CMSC 451: Reductions & NP-completeness

Slides By: Carl Kingsford

Department of Computer Science
University of Maryland, College Park

Based on Section 8.1 of *Algorithm Design* by Kleinberg & Tardos.

Reductions as tool for hardness

We want prove some problems are computationally difficult.

As a first step, we settle for relative judgements:

Problem X is at least as hard as problem Y

To prove such a statement, we reduce problem Y to problem X:

> *If you had a black box that can solve instances of problem X, how can you solve any instance of Y using polynomial number of steps, plus a polynomial number of calls to the black box that solves X?*

Polynomial Reductions

- If problem Y can be reduced to problem X, we denote this by $Y \leq_P X$.
- This means "Y is polynomal-time reducible to X."
- It also means that X is at least as hard as Y because if you can solve X, you can solve Y.
- Note: We reduce *to* the problem we want to show is the harder problem.

Polynomial Problems

Suppose:

- $Y \leq_P X$, and
- there is an polynomial time algorithm for X.

Then, there is a polynomial time algorithm for Y.

Why?

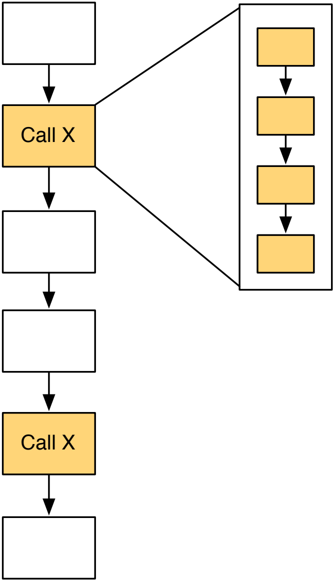
Polynomial Problems

Suppose:

- $Y \leq_P X$, and
- there is an polynomial time algorithm for X.

Then, there is a polynomial time algorithm for Y.

Why? Because polynomials compose.



We've Seen Reductions Before

Examples of Reductions:

- Max Bipartite Matching $\leq_P$ Max Network Flow.
- Image Segmentation $\leq_P$ Min-Cut.
- Survey Design $\leq_P$ Max Network Flow.
- Disjoint Paths $\leq_P$ Max Network Flow.

Reductions for Hardness

Theorem

If $Y \leq_P X$ and Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.

Why? If we *could* solve X in polynomial time, then we'd be able to solve Y in polynomial time using the reduction, contradicting the assumption.

So: If we could find one hard problem Y, we could prove that another problem X is hard by reducing Y to X.

Vertex Cover

Def. A vertex cover of a graph is a set S of nodes such that every edge has at least one endpoint in S.

In other words, we try to "cover" each of the edges by choosing at least one of its vertices.

Vertex Cover

Given a graph G and a number k, does G contain a vertex cover of size at most k.

Independent Set to Vertex Cover

Independent Set

Given graph G and a number k, does G contain a set of at least k independent vertices?

Can we reduce independent set to vertex cover?

Vertex Cover

Given a graph G and a number k, does G contain a vertex cover of size at most k.

Relation btw Vertex Cover and Indep. Set

Theorem

If $G = (V, E)$ is a graph, then S is an independent set $\iff$ $V - S$ is a vertex cover.

Proof. $\implies$ Suppose S is an independent set, and let $e = (u, v)$ be some edge. Only one of u, v can be in S. Hence, at least one of u, v is in $V - S$. So, $V - S$ is a vertex cover.

$\impliedby$ Suppose $V - S$ is a vertex cover, and let $u, v \in S$. There can't be an edge between u and v (otherwise, that edge wouldn't be covered in $V - S$). So, S is an independent set. $\square$

Independent Set $\leq_P$ Vertex Cover

Independent Set $\leq_P$ Vertex Cover

To show this, we change any instance of Independent Set into an instance of Vertex Cover:

- Given an instance of Independent Set $\langle G, k \rangle$,
- We ask our Vertex Cover black box if there is a vertex cover $V - S$ of size $\leq |V| - k$.

By our previous theorem, S is an independent set iff $V - S$ is a vertex cover. If the Vertex Cover black box said:

> `yes`: *then S must be an independent set of size $\geq k$.*
> `no`: *then there is no vertex cover $V - S$ of size $\leq |V| - k$, hence there is no independent set of size $\geq k$.*

Vertex Cover $\leq_P$ Independent Set

Actually, we also have:

Vertex Cover $\leq_P$ Independent Set

Proof. To decide if G has an vertex cover of size k, we ask if it has an independent set of size $n - k$. $\square$

So: Vertex Cover and Independent Set are equivalently difficult.

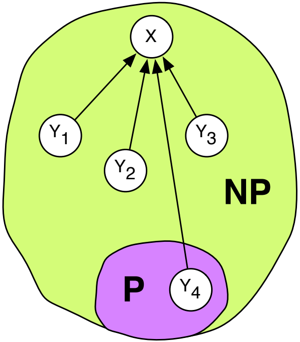
NP-completeness

Def. We say X is NP-complete if:

- $X \in \mathbf{NP}$
- for all $Y \in \mathbf{NP}$, $Y \leq_P X$.

If these hold, then X can be used to solve every problem in **NP**.

Therefore, X is definitely at least as hard as every problem in **NP**.



NP-completeness and P=NP

Theorem

If X is NP-complete, then X is solvable in polynomial time if and only if $\mathbf{P} = \mathbf{NP}$.

Proof. If $\mathbf{P} = \mathbf{NP}$, then X can be solved in polytime.

Suppose X is solvable in polytime, and let Y be any problem in $\mathbf{NP}$. We can solve Y in polynomial time: reduce it to X.

Therefore, every problem in $\mathbf{NP}$ has a polytime algorithm and $\mathbf{P} = \mathbf{NP}$.

Reductions and NP-completeness

Theorem

If Y is NP-complete, and

1. *X is in NP*
2. *$Y \leq_P X$*

then X is NP-complete.

In other words, we can prove a new problem is NP-complete by reducing some other NP-complete problem to it.

Proof. Let Z be any problem in **NP**. Since Y is NP-complete, $Z \leq_P Y$. By assumption, $Y \leq_P X$. Therefore: $Z \leq_P Y \leq_P X$. $\square$

Some First NP-complete problem

We need to find some first NP-complete problem.

Finding the first NP-complete problem was the result of the Cook-Levin theorem.

We'll deal with this later. For now, trust me that:

- Independent Set is a *packing problem* and is NP-complete.
- Vertex Cover is a *covering problem* and is NP-complete.

Set Cover

Another very general and useful covering problem:

Set Cover

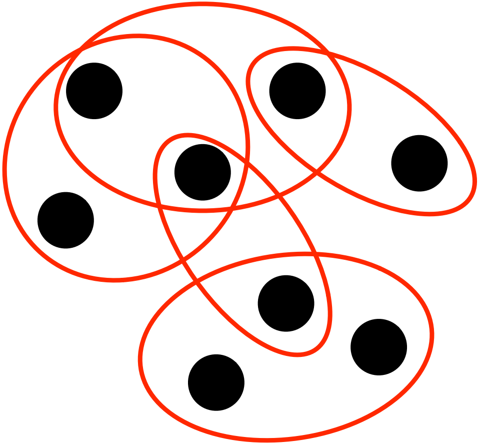
Given a set U of elements and a collection $S_1, \ldots, S_m$ of subsets of U, is there a collection of at most k of these sets whose union equals U?

We will show that

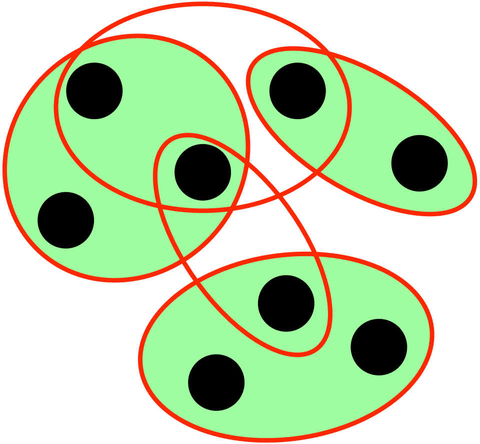
$$\textsc{Set Cover} \in NP$$
$$\textsc{Vertex Cover} \leq_P \textsc{Set Cover}$$

And therefore that SET COVER is NP-complete.

Set Cover, Figure



Set Cover, Figure



Vertex Cover $\leq_P$ Set Cover

Thm. Vertex Cover $\leq_P$ Set Cover

Proof. Let $G = (V, E)$ and k be an instance of Vertex Cover. Create an instance of Set Cover:

- $U = E$
- Create a S_u for for each $u \in V$, where S_u contains the edges adjacent to u.

U can be covered by $\leq k$ sets iff G has a vertex cover of size $\leq k$.

Why? If k sets $S_{u_1}, \ldots, S_{u_k}$ cover U then every edge is adjacent to at least one of the vertices $u_1, \ldots, u_k$, yielding a vertex cover of size k.

If $u_1, \ldots, u_k$ is a vertex cover, then sets $S_{u_1}, \ldots, S_{u_k}$ cover U. $\square$

Last Step:

We still have to show that Set Cover is in **NP**!

The certificate is a list of k sets from the given collection.

We can check in polytime whether they cover all of U.

Since we have a certificate that can be checked in polynomial time, Set Cover is in **NP**.

Summary

You can prove a problem is NP-complete by reducing a known NP-complete problem to it.

We know the following problems are NP-complete:

- Vertex Cover
- Independent Set
- Set Cover

Warning: You should reduce the *known* NP-complete problem to the problem you are interested in. (You *will* mistakenly do this backwards sometimes.)

CMSC 451: SAT, Coloring, Hamiltonian Cycle, TSP

Slides By: Carl Kingsford

Department of Computer Science
University of Maryland, College Park

Based on Sects. 8.2, 8.7, 8.5 of *Algorithm Design* by Kleinberg & Tardos.

Boolean Formulas

Boolean Formulas:

Variables: x_1, x_2, x_3 (can be either **true** or **false**)

Terms: $t_1, t_2, \ldots, t_\ell$: t_j is either x_i or $\bar{x}_i$
(meaning either x_i or **not** x_i).

Clauses: $t_1 \vee t_2 \vee \cdots \vee t_\ell$ ($\vee$ stands for "OR")
A clause is **true** if any term in it is **true**.

Example 1: $(x_1 \vee \bar{x_2}), (\bar{x_1} \vee \bar{x_3}), (x_2 \vee \bar{v_3})$

Example 2: $(x_1 \vee x_2 \vee \bar{x_3}), (\bar{x_2} \vee x_1)$

Boolean Formulas

Def. A truth assignment is a choice of **true** or **false** for each variable, ie, a function $v : X \to \{\textbf{true}, \textbf{false}\}$.

Def. A CNF formula is a conjunction of clauses:

$$C_1 \wedge C_2, \wedge \cdots \wedge C_k$$

Example: $(x_1 \vee \bar{x_2}) \wedge (\bar{x_1} \vee \bar{x_3}) \wedge (x_2 \vee \bar{v_3})$

Def. A truth assignment is a satisfying assignment for such a formula if it makes every clause **true**.

SAT and 3-SAT

Satisfiability (SAT)

Given a set of clauses $C_1, \ldots, C_k$ over variables $X = \{x_1, \ldots, x_n\}$ is there a satisfying assignment?

Satisfiability (3-SAT)

Given a set of clauses $C_1, \ldots, C_k$, each of length 3, over variables $X = \{x_1, \ldots, x_n\}$ is there a satisfying assignment?

Cook-Levin Theorem

Theorem (Cook-Levin)

3-SAT is NP-complete.

Proven in early 1970s by Cook. Slightly different proof by Levin independently.

Idea of the proof: encode the workings of a Nondeterministic Turing machine for an instance I of problem $X \in \mathbf{NP}$ as a SAT formula so that the formula is satisfiable if and only if the nondeterministic Turing machine would accept instance I.

We won't have time to prove this, but it gives us our first hard problem.

Reducing 3-SAT to Independent Set

Thm. 3-SAT $\leq_P$ Independent Set

Proof. Suppose we have an algorithm to solve Independent Set, how can we use it to solve 3-SAT?

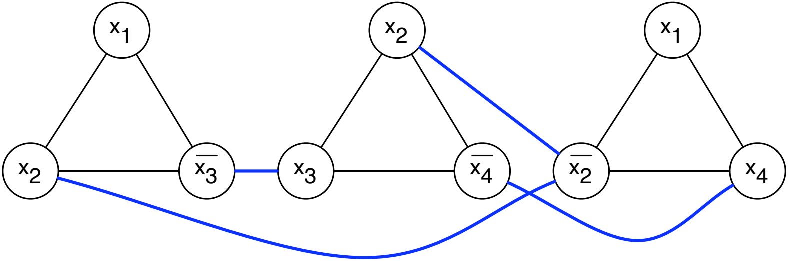
To solve 3-SAT:

- you have to choose a term from each clause to set to **true**,
- but you can't set both x_i and $\bar{x}_i$ to **true**.

How do we do the reduction?

3-SAT $\leq_P$ Independent Set

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



Proof

Theorem

This graph has an independent set of size k iff the formula is satisfiable.

Proof. $\Longrightarrow$ If the formula is satisfiable, there is at least one true literal in each clause. Let S be a set of one such true literal from each clause. $|S| = k$ and no two nodes in S are connected by an edge.

$\Longrightarrow$ If the graph has an independent set S of size k, we know that it has one node from each "clause triangle." Set those terms to **true**. This is possible because no 2 are negations of each other. $\square$

Graph Coloring

Graph Coloring Problem

Graph Coloring Problem

Given a graph G, can you color the nodes with $\leq k$ colors such that the endpoints of every edge are colored differently?

Notation: A k-coloring is a function $f : V \rightarrow \{1, \ldots, k\}$ such that for every edge $\{u, v\}$ we have $f(u) \neq f(v)$.

If such a function exists for a given graph G, then G is k-colorable.

Special case of $k = 2$

How can we test if a graph has a 2-coloring?

Special case of $k = 2$

How can we test if a graph has a 2-coloring?

Check if the graph is bipartite.

Unfortunately, for $k \geq 3$, the problem is NP-complete.

Theorem

3-Coloring is NP-complete.

Graph Coloring is NP-complete

3-Coloring $\in$ **NP**: A valid coloring gives a certificate.

We will show that:

$$\text{3-SAT} \leq_P \text{3-Coloring}$$

Let $x_1, \ldots, x_n, C_1, \ldots, C_k$ be an instance of 3-SAT.

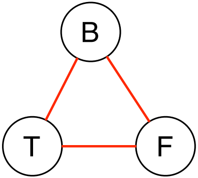
We show how to use 3-Coloring to solve it.

Reduction from 3-SAT

We construct a graph G that will be 3-colorable iff the 3-SAT instance is satisfiable.

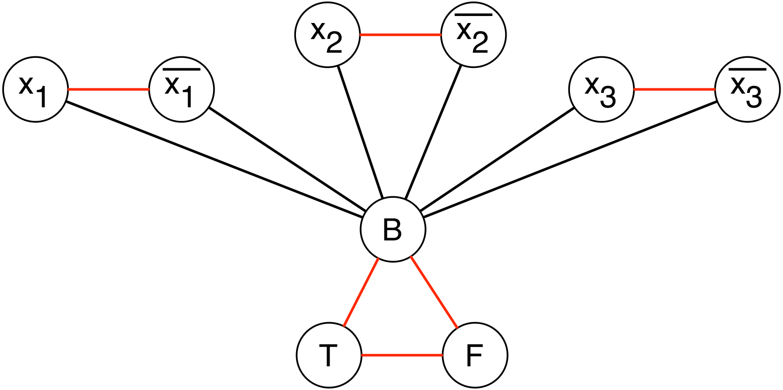
For every variable x_i, create 2 nodes in G, one for x_i and one for $\bar{x}_i$. Connect these nodes by an edge:

Create 3 *special nodes* T, F, and B, joined in a triangle:



Connecting them up

Connect every variable node to B:



Properties

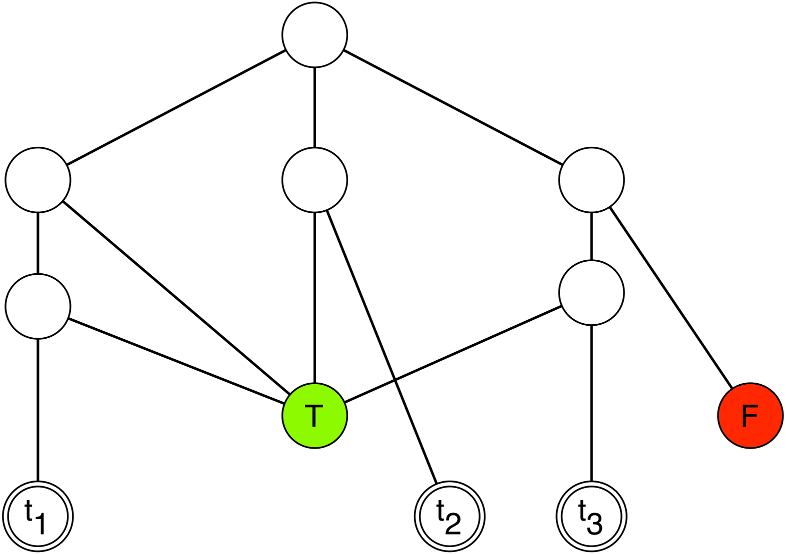
Properties:

- Each of x_i and $\bar{x}_i$ must get different colors
- Each must be different than the color of B.
- B, T, and F must get different colors.

Hence, any 3-coloring of this graph defines a valid truth assignment!

Still have to constrain the truth assignments to satisfy the given clauses, however.

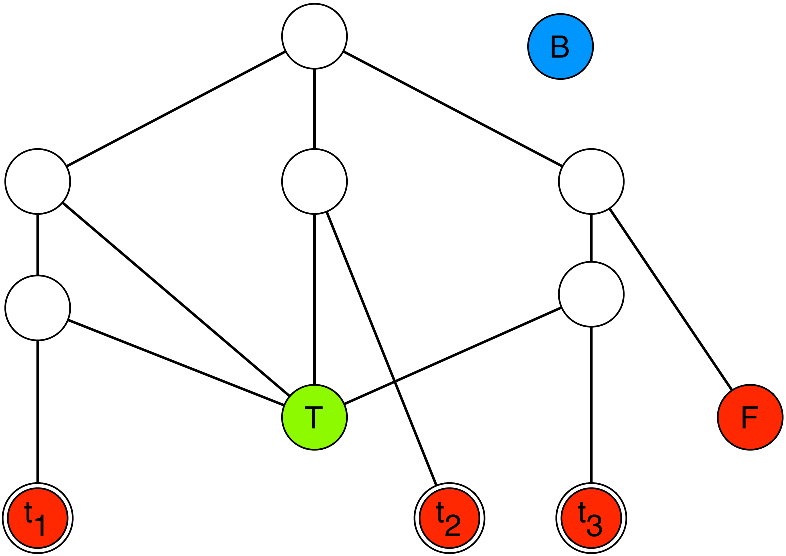
Connect Clause (t_1, t_2, t_3) up like this:



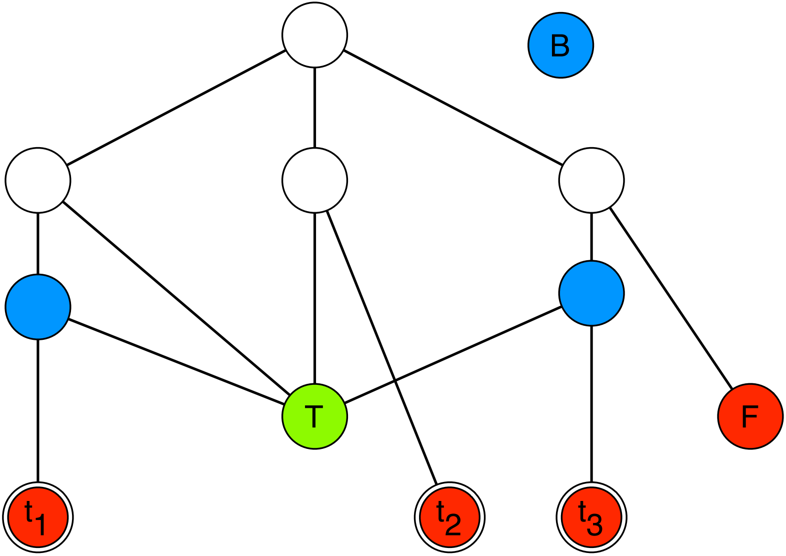
Suppose Every Term Was False

What if every term in the clause was assigned the **false** color?

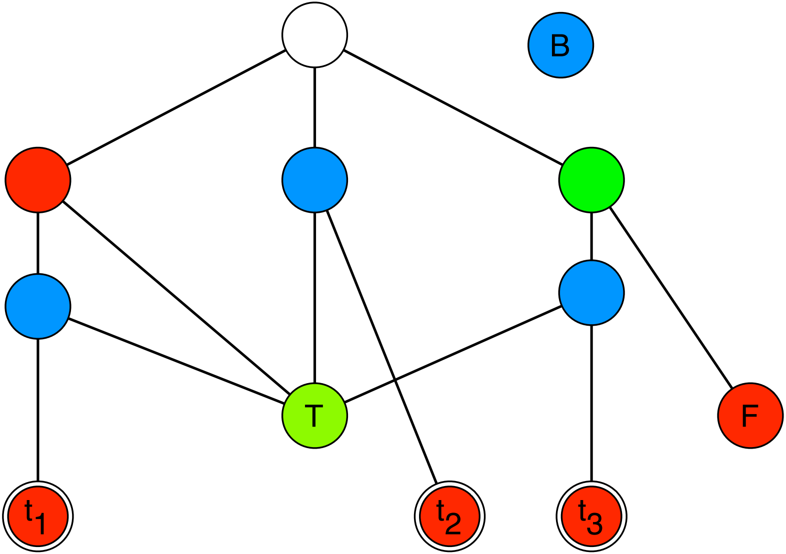
Connect Clause (t_1, t_2, t_3) up like this:



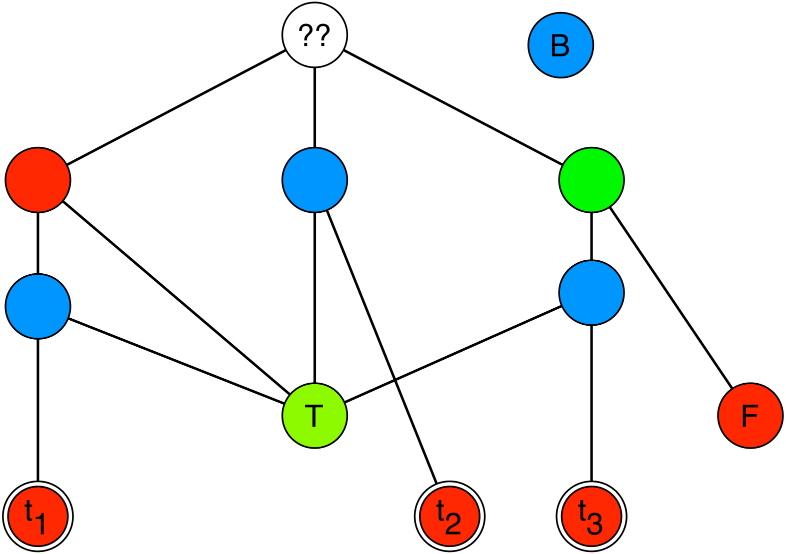
Connect Clause (t_1, t_2, t_3) up like this:



Connect Clause (t_1, t_2, t_3) up like this:



Connect Clause (t_1, t_2, t_3) up like this:



Suppose there is a 3-coloring

$\boxed{\text{Top node is colorable iff one of its terms gets the } \textbf{true} \text{ color.}}$

Suppose there is a 3-coloring.

We get a satisfying assignment by:

- Setting $x_i =$ **true** iff v_i is colored the same as T

Let C be any clause in the formula. At least 1 of its terms must be true, because if they were all false, we couldn't complete the coloring (as shown above).

Suppose there is a satisfying assignment

Suppose there is a satisfying assignment.

We get a 3-coloring of G by:

- Coloring T, F, B arbitrarily with 3 different colors
- If $x_i =$ **true**, color v_i with the same color as T and $\bar{v}_i$ with the color of F.
- If $x_i =$ **false**, do the opposite.
- Extend this coloring into the clause gadgets.

Hence: the graph is 3-colorable iff the formula it is derived from is satisfiable.

General Proof Strategy

General Strategy for Proving Something is NP-complete:

1. Must show that $X \in \mathbf{NP}$. Do this by showing there is an certificate that can be efficiently checked.

2. Look at some problems that are known to be NP-complete (there are thousands), and choose one Y that seems "similar" to your problem in some way.

3. Show that $Y \leq_P X$.

Strategy for Showing $Y \leq_P X$

One strategy for showing that $Y \leq_P X$ often works:

1. Let I_Y be any instance of problem Y.
2. Show how to construct an instance I_X of problem X in polynomial time such that:
   - If $I_Y \in Y$, then $I_X \in X$
   - If $I_X \in X$, then $I_Y \in Y$

Hamiltonian Cycle

Hamiltonian Cycle Problem

Hamiltonian Cycle

Given a directed graph G, is there a cycle that visits every vertex exactly once?

Such a cycle is called a Hamiltonian cycle.

Hamiltonian Cycle is NP-complete

Theorem

Hamiltonian Cycle is NP-complete.

Proof. First, HamCycle $\in$ **NP**. Why?

Second, we show 3-SAT $\leq_P$ Hamiltonian Cycle.

Suppose we have a black box to solve Hamiltonian Cycle, how do we solve 3-SAT?

In other words: how do we encode an instance I of 3-SAT as a graph G such that I is satisfiable exactly when G has a Hamiltonian cycle.

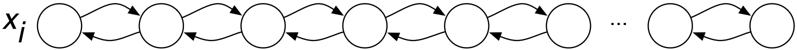
Consider an instance I of 3-SAT, with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_k$.

Reduction Idea

Reduction Idea (very high level):

- Create some graph structure (a "gadget") that represents the variables
- And some graph structure that represents the clauses
- Hook them up in some way that encodes the formula
- Show that this graph has a Ham. cycle iff the formula is satisfiable.

Gadget Representing the Variables



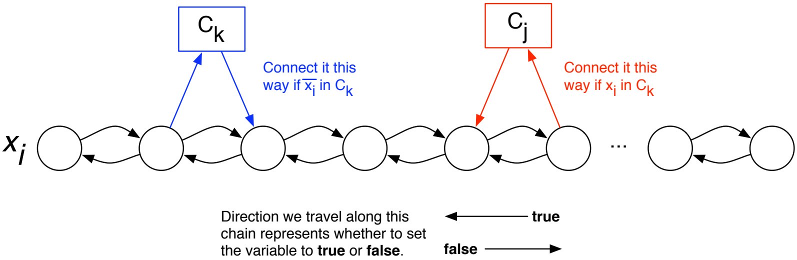
Direction we travel along this chain represents whether to set the variable to **true** or **false**.

← **true**

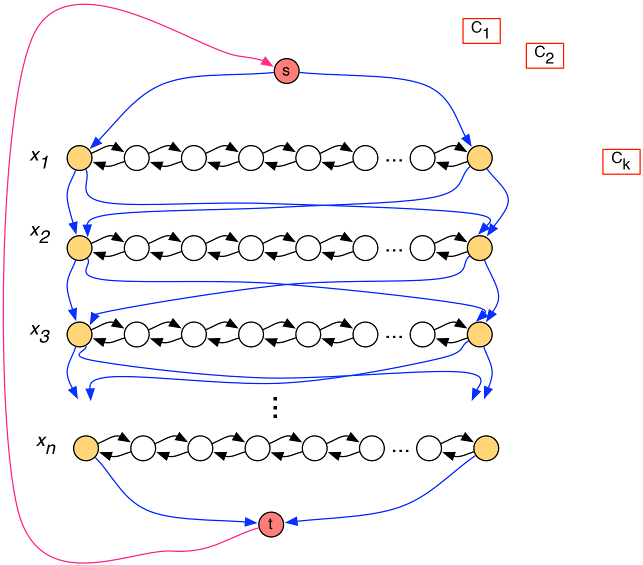
false →

Hooking in the Clauses

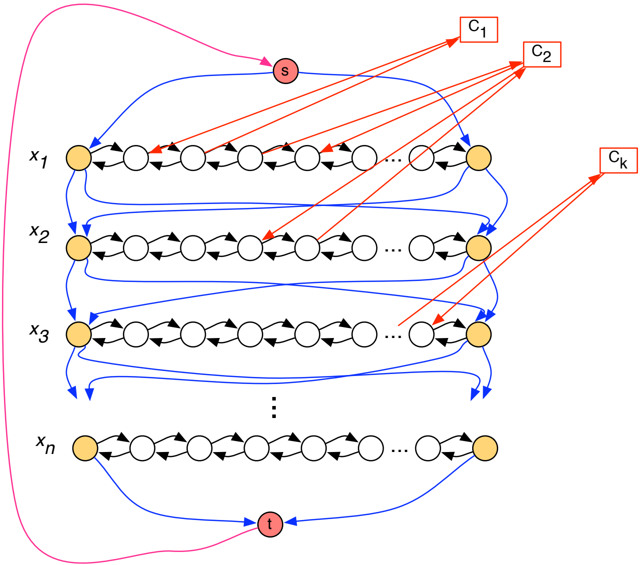
Add a new node for each clause:



Connecting up the paths



Connecting up the paths



Hamiltonian Cycle is NP-complete

- A Hamiltonian path encodes a truth assignment for the variables (depending on which direction each chain is traversed)
- For there to be a Hamiltonian cycle, we have to visit every clause node
- We can only visit a clause if we satisfy it (by setting one of its terms to true)
- Hence, if there is a Hamiltonian cycle, there is a satisfying assignment

Hamiltonian Path

Hamiltonian Path: Does G contain a path that visits every node exactly once?

How could you prove this problem is NP-complete?

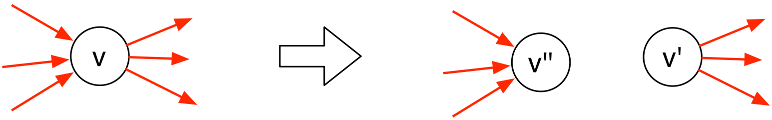
Hamiltonian Path

Hamiltonian Path: Does G contain a path that visits every node exactly once?

How could you prove this problem is NP-complete?

Reduce Hamiltonian Cycle to Hamiltonian Path.

Given instance of Hamiltonian Cycle G, choose an arbitrary node v and split it into two nodes to get graph G':



Now any Hamiltonian Path must start at v' and end at v''.

Hamiltonian Path

G'' has a Hamiltonian Path $\iff$ G has a Hamiltonian Cycle.

$\implies$ If G'' has a Hamiltonian Path, then the same ordering of nodes (after we glue v' and v'' back together) is a Hamiltonian cycle in G.

$\impliedby$ If G has a Hamiltonian Cycle, then the same ordering of nodes is a Hamiltonian path of G' if we split up v into v' and v''. $\square$

Hence, Hamiltonian Path is NP-complete.

Traveling Salesman Problem

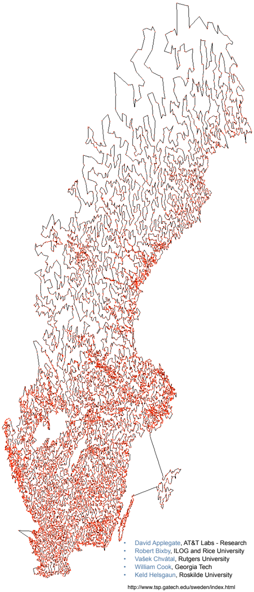
Traveling Salesman Problem

Given n cities, and distances $d(i, j)$ between each pair of cities, does there exist a path of length $\leq k$ that visits each city?

Notes:

- We have a distance between every pair of cities.
- In this version, $d(i, j)$ doesn't have to equal $d(j, i)$.
- And the distances don't have to obey the triangle inequality $(d(i, j) \leq d(i, k) + d(k, j)$ for all $i, j, k)$.

TSP large instance



- TSP visiting 24,978 (all) cities in Sweden.
- Solved by David Applegate, Robert Bixby, Vašek Chvátal, William Cook, and Keld Helsgaun
- `http://www.tsp.gatech.edu/sweden/index.html`
- Lots more cool TSP at `http://www.tsp.gatech.edu/`

Traveling Salesman is NP-complete

Thm. Traveling Salesman is NP-complete.

TSP seems a lot like Hamiltonian Cycle. We will show that

$$\textsc{Hamiltonian Cycle} \leq_P \mathit{TSP}$$

To do that:

Given: a graph $G = (V, E)$ that we want to test for a Hamiltonian cycle,

Create: an instance of TSP.

Creating a TSP instance

A TSP instance D consists of n cities, and $n(n-1)$ distances.

Cities We have a city c_i for every node v_i.

Distances Let $d(c_i, c_j) = \begin{cases} 1 & \text{if edge } (v_i, v_j) \in E \\ 2 & \text{otherwise} \end{cases}$

TSP Redcution

Theorem

G has a Hamiltonian cycle $\iff$ D has a tour of length $\leq n$.

Proof. If G has a Ham. Cycle, then this ordering of cities gives a tour of length $\leq n$ in D (only distances of length 1 are used).

Suppose D has a tour of length $\leq n$. The tour length is the sum of n terms, meaning each term must equal 1, and hence cities that are visited consecutively must be connected by an edge in G. $\square$

Also, TSP $\in$ **NP**: a certificate is simply an ordering of the n cities.

TSP is NP-complete

Hence, TSP is NP-complete.

Even TSP restricted to the case when the $d(i, j)$ values come from actual distances on a map is NP-complete.