6.045: Automata, Computability, and Complexity (GITCS)

Class 16

Nancy Lynch

Today: More NP-Completeness

- Topics:
  - 3SAT is NP-complete
  - Clique and VertexCover are NP-complete
  - More examples, overview
  - Hamiltonian path and Hamiltonian circuit
  - Traveling Salesman problem
  - More examples, revisited
- Reading:
  - Sipser Sections 7.4-7.5
  - Garey and Johnson
- Next:
  - Sipser Section 10.2

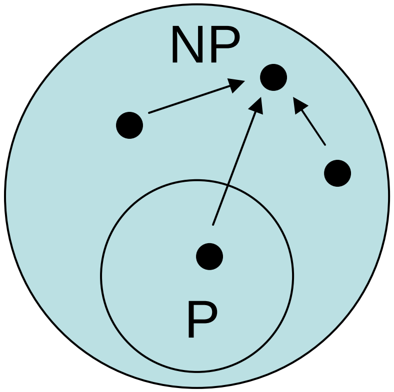
3SAT is NP-Complete

NP-Completeness

- Definition: Language B is NP-complete if both of the following hold:

  (a) B $\in$ NP, and

  (b) For any language A $\in$ NP, A $\leq_p$ B.



- Definition: Language B is NP-hard if, for any language A $\in$ NP, A $\leq_p$ B.

3SAT is NP-Complete

- SAT = { < ϕ > | ϕ is a satisfiable Boolean formula }
- Boolean formula: Constructed from literals using operations, e.g.:

  $$\phi = x \wedge ((y \wedge z) \vee (\neg y \wedge \neg z)) \wedge \neg(x \wedge z)$$

- A Boolean formula is satisfiable iff there is an assignment of 0s and 1s to the variables that makes the entire formula evaluate to 1 (true).
- Theorem: SAT is NP-complete.
- 3SAT: Satisfiable Boolean formulas of a restricted kind---conjunctive normal form (CNF) with exactly 3 literals per clause.
- Theorem: 3SAT is NP-complete.
- Proof:
  - 3SAT $\in$ NP: Obvious.
  - 3SAT is NP-hard: …

3SAT is NP-hard

- Clause: Disjunction of literals, e.g., $(\neg x_1 \vee x_2 \vee \neg x_3)$
- CNF: Conjunction of such clauses
- Example:

  $(\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (x_3)$
- 3-CNF:

  $\{ < \phi > \mid \phi$ is a CNF formula in which each clause has exactly 3 literals $\}$
- CNF-SAT: $\{ < \phi > \mid \phi$ is a satisfiable CNF formula $\}$
- 3-SAT: $\{ < \phi > \mid \phi$ is a satisfiable 3-CNF formula $\}$

  $=$ SAT $\cap$ 3-CNF
- Theorem: 3SAT is NP-hard.
- Proof: Show CNF-SAT is NP-hard, and CNF-SAT $\leq_p$ 3SAT.

CNF-SAT is NP-hard

- Theorem: CNF-SAT is NP-hard.
- Proof:
  - We won't show SAT $\leq_p$ CNF-SAT.
  - Instead, modify the proof that SAT is NP-hard, so that it shows A $\leq_p$ CNF-SAT, for an arbitrary A in NP, instead of just A $\leq_p$ SAT as before.
  - We've almost done this: formula ϕ_w is almost in CNF.
  - It's a conjunction $\phi_w = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$.
  - And each of these is itself in CNF, except ϕ_{move} .
  - ϕ_{move} is:
    - a conjunction over all (i,j)
    - of disjunctions over all tiles
    - of conjunctions of 6 conditions on the 6 cells:

$$x_{i,j,a1} \wedge x_{i,j+1,a2} \wedge x_{i,j+2,a3} \wedge x_{i+1,j,b1} \wedge x_{i+1,j+1,b2} \wedge x_{i+1,j+2,b3}$$

CNF-SAT is NP-hard

- Show $A \leq_p$ CNF-SAT.
- ϕ_w is a conjunction $\phi_w = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$, where each is in CNF, except ϕ_{move} .
- ϕ_{move} is:
  - a conjunction ($\wedge$) over all (i,j)
  - of disjunctions ($\vee$) over all tiles
  - of conjunctions ($\wedge$) of 6 conditions on the 6 cells:

    $x_{i,j,a1} \wedge x_{i,j+1,a2} \wedge x_{i,j+2,a3} \wedge x_{i+1,j,b1} \wedge x_{i+1,j+1,b2} \wedge x_{i+1,j+2,b3}$
- We want just $\wedge$ of $\vee$.
- Can use distributive laws to replace ($\vee$ of $\wedge$) with ($\wedge$ of $\vee$), which would yield overall $\wedge$ of $\vee$, as needed.
- In general, transforming ($\vee$ of $\wedge$) to ($\wedge$ of $\vee$), could cause formula size to grow too much (exponentially).
- However, in this situation, the clauses for each (i,j) have total size that depends only on the TM M, and not on w.
- So the size of the transformed formula is still poly in |w|.

CNF-SAT is NP-hard

- Theorem: CNF-SAT is NP-hard.
- Proof:
  - Modify the proof that SAT is NP-hard.
  - $\phi_w = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$.
  - Can be put into CNF, while keeping the size of the transformed formula poly in |w|.
  - Shows that A $\leq_p$ CNF-SAT.
  - Since A is any language in NP, CNF-SAT is NP-hard.

3SAT is NP-hard

- Proved: Theorem: CNF-SAT is NP-hard.
- Now: Theorem: 3SAT is NP-hard.
- Proof:
  - Use reduction, show CNF-SAT $\leq_p$ 3SAT.
  - Construct f, polynomial-time computable, such that w $\in$ CNF-SAT if and only if f(w) $\in$ 3SAT.
  - If w isn't a CNF formula, then f(w) isn't either.
  - If w is a CNF formula, then f(w) is another CNF formula, this one with 3 literals per clause, satisfiable iff w is satisfiable.
  - f works by converting each clause to a conjunction of clauses, each with $\leq$ 3 literals (add repeats to get 3).
  - Show by example: $(a \vee b \vee c \vee d \vee e)$ gets converted to $(a \vee r_1) \wedge (\neg r_1 \vee b \vee r_2) \wedge (\neg r_2 \vee c \vee r_3) \wedge (\neg r_3 \vee d \vee r_4) \wedge (\neg r_4 \vee e)$
  - f is polynomial-time computable.

3SAT is NP-hard

- Proof:
  - Show CNF-SAT $\leq_p$ 3SAT.
  - Construct f such that w $\in$ CNF-SAT iff f(w) $\in$ 3SAT; converts each clause to a conjunction of clauses.
  - f converts w = $(a \vee b \vee c \vee d \vee e)$ to f(w) =

  $(a \vee r_1) \wedge (\neg r_1 \vee b \vee r_2) \wedge (\neg r_2 \vee c \vee r_3) \wedge (\neg r_3 \vee d \vee r_4) \wedge (\neg r_4 \vee e)$
  - Claim w is satisfiable iff f(w) is satisfiable.
- $\Rightarrow$:
  - Given a satisfying assignment for w, add values for $r_1, r_2, \ldots,$ to satisfy f(w).
  - Start from a clause containing a literal with value 1---there must be one---make the new literals in that clause 0 and propagate consequences left and right.
  - Example: Above, if c = 1, a = b = d = e = 0 satisfy w, use:

  $(a \vee r_1) \wedge (\neg r_1 \vee b \vee r_2) \wedge (\neg r_2 \vee c \vee r_3) \wedge (\neg r_3 \vee d \vee r_4) \wedge (\neg r_4 \vee e)$

  0 1 0 0 1 0 1 0 1 0 0 1 0

3SAT is NP-hard

- Proof:
  - Show CNF-SAT $\leq_p$ 3SAT.
  - Construct f such that w $\in$ CNF-SAT iff f(w) $\in$ 3SAT; converts each clause to a conjunction of clauses.
  - f converts w = $(a \vee b \vee c \vee d \vee e)$ to f(w) =

  $(a \vee r_1) \wedge (\neg r_1 \vee b \vee r_2) \wedge (\neg r_2 \vee c \vee r_3) \wedge (\neg r_3 \vee d \vee r_4) \wedge (\neg r_4 \vee e)$
  - Claim w is satisfiable iff f(w) is satisfiable.
- $\Leftarrow$:
  - Given satisfying assignment for f(w), restrict to satisfy w.
  - Each r_i can make only one clause true.
  - There's one fewer r_i than clauses; so some clause must be made true by an original literal, i.e., some original literal must be true, satisfying w.

3SAT is NP-hard

- Theorem: CNF-SAT is NP-hard.
- Theorem: 3SAT is NP-hard.
- Proof:
  - Constructed polynomial-time-computable f such that w $\in$ CNF-SAT iff f(w) $\in$ 3SAT.
  - Thus, CNF-SAT $\leq_p$ 3SAT.
  - Since CNF-SAT is NP-hard, so is 3SAT.

CLIQUE and VERTEX-COVER are NP-Complete

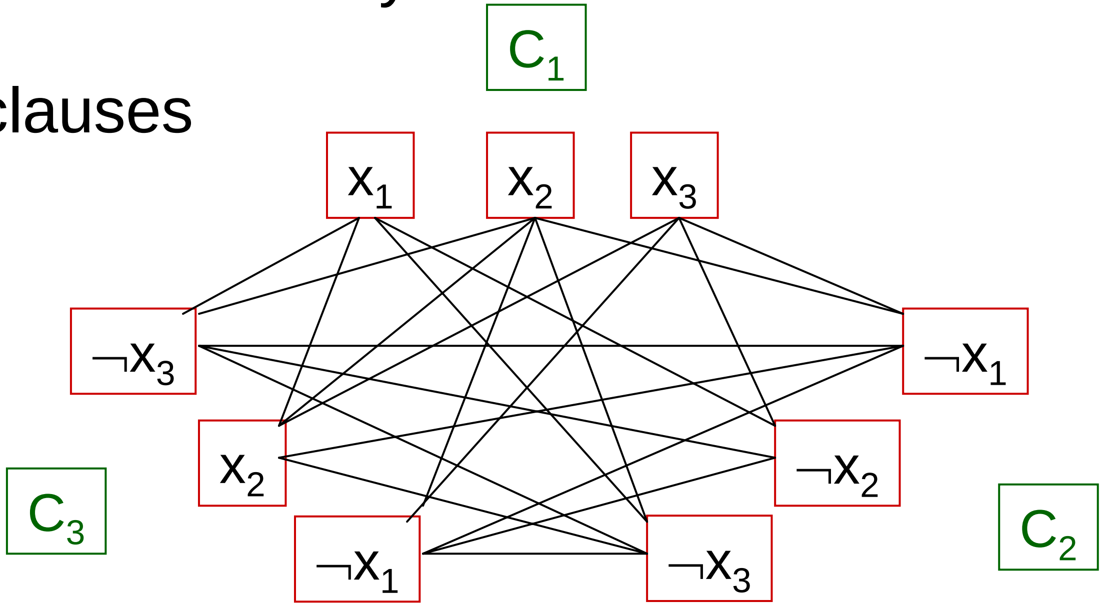
CLIQUE and VERTEX-COVER

- CLIQUE = { < G, k > | G is a graph with a k-clique }
- k-clique: k vertices with edges between all pairs in the clique.
- Theorem: CLIQUE is NP-complete.
- Proof:
  - CLIQUE $\in$ NP, already shown.
  - To show CLIQUE is NP-hard, show 3SAT $\leq_p$ CLIQUE.
  - Need poly-time-computable f, such that w $\in$ 3SAT iff f(w) $\in$ CLIQUE.
  - f must map a formula w in 3-CNF to <G, k> such that w is satisfiable iff G has a k-clique.
  - Show by example:

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

CLIQUE is NP-hard

- Proof:
  - Show 3SAT $\leq_p$ CLIQUE; construct f such that w $\in$ 3SAT iff f(w) $\in$ CLIQUE.
  - f maps a formula w in 3-CNF to <G, k> such that w is satisfiable iff G has a k-clique.
  - $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$
  - Graph G: Nodes for all (clause, literal) pairs, edges between all non-contradictory nodes in different clauses.
  - k: Number of clauses

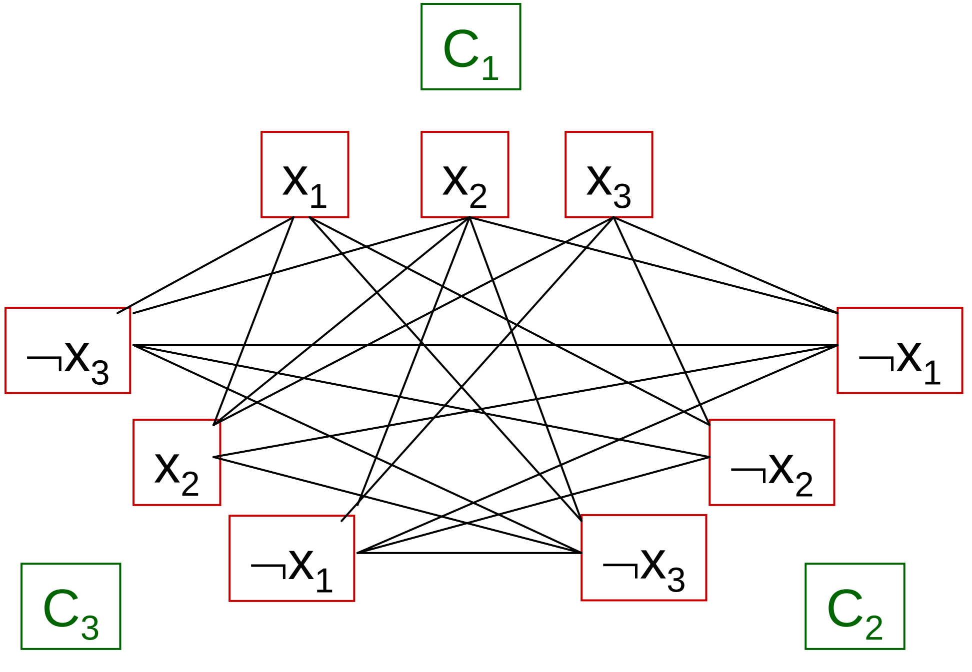


CLIQUE is NP-hard

- Graph G: Nodes for all (clause, literal) pairs, edges between all non-contradictory nodes in different clauses.
- k: Number of clauses

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

- Claim (general): w satisfiable iff G has a k-clique.
- $\Rightarrow$:
  - Assume the formula is satisfiable.
  - Satisfying assignment gives one literal in each clause, all with non-contradictory assignments.
  - Yields a k-clique.

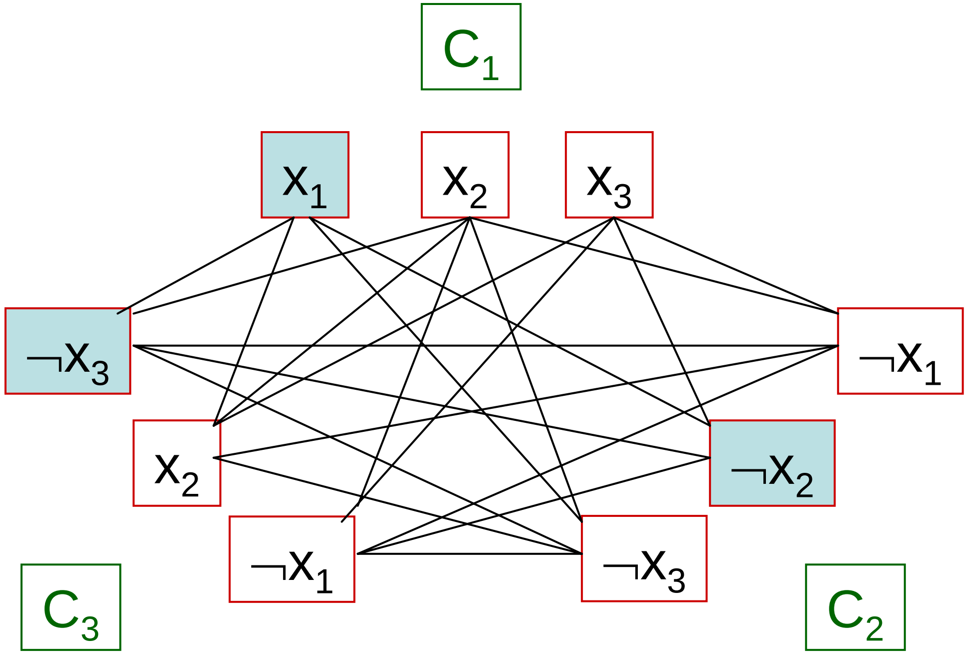


CLIQUE is NP-hard

- Example:

  $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$
- Satisfiable, with satisfying assignment $x_1 = 1$, $x_2 = x_3 = 0$
- Yields 3-clique:

- $\Rightarrow$:
  - Assume the formula is satisfiable.
  - Satisfying assignment gives one literal in each clause, all with non-contradictory assignments.
  - Yields a k-clique.

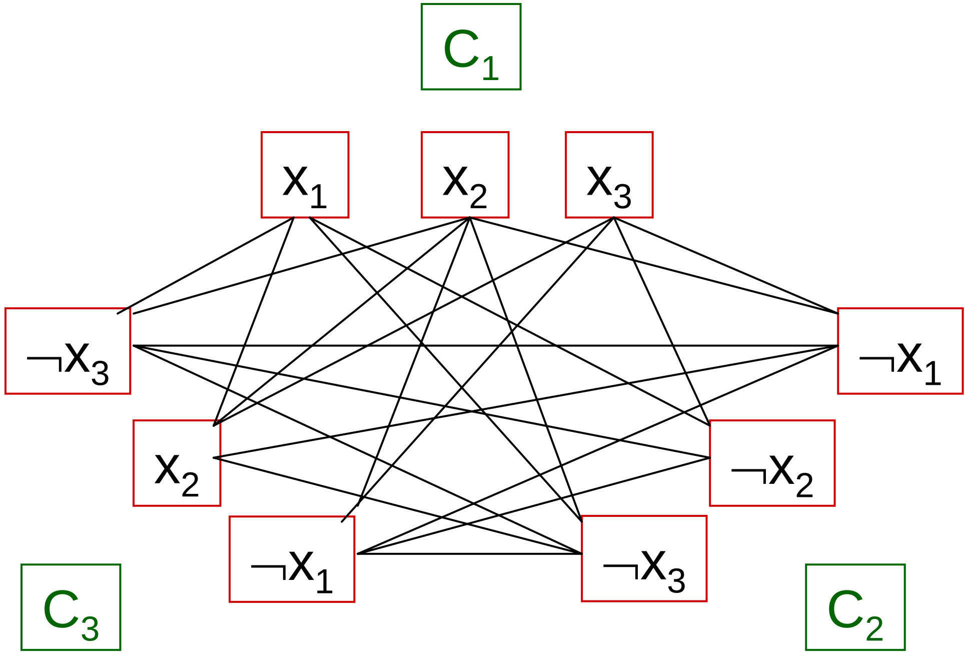


CLIQUE is NP-hard

- Graph G: Nodes for all (clause, literal) pairs, edges between all non-contradictory nodes in different clauses.
- k: Number of clauses

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

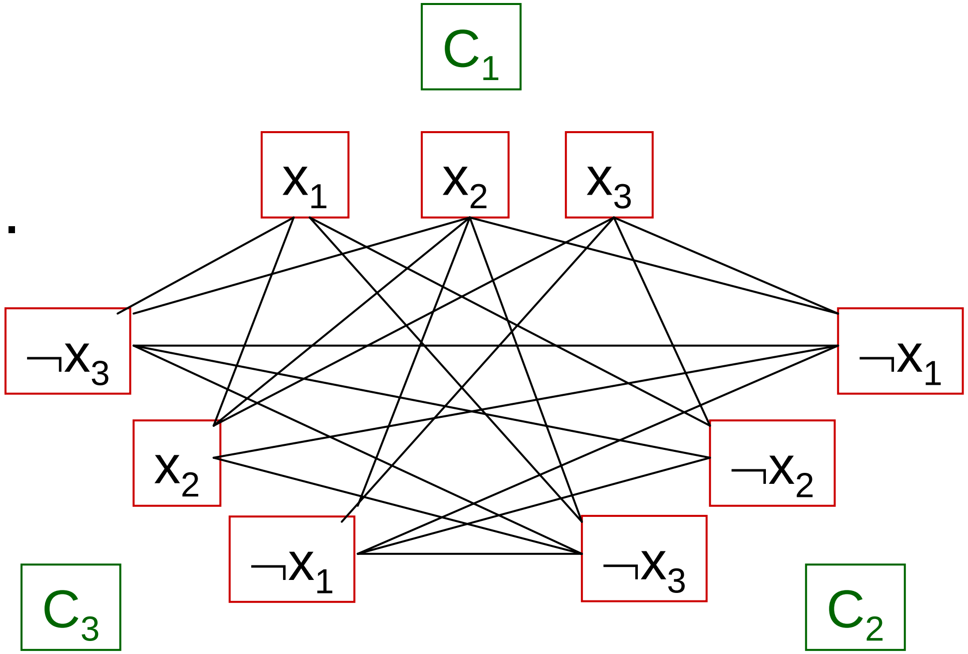
- Claim (general): w satisfiable iff G has a k-clique.
- $\Leftarrow$:
  - Assume a k-clique.
  - Yields one node per clause, none contradictory.
  - Yields a consistent assignment satisfying all clauses of w.



CLIQUE is NP-hard

- Graph G: Nodes for all (clause, literal) pairs, edges between all non-contradictory nodes in different clauses.
- k: Number of clauses
- Claim (general): w satisfiable iff G has a k-clique.

- So, 3SAT $\leq_p$ CLIQUE.
- Since 3SAT is NP-hard, so is CLIQUE.
- So CLIQUE is NP-complete.



VERTEX-COVER is NP-complete

- VERTEX-COVER =

  { < G, k > | G is a graph with a vertex cover of size k }
- Vertex cover of G = (V, E): A subset C of V such that, for every edge (u,v) in E, either u or v $\in$ C.
- Theorem: VERTEX-COVER is NP-complete.
- Proof:
  - VERTEX-COVER $\in$ NP, already shown.
  - Show VERTEX-COVER is NP-hard.
  - That is, if A $\in$ NP, then A $\leq_p$ VERTEX-COVER.
  - We know A $\leq_p$ CLIQUE, since CLIQUE is NP-hard.
  - Recall CLIQUE $\leq_p$ VERTEX-COVER.
  - By transitivity of $\leq_p$, A $\leq_p$ VERTEX-COVER, as needed.

VERTEX-COVER is NP-complete

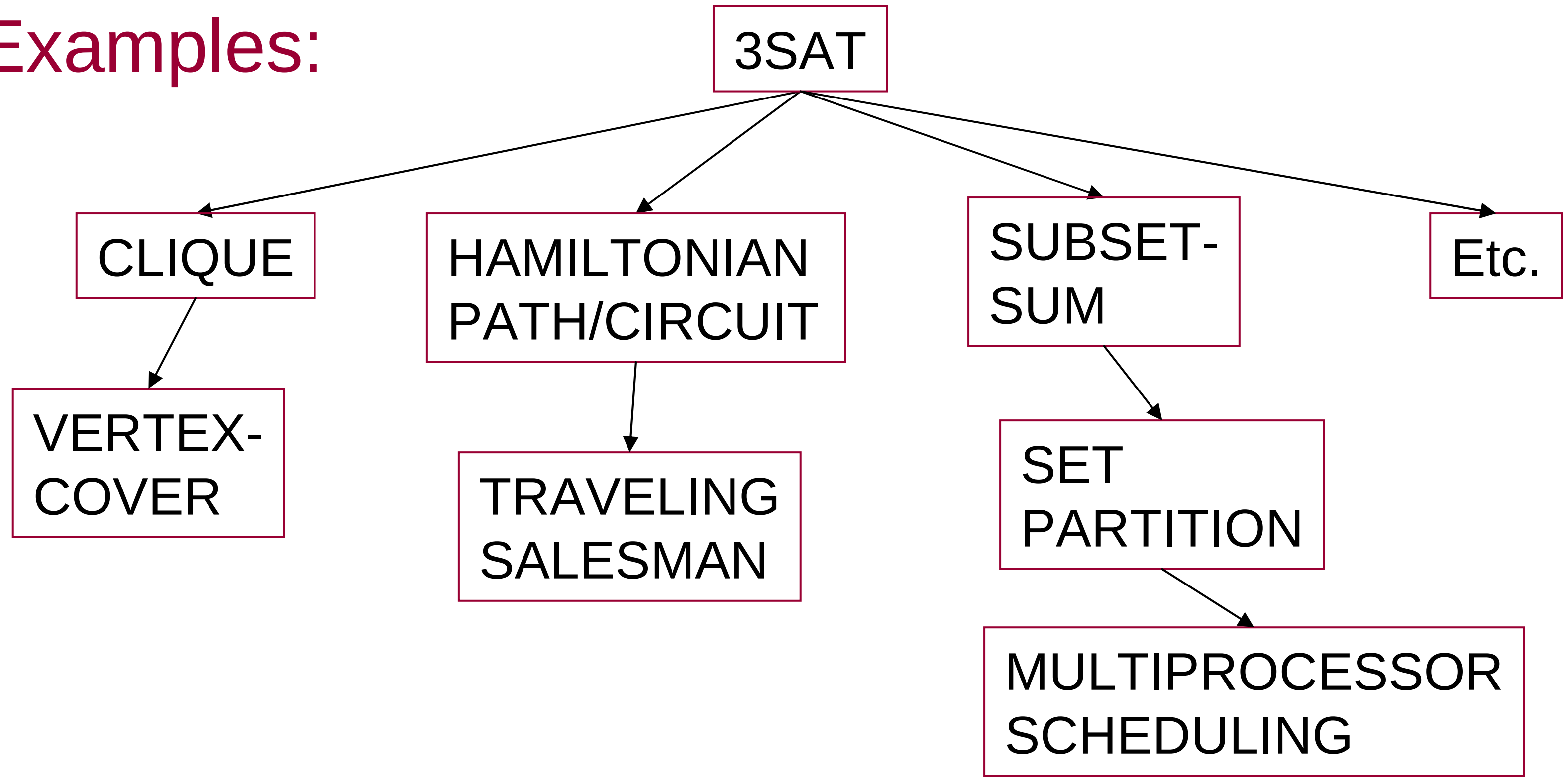
- Theorem: VERTEX-COVER is NP-complete.
- More succinct proof:
  - VC $\in$ NP; show VC is NP-hard.
  - CLIQUE is NP-hard.
  - CLIQUE $\leq_p$ VC.
  - So VC is NP-hard.

- In general, can show language B is NP-complete by:
  - Showing B $\in$ NP, and
  - Showing A $\leq_p$ B for some known NP-hard problem A.

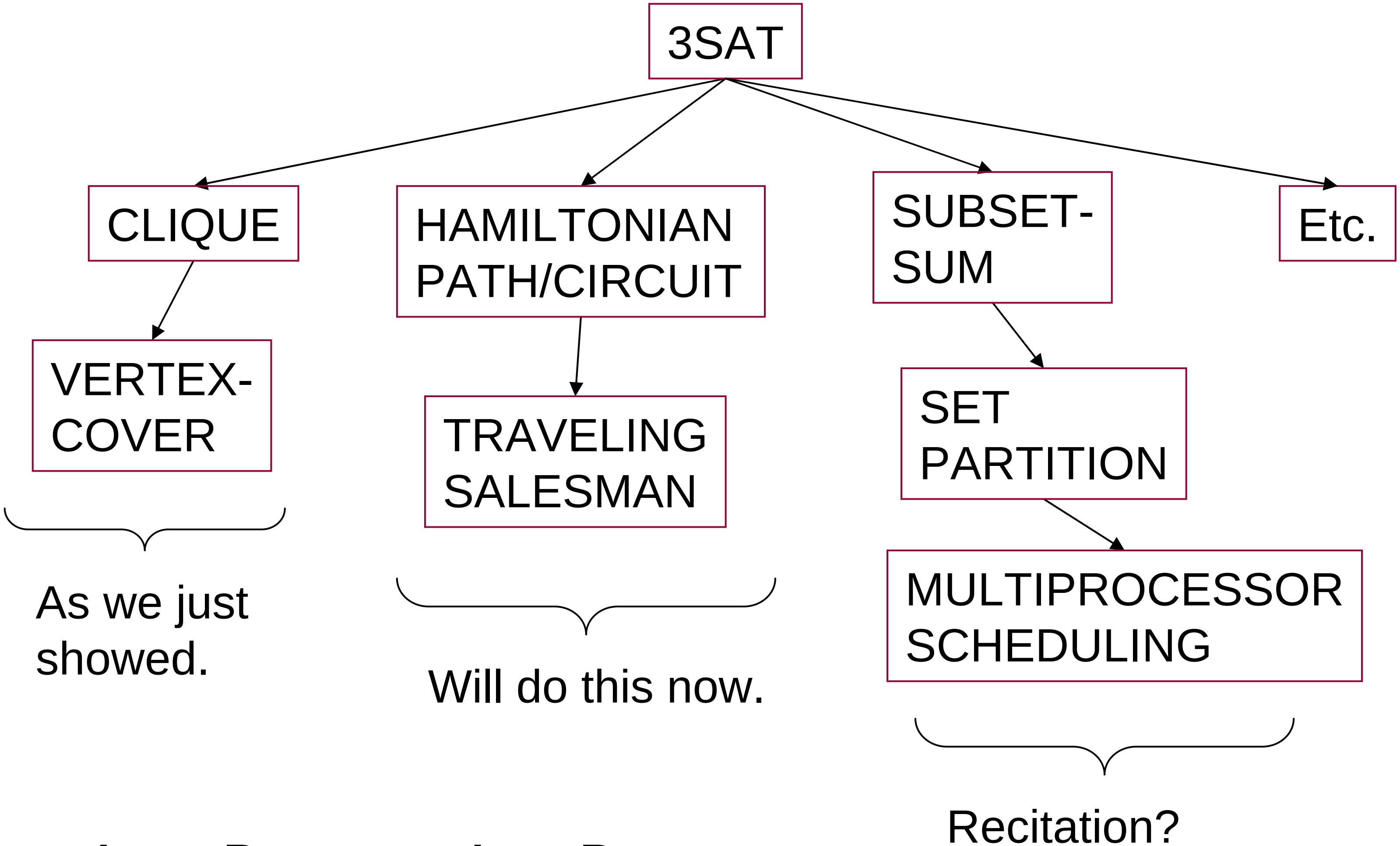
More Examples

More NP-Complete Problems

- [Garey, Johnson] show hundreds of problems are NP-complete.
- All but 3SAT use the polynomial-time reduction method.
- Examples:



More NP-Complete Problems

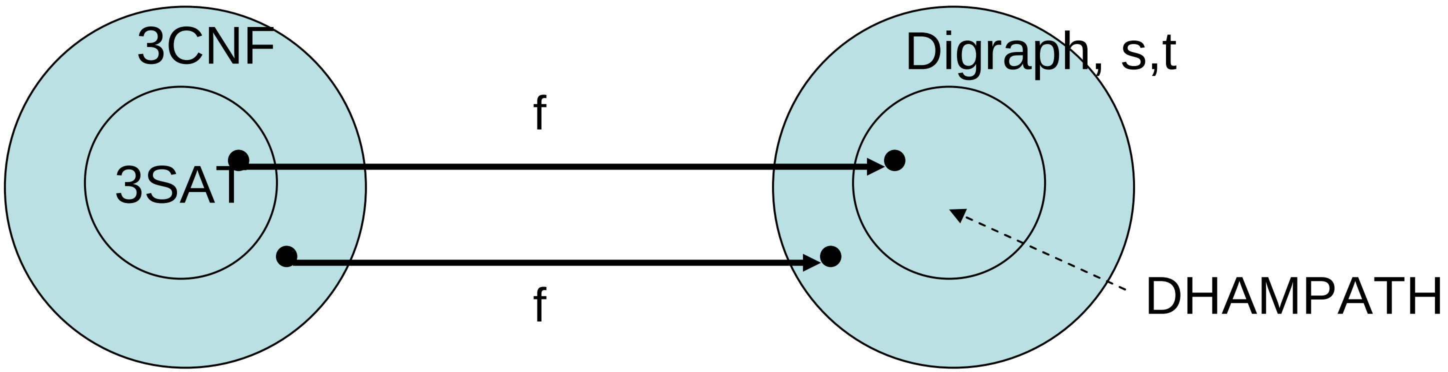


- A $\rightarrow$ B means A $\leq_p$ B.
- Hardness propagates to the right in $\leq_p$, downward along tree branches.

3SAT $\leq_p$ HAMILTONIAN PATH/CIRCUIT

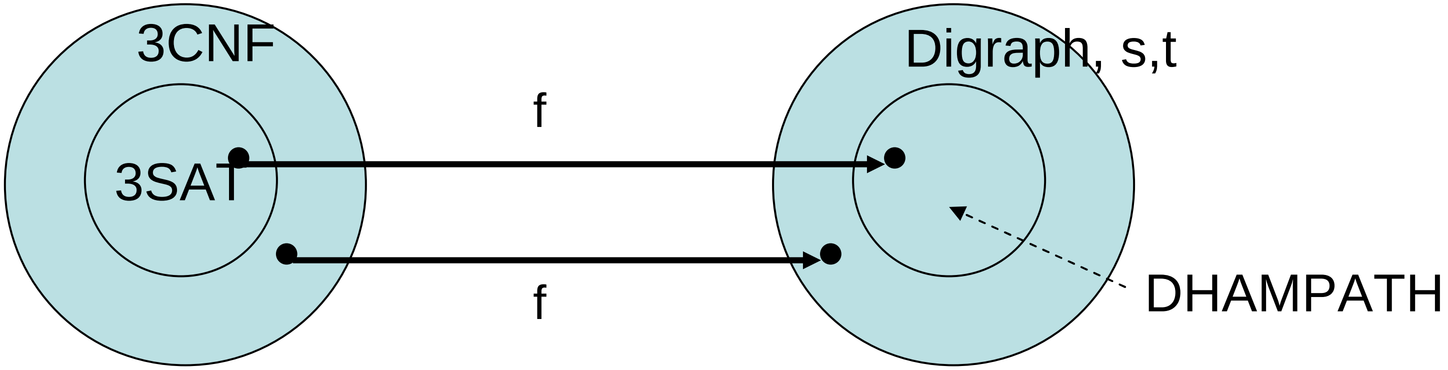
$3SAT \leq_p$ HAMILTONIAN PATH/CIRCUIT

- Two versions of the problem, for directed and undirected graphs.
- Consider directed version; undirected shown by reduction from directed version.
- DHAMPATH = { <G, s, t> | G is a directed graph, s and t are two distinct vertices, and there is a path from s to t in G that passes through each vertex of G exactly once }
- DHAMPATH $\in$ NP: Guess path and verify.
- $3SAT \leq_p$ DHAMPATH:



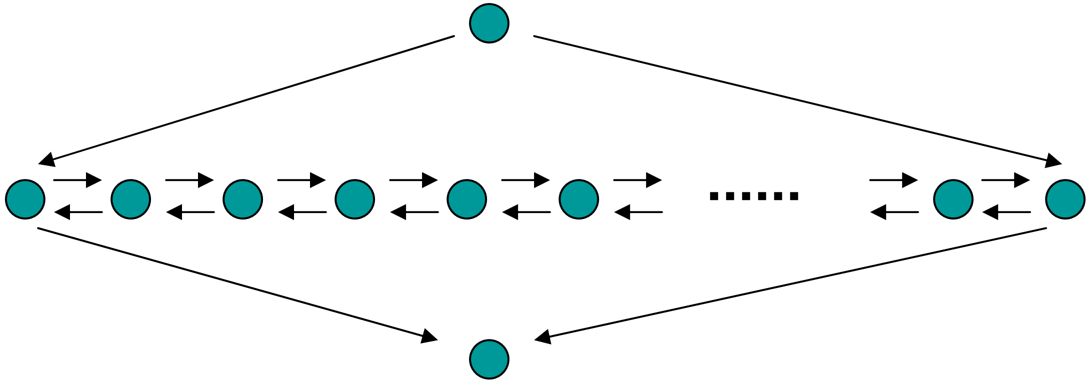
3SAT $\leq_p$ HAMILTONIAN PATH/CIRCUIT

- DHAMPATH = { <G, s, t> | G is a directed graph, s and t are two distinct vertices, and there is a path from s to t in G that passes through each vertex of G exactly once }
- 3SAT $\leq_p$ DHAMPATH:
  - Map a 3CNF formula ϕ to <G, s, t> so that ϕ is satisfiable if and only if G has a Hamiltonian path from s to t.
  - In fact, there will be a direct correspondence between a satisfying assignment for ϕ and a Hamiltonian path in G.



3SAT $\leq_p$ DHAMPATH

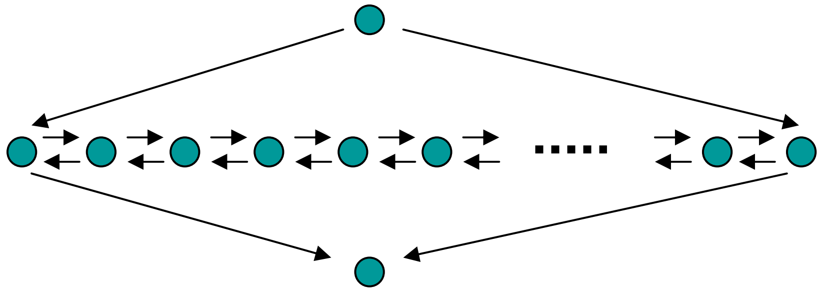
- Map a 3CNF formula ϕ to <G, s, t> so that ϕ is satisfiable if and only if G has a Hamiltonian path from s to t.
- Correspondence between satisfying assignment for ϕ and Hamiltonian path in G.
- Notation:
  - Write $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$
  - k clauses $C_1, C_2, \ldots, C_k$
  - Variables: $x_1, x_2, \ldots, x_l$
  - Each a_j, b_j, and c_j is either some x_i or some $\neg x_i$.
- Digraph is constructed from pieces (gadgets), one for each variable x_i and one for each clause C_j.
- Gadget for variable x_i:



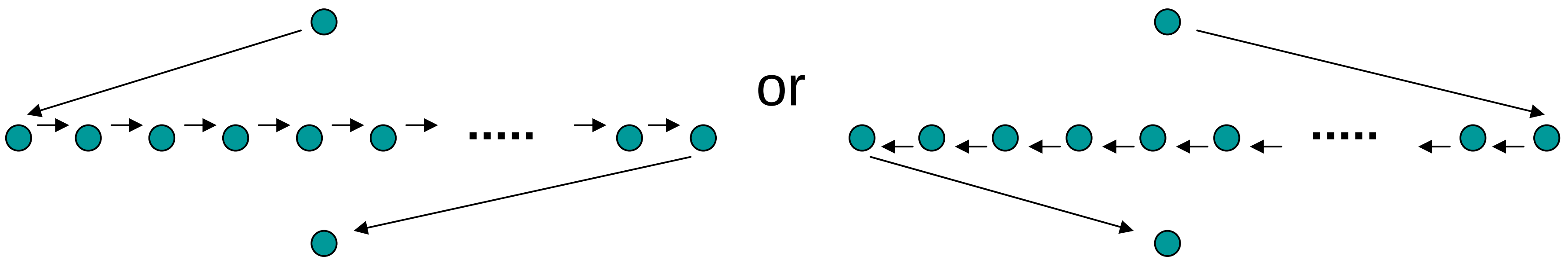
Row contains 3k+1 nodes, not counting endpoints.

$3SAT \leq_p DHAMPATH$

- Notation:
  - $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$
  - k clauses $C_1, C_2, \ldots, C_k$
  - Variables: $x_1, x_2, \ldots, x_l$
  - Each a_j, b_j, and c_j is either some x_i or some $\neg x_i$.
- Gadget for variable x_i:



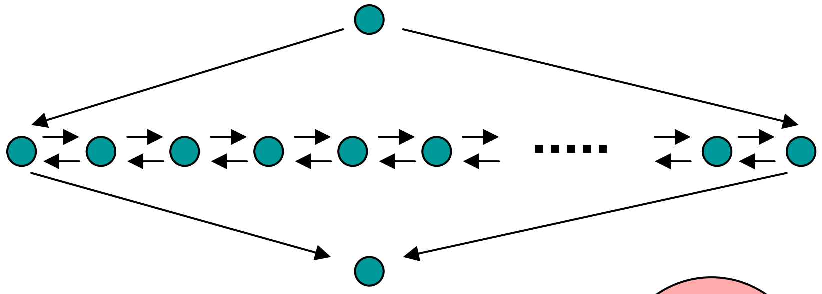
- Can get from top node to bottom node in two ways:



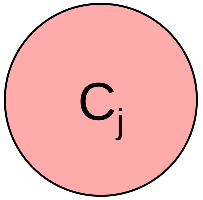
- Both ways visit all intermediate nodes.

3SAT $\leq_p$ DHAMPATH

- Notation:
  - $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$
  - k clauses $C_1, C_2, \ldots, C_k$
  - Variables: $x_1, x_2, \ldots, x_l$
  - Each a_j, b_j, and c_j is either some x_i or some $\neg x_i$.
- Gadget for variable x_i:



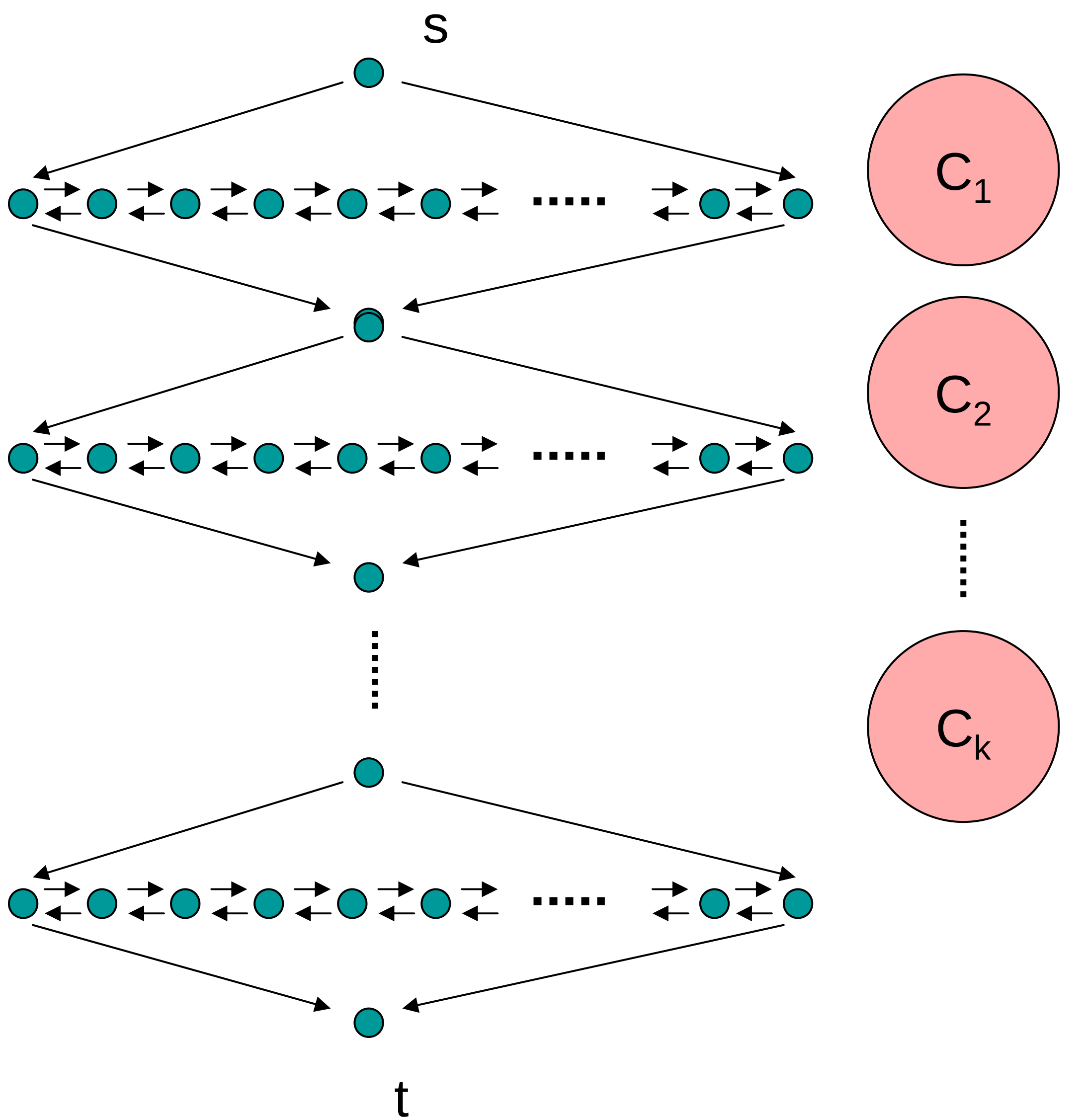
- Gadget for clause C_j:
  - Just a single node.



- Putting the pieces together:
  - Put variables' gadgets in order $x_1, x_2, \ldots, x_l$, top to bottom, identifying bottom node of each gadget with top node of the next.
  - Make s and t the overall top and bottom node, respectively

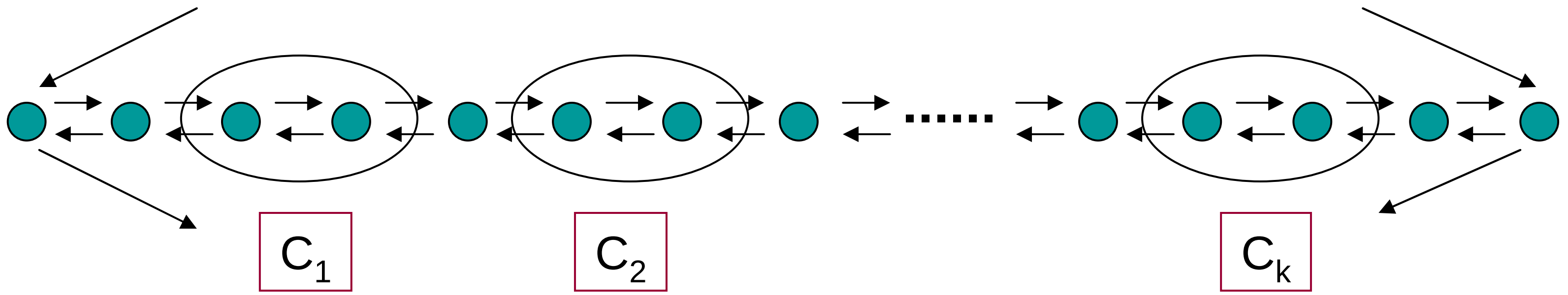
3SAT $\leq_p$ DHAMPATH

- Putting the pieces together:
  - Put variables' gadgets in order $x_1, x_2, \ldots, x_l$, identifying bottom node of each with top node of the next.
  - Make s and t the overall top and bottom node.
- We still must connect x-gadgets with C-gadgets.

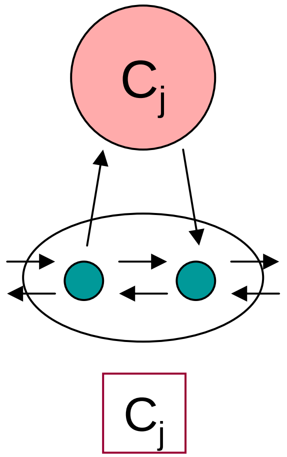


3SAT $\leq_p$ DHAMPATH

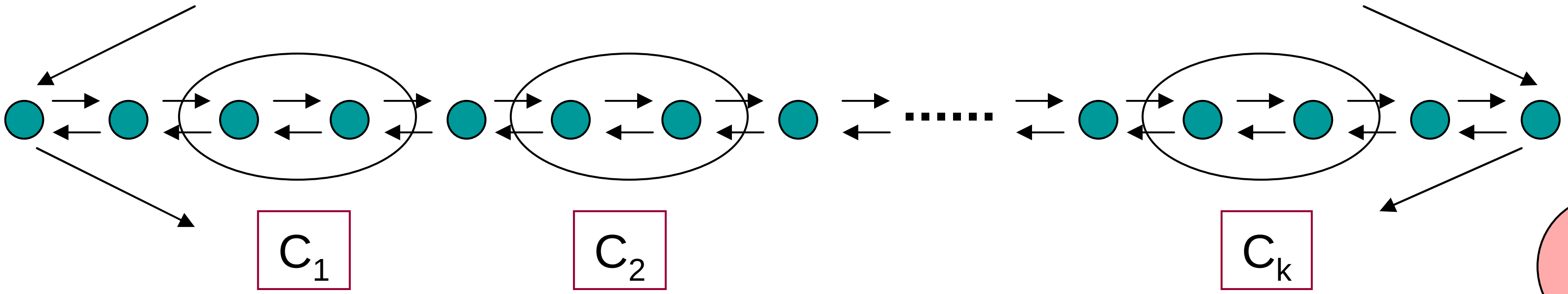
- We still must connect x-gadgets with C-gadgets.
- Divide the 3k+1 nodes in the cross-bar of x_i's gadget into k pairs, one per clause, separated by k+1 separator nodes:

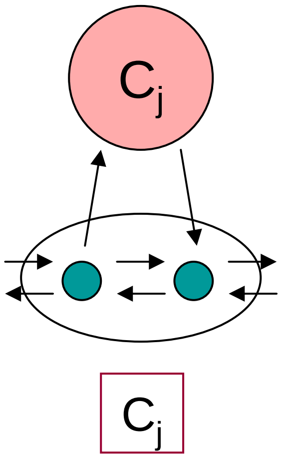


- If x_i appears in C_j, add edges between the C_j node and the nodes for C_j in the crossbar, going from left to right.
  - Allows detour to C_j while traversing crossbar left-to-right.

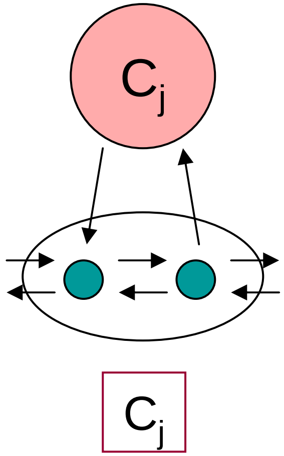


3SAT $\leq_p$ DHAMPATH





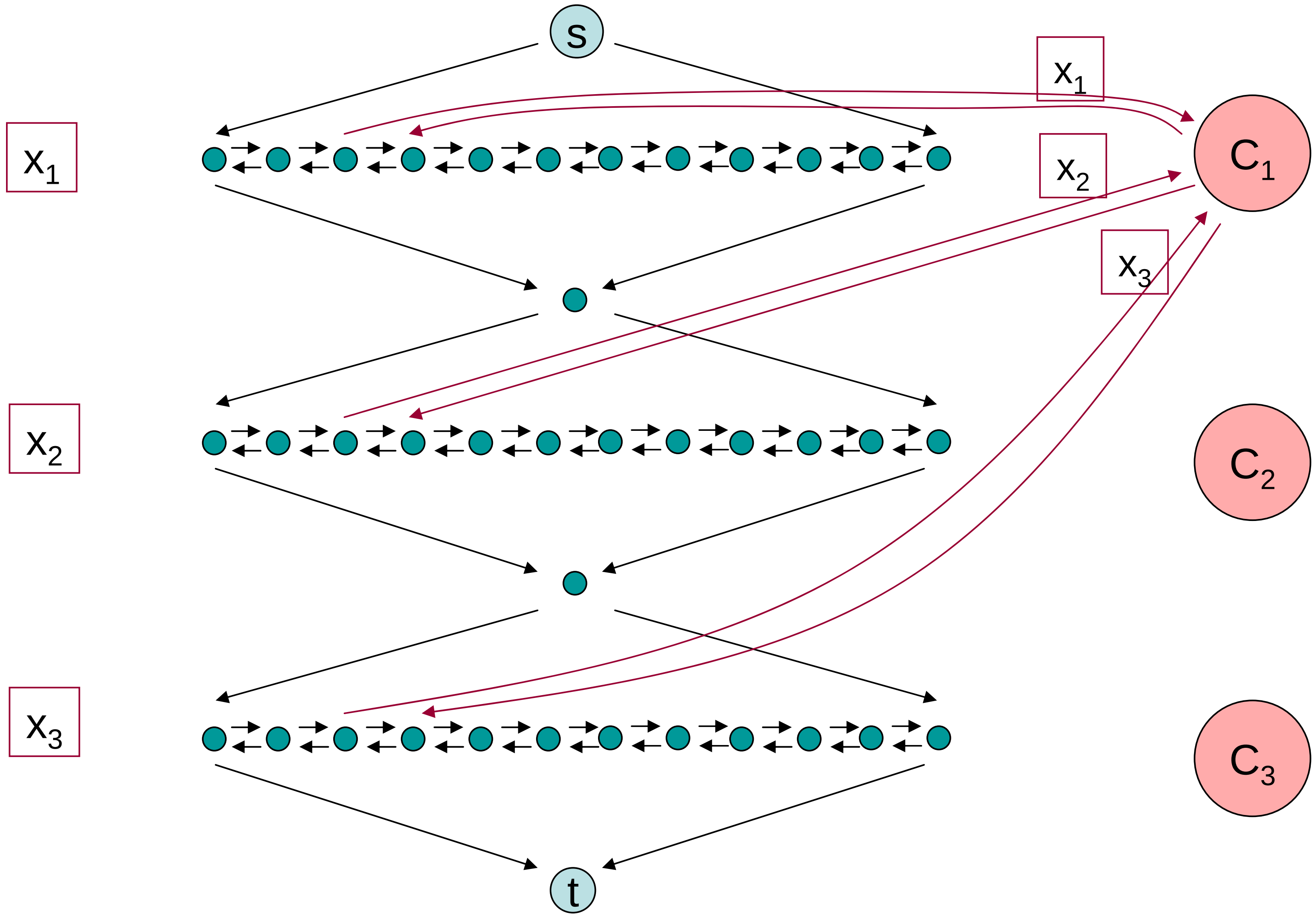
- If x_i appears in C_j, add edges L to R.
  - Allows detour to C_j while traversing crossbar L to R.



- If $\neg x_i$ appears in C_j, add edges R to L.
  - Allows detour to C_j while traversing crossbar R to L.
- If both x_i and $\neg x_i$ appear, add both sets of edges.
- This completes the construction of G, s, t.

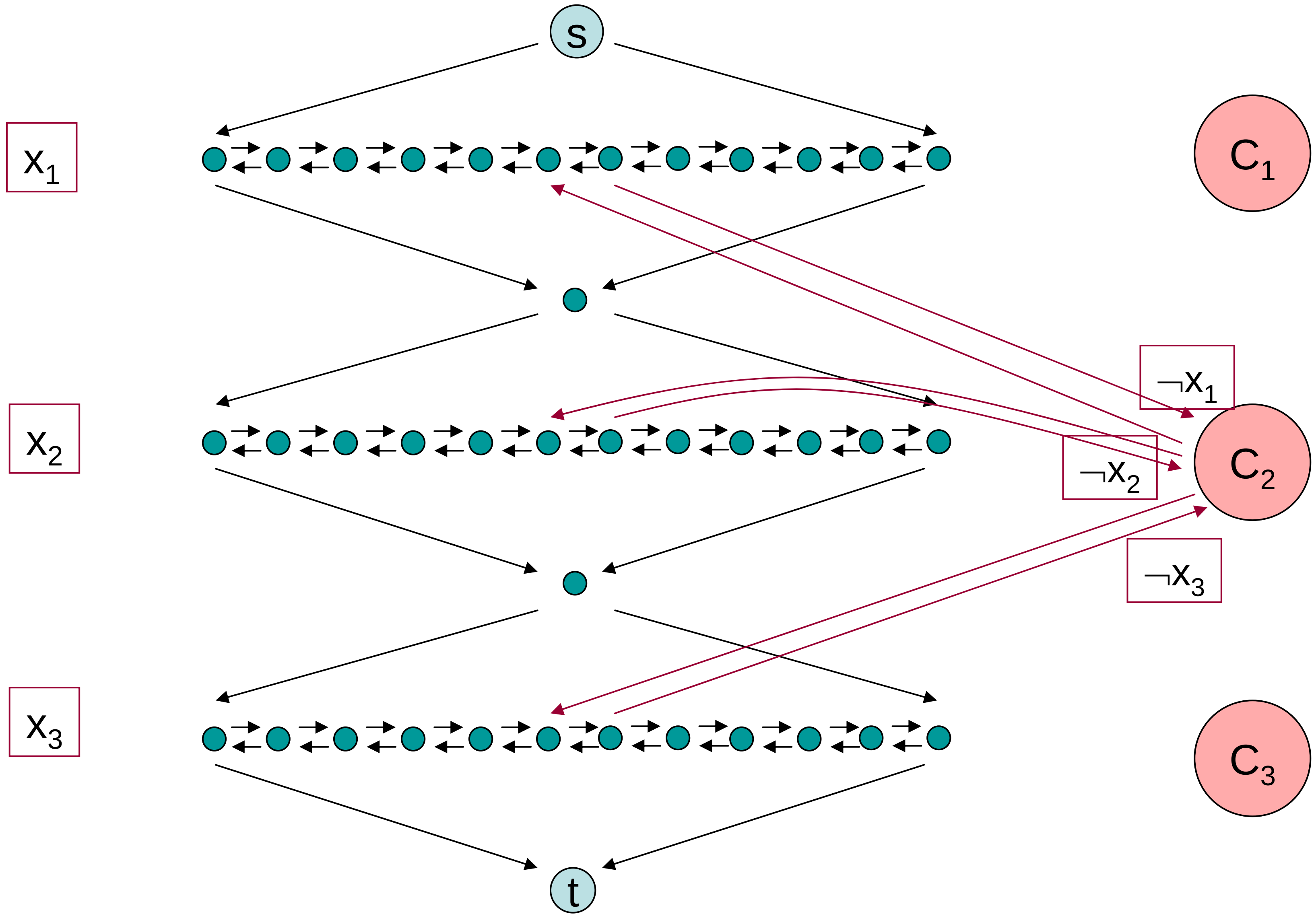
Example

- $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$



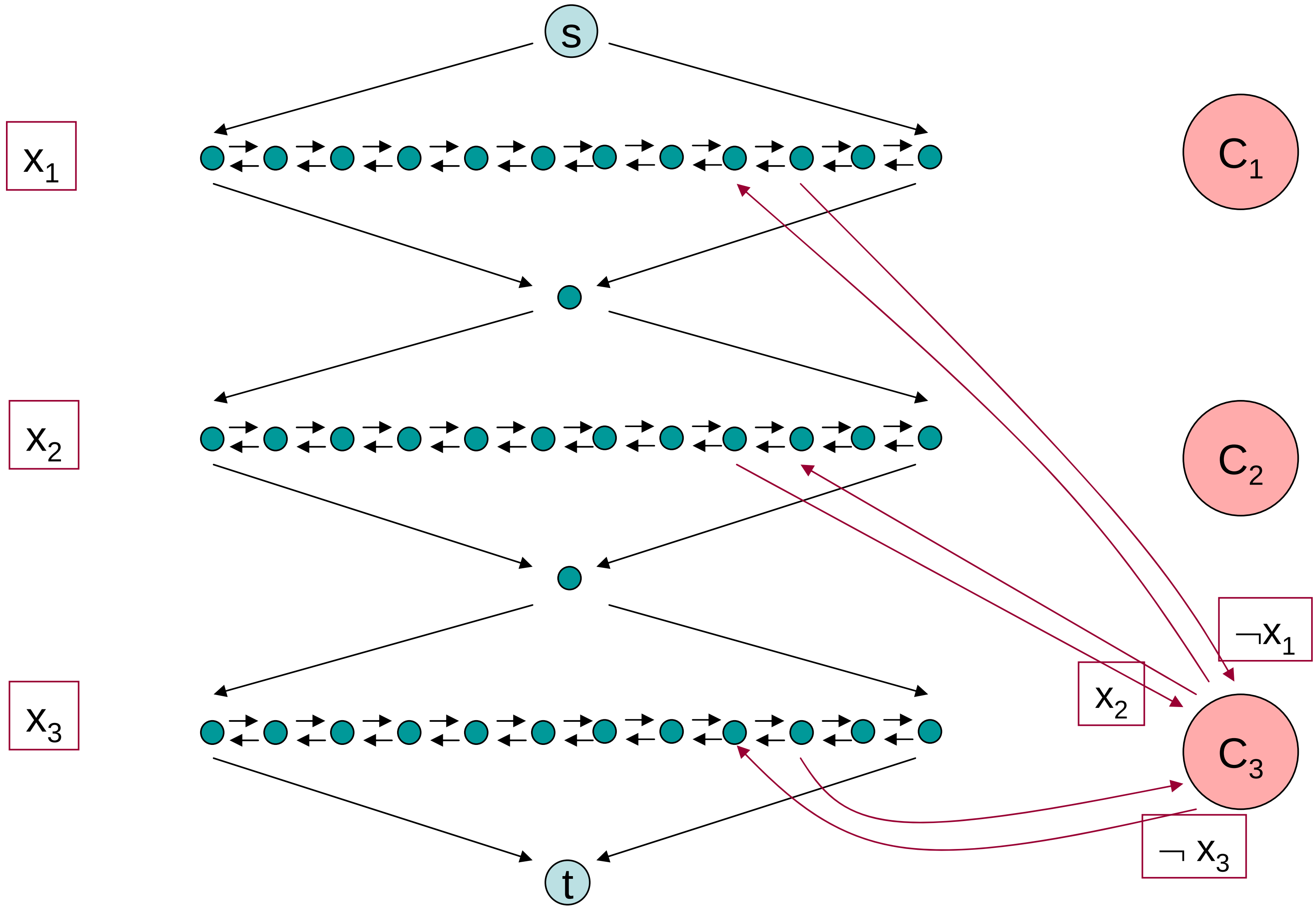
Example

- $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge \ldots \quad \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$



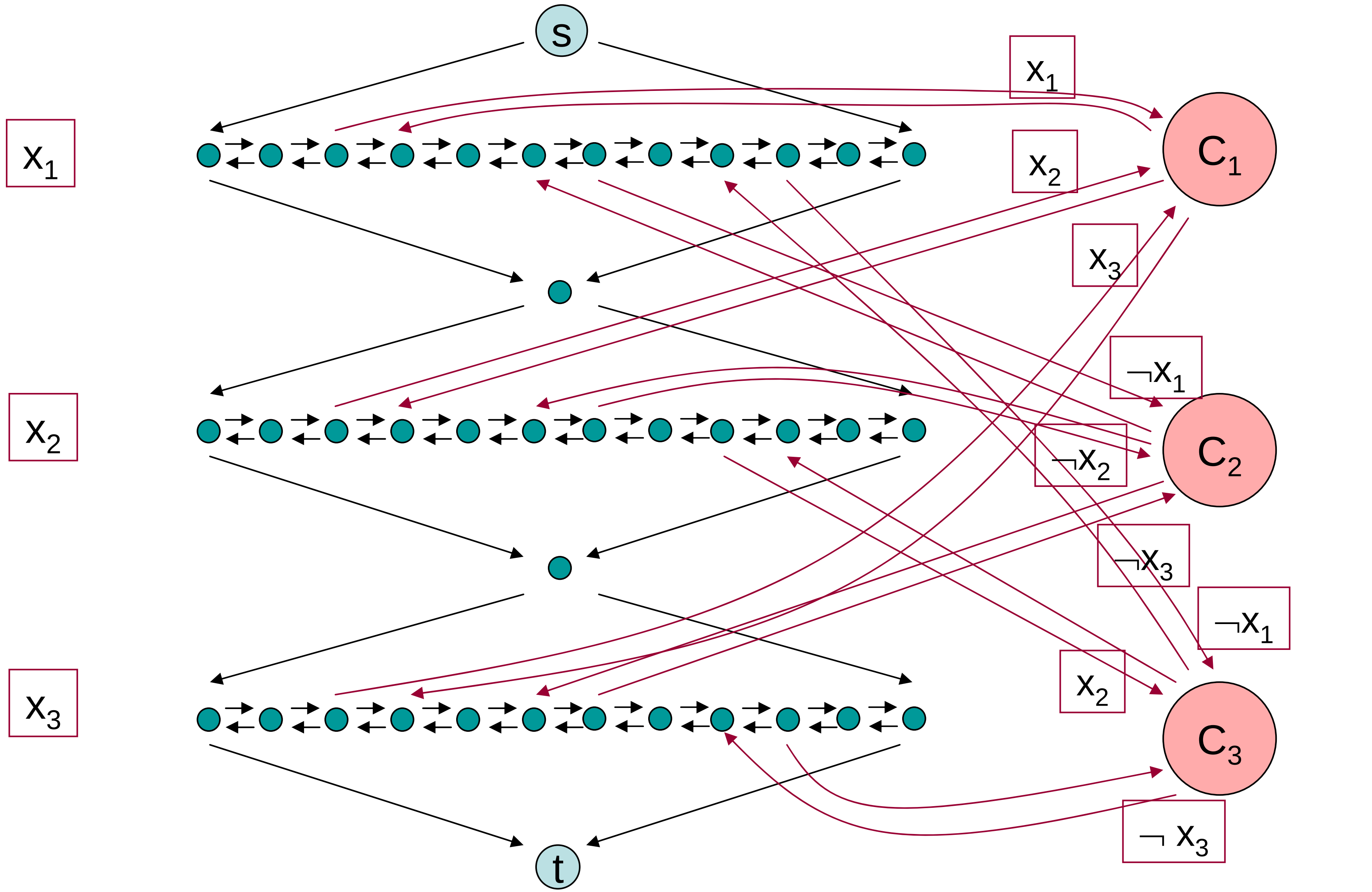
Example

- $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge \ldots \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$



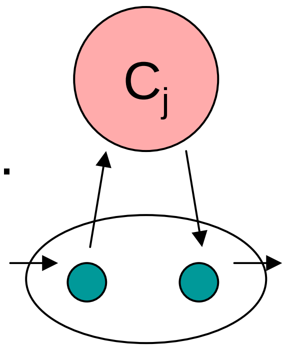
The entire graph G

- $\phi = (x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land \ldots \land (\neg x_1 \lor x_2 \lor \neg x_3)$



3SAT $\leq_p$ DHAMPATH

- Claim: ϕ is satisfiable iff the graph G has a Hamiltonian path from s to t.
- Proof: $\Rightarrow$
  - Assume ϕ is satisfiable; fix a particular satisfying assignment.
  - Follow path top-to-bottom, going
    - L to R through gadgets for x_is that are set true.
    - R to L through gadgets for x_is that are set false.
  - This visits all nodes of G except the C_j nodes.
  - For these, we must take detours.
  - For any particular clause C_j:
    - At least one of its literals must be set true; pick one.
    - If it's of the form x_i, then do:

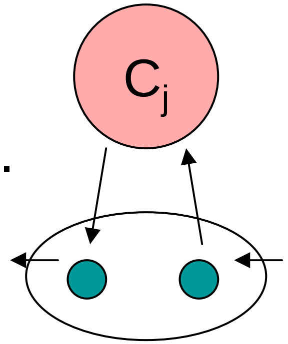


C_j pair in x_i row

- Works since x_i = true means we traverse this crossbar L to R.

3SAT $\leq_p$ DHAMPATH

- Claim: ϕ is satisfiable iff the graph G has a Hamiltonian path from s to t.
- Proof: $\Rightarrow$
  - Assume ϕ is satisfiable; fix a particular satisfying assignment.
  - Follow path top-to-bottom, going
    - L to R through gadgets for x_is that are set true.
    - R to L through gadgets for x_is that are set false.
  - This visits all nodes of G except the C_j nodes.
  - For these, we must take detours.
  - For any particular clause C_j:
    - At least one of its literals must be set true; pick one.
    - If it's of the form $\neg x_i$, then do:



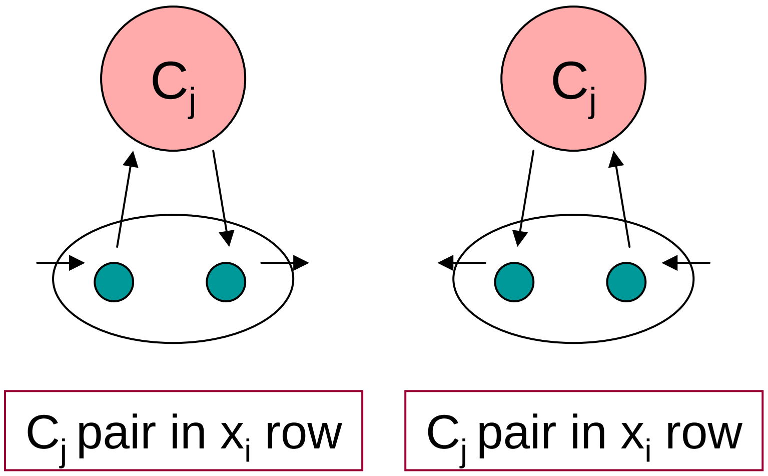
C_j pair in x_i row

    - Works since x_i = false means we traverse this crossbar R to L.

3SAT $\leq_p$ DHAMPATH

- Claim: ϕ is satisfiable iff the graph G has a Hamiltonian path from s to t.
- Proof: $\Leftarrow$
  - Assume G has a Hamiltonian path from s to t, get a satisfying assignment for ϕ.
  - If the path is “normal” (goes in order through the gadgets, top to bottom, going one way or the other through each crossbar, and detouring to pick up the C_j nodes), then define the assignment by:

    Set each x_i true if path goes L to R through x_i’s gadget, false if it goes R to L.
  - Why is this a satisfying assignment for ϕ?
  - Consider any clause C_j.
  - The path goes through its node in one of two ways:

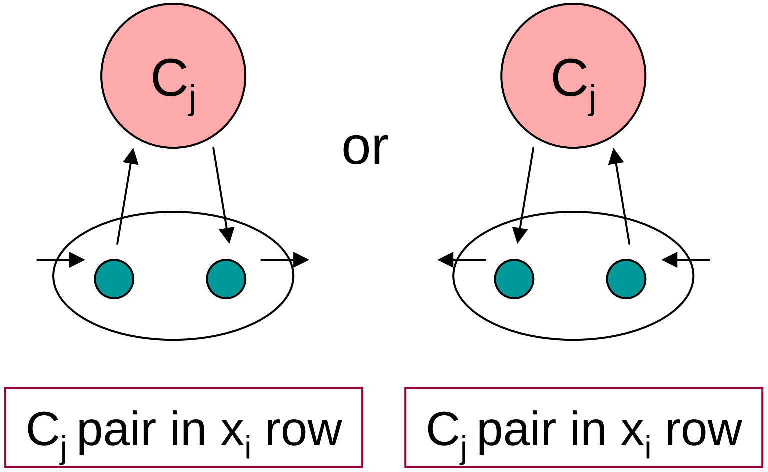


3SAT $\leq_p$ DHAMPATH

- Claim: ϕ is satisfiable iff the graph G has a Hamiltonian path from s to t.
- Proof: $\Leftarrow$
  - Assume G has a Hamiltonian path from s to t, get a satisfying assignment for ϕ.
  - If the path is "normal", then define the assignment by:

    Set each x_i true if path goes L to R through x_i's gadget, false if it goes R to L.

  - To see that this satisfies ϕ, consider any clause C_j.
  - The path goes through C_j's node by:
  - If the first, then:
    - x_i is true, since path goes L-R.
    - By the way the detour edges are set, C_j contains literal x_i.
    - So C_j is satisfied by x_i.

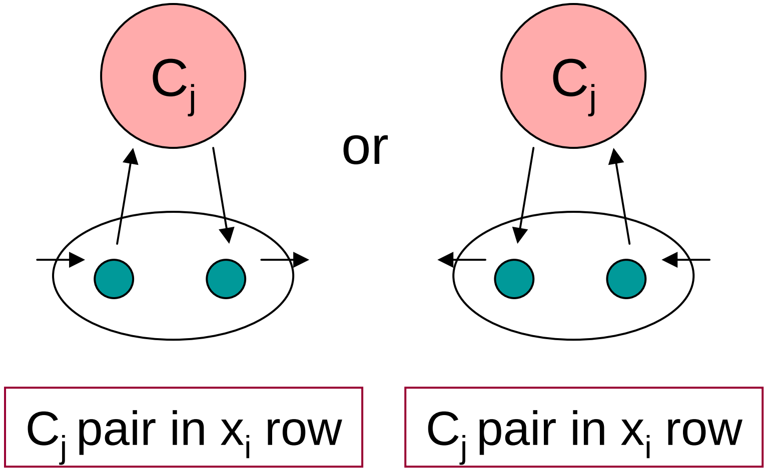


$3SAT \leq_p$ DHAMPATH

- Claim: ϕ is satisfiable iff the graph G has a Hamiltonian path from s to t.
- Proof: $\Leftarrow$
  - Assume G has a Hamiltonian path from s to t, get a satisfying assignment for ϕ.
  - If the path is "normal", then define the assignment by:

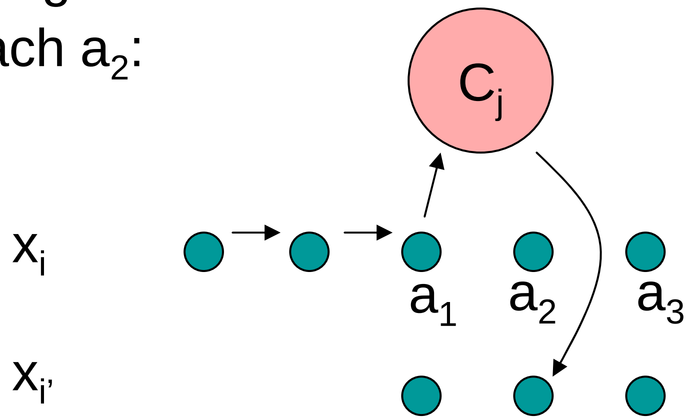
    Set each x_i true if path goes L to R through x_i's gadget, false if it goes R to L.

  - To see that this satisfies ϕ, consider any clause C_j.
  - The path goes through C_j's node by:
  - If the second, then:
    - x_i is false, since path goes R-L.
    - By the way the detour edges are set, C_j contains literal $\neg x_i$.
    - So C_j is satisfied by $\neg x_i$.



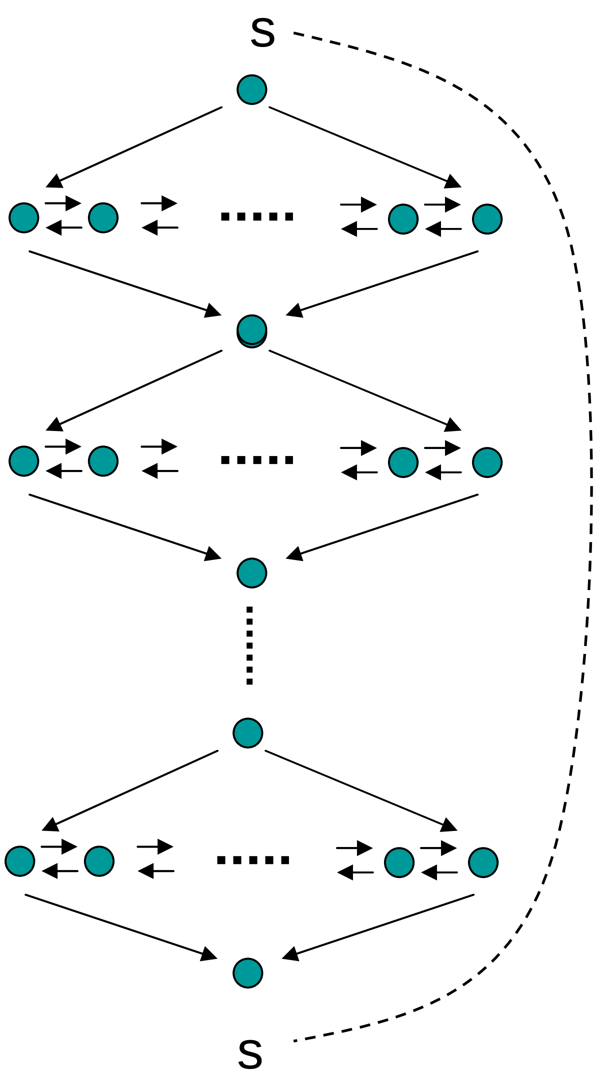
3SAT $\leq_p$ DHAMPATH

- Claim: ϕ is satisfiable iff the graph G has a Hamiltonian path from s to t.
- Proof: $\Leftarrow$
  - Assume G has a Hamiltonian path from s to t.
  - If the path is normal, then it yields a satisfying assignment.
  - It remains to show that the path is normal (goes in order through the gadgets, top to bottom, going one way or the other through each crossbar, and detouring to pick up the C_j nodes),
  - The only problem (hand-waving) is if a detour doesn't work right, but jumps from one gadget to another, e.g.:
  - But then the Ham. path could never reach a_2:
    - Can reach a_2 only from a_1, a_3, and (possibly) C_j.
    - But a_1 and C_j already lead elsewhere.
    - And reaching a_2 from a_3 leaves nowhere to go from a_2, stuck.



Summary: DHAMPATH

- We have proved 3SAT $\leq_p$ DHAMPATH.
- So DHAMPATH is NP-complete.
- Can prove similar result for DHAMCIRCUIT = { <G> | G is a directed graph, and there is a circuit in G that passes through each vertex of G exactly once }
- Theorem: 3SAT $\leq_p$ DHAMCIRCUIT.
- Proof:
  - Same construction, but wrap around, identifying s and t nodes.
  - Now a satisfying assignment for ϕ corresponds to a Hamiltonian circuit.



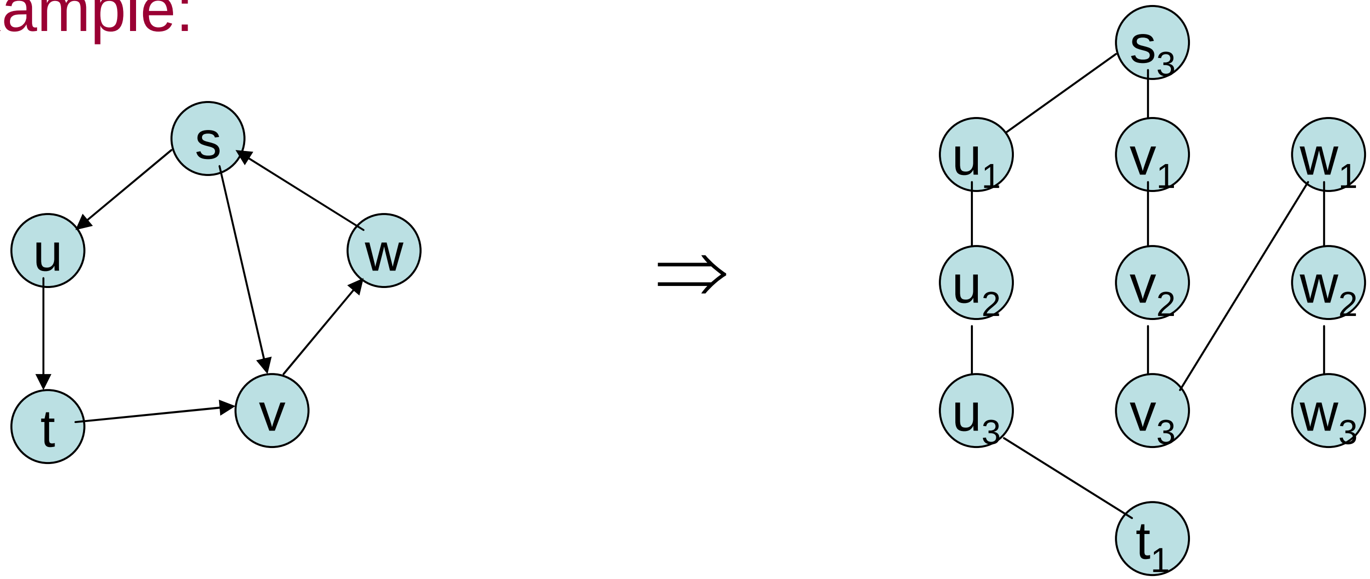
Identify these two s nodes.

UHAMPATH and UHAMCIRCUIT

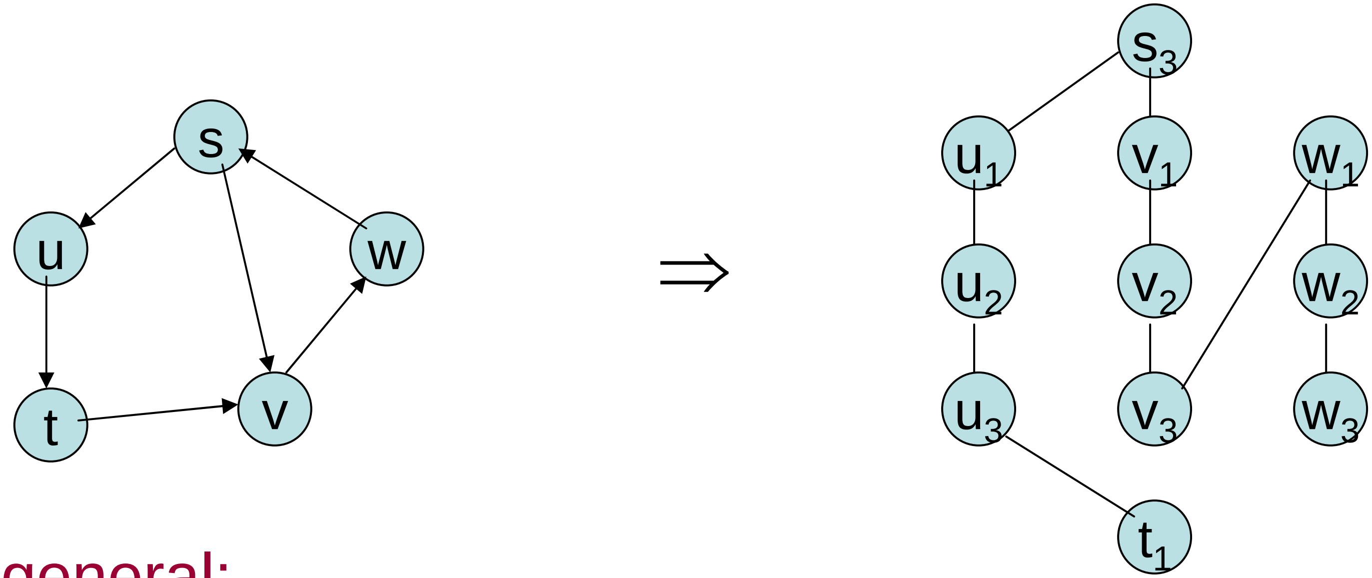
- Same questions about paths/circuits in undirected graphs.
- UHAMPATH = { <G, s, t> | G is an undirected graph, s and t are two distinct vertices, and there is a path from s to t in G that passes through each vertex of G exactly once }
- UHAMCIRCUIT = { <G> | G is an undirected graph, and there is a circuit in G that passes through each vertex of G exactly once }
- Theorem: Both are NP-complete.
- Obviously in NP.
- To show NP-hardness, reduce the digraph versions of the problems to the undirected versions---no need to consider Boolean formulas again.
  - DHAMPATH $\leq_p$ UHAMPATH
  - DHAMCIRCUIT $\leq_p$ UHAMCIRCUIT

DHAMPATH $\leq_p$ UHAMPATH

- UHAMPATH = { <G, s, t> | G is an undirected graph, s and t are two distinct vertices, and there is a path from s to t in G that passes through each vertex of G exactly once }
- Map <G, s, t> (directed) to <G′, s′, t′> (undirected) so that <G, s, t> $\in$ DHAMPATH iff <G′, s′, t′> $\in$ UHAMPATH.
- Example:

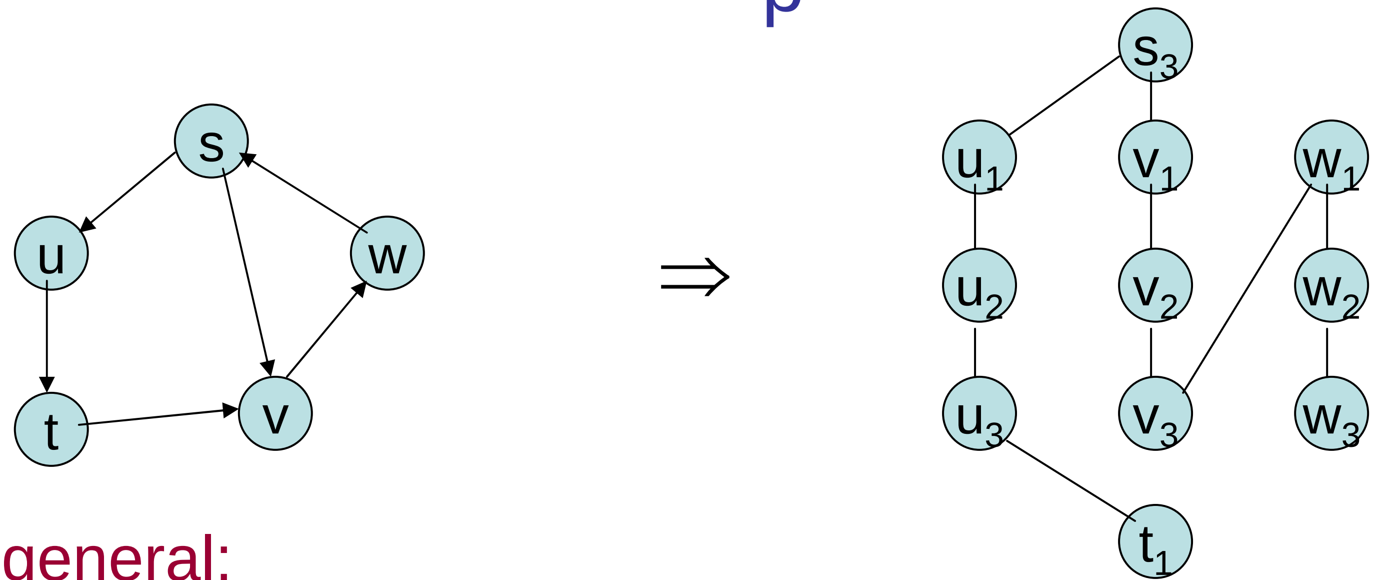


DHAMPATH $\leq_p$ UHAMPATH



- In general:
  - Replace each vertex x other than s, t with vertices x_1, x_2, x_3, connected in a line.
  - Replace s with just s_3, t with just t_1.
  - For each directed edge from x to y in G, except incoming edges of s and outgoing edges of t, include undirected edge between x_3 and y_1.
  - Don't include anything for incoming edges of s or outgoing edges of t---not needed since they can't be part of a Ham. path in G from s to t.

DHAMPATH $\leq_p$ UHAMPATH



- In general:
  - Replace each vertex x other than s, t with x_1---x_2---x_3.
  - Replace s with s_3, t with t_1.
  - For each directed edge from x to y in G, except incoming edges of s and outgoing edges of t, include x_3---y_1.
- G′ = the resulting undirected graph; s′ = s_3; t′ = t_1
- Claim G has directed Hamiltonian path from s to t iff G′ has an undirected Hamiltonian path from s′ to t′.
- Idea: Indices 1,2,3 enforce consistent direction of traversal.
- Proof LTTR (in book).

Summary: UHAMPATH

- We have proved DHAMPATH $\leq_p$ UHAMPATH.
- So UHAMPATH is NP-complete.
- Can prove similar result for

  UHAMCIRCUIT = { <G> | G is an undirected graph, and there is a circuit in G that passes through each vertex of G exactly once }
- Theorem: DHAMCIRCUIT $\leq_p$ UHAMCIRCUIT.
- Proof:
  - Similar construction.

The Traveling Salesman Problem

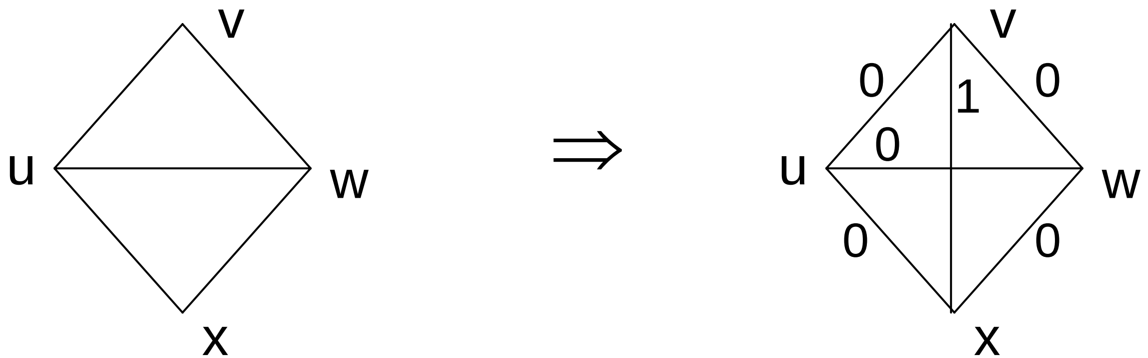
Traveling Salesman Problem (TSP)

- Variant of UHAMCIRCUIT.
- n cities = vertices, in a complete (undirected) graph.
- Each edge (u,v) has a cost, c(u,v), a nonnegative integer.
- Salesman should visit all cities, each just once, at low cost.
- Express as a language:

  TSP = { <G, c, k> | G = (V,E) is a complete graph, c: E $\rightarrow$ N, k $\in$ N, and G has a cycle visiting each node exactly once, with total cost $\leq$ k }
- Theorem: TSP is NP-complete.
- Proof:
  - TSP $\in$ NP: Guess tour and verify.
  - TSP is NP-hard: Show UHAMCIRCUIT $\leq_p$ TSP.
  - Map <G> (undirected graph) to <G′, c′, k′> so that G has a Ham. circuit iff G′ with cost function c′ has a tour of total cost at most k′.

UHAMCIRCUIT $\leq_p$ TSP

- TSP = { <G, c, k> | G = (V,E) is a complete graph, c: E $\rightarrow$ N, k $\in$ N, and G has a cycle visiting each node exactly once, with total cost $\leq$ k }
- Map <G> (undirected graph) to <G′, c′, k′> so that G has a Ham. circuit iff G′ with cost function c′ has a tour of total cost $\leq$ k′.
- Define mapping so that a Ham. circuit corresponds closely with a tour of cost $\leq$ k′.
  - G′ = (V′, E′), where V′ = V, all vertices of G, E′ = all edges (complete graph).
  - c′(u,v) = 1 if (u, v) $\notin$ E, 0 if (u,v) $\in$E.
  - k′ = 0.
- Example:



UHAMCIRCUIT $\leq_p$ TSP

- TSP = { <G, c, k> | G = (V,E) is a complete graph, c: E $\rightarrow$ N, k $\in$ N, and G has a cycle visiting each node exactly once, with total cost $\leq$ k }
- Map <G> (undirected graph) to <G′, c′, k′>:
  - G′ = (V′, E′), where V′ = V, all vertices of G, E′ = all edges (complete graph).
  - c′(u,v) = 1 if (u, v) $\notin$ E, 0 if (u,v) $\in$ E.
  - k′ = 0.
- Claim: G has a Ham. circuit iff G′ with cost function c′ has a tour of total cost $\leq$ k′.
- Proof:

  $\Rightarrow$ If G has a Ham. circuit, all its edges have cost 0 in G′ with c′, so we have a circuit of cost 0 in G′.

  $\Leftarrow$ Tour of cost 0 in G′ must consist of edges of cost 0, which are edges in G.

More Examples, Revisited

SUBSET-SUM

- SUBSET-SUM = {<S,t> | S is a multiset of N, t $\in$ N, and t is expressible as the sum of some of the elements of S }
- Example: S = { 2, 2, 4, 5, 5, 7 }, t = 13

  <S, t > $\in$ SUBSET-SUM, because 7 + 4 + 2 = 13
- Theorem: SUBSET-SUM is NP-complete.
- Proof:
  - Show 3SAT $\leq_p$ SUBSET-SUM.
  - Tricky, detailed, see book.

PARTITION

- PARTITION = { <S> | S is a multiset of N and S can be split into multisets S_1 and S_2 having equal sums }
- Example: S = { 2, 2, 4, 5, 5, 7 }

S $\notin$ PARTITION, since the sum is odd

- Example: T = { 2, 2, 5, 6, 9, 12 }

T $\in$ PARTITION, since 2 + 2 + 5 + 9 = 6 + 12.

- Theorem: PARTITION is NP-complete.
- Proof:
  - Show SUBSET-SUM $\leq_p$ PARTITION.
  - Simple…in recitation?

MULTIPROCESSOR SCHEDULING

- MPS = { <S, m, D > |
  – S is a multiset of N (represents durations for tasks),
  – m $\in$ N (number of processors), and
  – D $\in$ N (deadline),

  and S can be written as $S_1 \cup S_2 \cup \ldots \cup S_m$ such that, for every i, $\text{sum}(S_i) \le D$ }
- Theorem: MPS is NP-complete.
- Proof:
  – Show PARTITION $\le_p$ MPS.
  – Simple…in recitation?

Next time…

- Probabilistic Turing Machines and Probabilistic Time Complexity Classes
- Reading:
  - Sipser Section 10.2

6.045J / 18.400J Automata, Computability, and Complexity
Spring 2011