NP-Complete problems

Dr. Navjot Singh

Design and Analysis of Algorithms

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

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

Tractable vs. intractable problems

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

What about…

$O(n^{100})$?

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

Tractable vs. intractable problems

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

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

Why don't we worry about problems like this?

Tractable vs. intractable problems

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

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

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: Θ(*n* log *n*)

Enumerating all subsets

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

Tractable/intractable?

Solvable/unsolvable?

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?

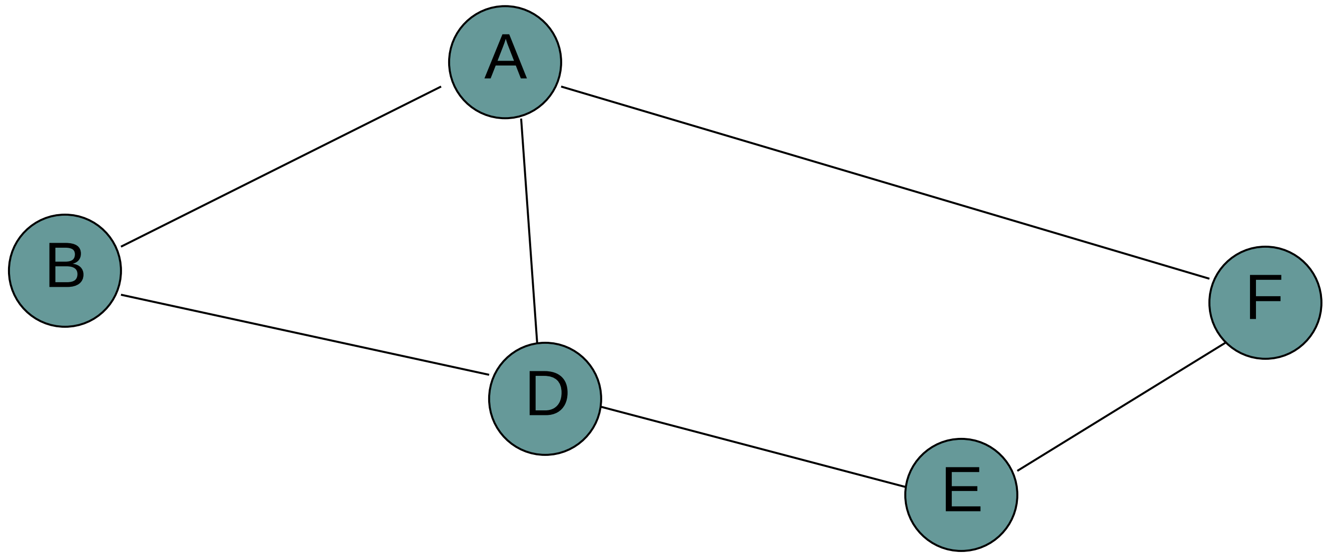
Given a polynomial equation, are there *integer* values of the variables such that the equation is true?

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

Unsolvable ☹

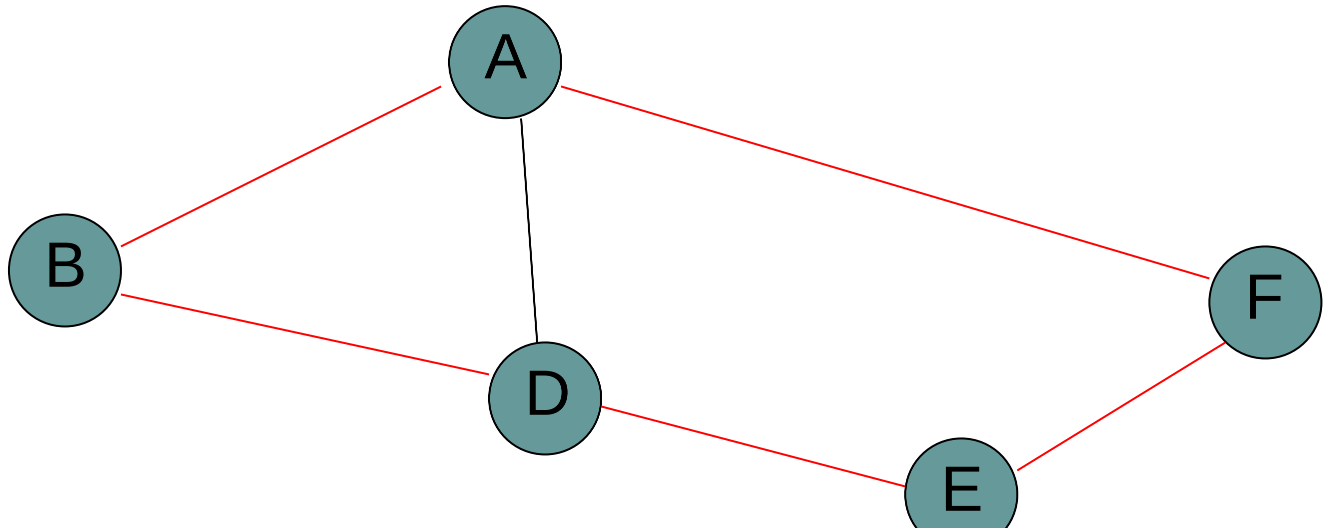
Hamiltonian cycle

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



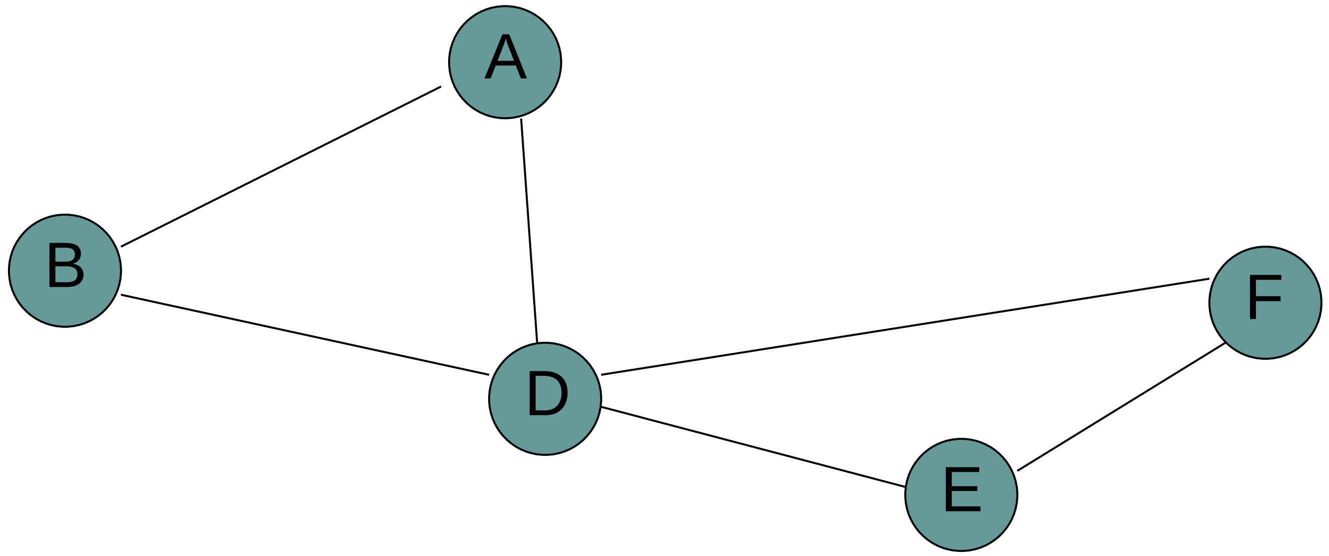
Hamiltonian cycle

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



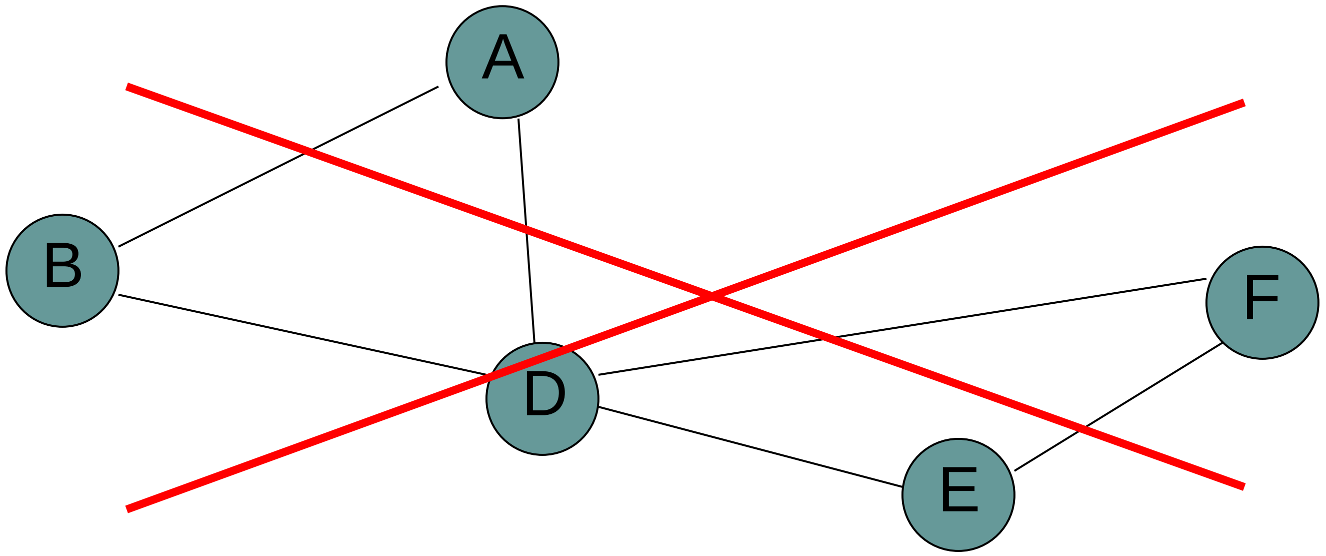
Hamiltonian cycle

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 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?

Tractable/intractable?

Solvable/unsolvable?

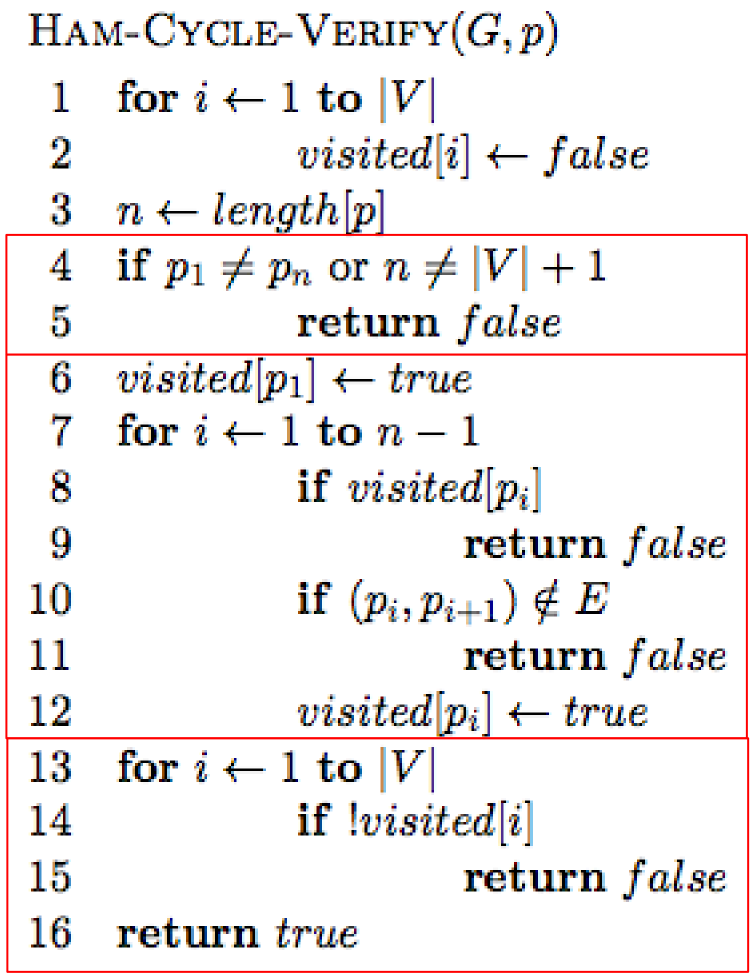
Hamiltonian cycle

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

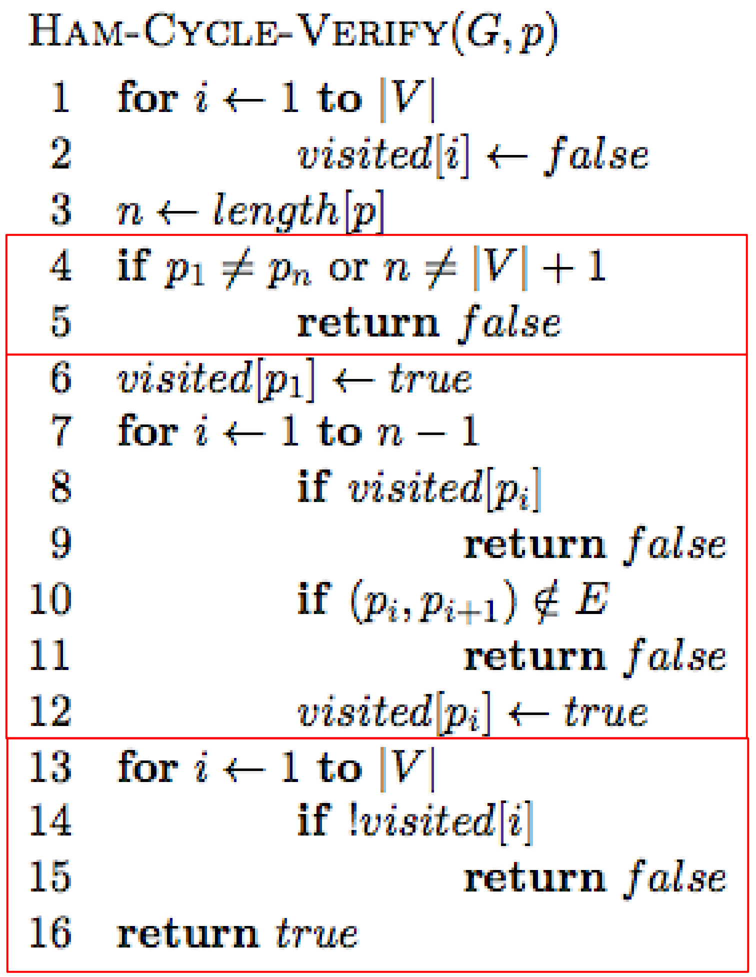
How would we do this check exactly, specifically given a graph and a path?

Checking hamiltonian cycles



```
HAM-CYCLE-VERIFY$(G, p)$
for $i \leftarrow 1$ to $|V|$
        $visited[i] \leftarrow false$
$n \leftarrow length[p]$
if $p_1 \neq p_n$ or $n \neq |V| + 1$
        return $false$
$visited[p_1] \leftarrow true$
for $i \leftarrow 1$ to $n - 1$
        if $visited[p_i]$
                return $false$
        if $(p_i, p_{i+1}) \notin E$
                return $false$
        $visited[p_i] \leftarrow true$
for $i \leftarrow 1$ to $|V|$
        if $!visited[i]$
                return $false$
return $true$
```

Checking hamiltonian cycles



```
HAM-CYCLE-VERIFY($G, p$)
for $i \leftarrow 1$ to $|V|$
        $visited[i] \leftarrow false$
$n \leftarrow length[p]$
if $p_1 \neq p_n$ or $n \neq |V| + 1$
        return $false$
$visited[p_1] \leftarrow true$
for $i \leftarrow 1$ to $n - 1$
        if $visited[p_i]$
                return $false$
        if $(p_i, p_{i+1}) \notin E$
                return $false$
        $visited[p_i] \leftarrow true$
for $i \leftarrow 1$ to $|V|$
        if $!visited[i]$
                return $false$
return $true$
```

Make sure the path starts and ends at the same vertex and is the right length

Can't revisit a vertex

Edge has to be in the graph

Check if we visited all the vertices

Checking hamiltonian cycles

```
HAM-CYCLE-VERIFY$(G, p)$
for $i \leftarrow 1$ to $|V|$
        $visited[i] \leftarrow false$
$n \leftarrow length[p]$
if $p_1 \neq p_n$ or $n \neq |V| + 1$
        return $false$
$visited[p_1] \leftarrow true$
for $i \leftarrow 1$ to $n - 1$
        if $visited[p_i]$
                return $false$
        if $(p_i, p_{i+1}) \notin E$
                return $false$
        $visited[p_i] \leftarrow true$
for $i \leftarrow 1$ to $|V|$
        if $!visited[i]$
                return $false$
return $true$
```

Running time?

O(V) adjacency matrix
O(V+E) adjacency list

What does that say about the hamilonian cycle problem?

It belongs to NP

P problems

P = problems with a polynomial runtime solution

Also, called “tractable” problems

(Basically, all of the problems in this class)

NP problems

NP is the set of problems that can be *verified* in polynomial time

A problem can be verified in polynomial time if you can check that a given solution is correct in polynomial time

(NP is an abbreviation for non-deterministic polynomial time)

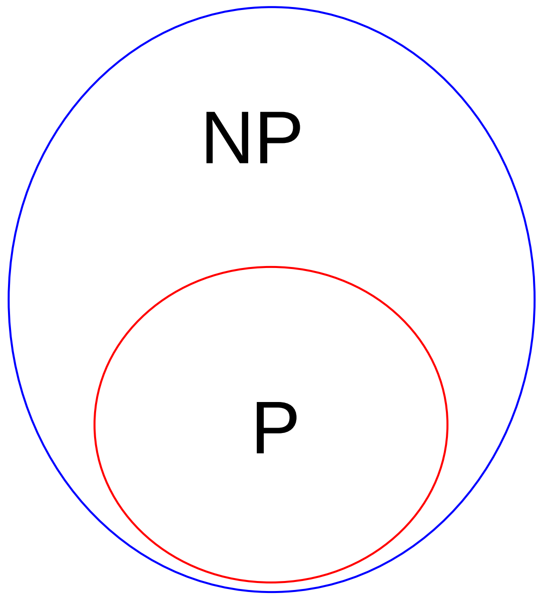
NP problems

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*

P and NP



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

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

$$\begin{tikzcd}
x \arrow[r] & \boxed{f} \arrow[r] & f(x) \\
P_1 \text{ instance} & & P_2 \text{ instance}
\end{tikzcd}$$

Reduction function

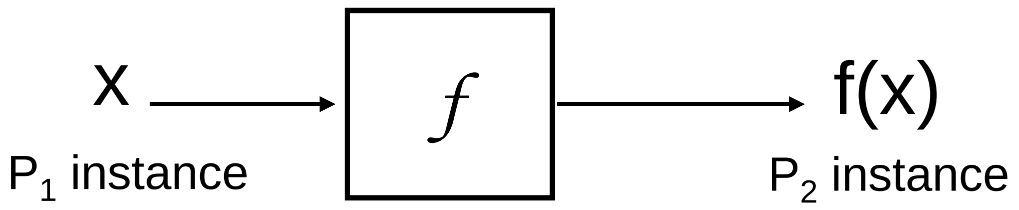
Where have we seen reductions before?

- Bipartite matching reduced to flow problem
- All pairs shortest path *through a particular vertex* reduced to single source shortest path

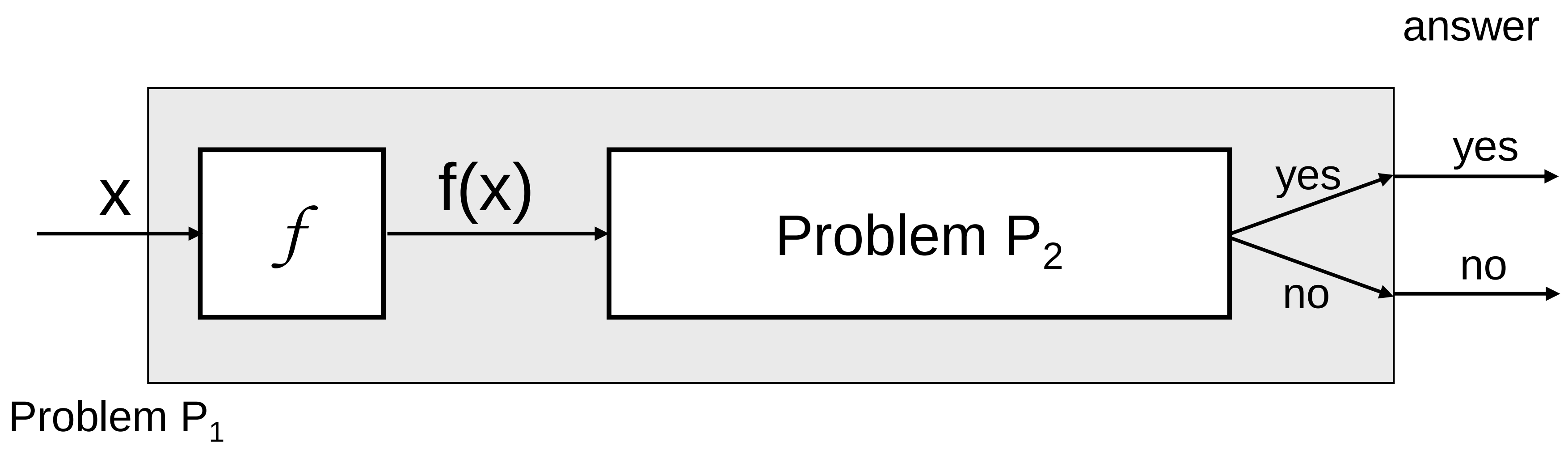
Why are they useful?

$$\begin{tikzcd}
x \arrow[r] & \boxed{f} \arrow[r] & f(x) \\
P_1 \text{ instance} & & P_2 \text{ instance}
\end{tikzcd}$$

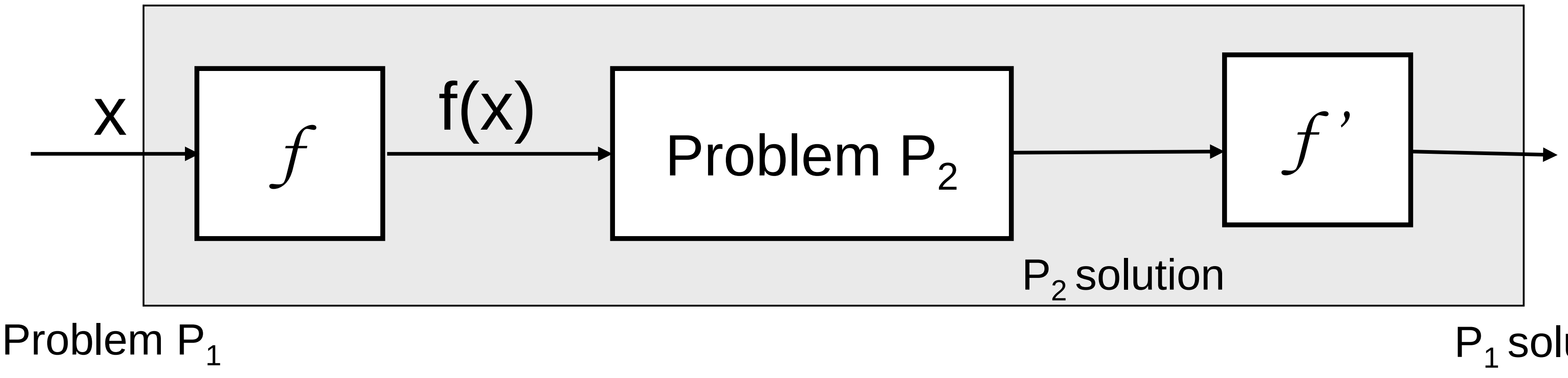
Reduction function



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

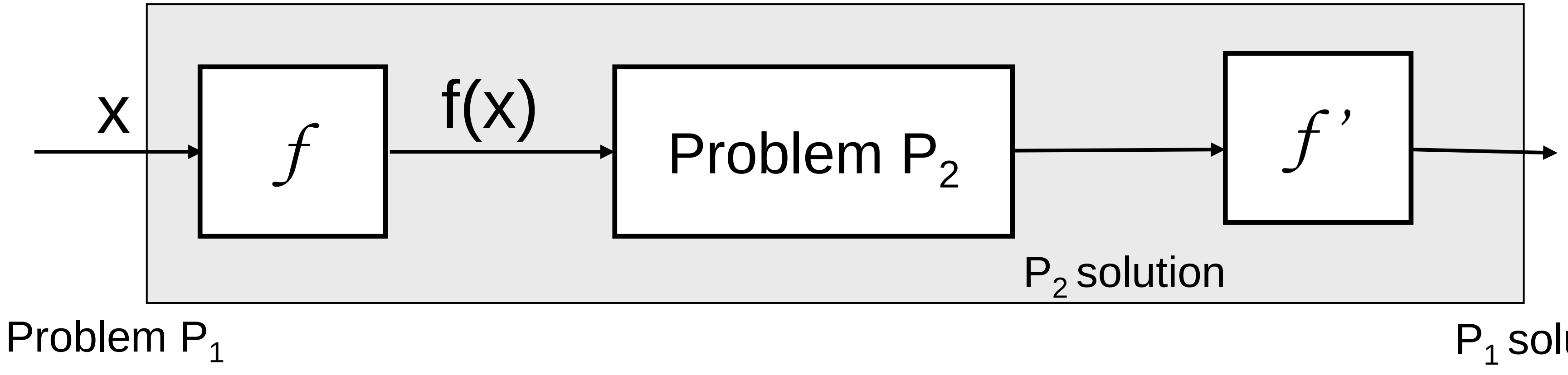


Reduction function



Most of the time we'll worry about yes no question, however, if we have more complicated answers we often just have to do a little work to the solution to the problem of P_2 to get the answer

Reduction function: Example

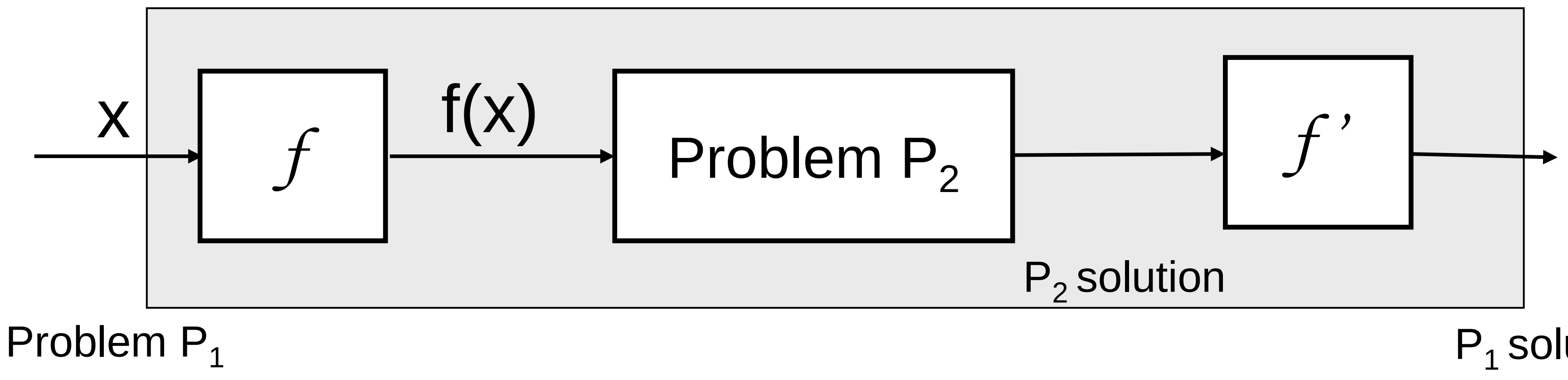


P1 = Bipartite matching
P2 = Network flow

Reduction function (f): Given *any* bipartite matching problem turn it into a network flow problem

What is *f* and what is *f'*?

Reduction function: Example



P1 = Bipartite matching
P2 = Network flow

Reduction function (f): Given *any* bipartite matching problem turn it into a network flow problem

A reduction function reduces problems instances

NP-Complete

A problem is *NP-complete* if:

1. it can be verified in polynomial time (i.e. in NP)
2. *any* NP-complete problem can be reduced to the problem in polynomial time (is NP-hard)

The hamiltonian cycle problem is NP-complete

What are the implications of this?
What does this say about how hard the hamiltonian cycle problem is compared to other NP-complete problems?

NP-Complete

A problem is *NP-complete* if:

1. it can be verified in polynomial time (i.e. in NP)
2. *any* NP-complete problem can be reduced to the problem in polynomial time (is NP-hard)

The hamiltonian cycle problem is NP-complete

It's *at least as hard* as *any* of the other NP-complete problems

NP-Complete

A problem is *NP-complete* if:

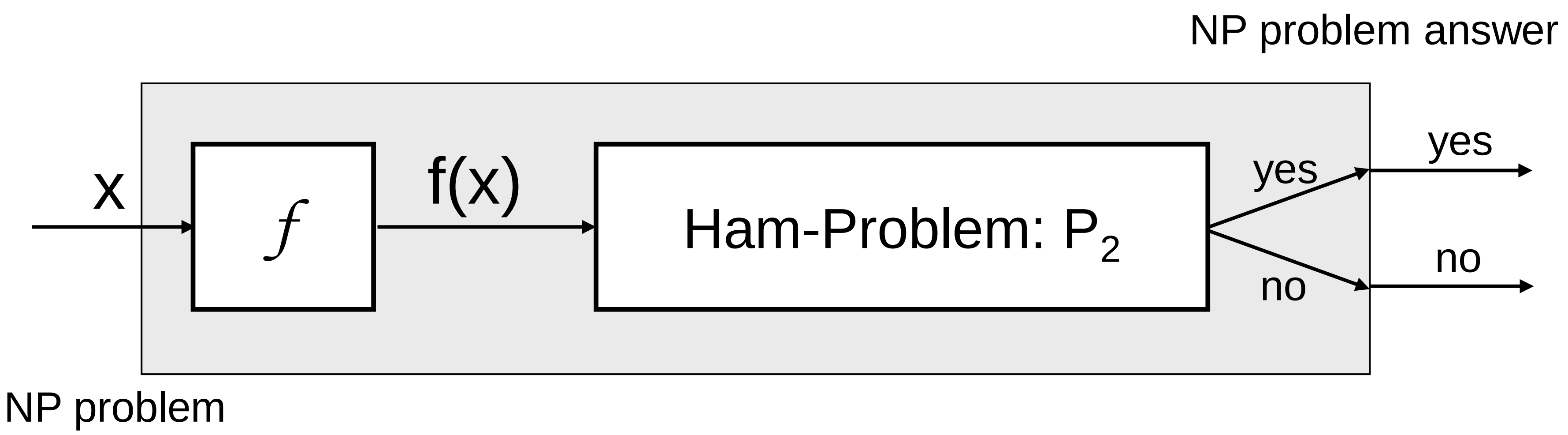
1. it can be verified in polynomial time (i.e. in NP)
2. *any* NP-complete problem can be reduced to the problem in polynomial time (is NP-hard)

If I found a polynomial-time solution to the hamiltonian cycle problem, what would this mean for the other NP-complete problems?

NP-complete

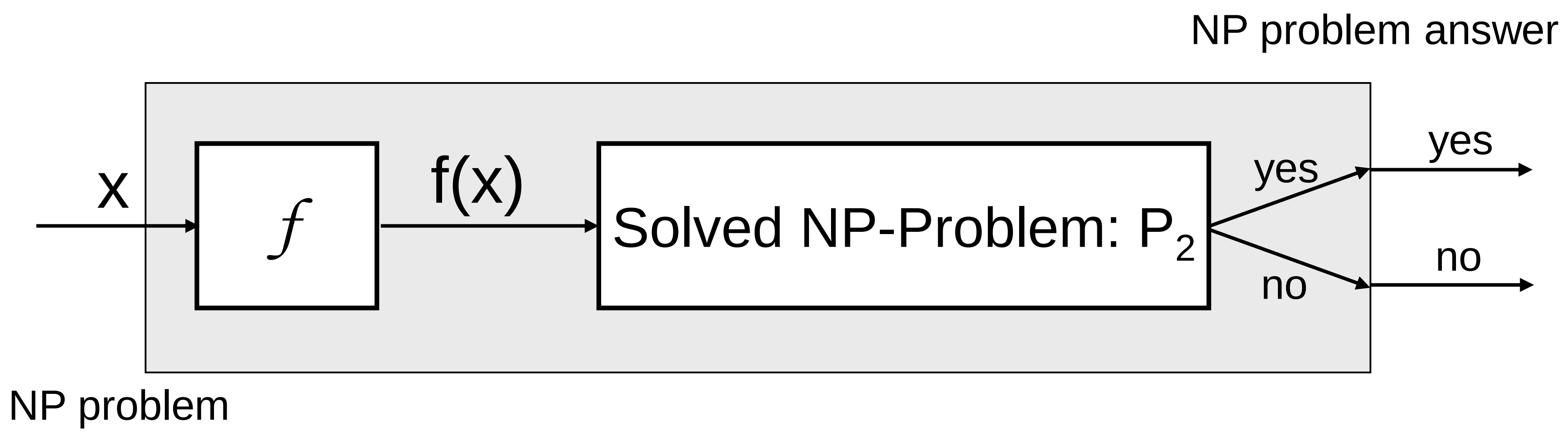
If a polynomial-time solution to the hamiltonian cycle problem is found, we would have a polynomial time solution to *any* NP-complete problem

- Take the input of the problem
- Convert it to the hamiltonian cycle problem (by definition, we know we can do this in polynomial time)
- Solve it
- If yes output yes, if no, output no



NP-complete

Similarly, if we found a polynomial time solution to *any* NP-complete problem we'd have a solution to *all* NP-complete problems



NP-complete problems

Longest path

Given a graph G with nonnegative edge weights does a simple path exist from *s* to *t* with weight at least *g*?

NP-complete problems

3D matching

- Bipartite matching: given two sets of things and pair constraints, find a matching between the sets
- 3D matching: given three sets of things and triplet constraints, find a matching between the sets

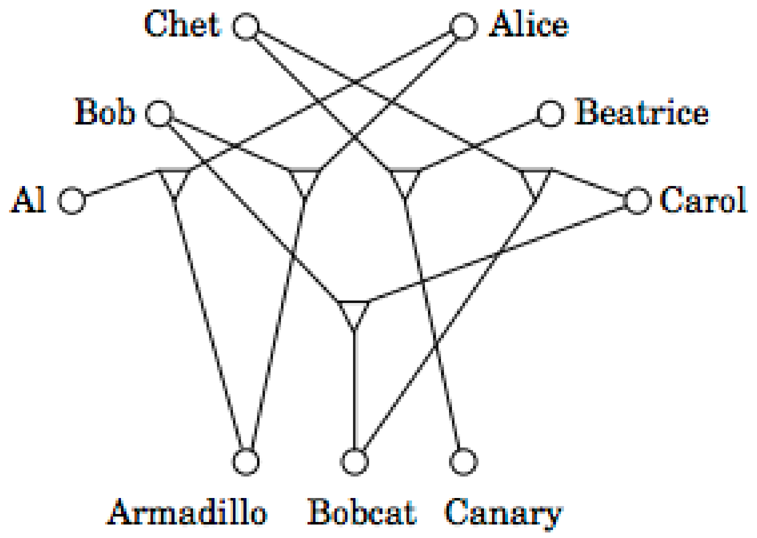


Figure from Dasgupta et. al 2008

P vs. NP

Polynomial time solutions exist	NP-complete (and no polynomial time solution currently exists)
Shortest path	Longest path
Bipartite matching	3D matching
Linear programming	Integer linear programming
Minimum cut	Balanced cut
…	…

Proving NP-completeness

A problem is *NP-complete* if:

1. it can be verified in polynomial time (i.e. in NP)
2. *any* NP-complete problem can be reduced to the problem in polynomial time (is NP-hard)

Ideas?

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

Proving NP-completeness

Show that all NP-complete problems are reducible to NEW in polynomial time

Why is it sufficient to show that one NP-complete problem reduces to the NEW problem?

Proving NP-completeness

Show that all NP-complete problems are reducible to NEW in polynomial time

All others can be reduced to NEW by first reducing to the one problem, then reducing to NEW. Two polynomial time reductions is still polynomial time!

Proving NP-completeness

Show that all NP-complete problems are reducible to NEW in polynomial time

$\Downarrow$

Show that *any* NP-complete problem is reducible to NEW in polynomial time

BE CAREFUL!

~~Show that NEW is reducible to any NP-complete problem in polynomial time~~

NP-complete: 3-SAT

A boolean formula is in *n-conjunctive normal form* (*n*-CNF) if:

- it is expressed as an AND of clauses
- where each clause is an OR of no more than *n* variables

$$(a \vee \neg a \vee \neg b) \wedge (c \vee b \vee d) \wedge (\neg a \vee \neg c \vee \neg d)$$

3-SAT: Given a 3-CNF boolean formula, is it satisfiable?

3-SAT is an NP-complete problem

NP-complete: SAT

Given a boolean formula of *n* boolean variables joined by *m* connectives (AND, OR or NOT) is there a setting of the variables such that the boolean formula evaluate to true?

$$(a \text{Ù} b)\text{Ú}(\text{Ø}a \text{Ù} \text{Ø}b)$$

$$((\text{Ø}(b \text{Ú} \text{Ø}c)\text{Ù} a)\text{Ú}(a \wedge b \wedge c)) \wedge c \wedge \text{Ø}b$$

Is SAT an NP-complete problem?

NP-complete: SAT

Given a boolean formula of *n* boolean variables joined by *m* connectives (AND, OR or NOT) is there a setting of the variables such that the boolean formula evaluate to true?

$$((\neg(b \vee \neg c) \wedge a) \vee (a \wedge b \wedge c)) \wedge c \wedge \neg b$$

1. Show that SAT 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 SAT in polynomial time
   a. Describe a reduction function *f* from a known NP-Complete problem to SAT
   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 SAT problem generate by f*

NP-Complete: SAT

1. Show that SAT is in NP
   a. Provide a verifier
   b. Show that the verifier runs in polynomial time

Verifier: A solution consists of an assignment of the variables

- If clause is a single variable:
  - return the value of the variable
- otherwise
  - for each clause:
    - call the verifier recursively
    - compute a running solution

polynomial run-time?

NP-Complete: SAT

Verifier: A solution consists of an assignment of the variables

- If clause is a single variable:
  - return the value of the variable
- otherwise
  - for each clause:
    - call the verifier recursively
    - compute a running solution

linear time

- at most a linear number of recursive calls (each call makes the problem smaller and no overlap)
- overall polynomial time

NP-Complete: SAT

1.
2. Show that all NP-complete problems are reducible to SAT in polynomial time
   a. Describe a reduction function *f* from a known NP-Complete problem to SAT
   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 SAT problem generate by f*

Reduce 3-SAT to SAT:
- Given an instance of 3-SAT, turn it into an instance of SAT

Reduction function:
- DONE ☺

- Runs in constant time! (or linear if you have to copy the problem)

NP-Complete: SAT

Show that a solution exists to the NP-Complete problem IFF a solution exists *to the NEW problem generated 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

- Assume we have a 3-SAT problem with a solution:
  - Because 3-SAT problems are a subset of SAT problems, then the SAT problem will also have a solution
- Assume we have a problem instance generated by our reduction with a solution:
  - Our reduction function simply does a copy, so it is already a 3-SAT problem
  - Therefore the variable assignment found by our SAT-solver will also be a solution to the original 3-SAT problem

NP-Complete problems

Why do we care about showing that a problem is NP-Complete?

- We know that the problem is hard (and we probably won’t find a polynomial time exact solver)

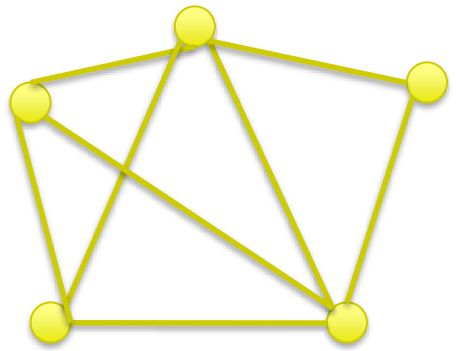
- We may need to compromise:
  - reformulate the problem
  - settle for an approximate solution

- Down the road, if a solution is found for an NP-complete problem, then we’d have one too…

CLIQUE

A *clique* in an undirected graph G = (V, E) is a subset V' ⊆ 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?

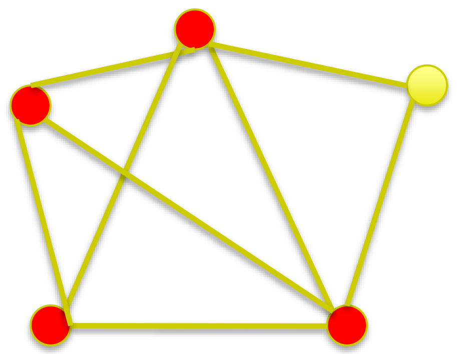


Is there a clique of size 4 in this graph?

CLIQUE

A *clique* in an undirected graph G = (V, E) is a subset V’ ⊆ 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?



CLIQUE is an NP-Complete problem

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'

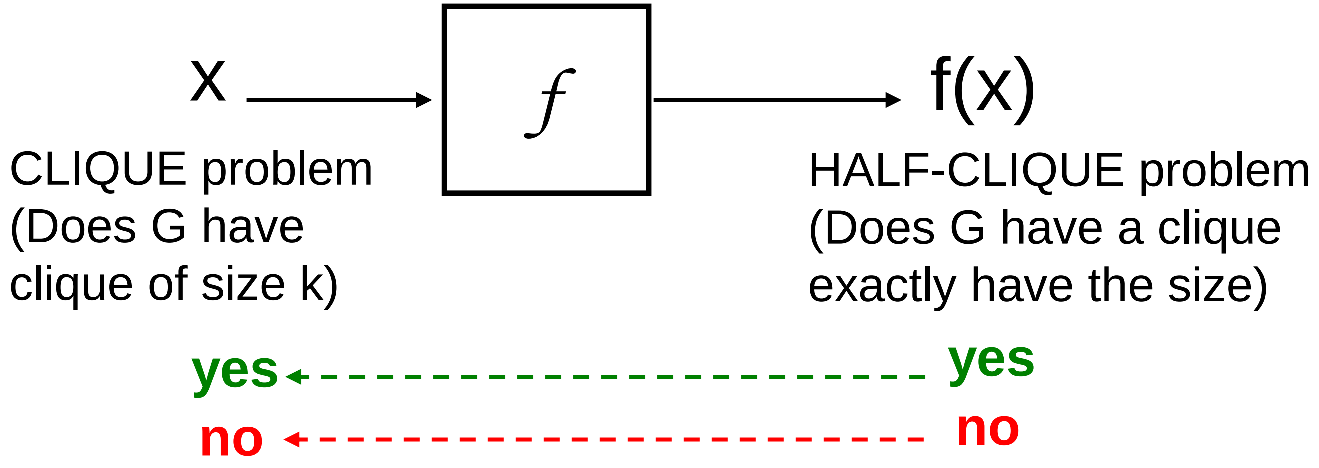
- check that |V '| = |V|/2
- for all pairs of u, v ∈ V'
  - there exists an edge (u,v) ∈ E

- 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



x
f
f(x)
CLIQUE problem (Does G have clique of size k)
HALF-CLIQUE problem (Does G have a clique exactly have the size)
yes
yes
no
no

HALF-CLIQUE

$$\begin{tikzcd} x \arrow[r] & \boxed{f} \arrow[r] & f(x) \end{tikzcd}$$

CLIQUE problem
(Does G have
clique of size k)

HALF-CLIQUE problem
(Does G have a clique
exactly have the size)

Three cases:

1. $k = |V|/2$

2. $k < |V|/2$

3. $k > |V|/2$

HALF-CLIQUE

Reduce CLIQUE to HALF-CLIQUE:
Given an instance of CLIQUE, turn it into an instance of HALF-CLIQUE

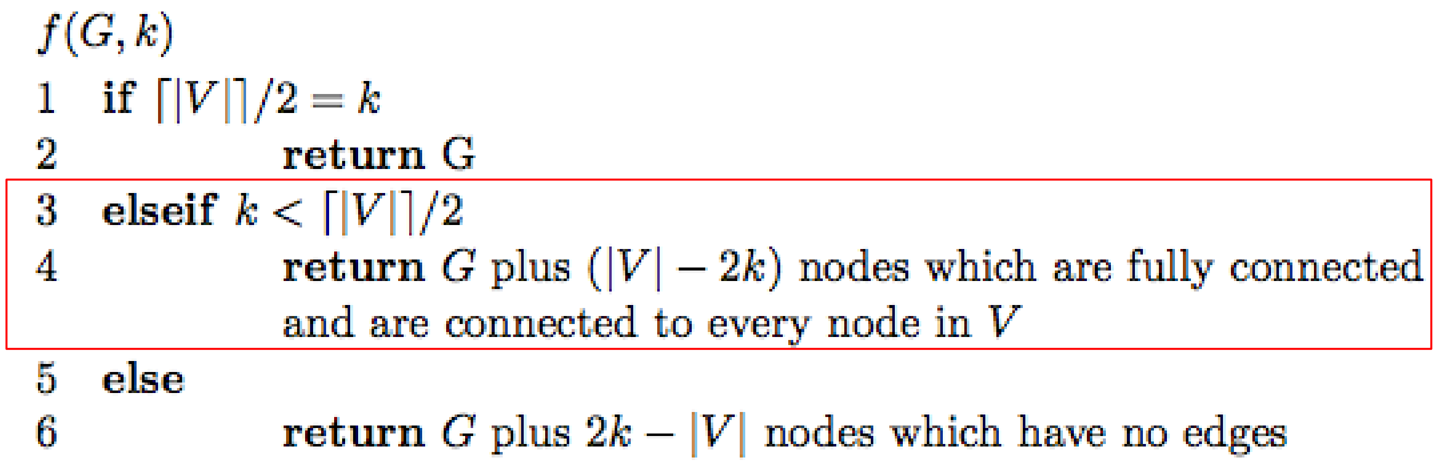
It’s already a half-clique problem

```
$f(G, k)$
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 smaller than half, so add an artificial clique to the graph and connect it up to all vertices

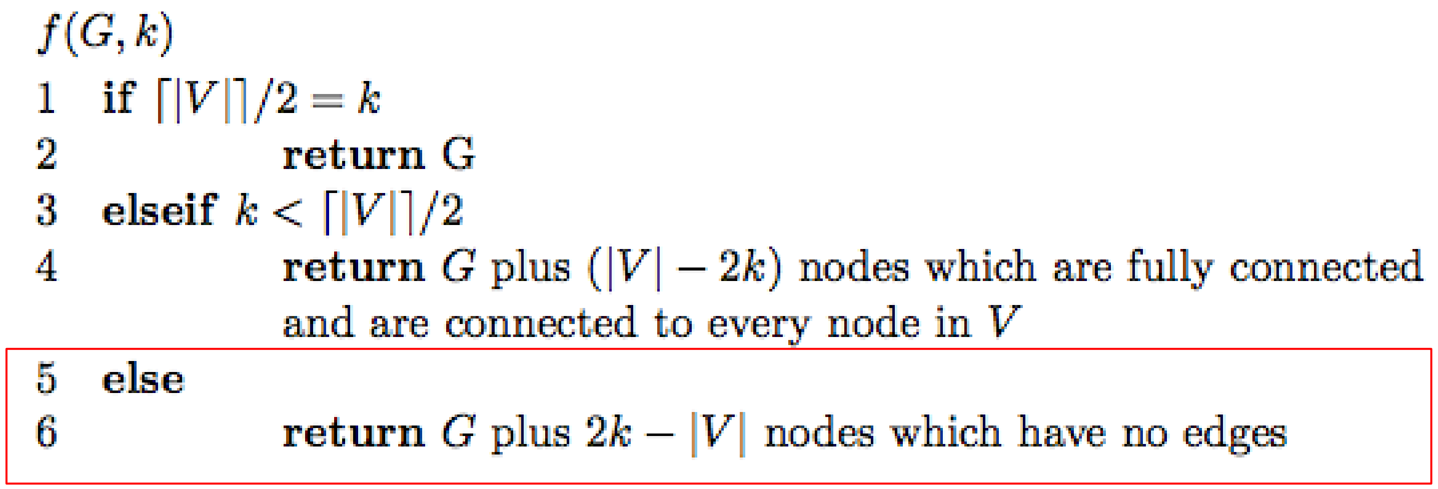


```
f(G, k)
1  if $\lceil |V| \rceil / 2 = k$
2          return G
3  elseif $k < \lceil |V| \rceil / 2$
4          return $G$ plus $(|V| - 2k)$ nodes which are fully connected
           and are connected to every node in $V$
5  else
6          return $G$ plus $2k - |V|$ nodes which have no edges
```

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



```
$f(G, k)$
if $\lceil |V| \rceil / 2 = k$
        return G
elseif $k < \lceil |V| \rceil / 2$
        return $G$ plus $(|V| - 2k)$ nodes which are fully connected
        and are connected to every node in $V$
else
        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

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

```
$f(G, k)$
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
```

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

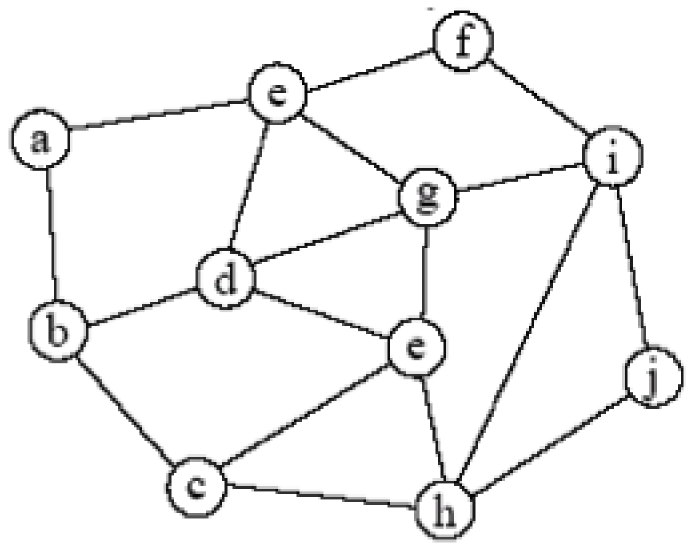
Given a graph f(G) that has a CLIQUE half the elements, show that G has a clique of size k

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

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

Independent-Set

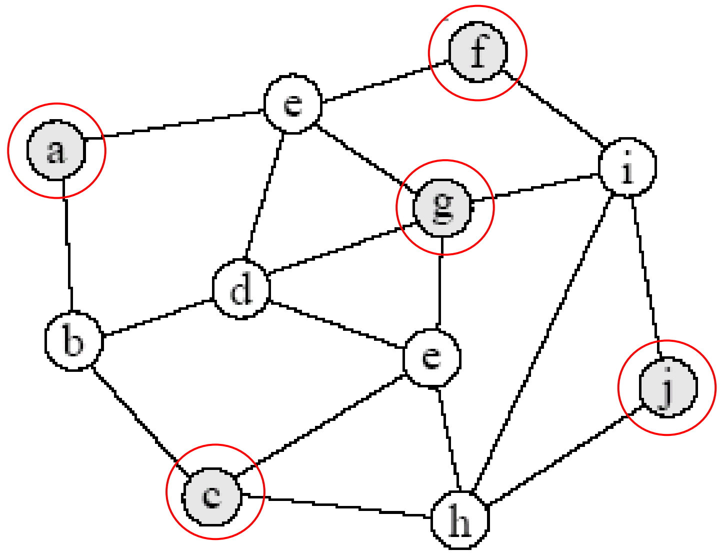
Given a graph G = (V, E) is there a subset V’⊆ V of vertices of size |V’| = k that are independent, i.e. for any pair of vertices u, v ∊ V’ there exists no edge between any of these vertices



Does the graph contain an independent set of size 5?

Independent-Set

Given a graph G = (V, E) is there a subset V’⊆ V of vertices of size |V ‘| = k that are independent, i.e. for any pair of vertices u, v ∊ V’ there exists no edge between any of these vertices

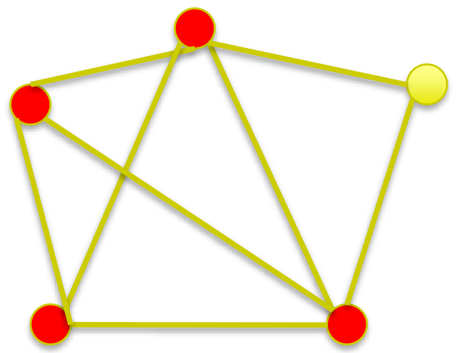


Independent-Set is NP-Complete

CLIQUE revisited

A *clique* in an undirected graph G = (V, E) is a subset V' ⊆ 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?

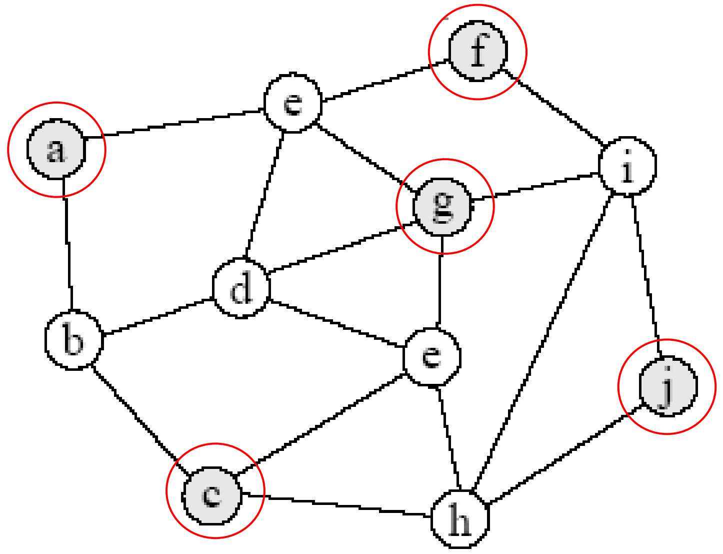
Is CLIQUE NP-Complete?

1. Show that CLIQUE 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 CLIQUE in polynomial time
   a. Describe a reduction function *f* from a known NP-Complete problem to 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 CLIQUE problem generate by f*

Given a graph G, does the graph contain a clique of size k?

Independent-Set

Given a graph G = (V, E) is there a subset V'⊆ V of vertices of size |V '| = k that are independent, i.e. for any pair of vertices u, v ∊ 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’⊆ V of vertices of size |V ‘| = k that are independent, i.e. for any pair of vertices u, v ∈ 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 a graph constructed by remove 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’
```

Reduction proof

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 Independent-Set problem instance that has a solution, show that the Clique problem instance generated by *f* has a solution
- Assume we have a problem instance of Clique *generated by f* that has a solution, show that we can derive a solution to Independent-Set problem instance

```
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

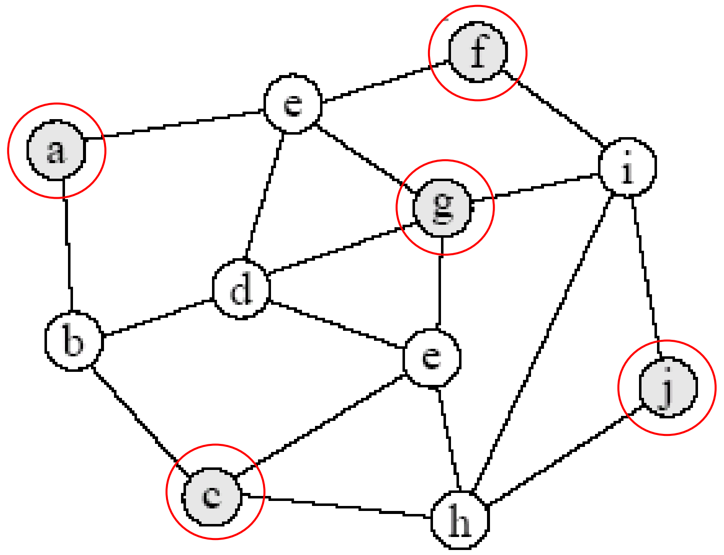
Proof

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

Independent-Set revisited

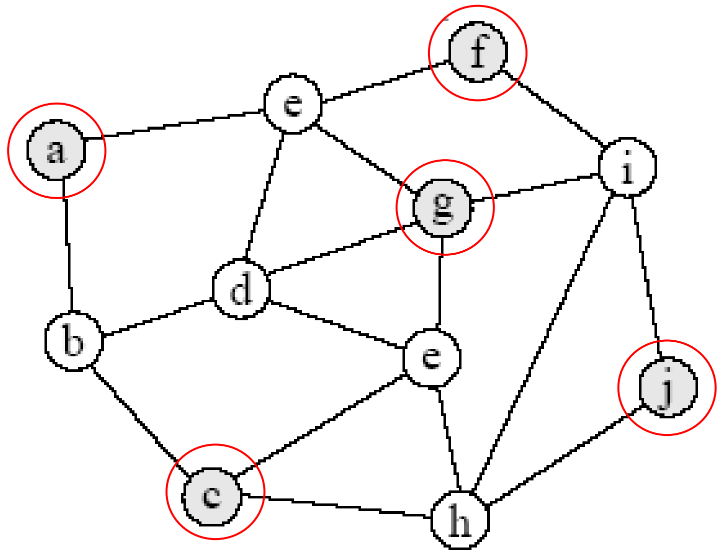
Given a graph G = (V, E) is there a subset V'⊆ V of vertices of size |V '| = k that are independent, i.e. for any pair of vertices u, v ∈ V' there exists no edge between any of these vertices



Is Independent-Set NP-Complete?

Independent-Set revisited

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



Reduce 3-SAT to Independent-Set

3-SAT to Independent-Set

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

$$(a \lor \neg a \lor \neg b) \land (c \lor b \lor d) \land (\neg a \lor \neg c \lor \neg d)$$

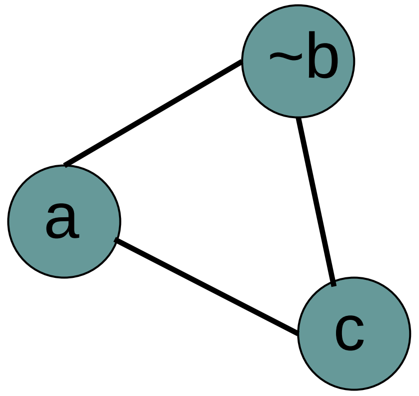
For the boolean formula in 3-SAT to be satisfied, at least one of the literals in each clause must be true

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



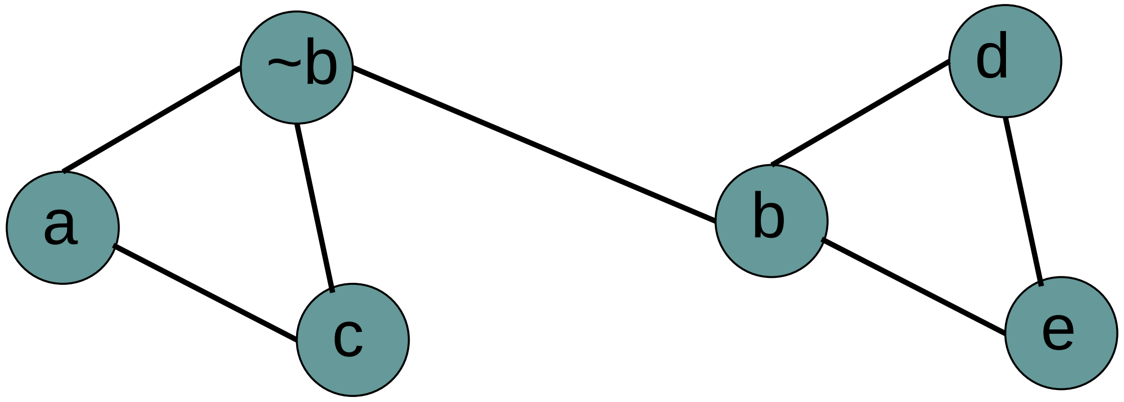
- for the Independent-Set problem to be satisfied it can only select one variable

- 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 ~x



Proof

Given a 3-SAT problem with k clauses and a valid truth assignment, show that f(3-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-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 on 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’⊆ 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’⊆V such that if (u,v)∈E then u∈V’ or v∈V’?
- The extra credit was to solve this problem for bipartite graphs

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

Search vs. Exists

All the problems we've looked at asked decision questions:

- Is there a hamiltonian cycle?
- Does the graph have a clique of size k?
- Does the graph has an independent set of size k?
- …

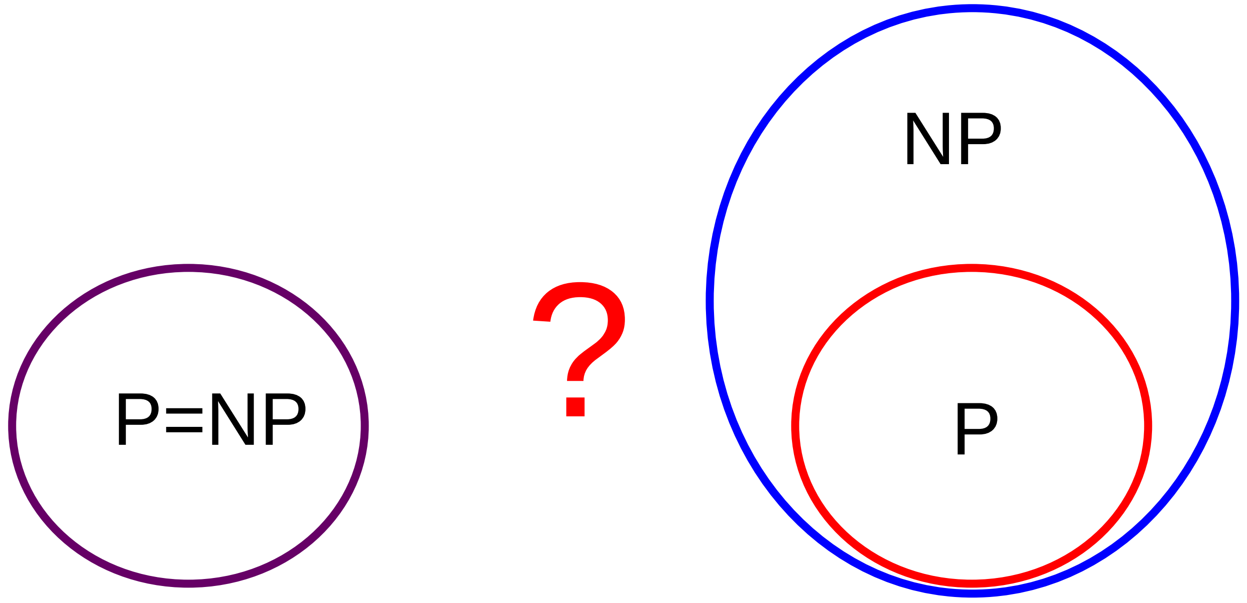
For many of the problems with a k in them, we really want to know what the largest/smallest one is

- What is the largest clique in the graph?
- What is the shortest path that visits all the vertices exactly once?

Why don't we care?

P vs. NP

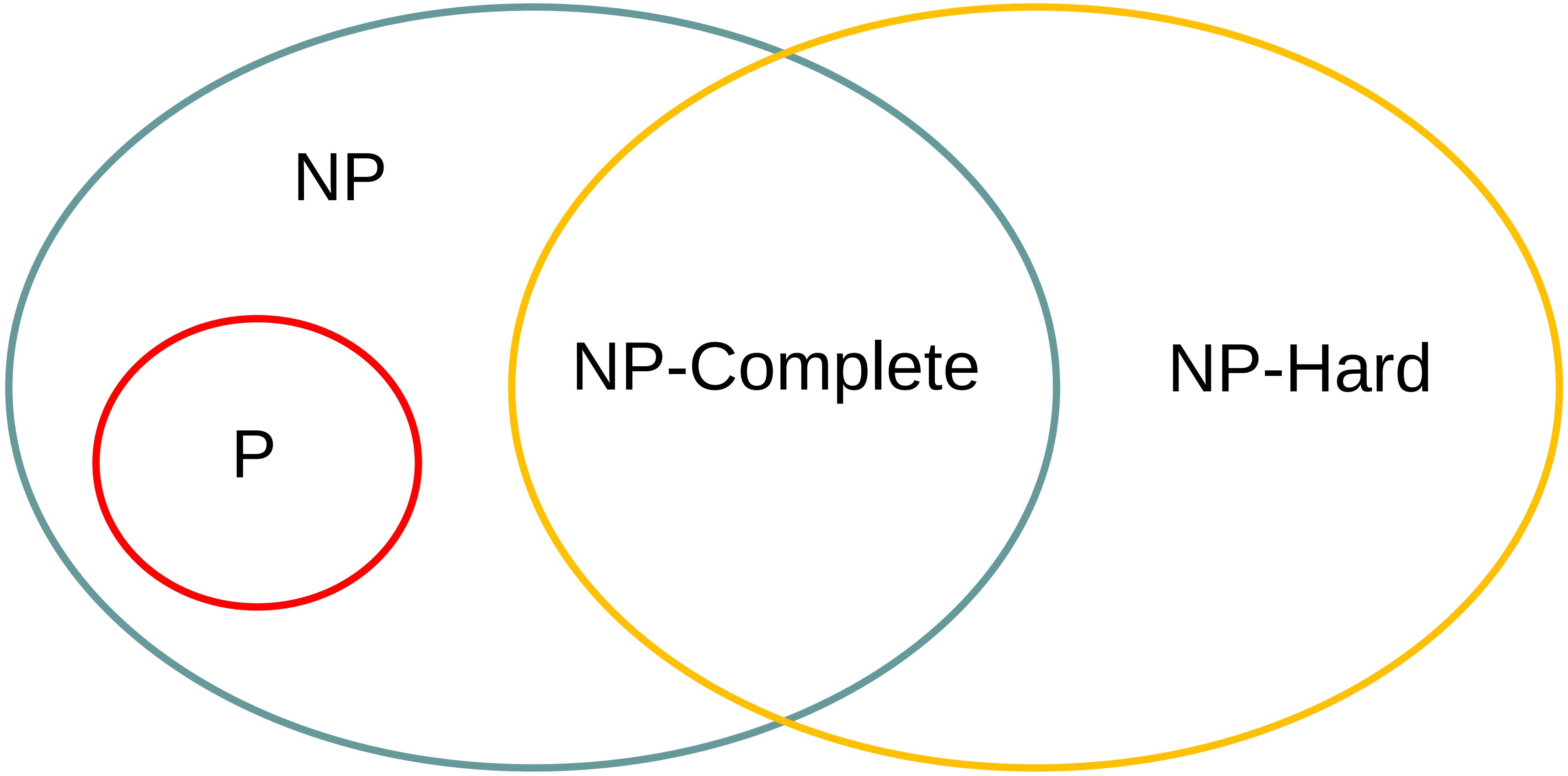
The big question:



Someone finds a polynomial time solution to one of the NP-Complete problems

NP-Complete problems are somehow harder and distinct

P vs. NP vs. NP-Complete vs. NP-Hard



P vs. NP vs. NP-Complete vs. NP-Hard

	P	NP	NP-Complete	NP-Hard
Solvable in polynomial time	√			
Solution verifiable in polynomial time	√	√	√	
Reduces any NP problem in polynomial time			√	√

Acknowledgements

- Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C., Introduction to algorithms. MIT press, 2009
- Dr. David Kauchak, Pomona College
- Prof. David Plaisted, The University of North Carolina at Chapel Hill