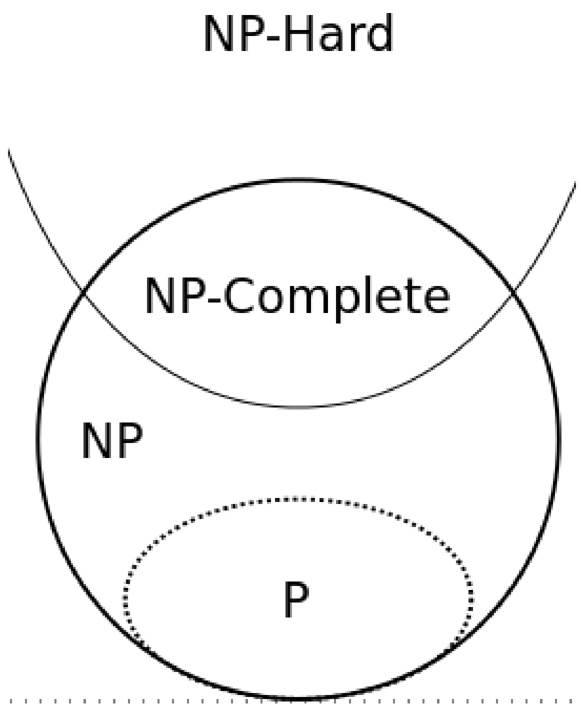
Lecture 10: NP and Computational Intractability



THE UNIVERSITY OF SYDNEY

Algorithms and hardness

Algorithmic techniques:

- Greedy algorithms [Lecture 3]
- Divide & Conquer algorithms [Lecture 4]
- Sweepline algorithms [Lecture 5]
- Dynamic programming algorithms [Lectures 6 and 7]
- Network flow algorithms [Lectures 8 and 9]

Hardness:

- NP-hardness [Lecture 10]: $O(n^c)$ algorithm is unlikely
- Undecidability: No algorithm possible (Halting problem)

Outline of today's lecture

- Reduction: polynomial time
  - Reduction by simple equivalence.
  - Reduction from special case to general case.
  - Reduction by encoding with gadgets.
- Definition of P and NP
- NP-completeness

Classify Problems According to Computational Requirements

Question: Which problems will we be able to solve in practice?

A working definition. [Cobham 1964, Edmonds 1965, Rabin 1966].
Those with polynomial-time algorithms.

Yes	Probably no
Shortest path	Longest path
Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover

Classify Problems

Aim: Classify problems according to those that can be solved in polynomial-time and those that cannot.

Frustrating news: Huge number of fundamental problems have defied classification for decades.

This lecture: Show that these fundamental problems (in the grey area) are "computationally equivalent" and appear to be different manifestations of one hard problem.

8.1 Polynomial-Time Reductions

Polynomial-Time Reduction

Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y, denoted $X \leq_P Y$, if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle that solves problem Y.

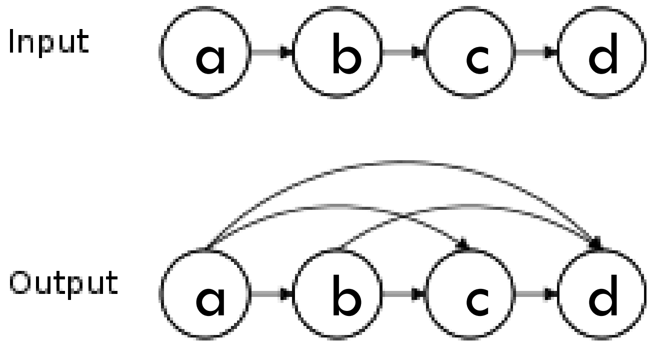
Polynomial-Time Reduction

Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y, denoted $X \leq_P Y$, if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle that solves problem Y.

Example: Transitive closure of a directed graph can be computed by n calls to BFS + the time to build the transitive closure.

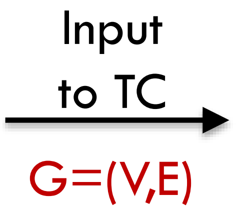


Polynomial-Time Reduction

Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y, denoted $X \leq_P Y$, if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle that solves problem Y.

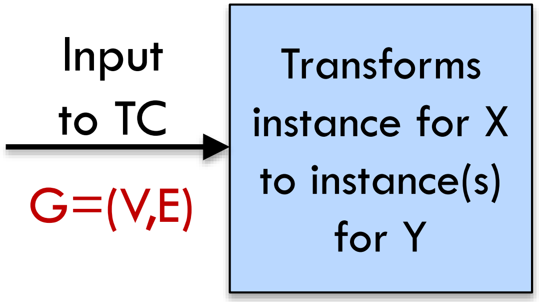


Polynomial-Time Reduction

Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y, denoted $X \leq_P Y$, if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle that solves problem Y.

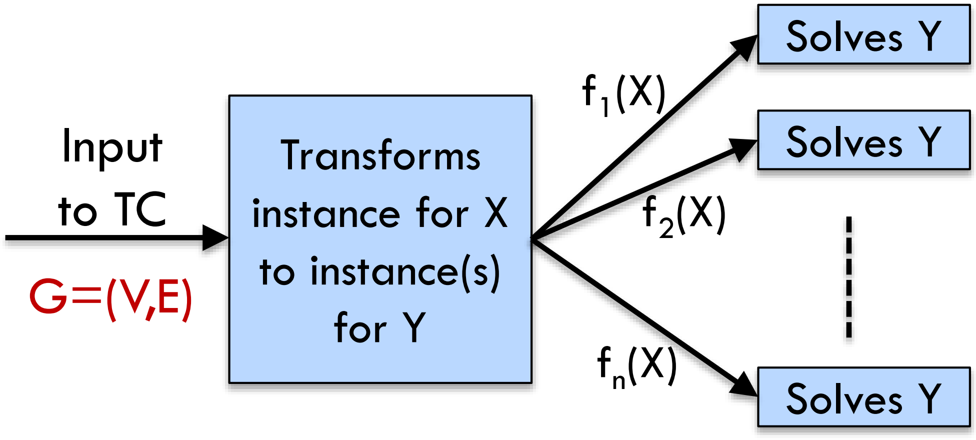


Polynomial-Time Reduction

Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y, denoted $X \leq_P Y$, if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle that solves problem Y.



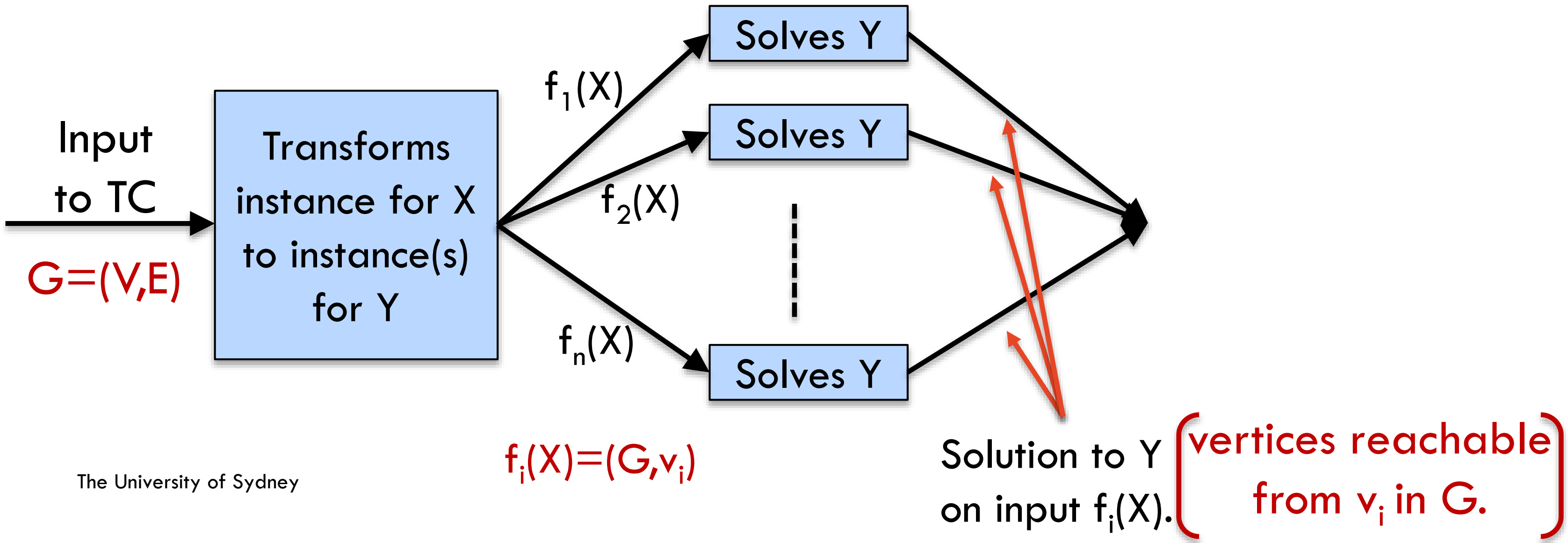
$f_i(X)=(G,v_i)$

Polynomial-Time Reduction

Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y, denoted $X \leq_P Y$, if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle that solves problem Y.



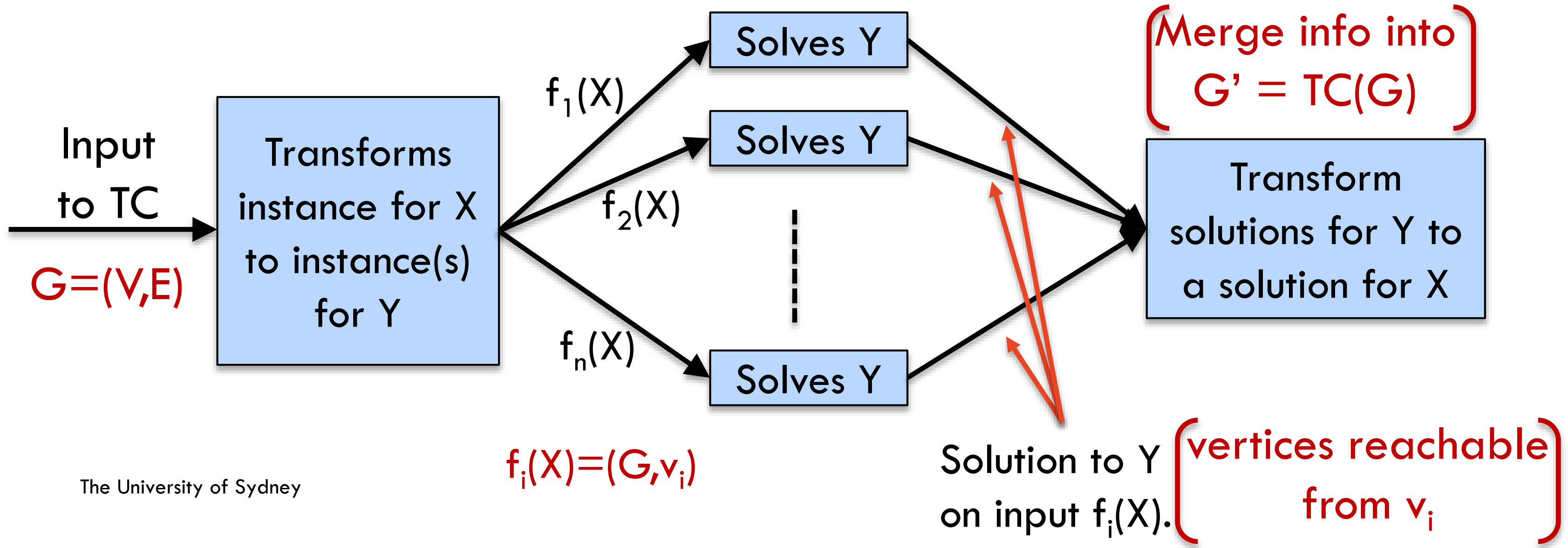
Input to TC
G=(V,E)
Transforms instance for X to instance(s) for Y
$f_1(X)$
$f_2(X)$
$f_n(X)$
Solves Y
Solves Y
Solves Y
$f_i(X)=(G,v_i)$
Solution to Y on input $f_i(X)$.
vertices reachable from v_i in G.

Polynomial-Time Reduction

Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y, denoted $X \leq_P Y$, if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle that solves problem Y.



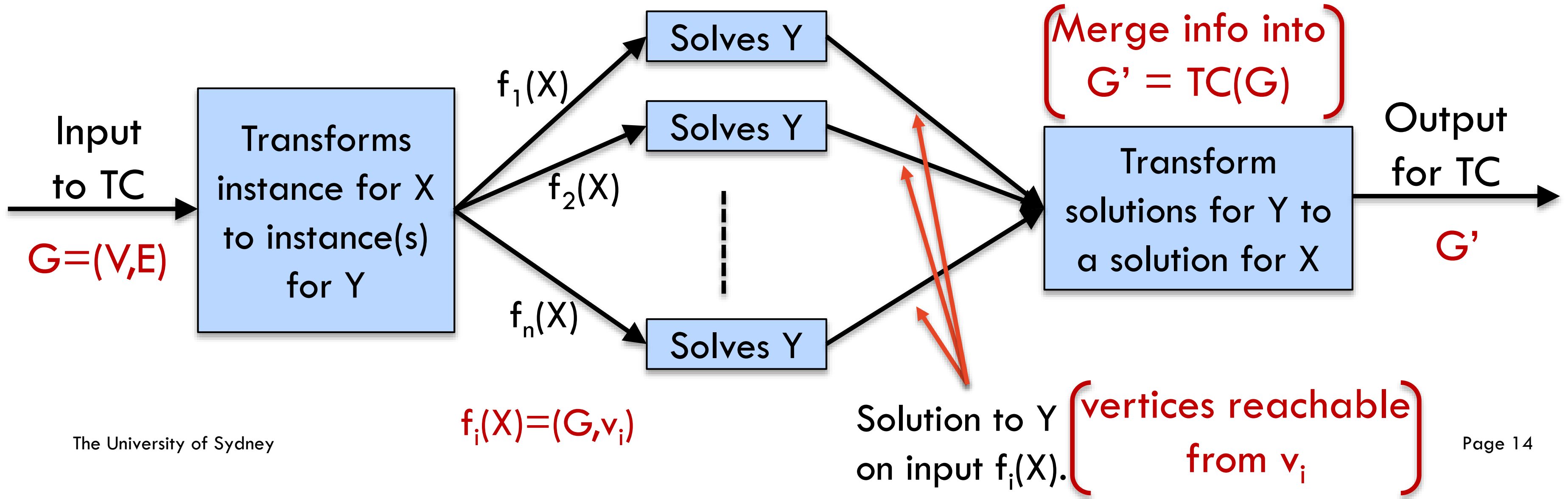
Input to TC
G=(V,E)
Transforms instance for X to instance(s) for Y
$f_1(X)$
$f_2(X)$
$f_n(X)$
Solves Y
Solves Y
Solves Y
$f_i(X)=(G,v_i)$
Merge info into G' = TC(G)
Transform solutions for Y to a solution for X
Solution to Y on input $f_i(X)$.
vertices reachable from v_i

Polynomial-Time Reduction

Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y, denoted $X \leq_P Y$, if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle that solves problem Y.



Input to TC
G=(V,E)
Transforms instance for X to instance(s) for Y
$f_1(X)$
$f_2(X)$
$f_n(X)$
Solves Y
Solves Y
Solves Y
$f_i(X)=(G,v_i)$
Solution to Y on input $f_i(X)$.
vertices reachable from v_i
Merge info into G' = TC(G)
Transform solutions for Y to a solution for X
Output for TC
G'

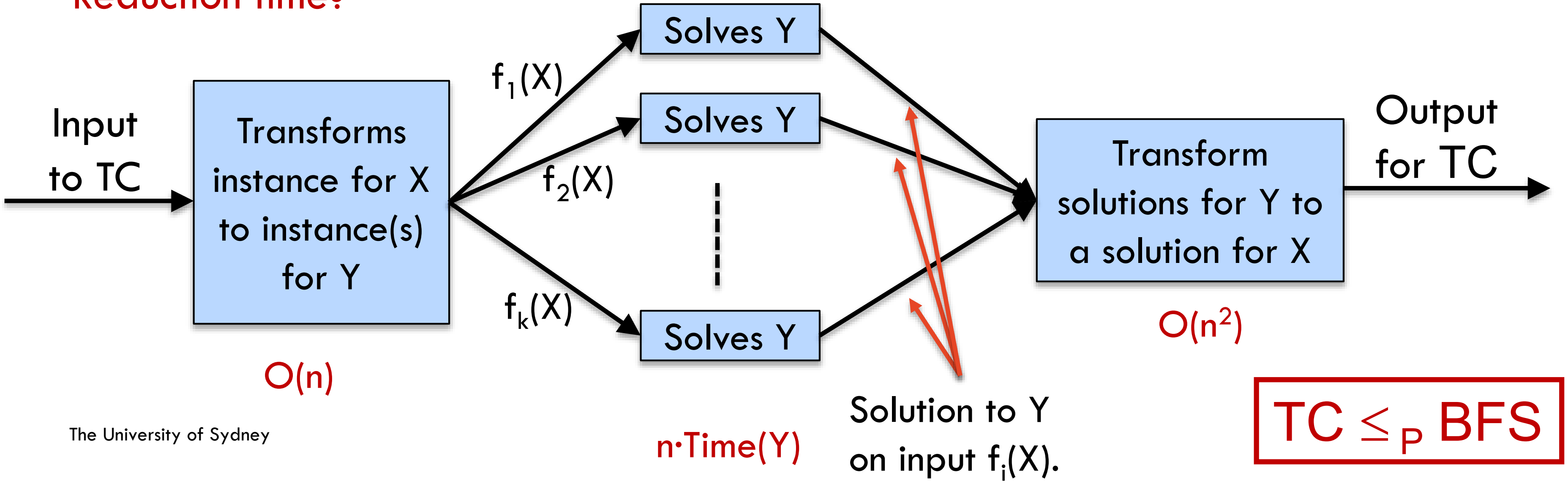
Polynomial-Time Reduction

Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y, denoted $X \leq_P Y$, if arbitrary instances of problem X can be solved using:

– Polynomial number of standard computational steps, plus
– Polynomial number of calls to an oracle that solves problem Y.

Reduction time?



Input to TC
Transforms instance for X to instance(s) for Y
$f_1(X)$
$f_2(X)$
$f_k(X)$
Solves Y
Solves Y
Solves Y
Transform solutions for Y to a solution for X
Output for TC
O(n)
n·Time(Y)
Solution to Y on input $f_i(X)$.
$O(n^2)$

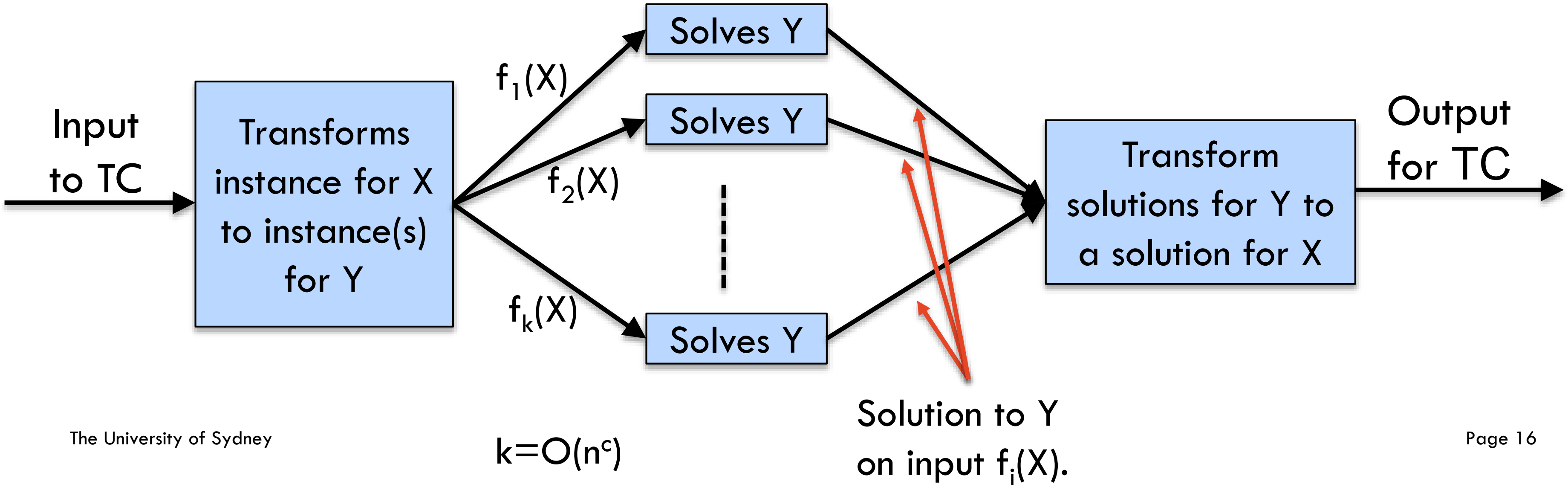
$\boxed{TC \leq_P BFS}$

Polynomial-Time Reduction

Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y, denoted $X \leq_P Y$, if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle that solves problem Y.



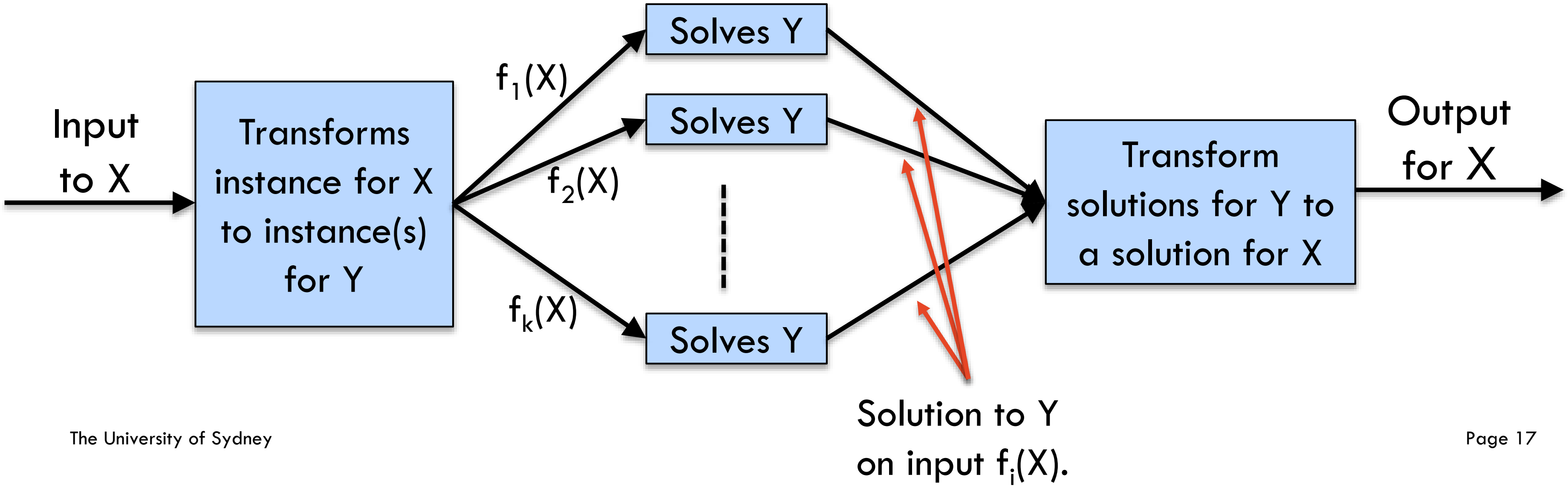
Input to TC
Transforms instance for X to instance(s) for Y
$f_1(X)$
$f_2(X)$
$f_k(X)$
Solves Y
Solves Y
Solves Y
Transform solutions for Y to a solution for X
Output for TC
Solution to Y on input $f_i(X)$.
$k=O(n^c)$

Polynomial-Time Reduction

Suppose we could solve problem Y in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y, denoted $X \leq_P Y$, if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle that solves problem Y.


Input to X
Transforms instance for X to instance(s) for Y
$f_1(X)$
$f_2(X)$
$f_k(X)$
Solves Y
Solves Y
Solves Y
Transform solutions for Y to a solution for X
Output for X
Solution to Y on input $f_i(X)$.

Polynomial-Time Reduction

Purpose. Classify problems according to relative difficulty.

1. Design algorithms. If $X \leq_P Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time.
2. Establish intractability. If $X \leq_P Y$ and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial time.

Establish equivalence: If $X \leq_P Y$ and $Y \leq_P X$, then $X \equiv_P Y$.

Reductions we have already seen

- Sorting reduces to convex hull
- Transitive closure reduces to breadth-first search
- Bipartite matching reduces to max flow
- Max flow reduces to min cut
- Circulation reduces to max flow
- Circulation with lower bounds reduces to circulation
- …

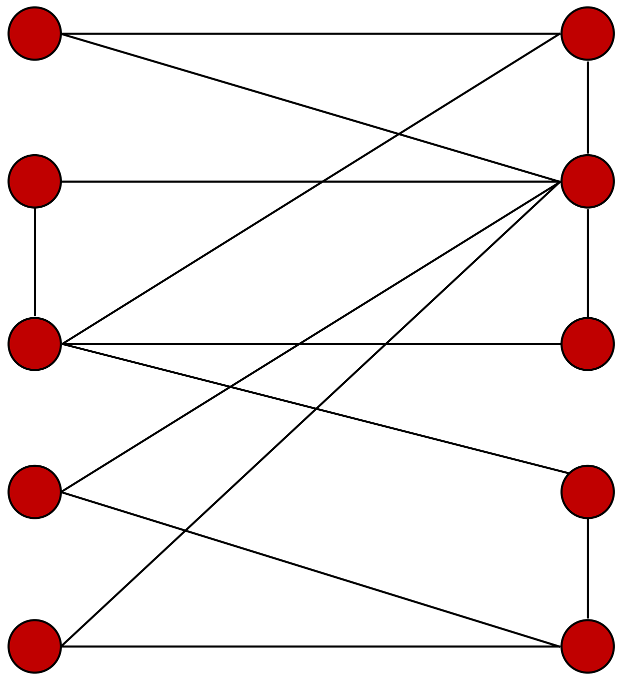
Reduction By Simple Equivalence

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Independent Set

INDEPENDENT-SET: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?



We will study decision problems

Decision problem:

Does there exist an independent set of size $\geq$ k?

Search problem: Find an independent set of maximum cardinality.

Self-reducibility: Search problem $\leq_P$ decision version.

- Applies to all problems in this lecture.
- Justifies our focus on decision problems.

Example: to find maximum cardinality independent set.

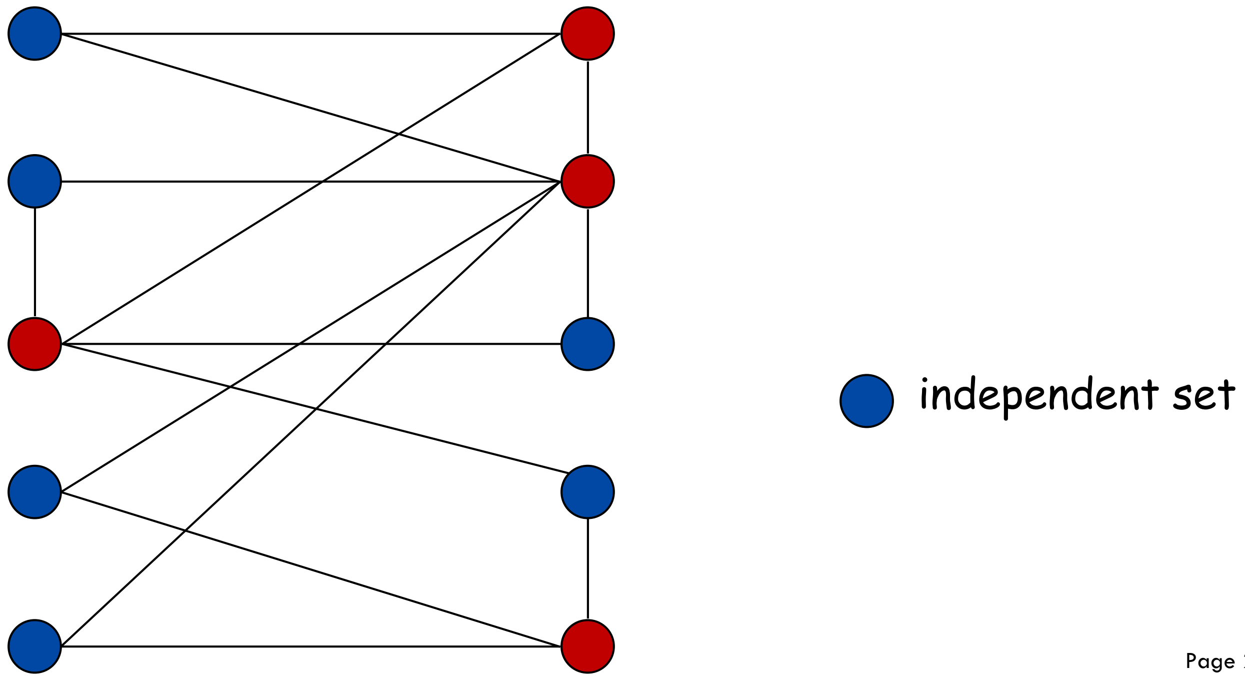
- (Binary) search for cardinality k of maximum independent set.

Independent Set

INDEPENDENT-SET: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?

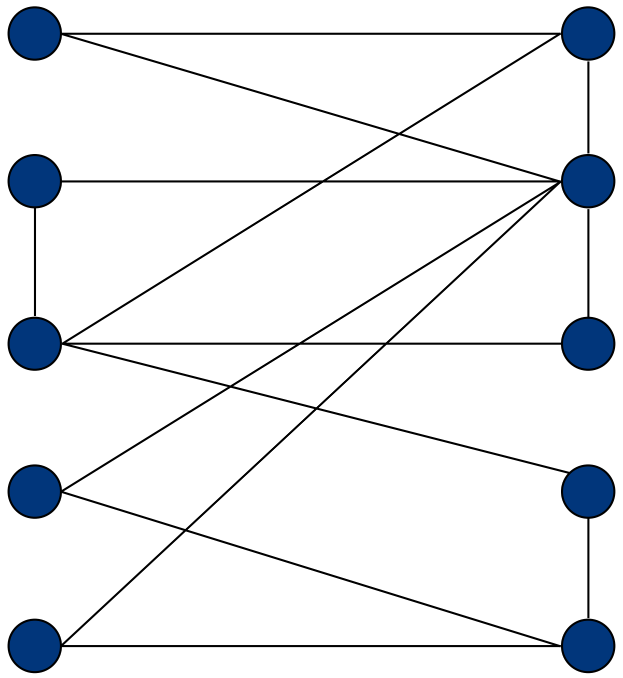
Ex. Is there an independent set of size ≥ 6? Yes.

Ex. Is there an independent set of size ≥ 7? No.



Vertex Cover

VERTEX-COVER: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?

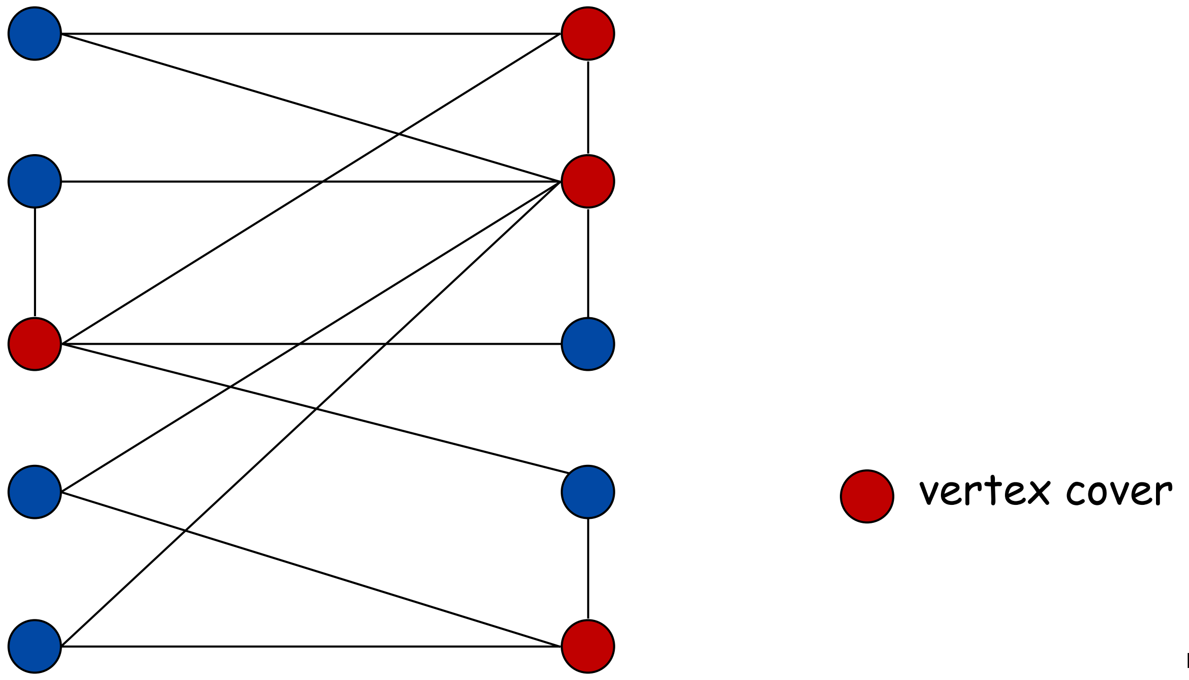


Vertex Cover

VERTEX-COVER: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?

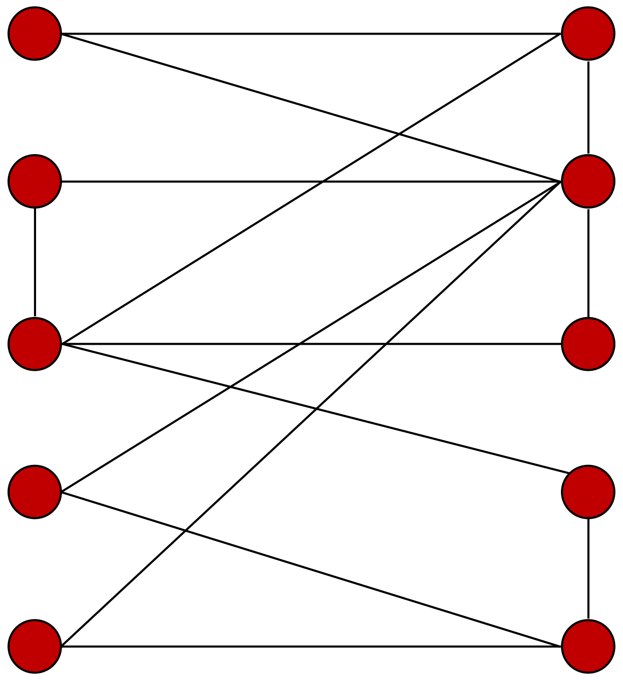
Ex. Is there a vertex cover of size ≤ 4? Yes.

Ex. Is there a vertex cover of size ≤ 3? No.



Vertex Cover and Independent Set

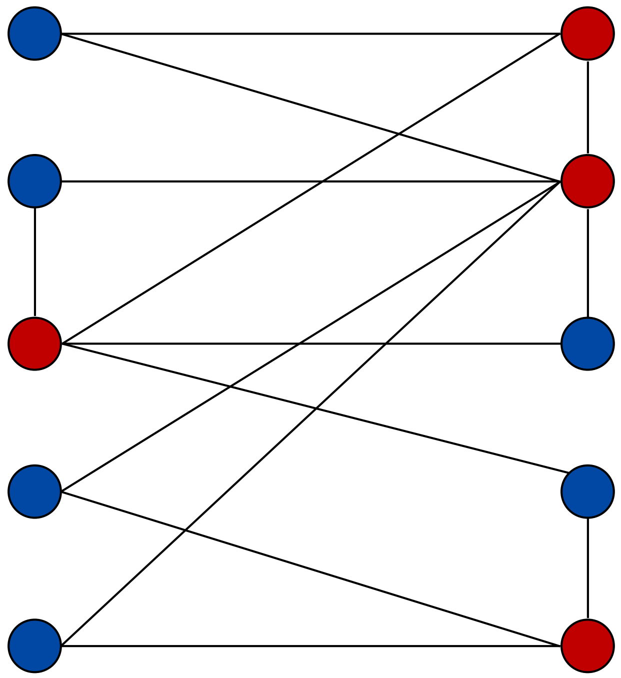
Theorem: VERTEX-COVER $\equiv_P$ INDEPENDENT-SET. (VC $\leq_P$ IS and IS $\leq_P$ VC)

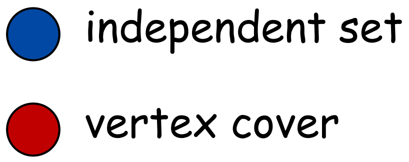


Vertex Cover and Independent Set

Theorem: VERTEX-COVER $\equiv_P$ INDEPENDENT-SET. (VC $\leq_P$ IS and IS $\leq_P$ VC)

Proof: We show S is an independent set iff V\S is a vertex cover.

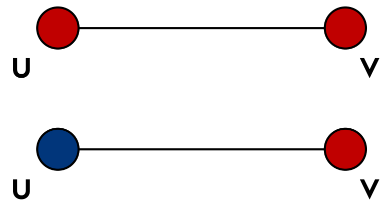




Vertex Cover and Independent Set

Theorem: VERTEX-COVER $\equiv_P$ INDEPENDENT-SET. (VC $\leq_P$ IS and IS $\leq_P$ VC)

Proof: We show S is an independent set <u>iff</u> V\S is a vertex cover.



$\Rightarrow$

- Let S be any independent set.
- Consider an arbitrary edge (u, v).
- S independent $\Rightarrow$ $u \notin S$ or $v \notin S$ $\Rightarrow$ $u \in V\setminus S$ or $v \in V\setminus S$.
- Thus, V\S covers (u, v) $\Rightarrow$ V\S vertex cover.

[VC $\leq_P$ IS]

$\Leftarrow$

- Let V\S be any vertex cover.
- Consider two nodes $u \in S$ and $v \in S$.
- Observe that $(u, v) \notin E$ since V\S is a vertex cover.
- Thus, no two nodes in S are joined by an edge $\Rightarrow$ S independent set. ▪

[IS $\leq_P$ VC]

Reduction from Special Case to General Case

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Set Cover

SET-COVER: Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k, does there exist a collection of k of these sets whose union is equal to U?

Sample application.

- m available pieces of software.
- Set U of n capabilities that we would like our system to have.
- The ith piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all n capabilities using fewest pieces of software.

Example:

$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$k = 2$

$S_1 = \{3, 7\}$	$S_4 = \{2, 4\}$
$S_2 = \{3, 4, 5, 6\}$	$S_5 = \{5\}$
$S_3 = \{1\}$	$S_6 = \{1, 2, 6, 7\}$

Set Cover

SET-COVER: Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k, does there exist a collection of k of these sets whose union is equal to U?

Sample application.

- m available pieces of software.
- Set U of n capabilities that we would like our system to have.
- The ith piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all n capabilities using fewest pieces of software.

Example:

$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$k = 2$

$S_1 = \{3, 7\}$	$S_4 = \{2, 4\}$
$\boxed{S_2 = \{3, 4, 5, 6\}}$	$S_5 = \{5\}$
$S_3 = \{1\}$	$\boxed{S_6 = \{1, 2, 6, 7\}}$

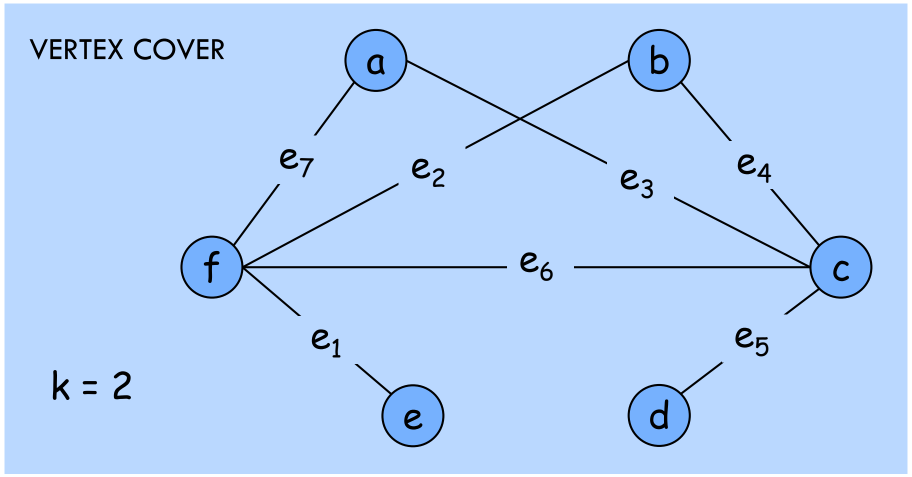
Vertex Cover Reduces to Set Cover

Theorem: VERTEX-COVER $\leq_P$ SET-COVER.

Proof: Given a VERTEX-COVER instance G = (V, E), k, we construct a set cover instance whose size equals the size of the vertex cover instance.

Construction.

- Create SET-COVER instance:
  - $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$
- Set-cover of size $\leq k$ iff vertex cover of size $\leq k$. ▪



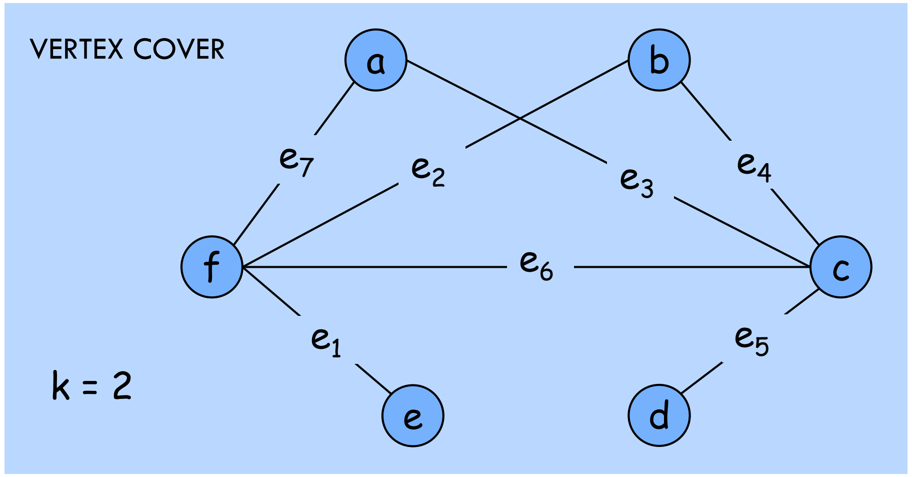
Vertex Cover Reduces to Set Cover

Theorem: VERTEX-COVER $\leq_P$ SET-COVER.

Proof: Given a VERTEX-COVER instance G = (V, E), k, we construct a set cover instance whose size equals the size of the vertex cover instance.

Construction.

- Create SET-COVER instance:
  - $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$
- Set-cover of size $\leq k$ iff vertex cover of size $\leq k$. ▪



SET COVER

$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$
$k = 2$

$S_a = \{3, 7\}$	$S_b = \{2, 4\}$
$S_c = \{3, 4, 5, 6\}$	$S_d = \{5\}$
$S_e = \{1\}$	$S_f = \{1, 2, 6, 7\}$

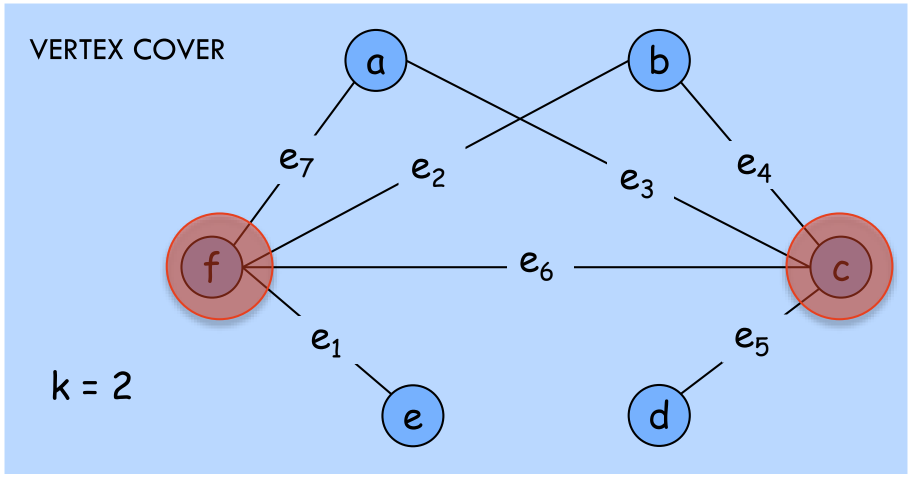
Vertex Cover Reduces to Set Cover

Theorem: VERTEX-COVER $\leq_P$ SET-COVER.

Proof: Given a VERTEX-COVER instance G = (V, E), k, we construct a set cover instance whose size equals the size of the vertex cover instance.

Construction.

- Create SET-COVER instance:
  - $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$
- Set-cover of size $\leq k$ iff vertex cover of size $\leq k$. ▪



SET COVER

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2

$S_a = \{3, 7\}$ $S_b = \{2, 4\}$
$\boxed{S_c = \{3, 4, 5, 6\}}$ $S_d = \{5\}$
$S_e = \{1\}$ $\boxed{S_f = \{1, 2, 6, 7\}}$

8.2 Reductions via "Gadgets"

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Satisfiability

Literal: A Boolean variable or its negation. x_i or $\overline{x_i}$

Clause: A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$

Conjunctive normal form (CNF): A propositional formula Φ that is the conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT: Given CNF formula Φ, does it have a satisfying truth assignment?

3-SAT: SAT where each clause contains exactly 3 literals.

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

Yes: x_1 = true, x_2 = true x_3 = false.

3 Satisfiability Reduces to Independent Set

Theorem: 3-SAT $\leq_P$ INDEPENDENT-SET.

INDEPENDENT-SET: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?

3 Satisfiability Reduces to Independent Set

Theorem: 3-SAT $\leq_P$ INDEPENDENT-SET.

Proof: Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT-SET that has an independent set of size k iff Φ is satisfiable.

3 Satisfiability Reduces to Independent Set

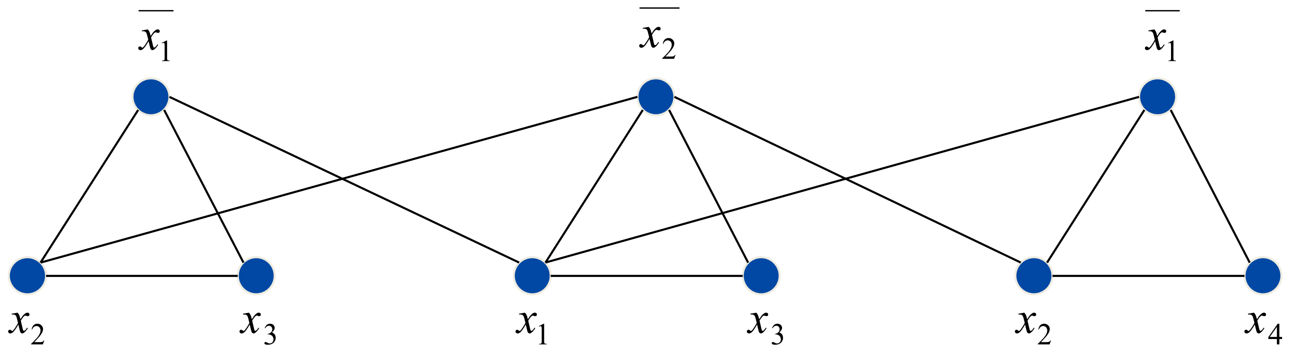
Theorem: 3-SAT $\leq_P$ INDEPENDENT-SET.

Proof: Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT-SET that has an independent set of size k iff Φ is satisfiable.

Construction.

- G contains 3 vertices for each of the k clauses, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.

(G, k = 3)



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

3 Satisfiability Reduces to Independent Set

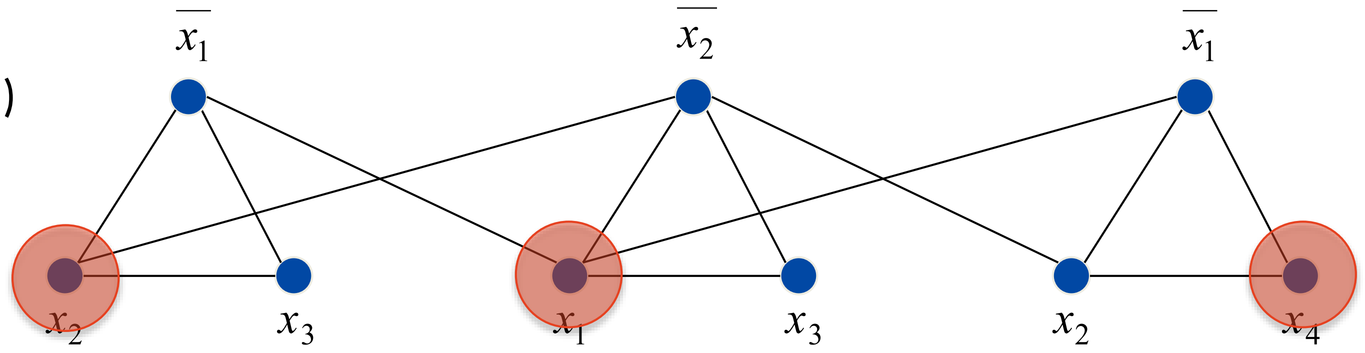
Claim. G contains independent set of size k = $|\Phi|$ iff Φ is satisfiable.

Proof: $\Rightarrow$ Let S be independent set of size k.

- S must contain exactly one vertex in each triangle.
- Set these literals to true.
- Truth assignment is consistent and all clauses are satisfied.

Proof: $\Leftarrow$ Given satisfying assignment, select one true literal from each triangle. This is an independent set of size k. ▪

(G, k = 3)



$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

3 Satisfiability Reduces to Independent Set

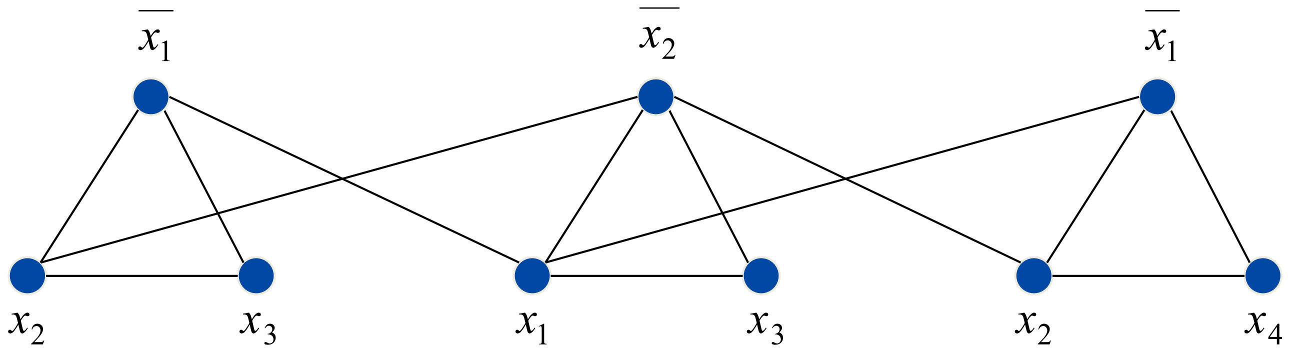
Claim. G contains independent set of size k = $|\Phi|$ iff Φ is satisfiable.

Proof: $\Rightarrow$ Let S be independent set of size k.

- S must contain exactly one vertex in each triangle.
- Set these literals to true.
- Truth assignment is consistent and all clauses are satisfied.

Proof: $\Leftarrow$ Given satisfying assignment, select one true literal from each triangle. This is an independent set of size k. ▪

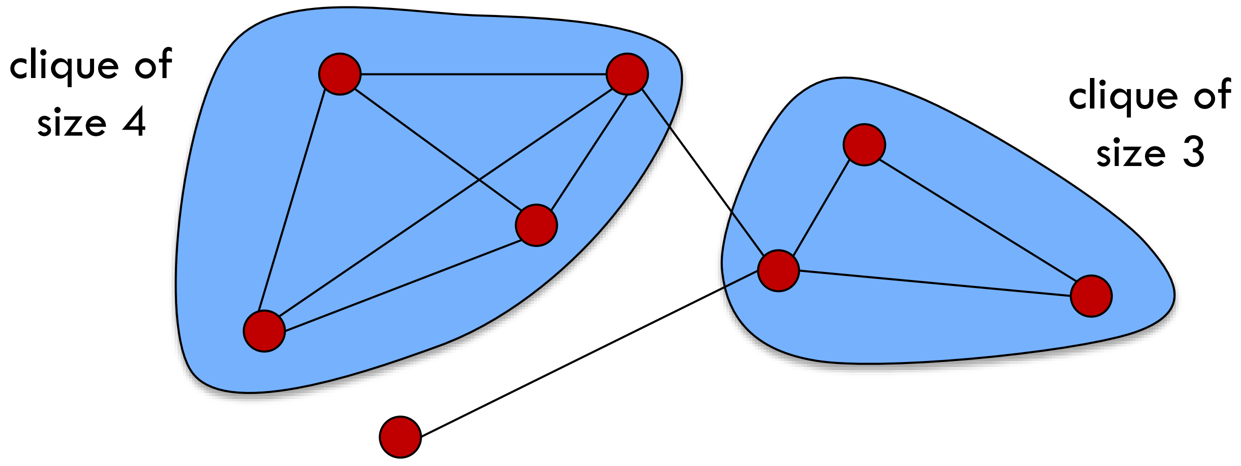
(G, k = 3)



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

Clique

A clique of a graph G is a complete subgraph of G.



CLIQUE: Given a graph G=(V,E) and an integer k, does G=(V,E) contain a clique of size *k*?

3 Satisfiability Reduces to Clique

Theorem: 3-SAT $\leq_P$ CLIQUE.

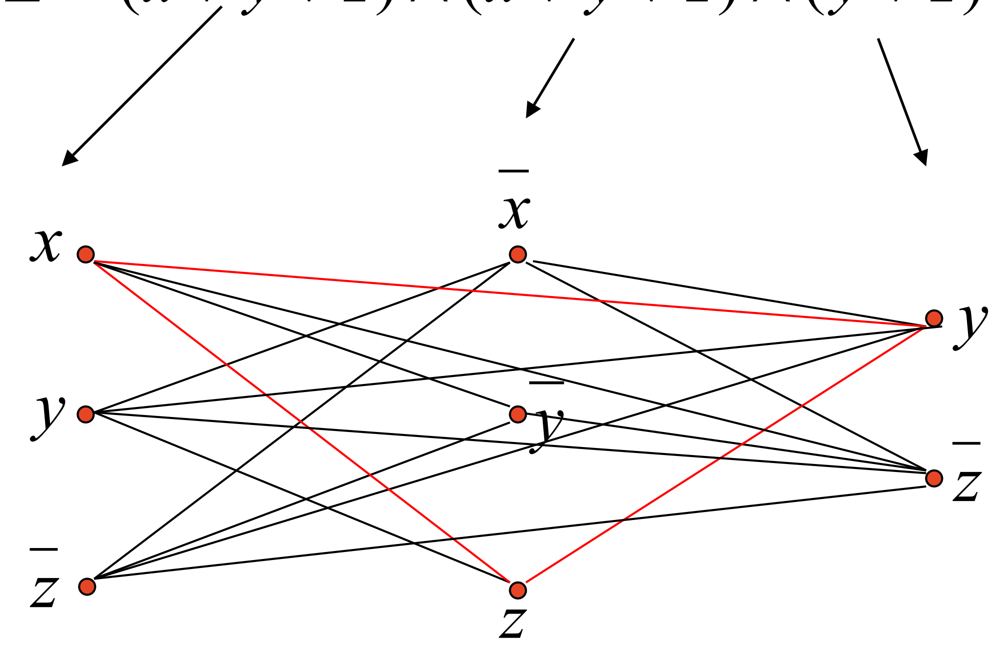
Idea:

- Make "column" for each of the k clauses.
- No edge within a column.
- All other edges present except between x and $\overline{x}$.

3 Satisfiability Reduces to Clique

Example: $E = (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (y \vee \bar{z})$

G =



Observation: G has a *k*-clique, if and only if E is satisfiable.

Review

Basic reduction strategies.

- Simple equivalence: INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
- Special case to general case: VERTEX-COVER $\leq_P$ SET-COVER.
- Encoding with gadgets: 3-SAT $\leq_P$ INDEPENDENT-SET.
  3-SAT $\leq_P$ CLIQUE

Transitivity. If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.
Proof idea: Compose the two algorithms.

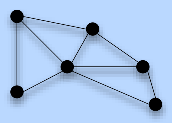
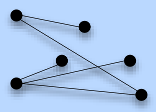
Example: 3-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ SET-COVER.

8.3 Definition of NP

- Class P
- Class NP
- Class NP-complete

Definition of the class P

Class P: Decision problems for which there is a poly-time algorithm.

Problem	Description	Algorithm	Yes	No
MULTIPLE	Is x a multiple of y?	Grade school division	51, 17	51, 16
RELPRIME	Are x and y relatively prime?	Euclid (300 BCE)	34, 39	34, 51
PRIMES	Is x prime?	AKS (2002)	53	51
RNA secondary structure	Is there an RNA secondary structure of weight at most 3?	Dynamic programming	accgguagu	aaaaggggg
MST	Is there a MST of weight 5?	Prim's		

Definition of the class NP

Certification algorithm intuition.

- Certifier views things from "managerial" viewpoint.
- Certifier does not solve the problem by its own;
  rather, it checks if a proposed proof t is a valid solution.

Definition: Algorithm C(s,t) is a certifier for problem X if for every input instance s and proposed proof t, C(s, t) = `yes'. If and only if t is a valid solution to X.

"certificate" or "witness"

Class NP: Decision problems for which there exists a poly-time certifier.

Certifiers and Certificates: Set Cover

SET-COVER: Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k, does there exist a collection of k of these sets whose union is equal to U?

Certificate: $t=\{S_{i1}, S_{i2}, \ldots, S_{ik}\}$

Certifier:

```
boolean C(s,t) {
   if the number of sets > k
      return false
   else if (∀v∈U: v ∈ a set in t)
      return true
   else
      return false}
```

Example: $U=\{1,2,3,4\}$, $S_1=\{1\}$, $S_2=\{2,3\}$, $S_3=\{1,4\}$ and $k=2$

certificate $t=\{S_2,S_3\}$

Conclusion: SET-COVER is in NP.

Certifiers and Certificates: 3-Satisfiability

SAT: Given a CNF formula Φ, 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.

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

instance s

x_1, x_2, x_4=true, x_3=false

certificate t

Conclusion: SAT is in NP.

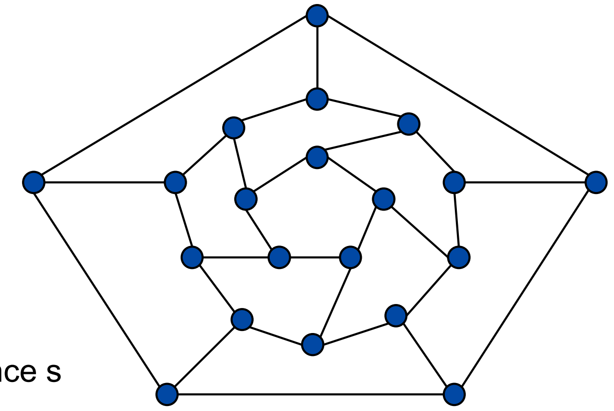
Certifiers and Certificates: Hamiltonian Cycle

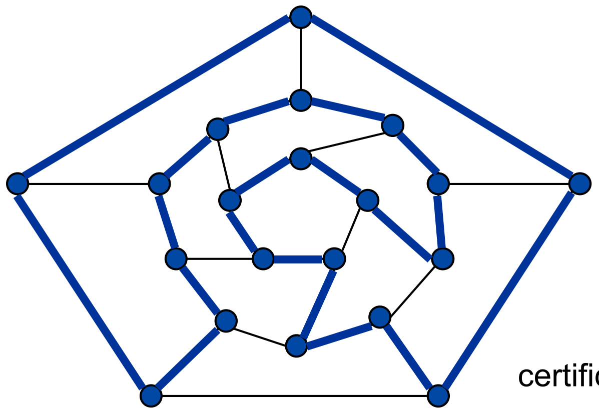
HAM-CYCLE: Given an undirected graph G = (V, E), does there exist a simple cycle C that visits every node?

Certificate: t = a permutation of the n nodes.

Certifier: Check that the permutation contains each node in V exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.

Conclusion: HAM-CYCLE is in NP.


instance s


certificate t

P, NP

P: Decision problems for which there is a poly-time algorithm.
NP: Decision problems for which there is a poly-time certifier.

Claim: P $\subseteq$ NP.

P, NP, EXP

P: Decision problems for which there is a poly-time algorithm.
NP: Decision problems for which there is a poly-time certifier.
EXP: Decision problems for which there is an exponential-time algorithm.

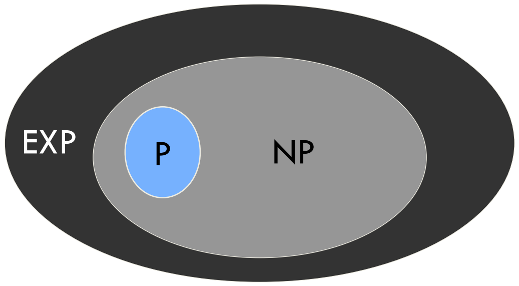
Claim: P $\subseteq$ NP.

Claim: NP $\subseteq$ EXP.

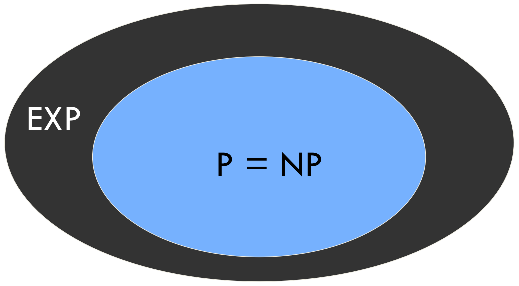
The Main Question: P Versus NP

Is P = NP? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

- Is the decision problem as easy as the certification problem?
- One of the seven Millennium Prize problems: $1 million prize if solved.



If P ≠ NP



If P = NP

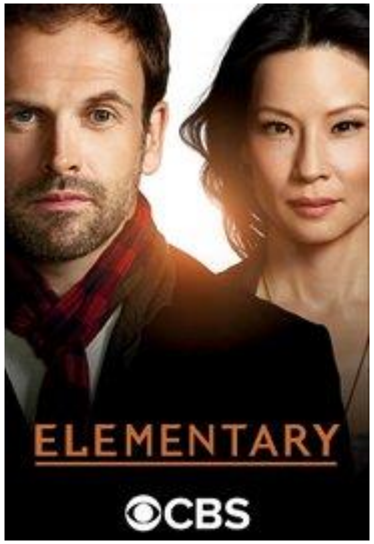
would break RSA cryptography

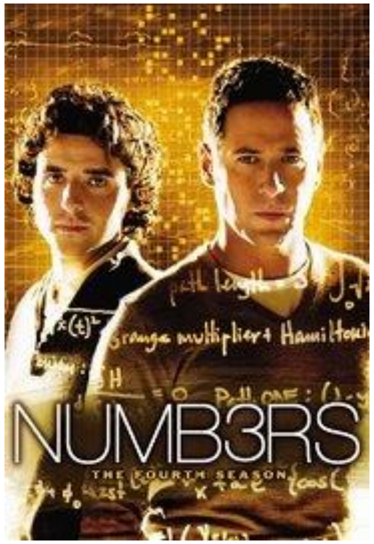
P=NP: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, …

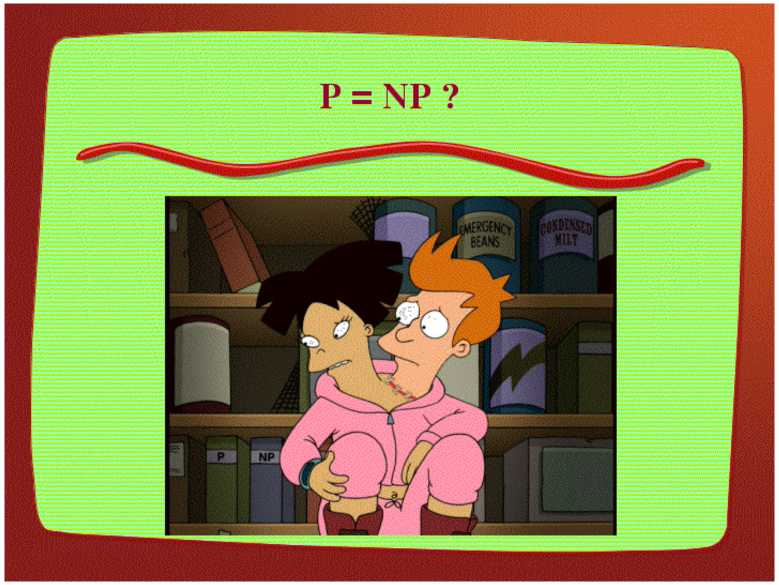
P ≠ NP: No efficient algorithms possible for 3-COLOR, TSP, SAT, …

Consensus opinion on P = NP? Probably no.

P=NP
$e^{\pi i} = -1$


ELEMENTARY
CBS


NUMB3RS
THE FOURTH SEASON


P = NP ?
EMERGENCY BEANS
CONDENSED MILK
P
NP

8.4 NP-Completeness

Polynomial Transformation

Definition: Problem X polynomial reduces (Cook) to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

Definition: Problem X polynomial transforms (Karp) to problem Y if given any input x to X, we can construct an input y such that x is a `yes` instance of X iff y is a `yes` instance of Y.

Note. Polynomial transformation is polynomial reduction with just one call to oracle for Y, exactly at the end of the algorithm for X. Almost all previous reductions were of this form.

Open question. Are these two concepts the same?

$\uparrow$

we abuse notation $\leq_p$ and blur distinction

Class NP-Complete

NP-complete: A problem Y in NP with the property that for every problem X in NP, $X \leq_p Y$.

Theorem: Suppose Y is an NP-complete problem. Then Y is solvable in poly-time iff P = NP.

Proof:

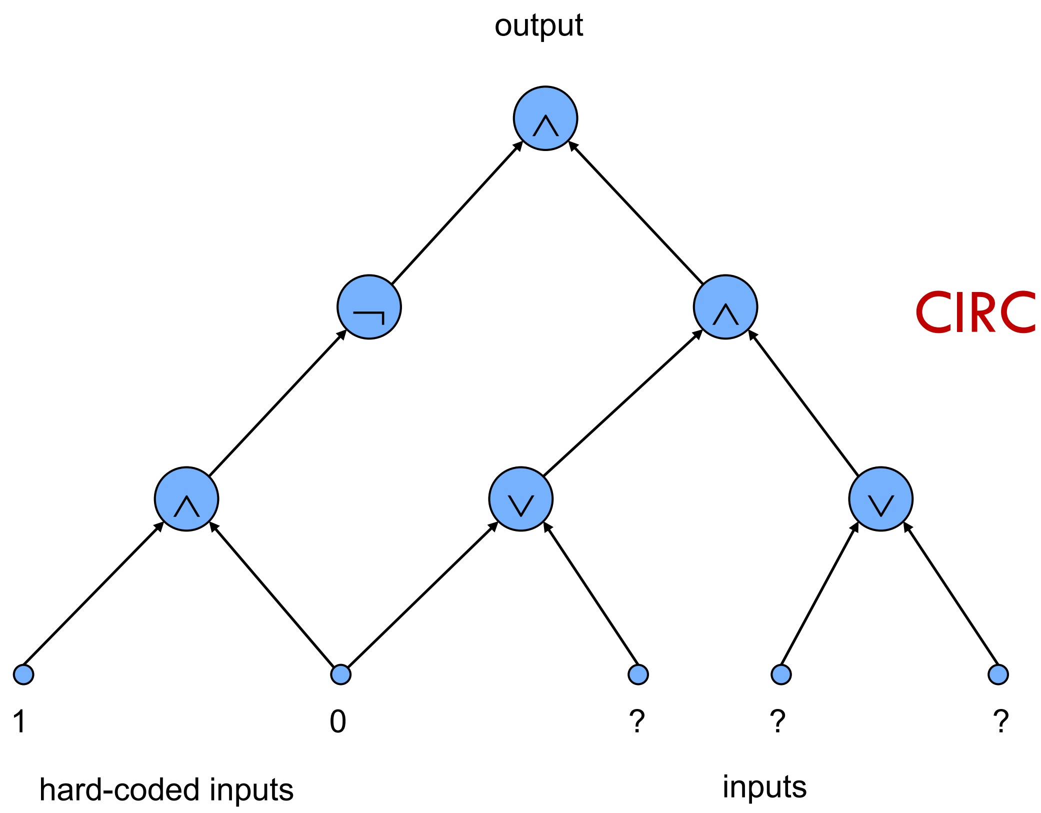
$\Leftarrow$ If P = NP then Y can be solved in poly-time since Y is in NP=P.

$\Rightarrow$ Suppose Y can be solved in poly-time.

- Let X be any problem in NP. Since $X \leq_p Y$, we can solve X in poly-time. This implies NP $\subseteq$ P.
- We already know P $\subseteq$ NP. Thus P = NP. ▪

Circuit Satisfiability

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?



CIRCUIT-SAT $\in$ NP

Yes: 1 0 1

The "First" NP-Complete Problem

Theorem: CIRCUIT-SAT is NP-complete. [Cook 1971, Levin 1973]

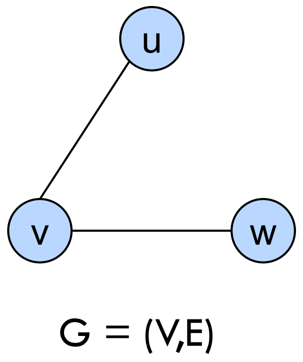
Proof: (main idea)

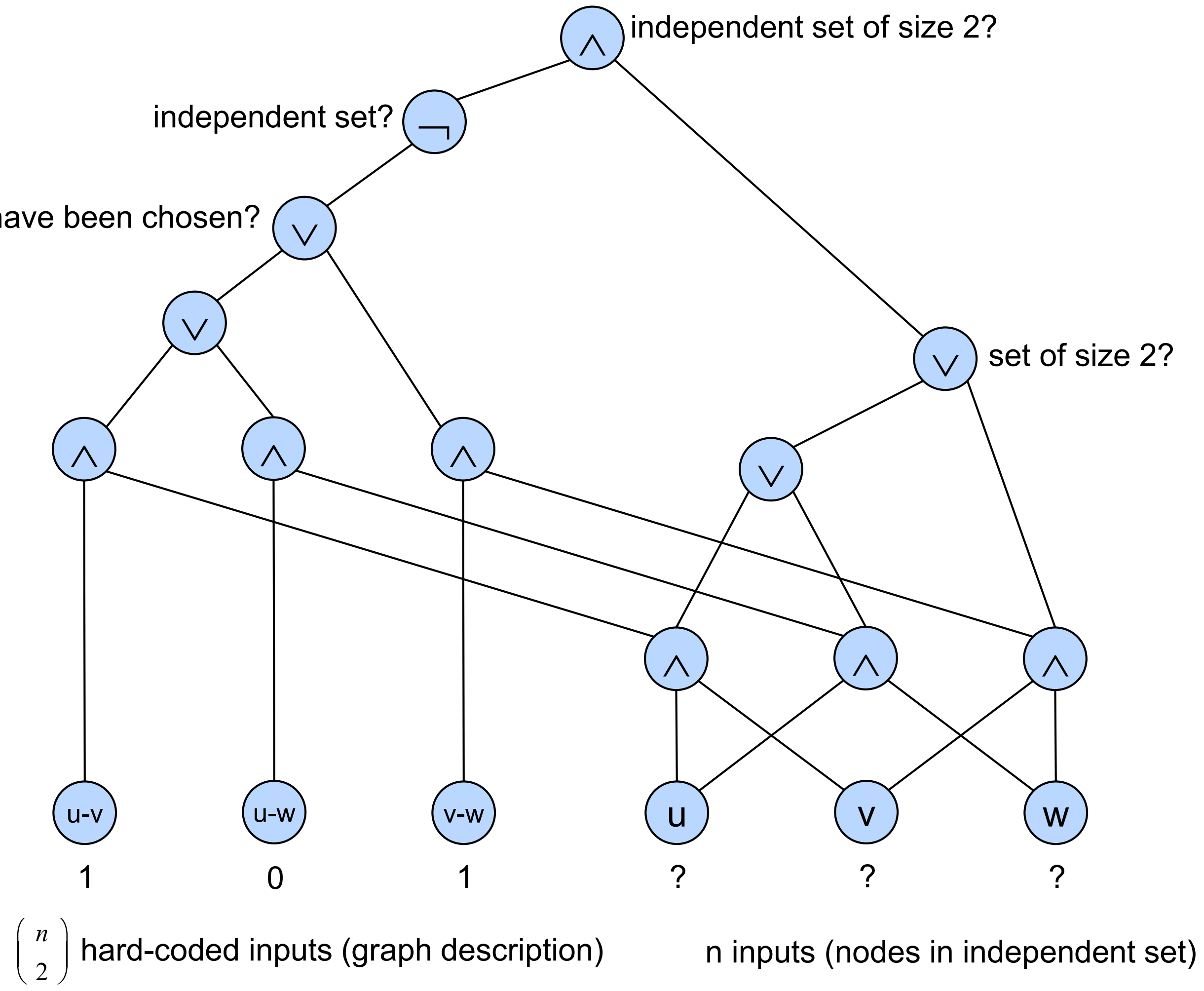
- Any algorithm that takes a fixed number of bits n as input and produces a yes/no answer can be represented by such a circuit. Moreover, if algorithm takes poly-time, then circuit is of poly-size.

Proof not part of the course.

Example

Construction below creates a circuit K whose inputs can be set so that K outputs true iff graph G has an independent set of size 2.





independent set of size 2?
independent set?
both endpoints of some edge have been chosen?
set of size 2?
u-v
u-w
v-w
u
v
w
1
0
1
?
?
?
$\binom{n}{2}$ hard-coded inputs (graph description)
n inputs (nodes in independent set)

Establishing NP-Completeness

Remark: Once we establish first "natural" NP-complete problem, others fall like dominoes.

Recipe to establish NP-completeness of problem Y.

- Step 1. Show that Y is in NP.
- Step 2. Choose an NP-complete problem X.
- Step 3. Prove that $X \le_p Y$.

Justification: If X is an NP-complete problem, and Y is a problem in NP with the property that $X \le_P Y$ then Y is NP-complete.

Proof: Let W be any problem in NP. Then $W \underset{\substack{\uparrow \\ \text{by definition of} \\ \text{NP-complete}}}{\le_P} X \underset{\substack{\uparrow \\ \text{by assumption}}}{\le_P} Y$.

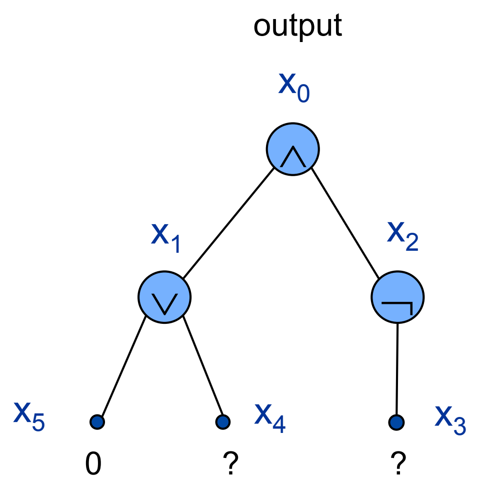
- By transitivity, $W \le_P Y$.
- Hence Y is NP-complete. ▪

3-SAT is NP-Complete

Theorem. 3-SAT is NP-complete.

Proof: Suffices to show that CIRCUIT-SAT $\leq_P$ 3-SAT since 3-SAT is in NP.

- Let K be any circuit.
- Create a 3-SAT variable x_i for each circuit element i.
- Make circuit compute correct values at each node:
  - $x_2 = \neg x_3 \Rightarrow$ add 2 clauses: $x_2 \vee x_3$, $\overline{x_2} \vee \overline{x_3}$
  - $x_1 = x_4 \vee x_5 \Rightarrow$ add 3 clauses: $x_1 \vee \overline{x_4}$, $x_1 \vee \overline{x_5}$, $\overline{x_1} \vee x_4 \vee x_5$
  - $x_0 = x_1 \wedge x_2 \Rightarrow$ add 3 clauses: $\overline{x_0} \vee x_1$, $\overline{x_0} \vee x_2$, $x_0 \vee \overline{x_1} \vee \overline{x_2}$
- Hard-coded input values and output value.
  - $x_5 = 0 \Rightarrow$ add 1 clause: $\overline{x_5}$
  - $x_0 = 1 \Rightarrow$ add 1 clause: x_0
- Final step: turn clauses of length < 3 into clauses of length exactly 3. ▪



Using transitivity

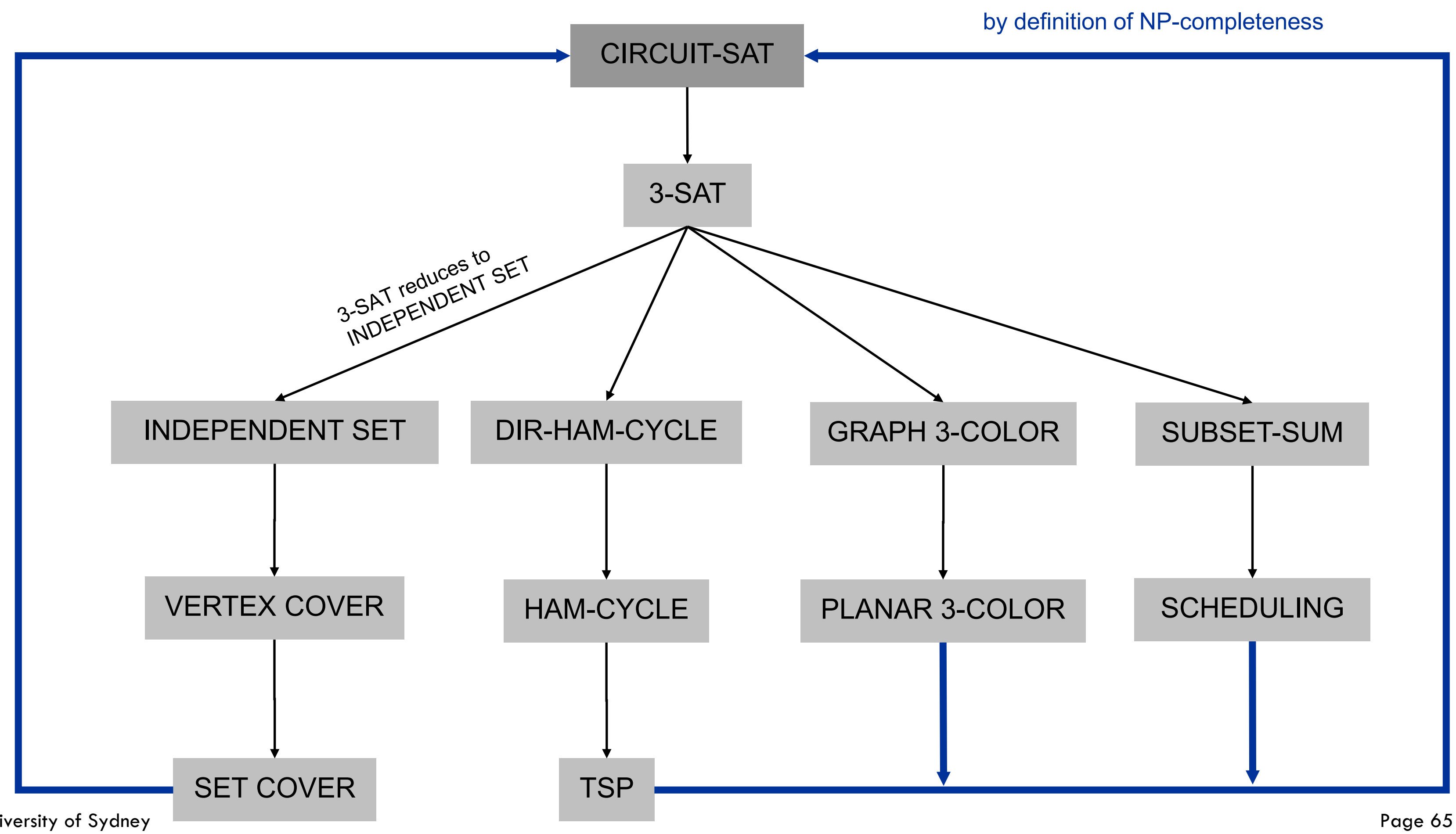
- 3-SAT is NP-complete
- 3-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ SET-COVER

Corollary:

INDEPENDENT-SET, VERTEX-COVER and SET-COVER are NP-complete.

NP-Completeness

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



Some NP-Complete Problems

Six basic genres of NP-complete problems and paradigmatic examples.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Extent and Impact of NP-Completeness

- Extent of NP-completeness. [Papadimitriou 1995]
  - Prime intellectual export of CS to other disciplines.
  - 6,000 citations per year (title, abstract, keywords).
    - more than "compiler", "operating system", "database"
  - Broad applicability and classification power.

More Hard Computational Problems

- Aerospace engineering: optimal mesh partitioning for finite elements.
- Biology: protein folding.
- Chemical engineering: heat exchanger network synthesis.
- Civil engineering: equilibrium of urban traffic flow.
- Economics: computation of arbitrage in financial markets with friction.
- Electrical engineering: VLSI layout.
- Environmental engineering: optimal placement of contaminant sensors.
- Financial engineering: find minimum risk portfolio of given return.
- Game theory: find Nash equilibrium that maximizes social welfare.
- Genomics: phylogeny reconstruction.
- Mechanical engineering: structure of turbulence in sheared flows.
- Medicine: reconstructing 3-D shape from biplane angiocardiogram.
- Operations research: optimal resource allocation.
- Physics: partition function of 3-D Ising model in statistical mechanics.
- Politics: Shapley-Shubik voting power.
- Pop culture: Minesweeper consistency.
- Statistics: optimal experimental design.

Games

The Eternity II puzzle, is a puzzle competition which was released in 2007.

A $2 million prize was offered for the first complete solution.

The Eternity II puzzle is an edge-matching puzzle which involves placing 256 square puzzle pieces into a 16 by 16 grid, constrained by the requirement to match adjacent edges.
The problem is NP-complete.

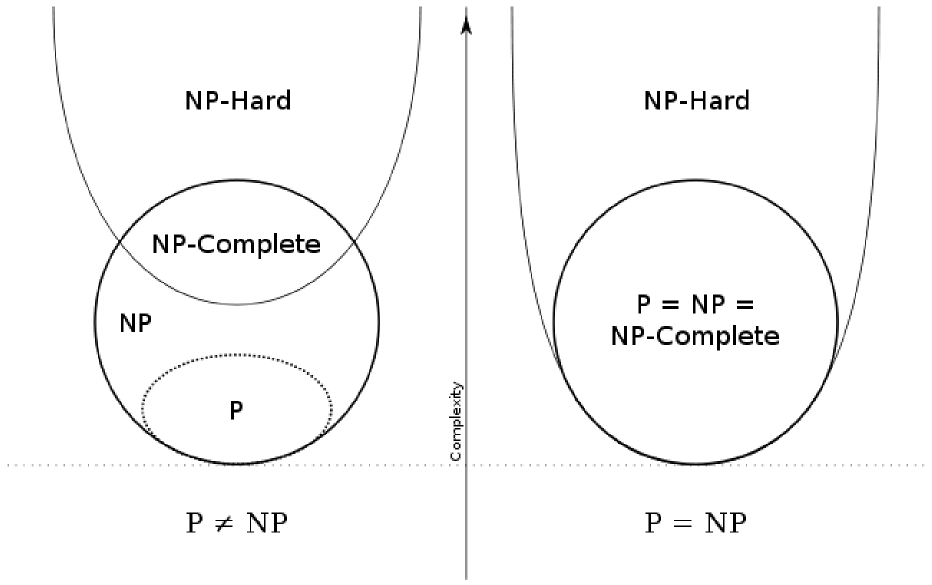
The competition ended at noon on 31 December 2010, with no solution being found.

Class NP-hard

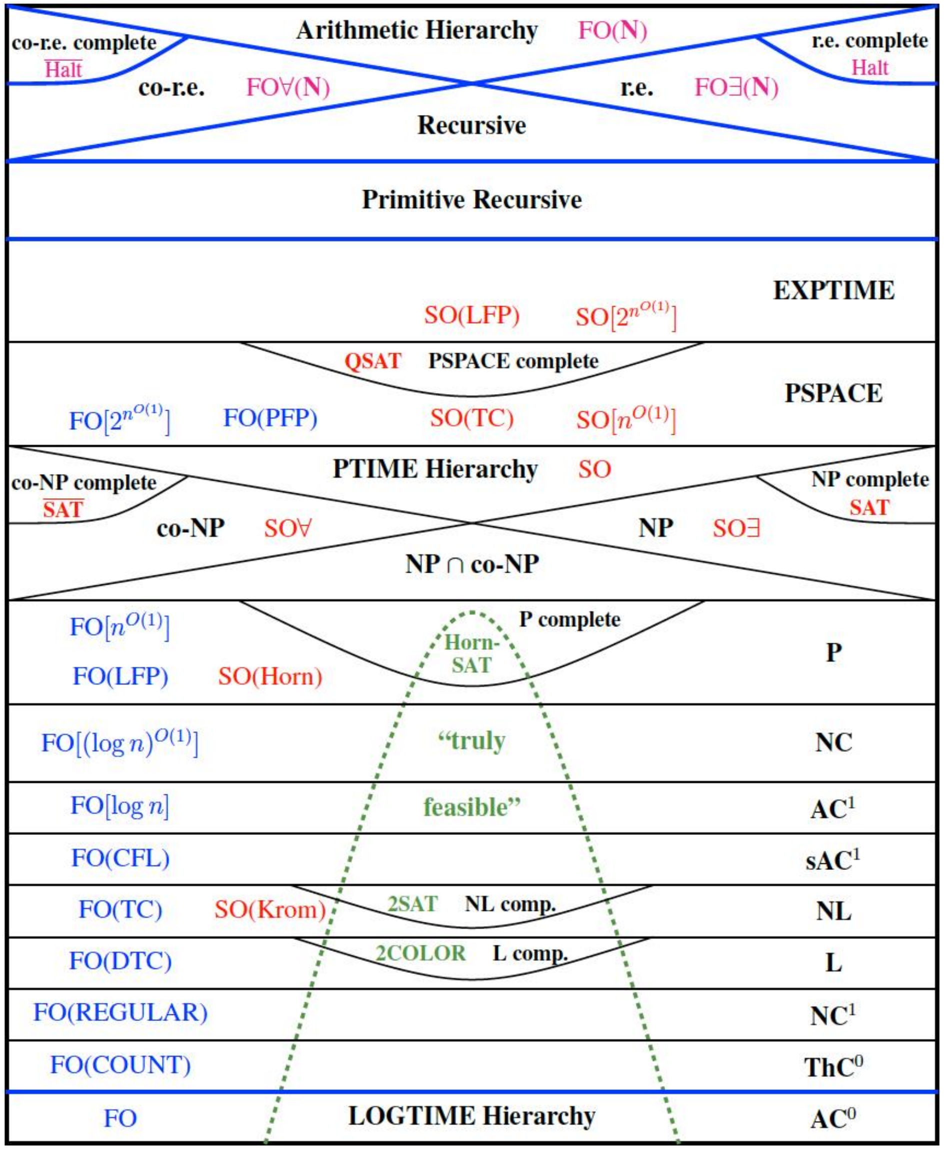
Class NP-complete: A problem in NP such that every problem in NP polynomial reduces to it.

Class NP-hard:
A decision problem such that every problem in NP reduces to it.

not necessarily in NP



Many classes?



8.5 Sequencing Problems

Six Basic genres

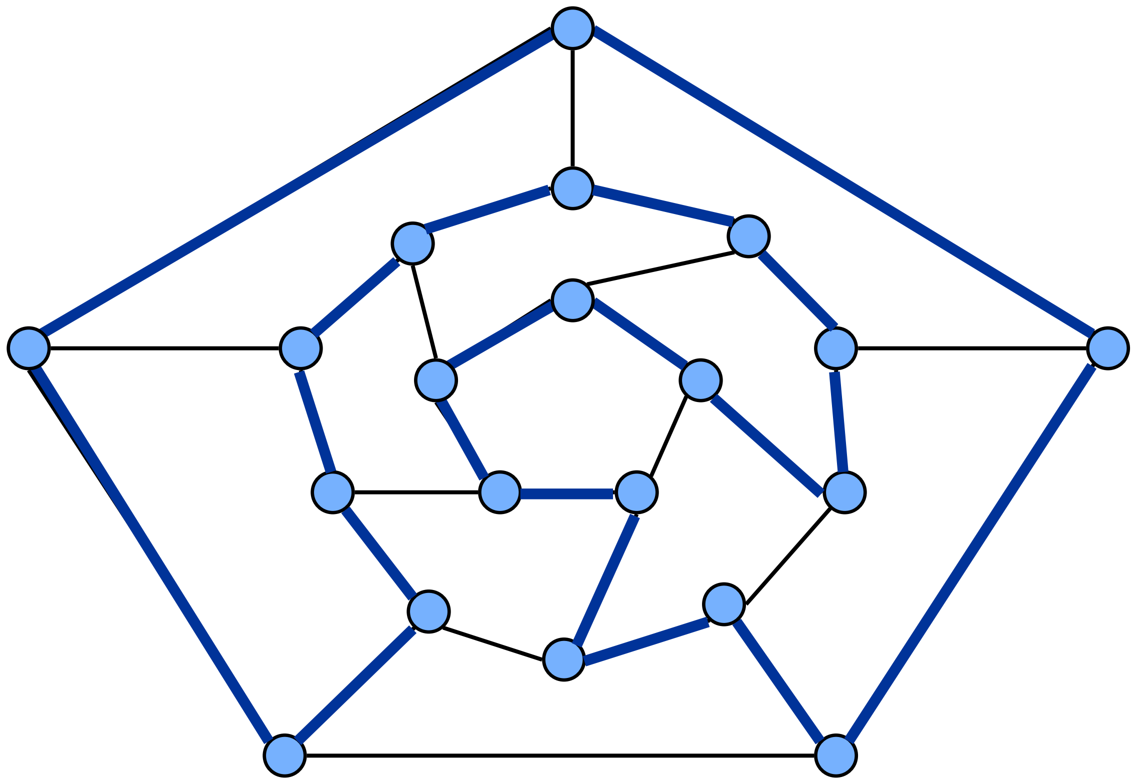
- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.

  3-SAT $\leq_P$ DIR HAMILTONIAN CYCLE $\leq_P$ HAMILTONIAN CYCLE $\leq_P$ TSP

- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Hamiltonian Cycle

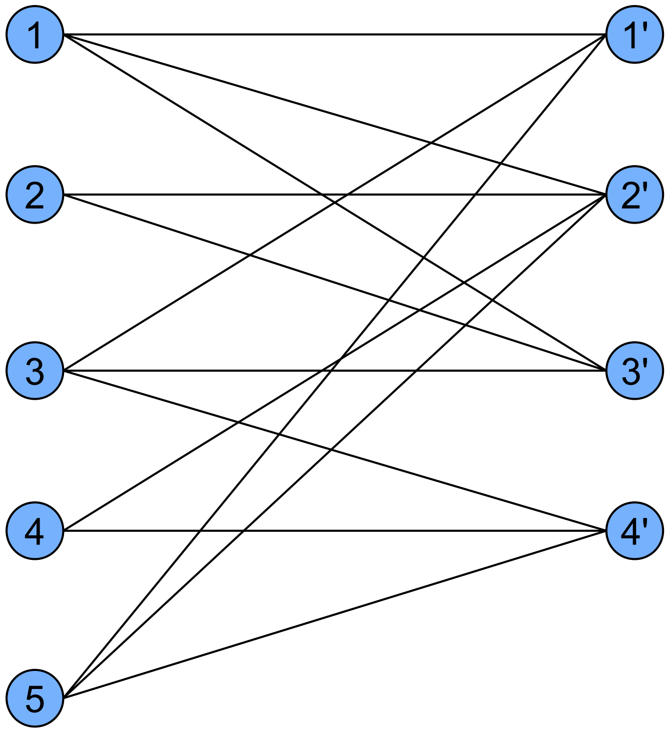
HAM-CYCLE: given an undirected graph $G = (V, E)$, does there exist a simple cycle Γ that contains every node in V.



YES: vertices and faces of a dodecahedron.

Hamiltonian Cycle

HAM-CYCLE: given an undirected graph G = (V, E), does there exist a simple cycle Γ that contains every node in V.



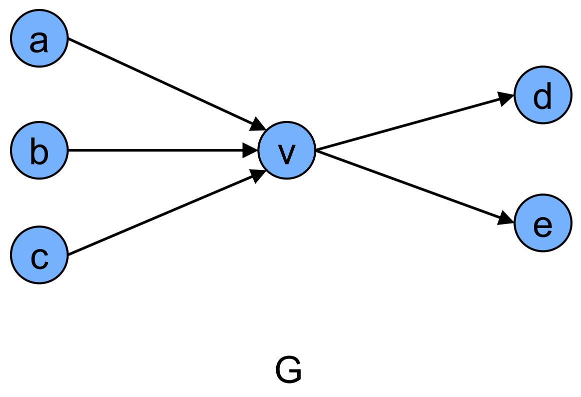
NO: bipartite graph with odd number of nodes.

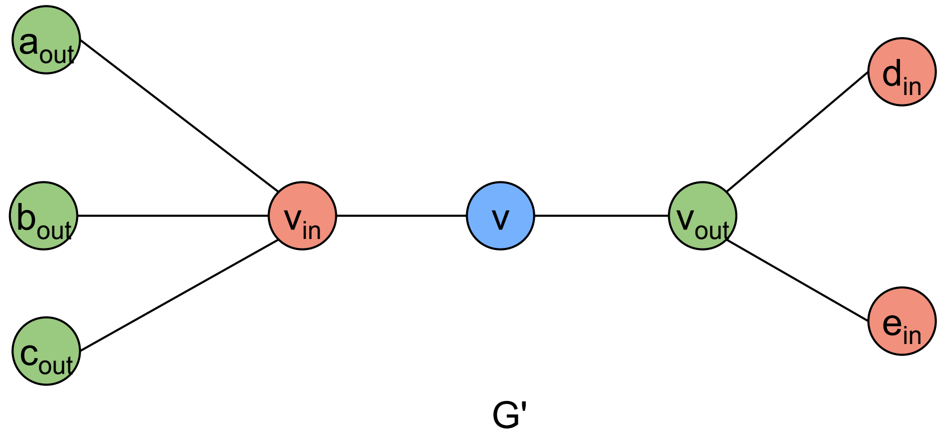
Directed Hamiltonian Cycle

DIR-HAM-CYCLE: Given a directed graph G = (V, E), does there exists a simple directed cycle Γ that contains every node in V?

Theorem: DIR-HAM-CYCLE $\leq_P$ HAM-CYCLE.

Proof idea: Given a directed graph G = (V, E), construct an undirected graph G' with 3n vertices.





Directed Hamiltonian Cycle

Claim: G has a Hamiltonian cycle iff G' does.

Proof:

$\Rightarrow$ – Suppose G has a directed Hamiltonian cycle Γ.

– Then G' has an undirected Hamiltonian cycle (same order).

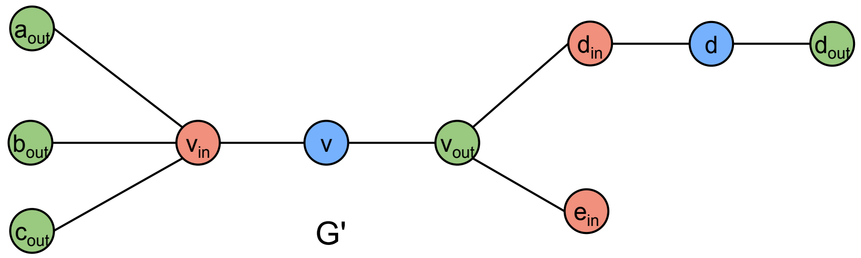
$\Leftarrow$ – Suppose G' has an undirected Hamiltonian cycle Γ'.

– Γ' must visit nodes in G' using one of two orders:

…, B, G, R, B, G, R, B, G, R, B, …

…, B, R, G, B, R, G, B, R, G, B, …

– Blue nodes in Γ' make up directed Hamiltonian cycle Γ in G, or reverse of one. ▪



3-SAT Reduces to Directed Hamiltonian Cycle

Theorem: 3-SAT $\leq_P$ DIR-HAM-CYCLE.

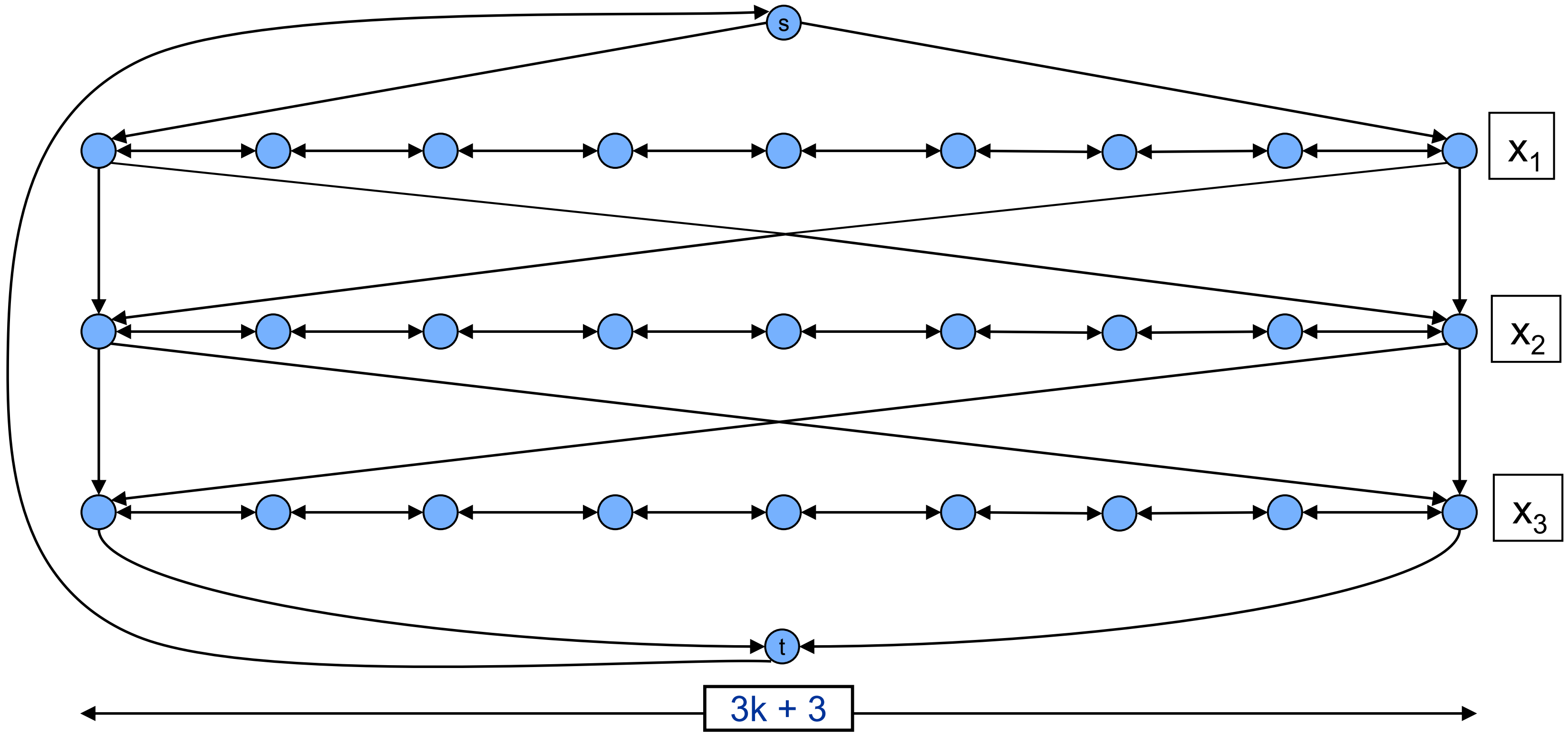
Proof: Given an instance Φ of 3-SAT, we construct an instance of DIR-HAM-CYCLE that has a Hamiltonian cycle iff Φ is satisfiable.

Construction. First, create graph that has 2^n Hamiltonian cycles which correspond in a natural way to 2^n possible truth assignments.

3-SAT Reduces to Directed Hamiltonian Cycle

Construction: Given a 3-SAT instance Φ with n variables x_i and k clauses.

- Construct G to have 2^n Hamiltonian cycles.
- Intuition: Traverse path i from left to right $\Leftrightarrow$ set variable $x_i = 1$.

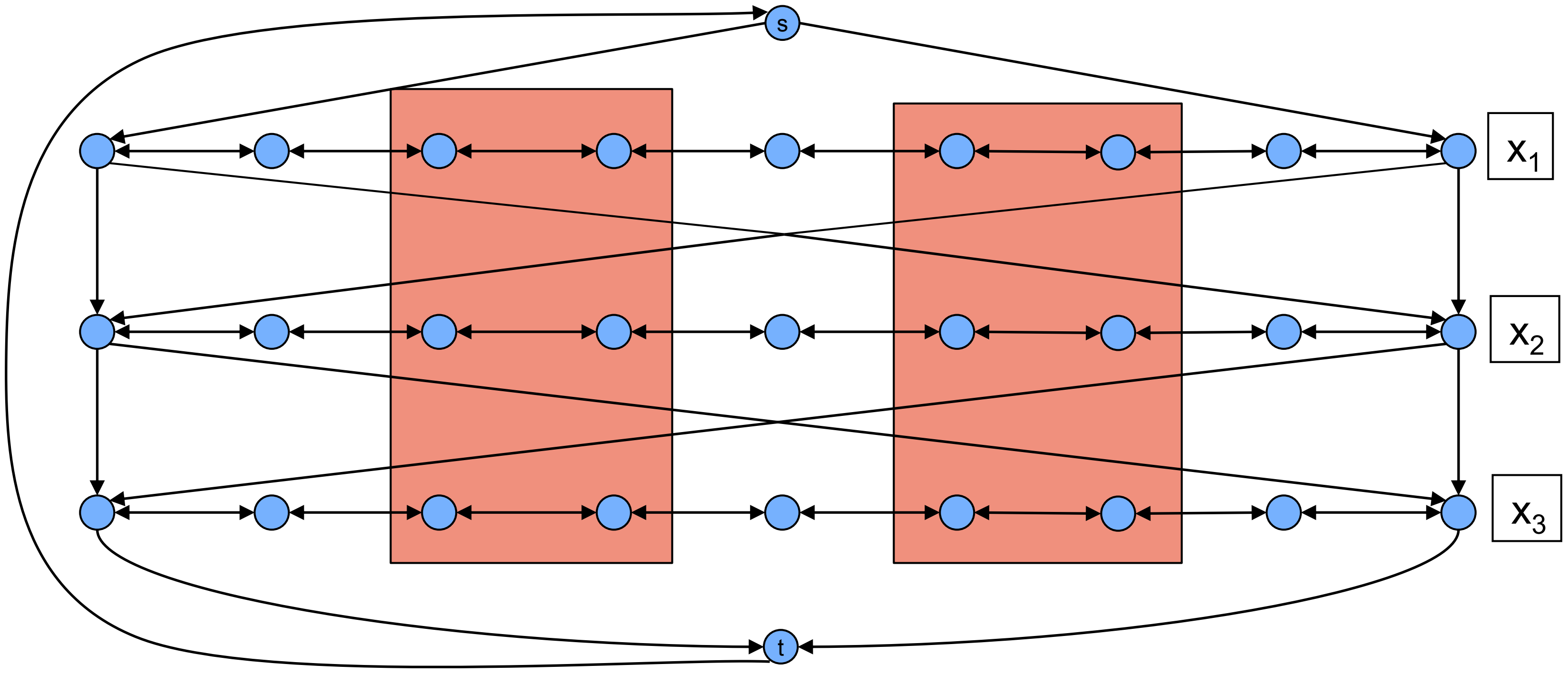


3-SAT Reduces to Directed Hamiltonian Cycle

- Construction. Given 3-SAT instance Φ with n variables x_i and k clauses.
  - For each clause: add a node and 6 edges.

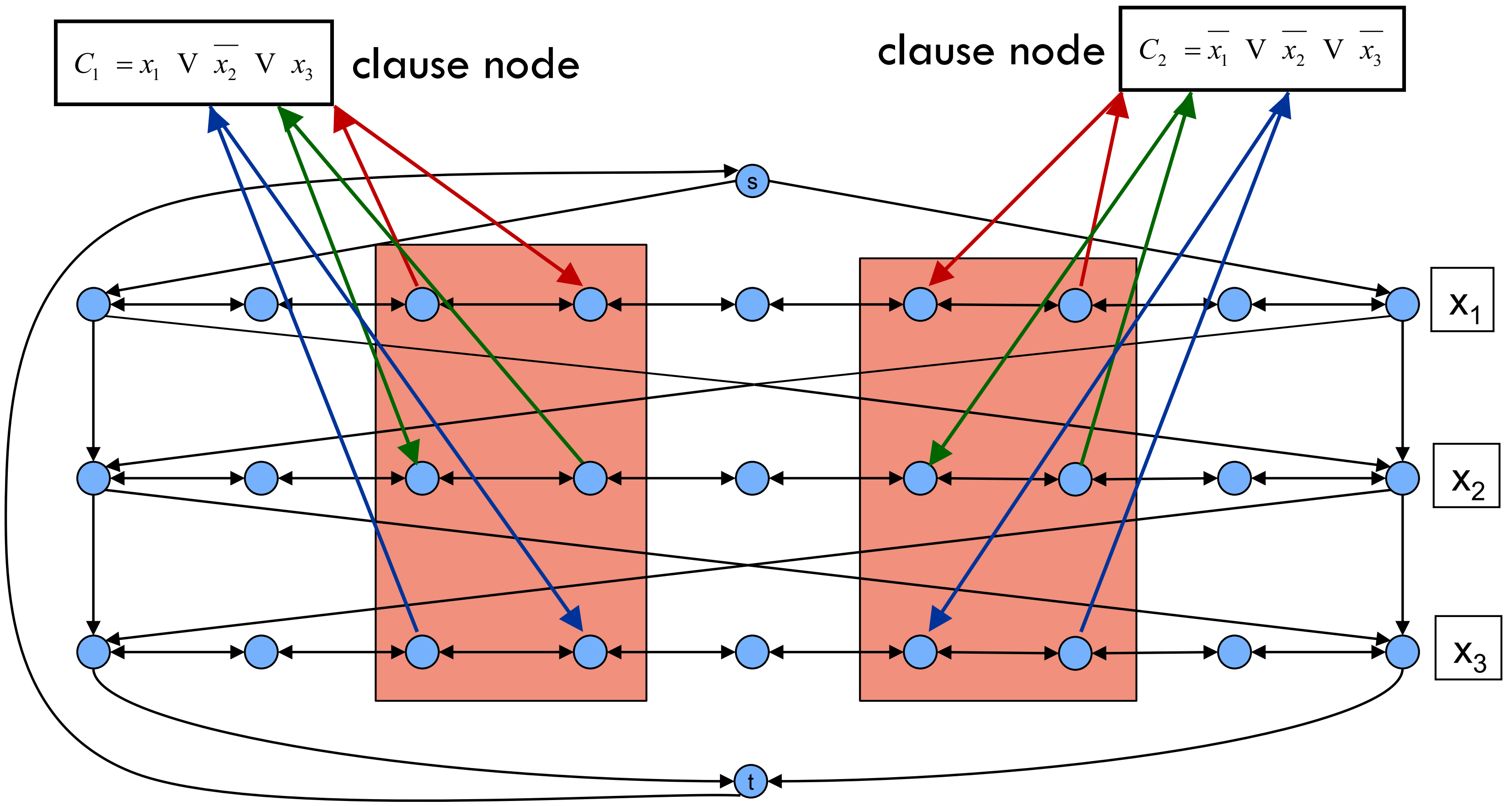
$\boxed{C_1 = x_1 \vee \overline{x_2} \vee x_3}$ clause node

clause node $\boxed{C_2 = \overline{x_1} \vee \overline{x_2} \vee \overline{x_3}}$



3-SAT Reduces to Directed Hamiltonian Cycle

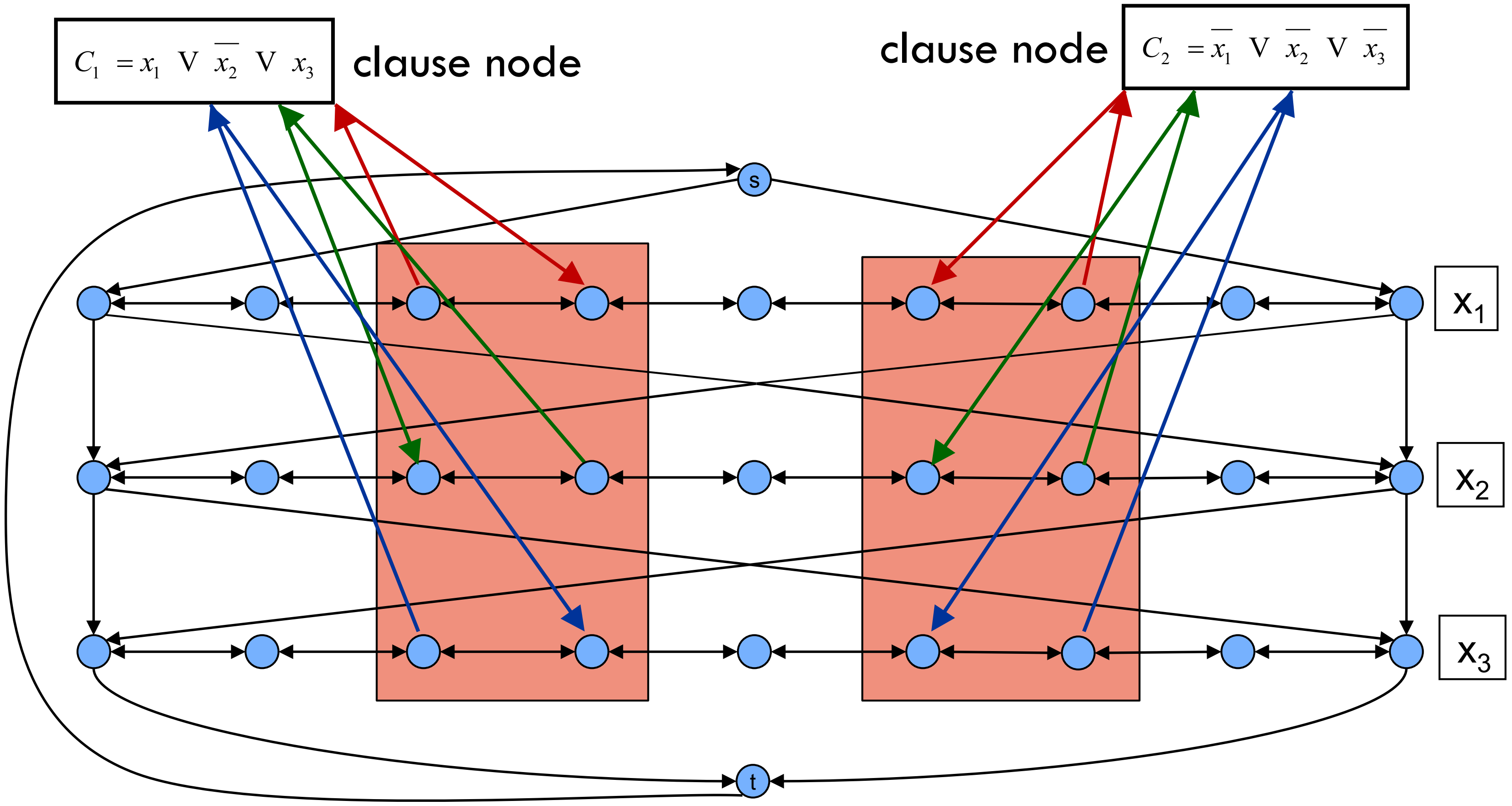
- Construction. Given 3-SAT instance Φ with n variables x_i and k clauses.
  - For each clause: add a node and 6 edges.



$C_1 = x_1 \vee \overline{x_2} \vee x_3$
clause node
clause node
$C_2 = \overline{x_1} \vee \overline{x_2} \vee \overline{x_3}$
s
t
x_1
x_2
x_3

3-SAT Reduces to Directed Hamiltonian Cycle

- Construction. Given 3-SAT instance Φ with n variables x_i and k clauses.
  - For each clause: add a node and 6 edges.



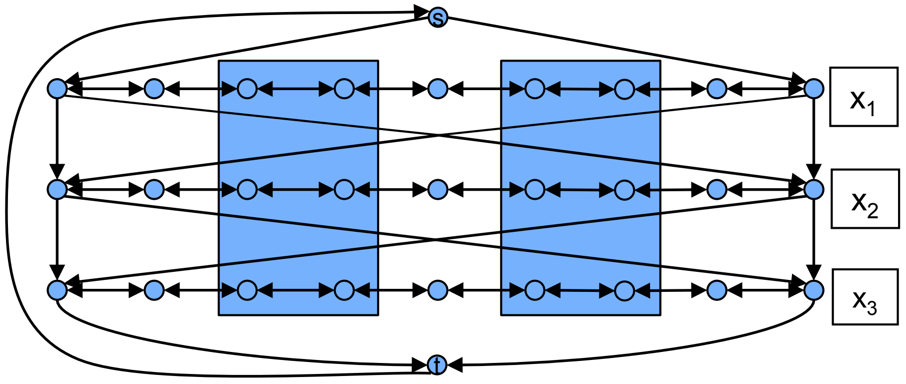
$C_1 = x_1 \lor \overline{x_2} \lor x_3$
clause node
clause node
$C_2 = \overline{x_1} \lor \overline{x_2} \lor \overline{x_3}$
s
t
x_1
x_2
x_3

3-SAT Reduces to Directed Hamiltonian Cycle

Claim: Φ is satisfiable iff G has a Hamiltonian cycle.

Proof: $\Rightarrow$

- Suppose 3-SAT instance has satisfying assignment x*.
- Then, define Hamiltonian cycle in G as follows:
  - if $x^*_i = 1$, traverse row i from left to right
  - if $x^*_i = 0$, traverse row i from right to left
  - for each clause C_j, there will be at least one row i in which we are going in "correct" direction to splice node C_j into tour



3-SAT Reduces to Directed Hamiltonian Cycle

Claim: Φ is satisfiable iff G has a Hamiltonian cycle.

Proof: $\Leftarrow$

- Suppose G has a Hamiltonian cycle Γ.
- If Γ enters clause node C_j, it must depart on mate edge.
  - thus, nodes immediately before and after C_j are connected by an edge e in G
  - removing C_j from cycle, and replacing it with edge e yields Hamiltonian cycle on $G \setminus \{ C_j \}$
- Continuing in this way, we are left with Hamiltonian cycle Γ' in $G - \{ C_1, C_2, \ldots, C_k \}$.
- Set $x^*_i = 1$ iff Γ' traverses row i left to right.
- Since Γ visits each clause node C_j, at least one of the paths is traversed in "correct" direction, and each clause is satisfied. ▪

Longest Path

SHORTEST-PATH: Given a digraph G = (V, E), does there exists a simple path of length at most k edges?

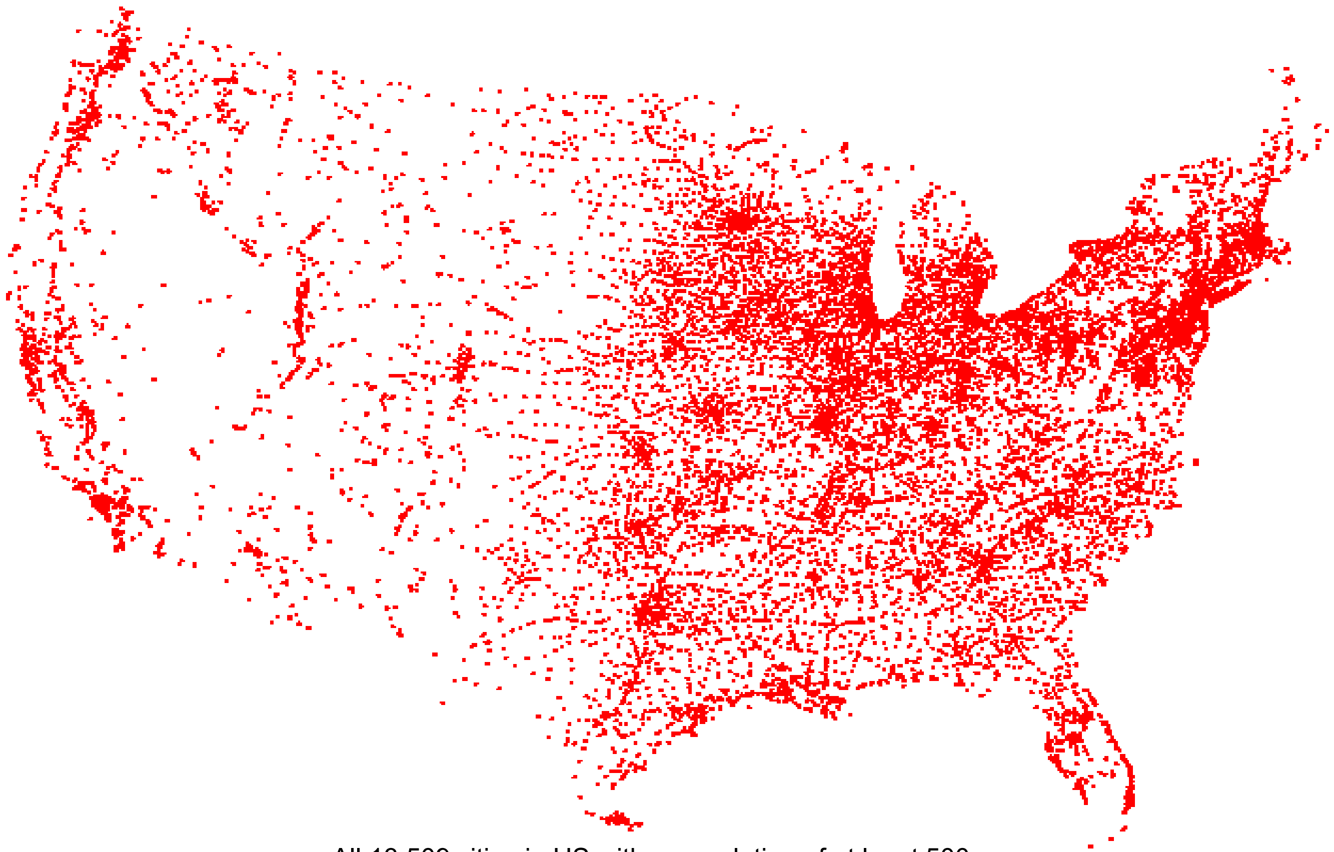
LONGEST-PATH: Given a digraph G = (V, E), does there exists a simple path of length at least k edges?

Theorem: 3-SAT $\leq_P$ LONGEST-PATH.

Proof: Redo the proof for DIR-HAM-CYCLE, ignoring back-edge from t to s.

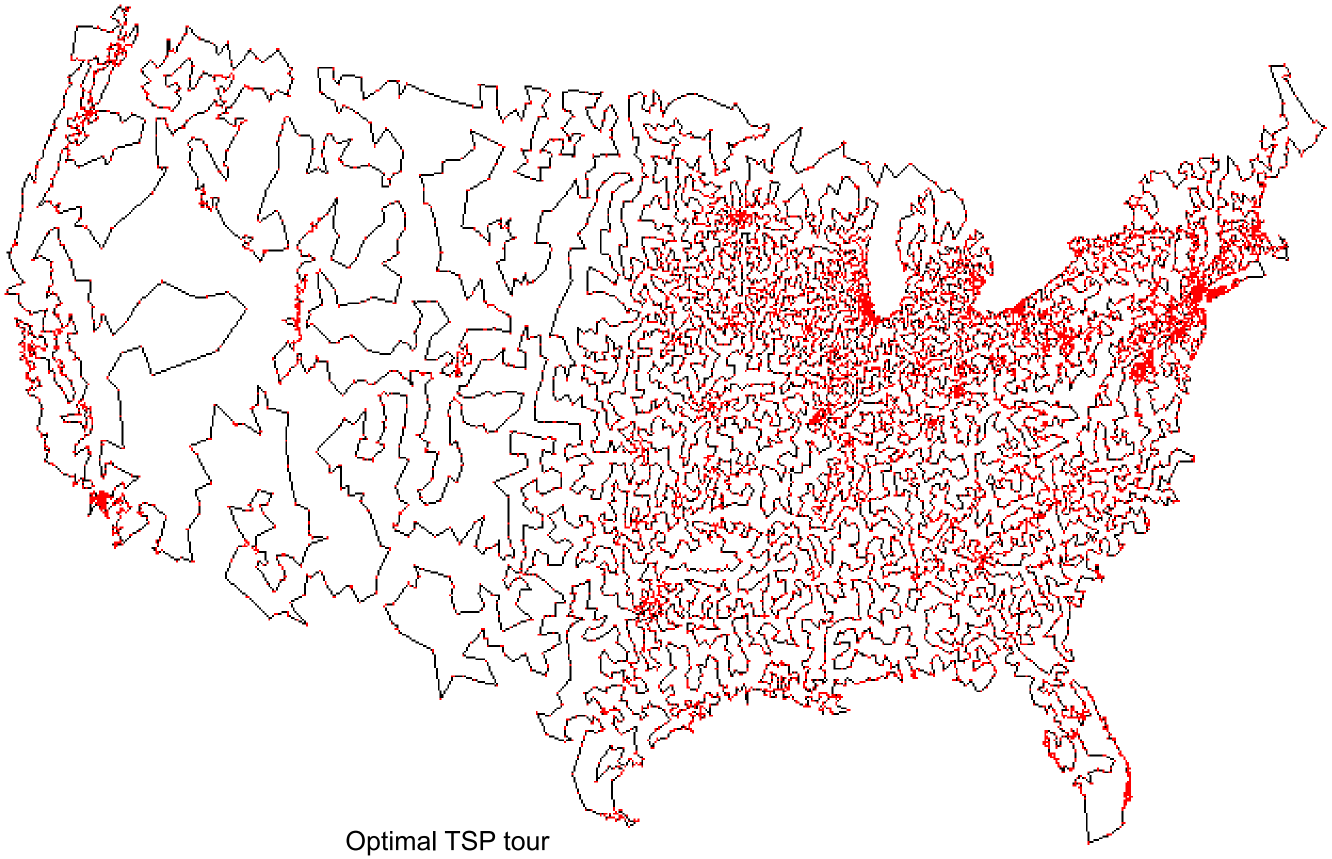
Travelling Salesperson Problem

TSP: Given a set of n cities and a pairwise distance function d(u, v), is there a tour of length ≤ D?


All 13,509 cities in US with a population of at least 500
Reference: http://www.tsp.gatech.edu

Travelling Salesperson Problem

TSP: Given a set of n cities and a pairwise distance function d(u, v), is there a tour of length $\leq$ D?



Optimal TSP tour
Reference: http://www.tsp.gatech.edu

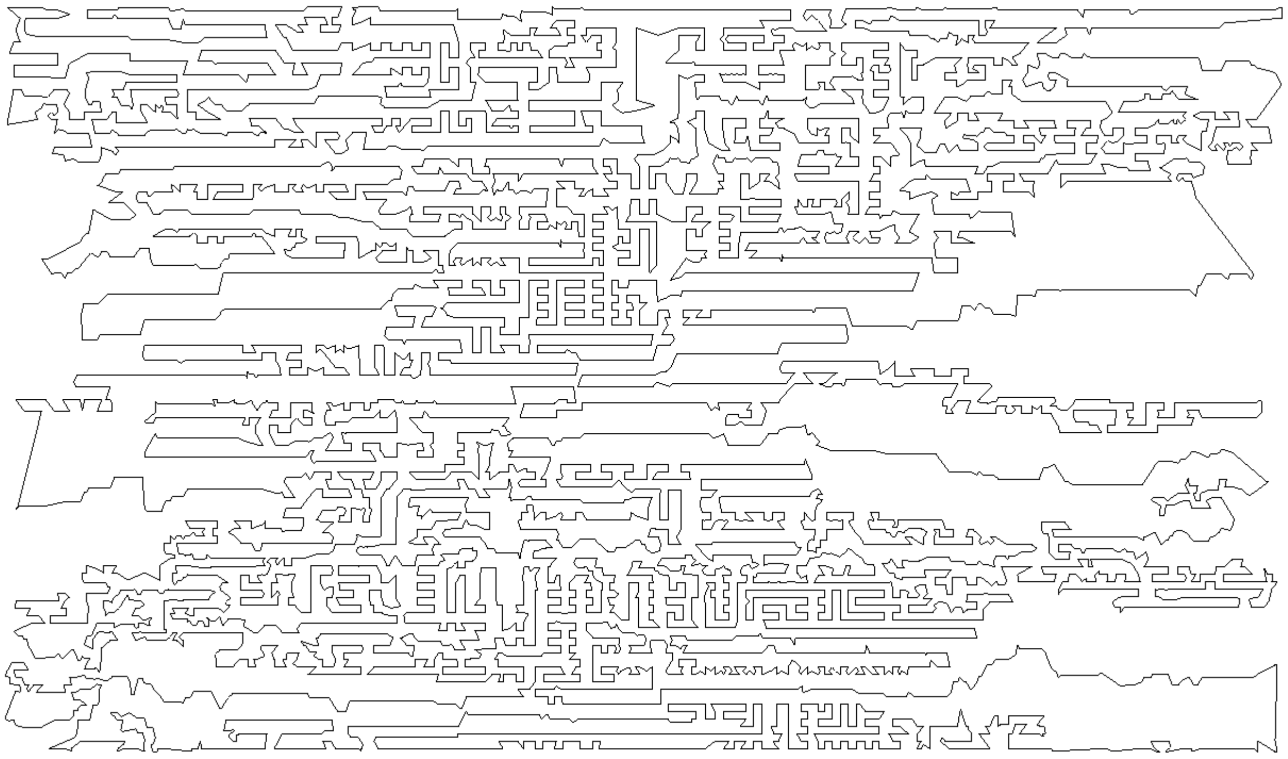
Travelling Salesperson Problem

TSP: Given a set of n cities and a pairwise distance function d(u, v), is there a tour of length $\leq$ D?

11,849 holes to drill in a programmed logic array
Reference: http://www.tsp.gatech.edu

Travelling Salesperson Problem

TSP: Given a set of n cities and a pairwise distance function d(u, v), is there a tour of length $\leq$ D?



Optimal TSP tour
Reference: http://www.tsp.gatech.edu

Travelling Salesperson Problem

TSP: Given a set of n cities and a pairwise distance function d(u, v), is there a tour of length $\leq$ D?

HAM-CYCLE: given a graph G = (V, E), does there exists a simple cycle that contains every node in V?

Theorem: HAM-CYCLE $\leq_P$ TSP.

Proof:

- Given instance G = (V, E) of HAM-CYCLE, create n cities with distance function

$$d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \notin E \end{cases}$$

- TSP instance has tour of length $\leq$ n iff G is Hamiltonian. ▪

NP-complete games and puzzles

- Battleship
- Candy Crush Saga
- Donkey Kong
- Eternity II
- (Generalized) FreeCell
- Lemmings
- Minesweeper Consistency Problem
- Pokémon
- SameGame
- (Generalized) Sudoku
- (generalized) Tetris
- Rush Hour
- Hex
- (Generalized) Super Mario Bros

Polynomial-Time Reductions

Dick Karp (1972)
1985 Turing Award

constraint satisfaction

$$\begin{tikzcd}
& \text{3-SAT} \arrow[dl, "\text{3-SAT reduces to INDEPENDENT SET}"'] \arrow[d] \arrow[dr] \arrow[drr] & & \\
\text{INDEPENDENT SET} \arrow[d] & \text{DIR-HAM-CYCLE} \arrow[d] & \text{GRAPH 3-COLOR} \arrow[d] & \text{SUBSET-SUM} \arrow[d] \\
\text{VERTEX COVER} \arrow[d] & \text{HAM-CYCLE} \arrow[d] & \text{PLANAR 3-COLOR} & \text{SCHEDULING} \\
\text{SET COVER} & \text{TSP} & &
\end{tikzcd}$$

packing and covering | sequencing | partitioning | numerical

Summary

- Polynomial time reductions

  3-SAT $\leq_P$ DIR HAMILTONIAN CYCLE $\leq_P$ HAMILTONIAN CYCLE $\leq_P$ TSP

  3-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ SET-COVER

- Complexity classes:

  P: Decision problems for which there is a poly-time algorithm.

  NP: Decision problems for which there is a poly-time certifier.

  NP-complete: A problem in NP such that every problem in NP polynomial reduces to it.

  NP-hard: A problem such that every problem in NP polynomial reduces to it.

- Lots of problems are NP-complete

  See https://www.nada.kth.se/~viggo/wwwcompendium/