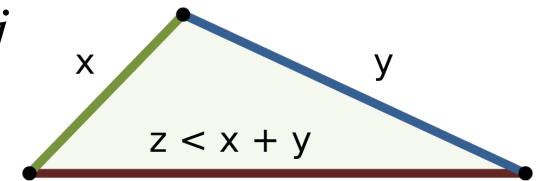
Approximation Ratio for Euclidean Instance

➤ If $P \neq NP$, there exists no constant-factor approximation algorithm for TSP.

➤ TSP is *Euclidean*, if its distances satisfy two conditions:

➤ *symmetry* $d[i, j] = d[j, i]$ for any cities i and j

➤ *triangle inequality* $d[i, j] \leq d[i, k] + d[k, j]$



➤ **MST based alg for Euclidean TSP has approximation ratio 2**

➤ $MST \leq OPT$ (MST is the length of MST)

➤ Length of path that goes twice around the tree $\leq 2MST \leq 2OPT$

➤ Due to triangle inequality, shortcutting reduces the above cost.

➤ Hence, approx. ratio ≤ 2 . $OPT/OPT = 2$.