

CSC 611: Analysis of Algorithms

Lecture 16

NP-Completeness

NP-Completeness

- **Polynomial-time algorithms**
 - on inputs of size n , worst-case running time is $O(n^k)$, for a constant k
- Not all problems can be solved in polynomial time
 - Some problems cannot be solved by any computer no matter how much time is provided (Turing's Halting problem) – such problems are called **undecidable**
 - Some problems can be solved but not in $O(n^k)$

Class of “P” Problems

- **Class P** consists of (decision) problems that are solvable in polynomial time:

there exists an algorithm that can solve the problem in $O(n^k)$, k constant

- Problems in P are also called **tractable**
- Problems not in P are also called **intractable**
 - Can be solved in reasonable time only for small inputs

Optimization & Decision Problems

- **Decision problems**
 - Given an input and a question regarding a problem, determine if the answer is yes or no
- **Optimization problems**
 - Find a solution with the “best” value
- Optimization problems can be cast as decision problems that are easier to study
 - *E.g.:* Shortest path: G = unweighted directed graph
 - Find a path between u and v that uses the fewest edges
 - *Does a path exist from u to v consisting of at most k edges?*

Nondeterministic Algorithms

Nondeterministic algorithm = two stage procedure:

- 1) Nondeterministic (“guessing”) stage:
generate an arbitrary string that can be thought of as a candidate solution (“certificate”)
 - 2) Deterministic (“verification”) stage:
take the certificate and the instance to the problem and return YES if the certificate represents a solution
- **Nondeterministic polynomial (NP)** = verification stage is polynomial

Class of “NP” Problems

- **Class NP** consists of problems that are verifiable in polynomial time (i.e., could be solved by nondeterministic polynomial algorithms)
 - If we were given a “certificate” of a solution, we could verify that the certificate is correct in time polynomial to the size of the input

E.g.: Hamiltonian Cycle

- **Given:** a directed graph $G = (V, E)$,
determine a simple cycle that contains
each vertex in V

- Each vertex can only be visited once

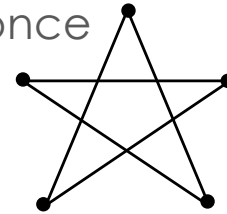
- **Certificate:**

- Sequence: $\langle v_1, v_2, v_3, \dots, v_{|V|} \rangle$

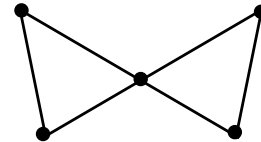
- **Verification:**

- 1) $(v_i, v_{i+1}) \in E$ for $i = 1, \dots, |V|$

- 2) $(v_{|V|}, v_1) \in E$

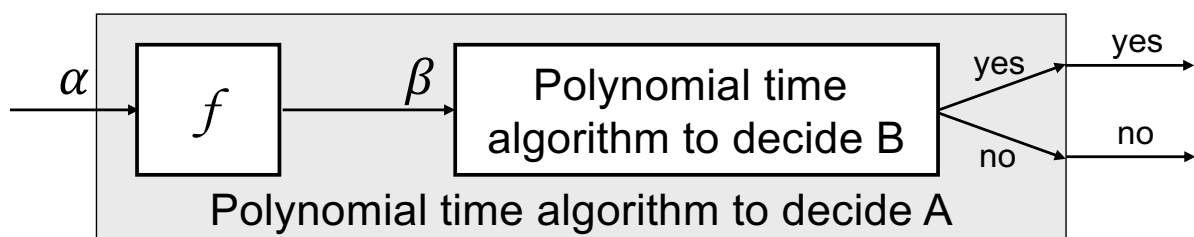


hamiltonian



not
hamiltonian

Polynomial Reduction Algorithm



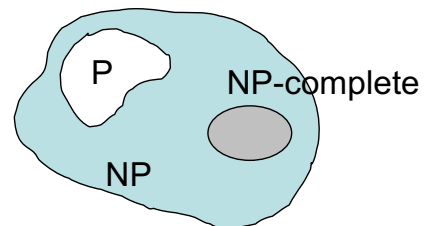
- To solve a decision problem A in polynomial time
 1. Use a polynomial time reduction algorithm to transform A into B
 2. Run a known polynomial time algorithm for B
 3. Use the answer for B as the answer for A

Reductions

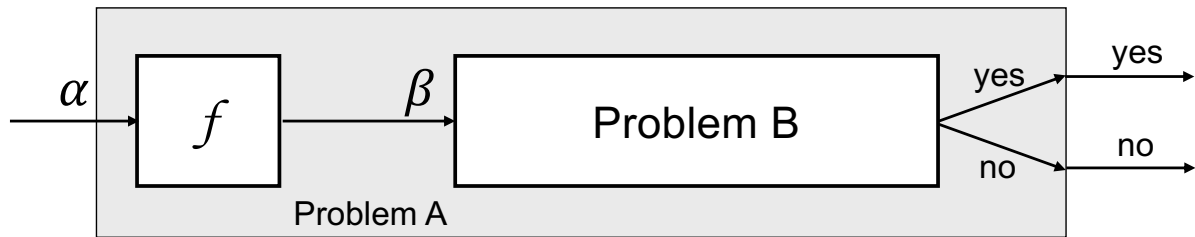
- Given two problems A, B, we say that A is **reducible** to B ($A \leq_p B$) if:
 - There exists a function f that converts the input of A to an input of B in polynomial time
 - $A(i) = \text{YES} \Leftrightarrow B(f(i)) = \text{YES}$ (for every input i)

NP-Completeness

- A problem B is **NP-complete** if:
 - $B \in \mathbf{NP}$
 - $A \leq_p B$ for all $A \in \mathbf{NP}$
- If B satisfies only property 2) we say that B is **NP-hard**
- No polynomial time algorithm has been discovered for an **NP-Complete** problem
- No one has ever proven that no polynomial time algorithm can exist for any **NP-Complete** problem



Reduction and NP-Completeness



- Suppose we know:
 - No polynomial time algorithm exists for problem A
 - We have a polynomial reduction f from A to B
- \Rightarrow No polynomial time algorithm exists for B

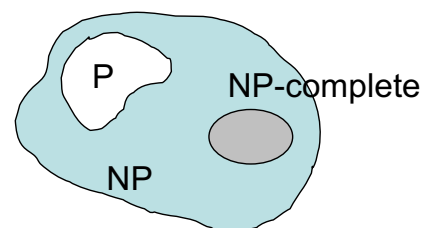
Proving NP-Completeness

Theorem: If A is NP-Complete and $A \leq_p B$

\Rightarrow B is NP-Hard

In addition, if $B \in \text{NP}$

\Rightarrow B is NP-Complete



Proof: Assume that $B \in \text{P}$

Since $A \leq_p B \Rightarrow A \in \text{P}$ contradiction, so $B \notin \text{P}$

If $B \in \text{NP} \Rightarrow B \in \text{NP-Complete}$ (by definition of NP-C)

If $B \notin \text{NP} \Rightarrow B \in \text{NP-Hard}$ (by definition of NP-H)

Proving NP-Completeness

1. Prove that the problem B is in NP
 - A randomly generated string can be checked in polynomial time to determine if it represents a solution
2. Show that **one known** NP-Complete problem can be transformed to B in polynomial time
 - No need to check that **all** NP-Complete problems are reducible to B

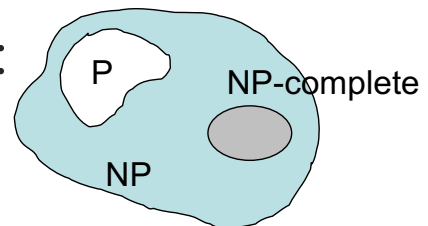
CSC611/Lecture 17

13

Is $P = NP$?

- Any problem in P is also in NP:

$$P \subseteq NP$$



- We can solve problems in P, even without having a certificate
- The big (and open question) is whether $P = NP$

Theorem: If any NP-Complete problem can be solved in polynomial time \Rightarrow then $P = NP$.

CSC611/Lecture 17

14

P & NP-Complete Problems

- **Shortest simple path**

- Given a graph $G = (V, E)$ find a **shortest** path from a source to all other vertices
- Polynomial solution: $O(VE)$

- **Longest simple path**

- Given a graph $G = (V, E)$ find a **longest** path from a source to all other vertices
- NP-complete

CSC611/Lecture 17

15

P & NP-Complete Problems

- **Euler tour**

- Given $G = (V, E)$ a connected, directed graph, find a cycle that traverses each edge of G exactly once (may visit a vertex multiple times)
- Polynomial solution $O(E)$

- **Hamiltonian cycle**

- $G = (V, E)$ a connected, directed graph find a cycle that visits each vertex of G exactly once
- NP-complete

CSC611/Lecture 17

16

Boolean Formula Satisfiability

Formula Satisfiability Problem: a boolean formula Φ composed of

1. n boolean variables: x_1, x_2, \dots, x_n
2. m boolean connectives: \wedge (AND), \vee (OR), \neg (NOT), \rightarrow (implication), \leftrightarrow (equivalence, “if and only if”)
3. Parentheses

Satisfying assignment: an assignment of values (0, 1) to variables x_i that causes Φ to evaluate to 1

E.g.: $\Phi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$

Certificate: $x_1 = 1, x_2 = 0 \Rightarrow \Phi = 1 \wedge 1 \wedge 1 = 1$

- Formula Satisfiability is first to be proven NP-Complete

3-CNF Satisfiability

3-CNF (clause normal form) Satisfiability Problem:

- n boolean variables: x_1, x_2, \dots, x_n
- **Literal:** x_i or $\neg x_i$ (a variable or its negation)
- **Clause:** c_j = an **OR** of **three literals**
- Formula: $\Phi = c_1 \wedge c_2 \wedge \dots \wedge c_m$ (m clauses)

• *E.g.:*

$$\Phi = (x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

- **3-CNF** is NP-Complete

Clique

Clique Problem:

- Undirected graph $G = (V, E)$
- **Clique:** a subset of vertices in V all connected to each other by edges in E (i.e., forming a complete graph)
- **Size of a clique:** number of vertices it contains

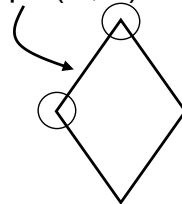
Optimization problem:

- Find a clique of maximum size

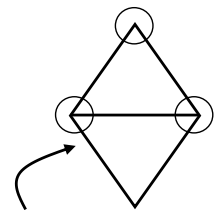
Decision problem:

- Does G have a clique of size k ?

Clique($G, 2$) = YES
Clique($G, 3$) = NO

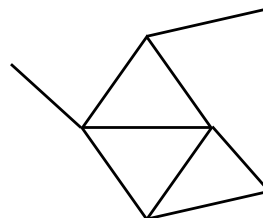


Clique($G, 3$) = YES
Clique($G, 4$) = NO



Clique Verifier

- **Given:** an undirected graph $G = (V, E)$
- **Problem:** Does G have a clique of size k ?
- **Certificate:**
 - A set of k nodes
- **Verifier:**
 - Verify that for all pairs of vertices in this set there exists an edge in E
- Let's prove that the clique problem is NP-Complete



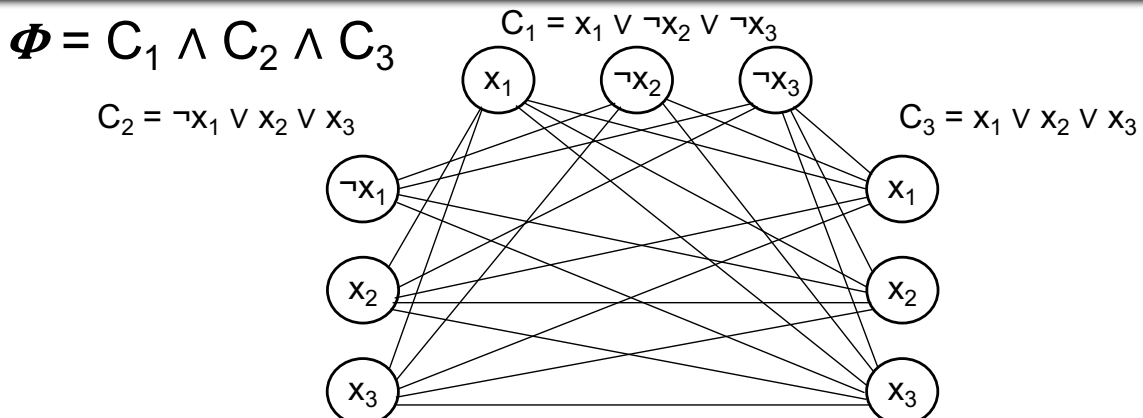
3-CNF \leq_p Clique

- Start with an instance of 3-CNF:
 - $\Phi = C_1 \wedge C_2 \wedge \dots \wedge C_k$ (k clauses)
 - Each clause C_r has three literals: $C_r = l_1^r \vee l_2^r \vee l_3^r$
- **Idea:**
 - Construct a graph G such that Φ is satisfiable if and only if G has a clique of size k

CSC611/Lecture 17

21

3-CNF \leq_p Clique



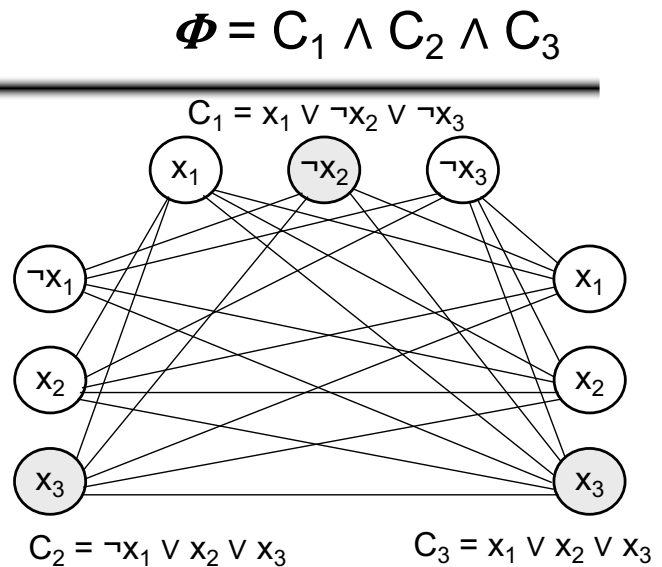
- For each clause $C_r = l_1^r \vee l_2^r \vee l_3^r$ place a triple of vertices v_1^r, v_2^r, v_3^r in V
- Put an edge between two vertices v_i^r and v_j^s if:
 - v_i^r and v_j^s are in different triples
 - l_i^r is not the negation of l_j^s

CSC611/Lecture 17

22

3-CNF \leq_p Clique

- Suppose Φ has a satisfying assignment
 - Each clause C_r has some literal assigned to 1 – this corresponds to a vertex v_i^r
 - Picking one such literal from each $C_r \Rightarrow$ a set V' of k vertices



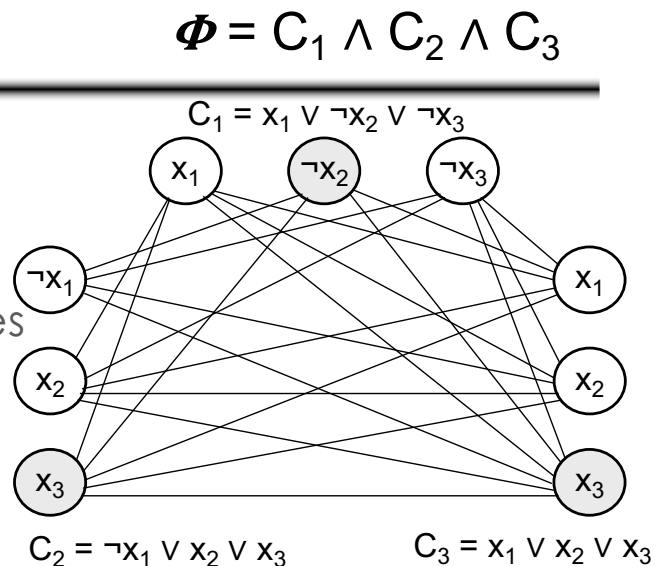
- Claim: V' is a clique
 - $\forall v_i^r, v_j^s \in V'$ the corresponding literals are 1 \Rightarrow cannot be complements
 - by the design of G the edge $(v_i^r, v_j^s) \in E$

CSC611/Lecture 17

23

3-CNF \leq_p Clique

- Suppose G has a clique of size k
 - No edges between nodes in the same clause
 - Clique contains only one vertex from each clause
 - Assign 1 to vertices in the clique (we can do it because the literals of these vertices cannot belong to complementary literals)
 - Each clause is satisfied $\Rightarrow \Phi$ is satisfied

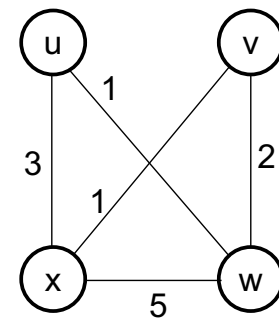


CSC611/Lecture 17

24

The Traveling Salesman Problem

- $G = (V, E)$, $|V| = n$, vertices represent cities
- **Cost:** $c(i, j)$ = cost of travel from city i to city j
- **Problem:** salesman should make a tour (hamiltonian cycle):
 - Visit each city only once
 - Finish at the city he started from
 - Total cost is minimum
- TSP = tour with cost at most k



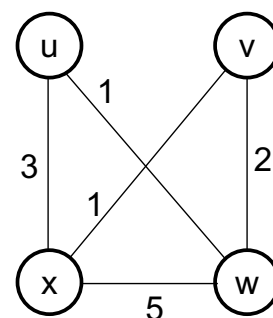
$\langle u, w, v, x \rangle$

CSC611/Lecture 17

25

TSP \in NP

- **Certificate:**
 - Sequence of n vertices, cost
 - E.g.: $\langle u, w, v, x \rangle, 7$
- **Verification:**
 - Each vertex occurs only once
 - Sum of costs is at most k



CSC611/Lecture 17

26

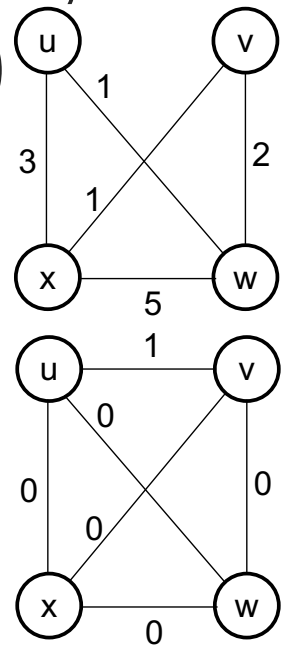
HAM-CYCLE \leq_p TSP

- Start with a Hamiltonian cycle $G = (V, E)$
- Form the complete graph $G' = (V, E')$

$$E' = \{(i, j) : i, j \in V \text{ and } i \neq j\}$$

$$c(i, j) = \begin{cases} 0 & \text{if } (i, j) \in E \\ 1 & \text{if } (i, j) \notin E \end{cases}$$

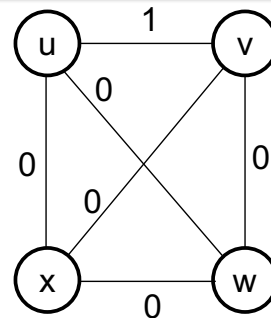
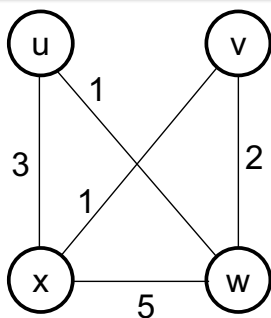
- Let's prove that:
- G has a hamiltonian cycle \Leftrightarrow
 G' has a tour of cost at most 0



CSC611/Lecture 17

27

HAM-CYCLE \leq_p TSP



- G has a hamiltonian cycle h
 - \Rightarrow Each edge in $h \in E \Rightarrow$ has cost 0 in G'
 - $\Rightarrow h$ is a tour in G' with cost 0
- G' has a tour h' of cost at most 0
 - \Rightarrow Each edge on tour must have cost 0
 - $\Rightarrow h'$ contains only edges in E

CSC611/Lecture 17

28

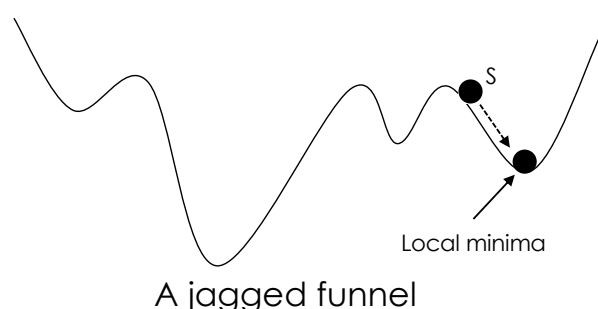
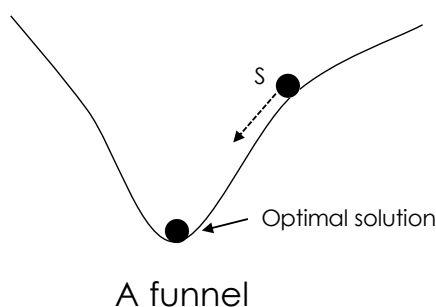
Approximation Algorithms

Various ways to get around NP-completeness:

1. If inputs are small, an algorithm with exponential time may be satisfactory
2. Isolate special cases, solvable in polynomial time
3. Find near-optimal solutions in polynomial time
 - Approximation algorithms
 - Local search (hill climbing)

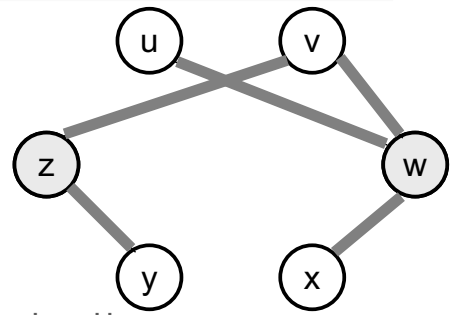
Local Search (Hill Climbing, Gradient Descent)

- Explore the space of possible solutions, moving from a current solution to a "nearby" one
 1. Let S denote current solution
 2. If there is a neighbor S' of S with strictly lower cost, replace S with the neighbor whose cost is as small as possible
 3. Otherwise, terminate the algorithm



Vertex Cover

- $G = (V, E)$, undirected graph
- **Vertex cover** = a subset $V' \subseteq V$ which covers all the edges
 - if $(u, v) \in E$ then $u \in V'$ or $v \in V'$ or both.



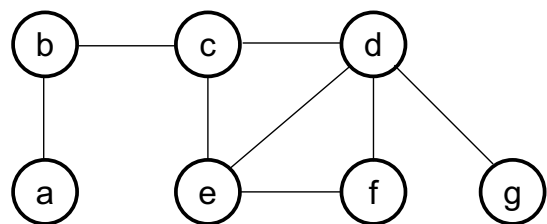
- **Size** of a vertex cover = number of vertices in it

Problem:

- Find a vertex cover of minimum size
- Does graph G have a vertex cover of size k ?

The Vertex-Cover Problem

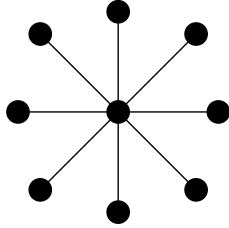
- Vertex cover of $G = (V, E)$, undirected graph
 - A subset $V' \subseteq V$ that covers all the edges in G



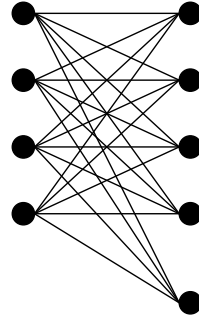
- **Hill climbing (gradient descent) idea:**
 - Start with a solution $S = V$
 - If there is a neighbor S' that is a vertex cover and has lower cardinality, replace S with S' .
 - Algorithm ends after at most n steps (each update decreases the size of the cover by one)

Gradient Descent: Vertex Cover

- Local optimum. No neighbor is strictly better.



optimum = center node only
local optimum = all other nodes



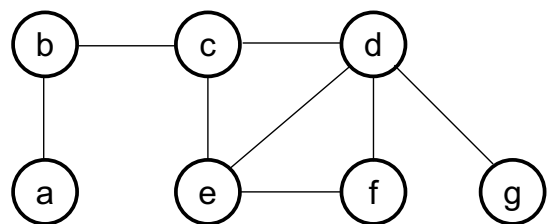
optimum = all nodes on left side
local optimum = all nodes on right side



optimum = even nodes
local optimum = omit every third node

The Vertex-Cover Problem

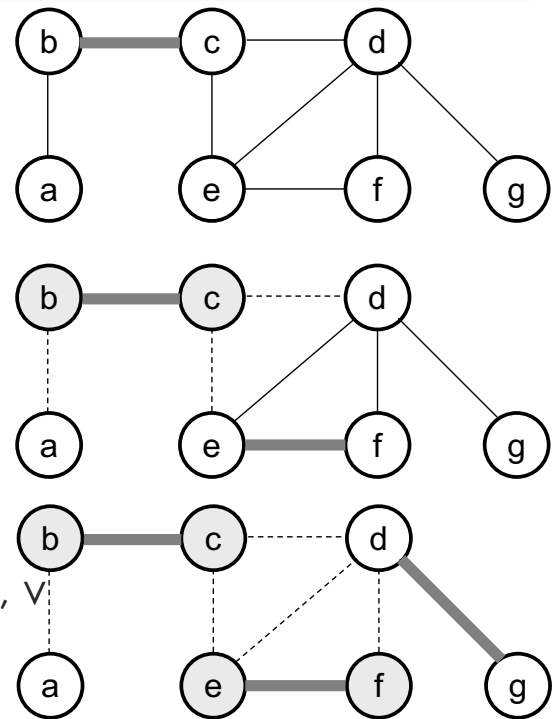
- Vertex cover of $G = (V, E)$, undirected graph
 - A subset $V' \subseteq V$ that covers all the edges in G



- Approximate solution (greedy):**
 - Start with a list of all edges
 - Repeatedly pick an arbitrary edge (u, v)
 - Add its endpoints u and v to the vertex-cover set
 - Remove from the list all edges incident on u or v

APPROX-VERTEX-COVER(G)

1. $C \leftarrow \emptyset$
2. $E' \leftarrow E[G]$
3. **while** $E' \neq \emptyset$
4. **do** choose (u, v)
 arbitrary from E'
5. $C \leftarrow C \cup \{u, v\}$
6. remove from E' all
 edges incident on u, v
7. **return** C

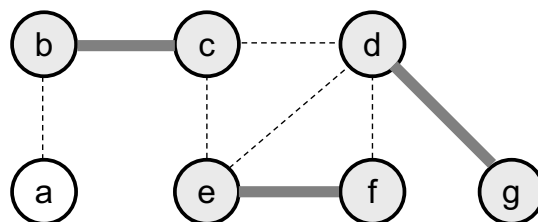


CSC611/Lecture 17

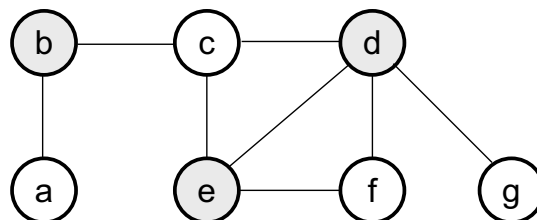
35

APPROX-VERTEX-COVER(G)

APPROX-VERTEX-COVER:



Optimal VERTEX-COVER:



It can be proven that the approximation algorithm returns a solution that is no more than twice the optimal vertex cover.

CSC611/Lecture 17

36

The Set Covering Problem

- Finite set X
- Family \mathcal{F} of subsets of X : $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$

$$X = \bigcup_{S \in \mathcal{F}} S$$

- Find a minimum-size subset $C \subseteq \mathcal{F}$ that covers all the elements in X
- Decision: given a number k find if there exist k sets $S_{i1}, S_{i2}, \dots, S_{ik}$ such that:

$$S_{i1} \cup S_{i2} \cup \dots \cup S_{ik} = X$$

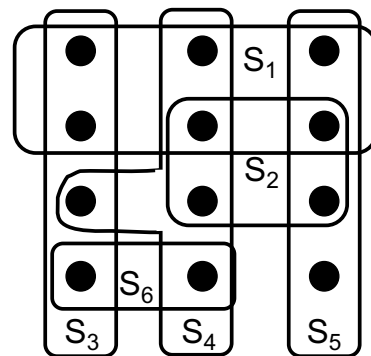
CSC611/Lecture 17

37

Greedy Set Covering

Idea:

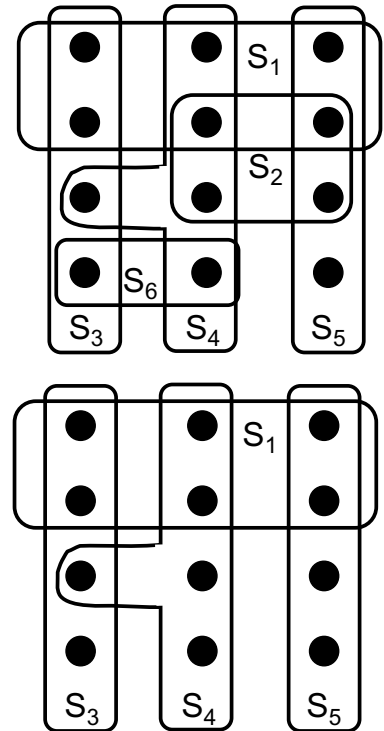
- At each step pick a set S that covers the greatest number of remaining elements



Optimal: $C = \{S_3, S_4, S_5\}$

GREEDY-SET-COVER(X, \mathcal{F})

1. $U \leftarrow X$
2. $C \leftarrow \emptyset$
3. **while** $U \neq \emptyset$
4. **do** select an $S \in F$ that
 maximizes $|S \cap U|$
5. $U \leftarrow U - S$
6. $C \leftarrow C \cup \{S\}$
7. **return** C

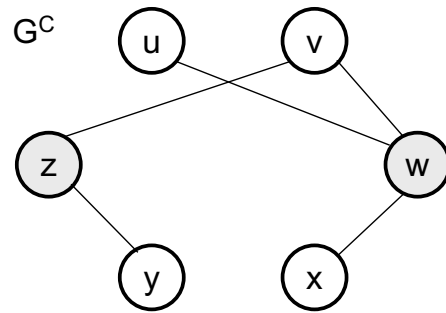
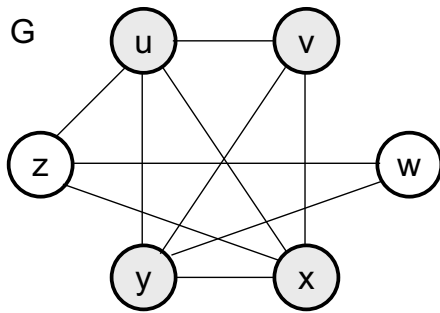


CSC611/Lecture 17

39

Additional Examples

Clique \leq_p Vertex Cover

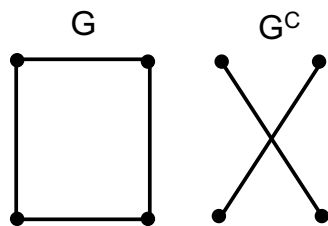


- $G = (V, E) \Rightarrow$ complement graph $G^C = (V, E^C)$
 $E^C = \{(u, v) : u, v \in V, \text{ and } (u, v) \notin E\}$

Idea:

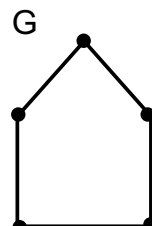
$\langle G, k \rangle$ (clique) $\rightarrow \langle G^C, |V| - k \rangle$ (vertex cover)

Clique \leq_p Vertex Cover (VC)

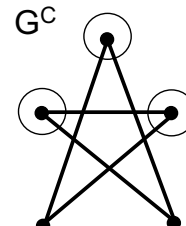


Clique = 2

VC = 2



Clique = 2

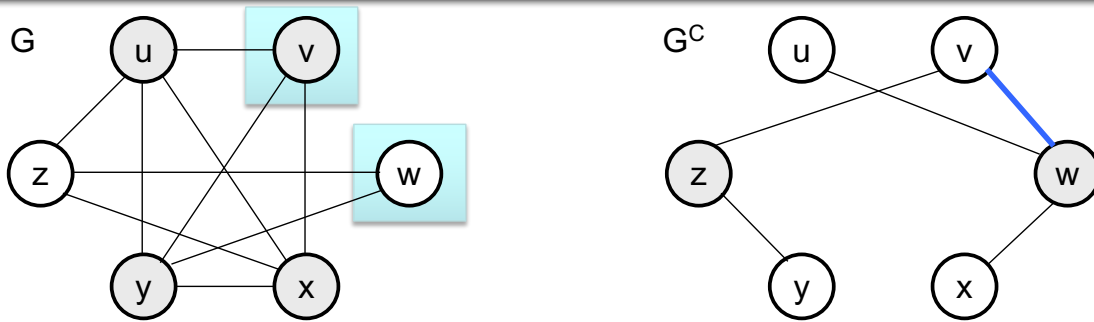


VC = 3

$$\text{Size}[\text{Clique}](G) + \text{Size}[\text{Vertex Cover}](G^C) = n$$

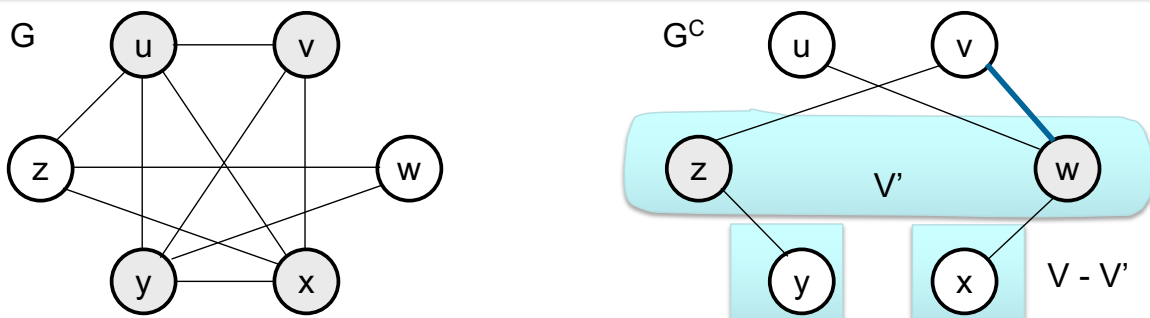
- G has a **clique** of size $k \Leftrightarrow G^C$ has a **vertex cover** of size $n - k$
- S is a clique in $G \Leftrightarrow V - S$ is a vertex cover in G^C

Clique \leq_p Vertex Cover



- Prove: G has a clique $V' \subseteq V$, $|V'| = k \Rightarrow V - V'$ is a VC in G^C
- Let $(v, w) \in E^C \Rightarrow (v, w) \notin E$
 $\Rightarrow v$ and w were not connected in E
 \Rightarrow at least one of v or w does not belong in the clique V'
 \Rightarrow at least one of v or w belongs in $V - V'$
 \Rightarrow edge (v, w) is covered by $V - V'$
 \Rightarrow edge (v, w) was arbitrary \Rightarrow every edge of E^C is covered

Clique \leq_p Vertex Cover

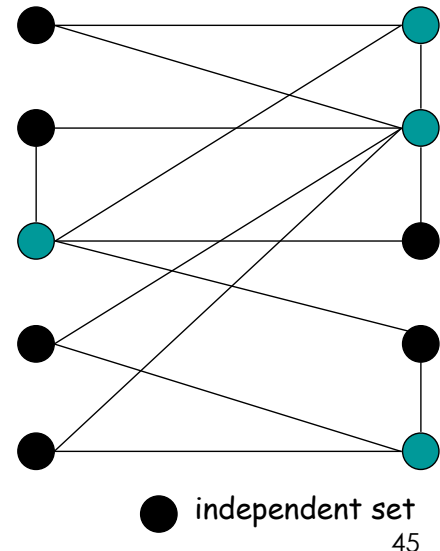


- Prove: G^C has a vertex cover $V' \subseteq V$, $|V'| = |V| - k \Rightarrow V - V'$ is a clique in G
- For all $v, w \in V$, if $(v, w) \in E^C$
 $\Rightarrow v \in V'$ or $w \in V'$ or both $\in V'$
 \Rightarrow For all $x, y \in V$, if $x \notin V'$ and $y \notin V'$:
 \Rightarrow no edge between x, y in $E^C \Rightarrow (x, y) \in E$
 $\Rightarrow V - V'$ is a clique, of size $|V| - |V'| = k$

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 ?

- Is there an independent set of size ≥ 6 ?
 - Yes.
- Is there an independent set of size ≥ 7 ?
 - No.

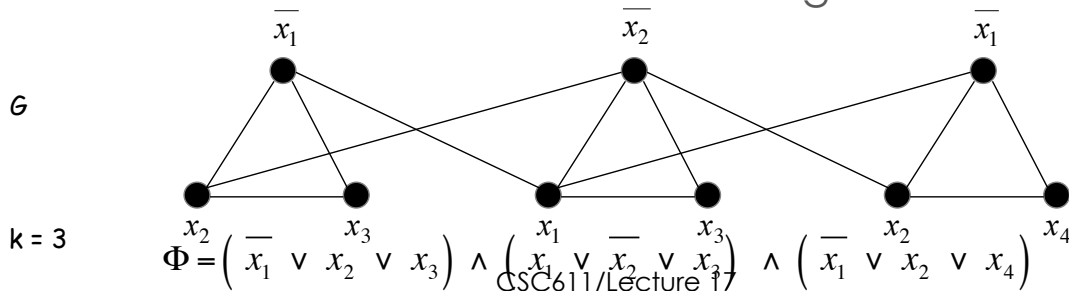


CSC611/Lecture 17

45

$3\text{-CNF} \leq_p \text{INDEPENDENT-SET}$

- Given an instance Φ of 3-CNF, 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 clause, one for each literal.
 - Connect 3 literals in a clause in a triangle.
 - Connect literal to each of its negations.

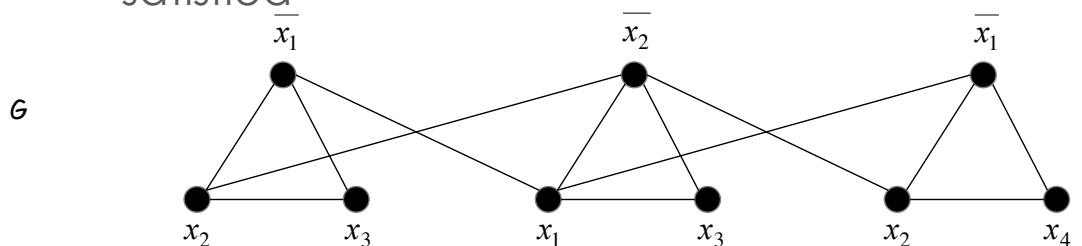


CSC611/Lecture 17

46

3-CNF \leq_p 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



$k = 3$

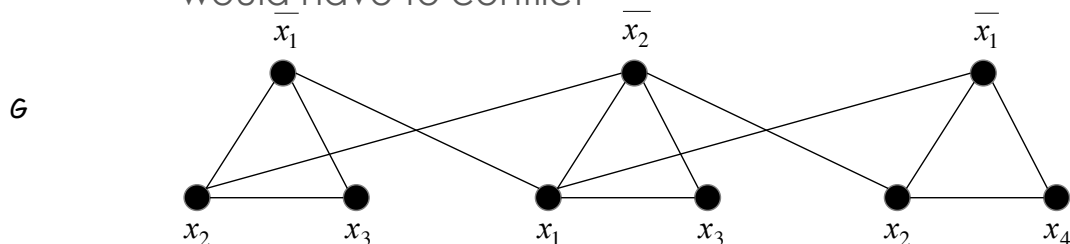
$$\Phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$$

CSC611/Lecture 17

47

3-CNF \leq_p INDEPENDENT-SET

- Claim: G contains independent set of size $k = |\Phi|$ iff Φ is satisfiable
- Proof: “ \Leftarrow ”
 - Each triangle has a literal that evaluates to 1
 - This is an independent set S of size k
 - If there would be an edge between vertices in S , they would have to conflict



$k = 3$

$$\Phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$$

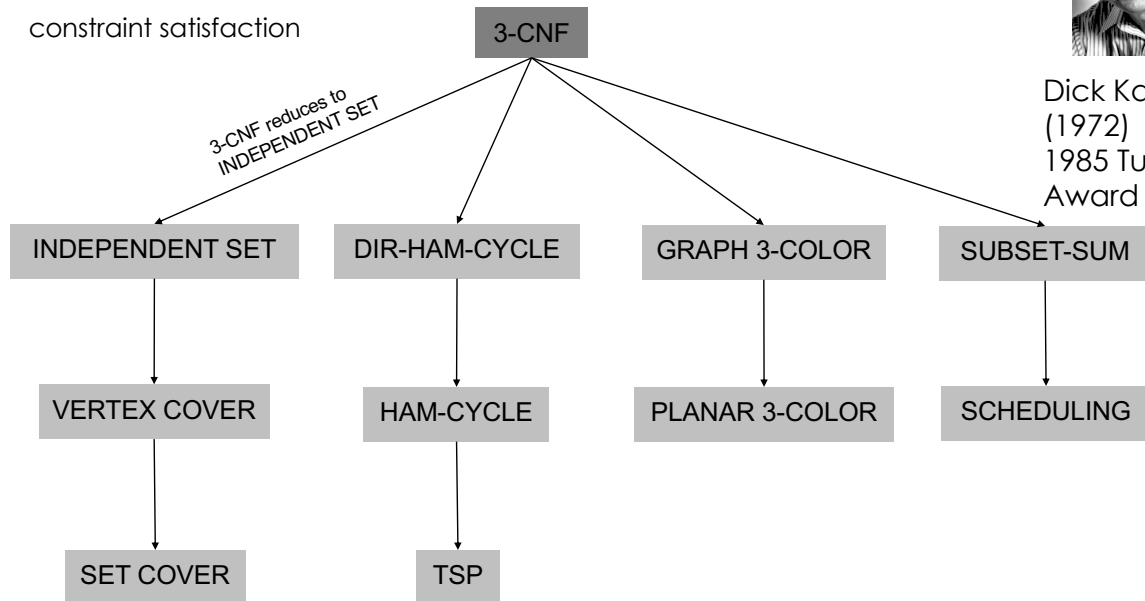
CSC611/Lecture 17

48

Polynomial-Time Reductions



Dick Karp
(1972)
1985 Turing
Award



packing and covering

sequencing

partitioning

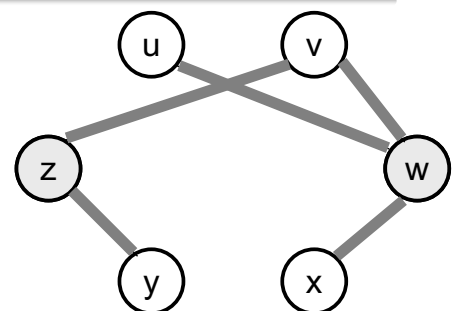
numerical

CSC611/Lecture 17

49

Vertex Cover

- $G = (V, E)$, undirected graph
- **Vertex cover** = a subset $V' \subseteq V$ which covers all the edges
 - if $(u, v) \in E$ then $u \in V'$ or $v \in V'$ or both.
- **Size** of a vertex cover = number of vertices in it



Problem:

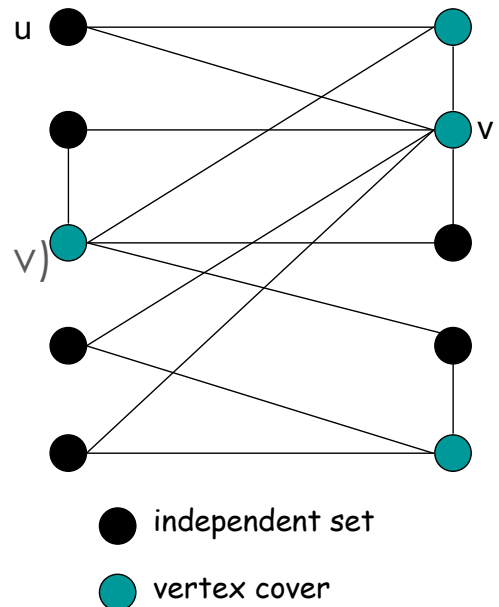
- Find a vertex cover of minimum size
- Does graph G have a vertex cover of size k ?

INDEPENDENT-SET \leq_p VERTEX-COVER

- We show S is an independent set iff $V \setminus S \Leftrightarrow S$ is a vertex cover

Proof " \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, $V \setminus S$ covers (u, v)

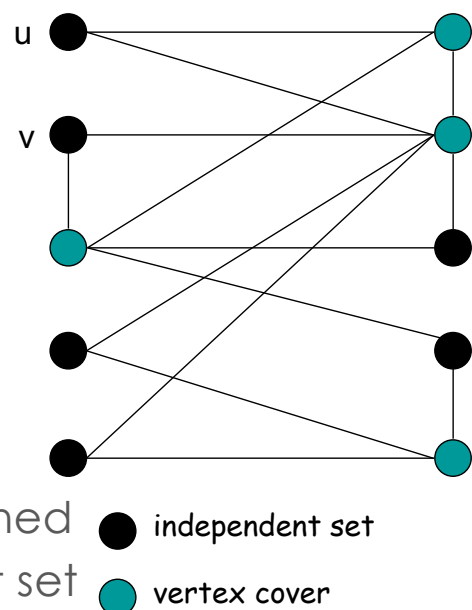


INDEPENDENT-SET \leq_p VERTEX-COVER

- We show S is an independent set iff $V \setminus S \Leftrightarrow S$ is a vertex cover

Proof " \Leftarrow "

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



Set Cover

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

- Example

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

$k = 2$

$S_1 = \{3, 7\}$

$S_4 = \{2, 4\}$

$S_2 = \{3, 4, 5, 6\}$

$S_5 = \{5\}$

$S_3 = \{1\}$

$S_6 = \{1, 2, 6, 7\}$

CSC611/Lecture 17

53

Set Cover

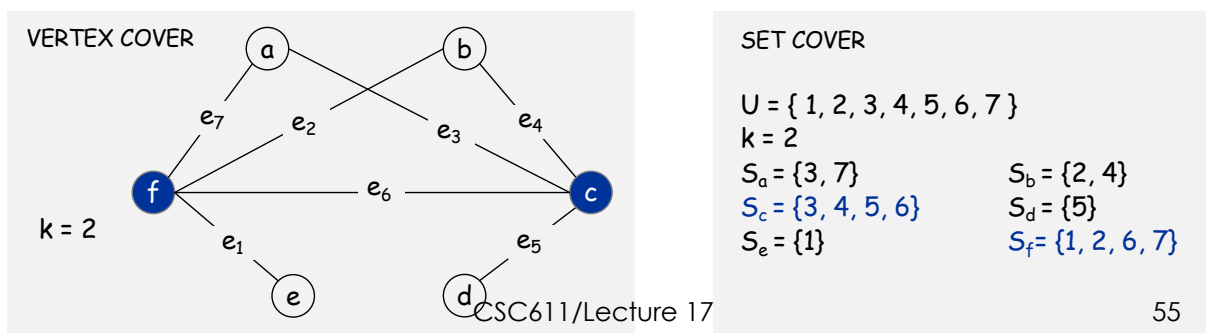
- Given a set U of elements, a collection S_1, S_2, \dots, S_m of subsets of U , and an integer k , does there exist a collection of $\leq k$ of these sets whose union is equal to U ?
- Sample application
 - m available pieces of software
 - Set U of n capabilities that the system should have
 - The i -th piece of software provides the set $S_i \subseteq U$ of capabilities
 - Goal: achieve all n capabilities using fewest pieces of software

CSC611/Lecture 17

54

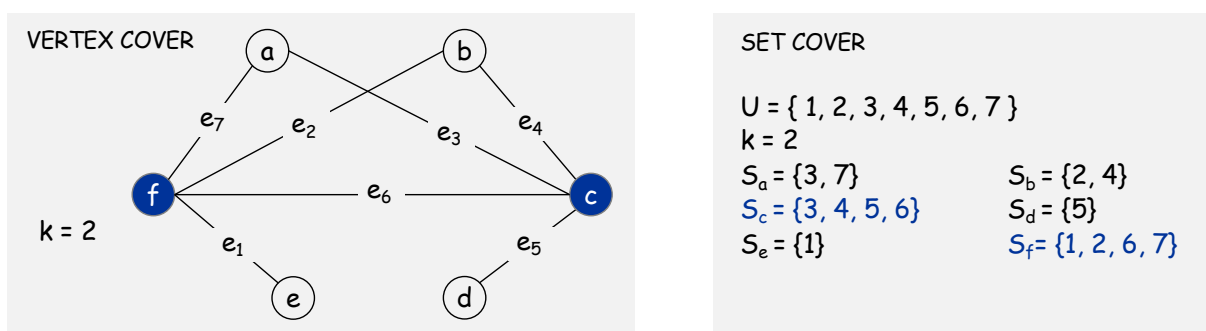
VERTEX-COVER \leq_p SET-COVER

- 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 $\leq k$



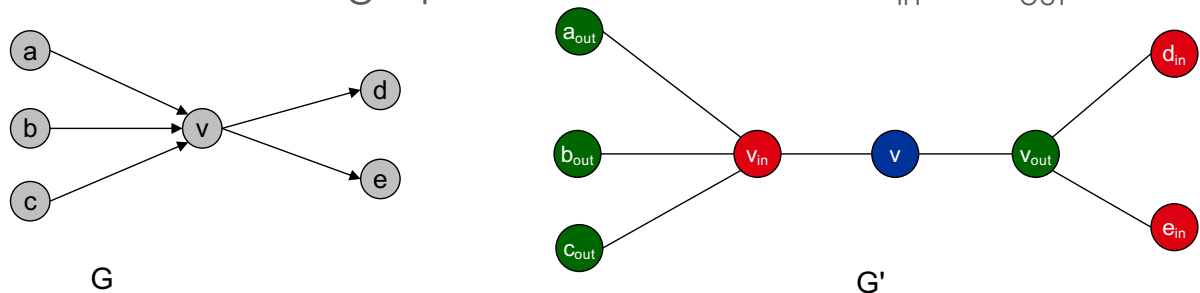
VERTEX-COVER \leq_p SET-COVER

- Set-cover of size $\leq k$ iff vertex cover of size $\leq k$
- Proof " \Rightarrow " (S_{i_1}, \dots, S_{i_l} are $l \leq k$ sets that cover U)
 - Every edge in G is incident on one of the vertices i_1, \dots, i_l , so $\{i_1, \dots, i_l\}$ is a vertex cover of size $l \leq k$
- Proof " \Leftarrow " $\{i_1, \dots, i_l\}$ is a vertex cover of size $l \leq k$
 - Then, the sets S_{i_1}, \dots, S_{i_l} cover U



Hamiltonian Cycle

- Given an undirected graph $G = (V, E)$, does there exist a simple directed cycle Γ that contains every node in V ?
- Claim: $\text{DIR-HAM-CYCLE} \leq_p \text{HAM-CYCLE}$
- Construction
 - Given a directed graph $G = (V, E)$, construct an undirected graph G' with $3n$ nodes: $v_{\text{in}}, v, v_{\text{out}}$

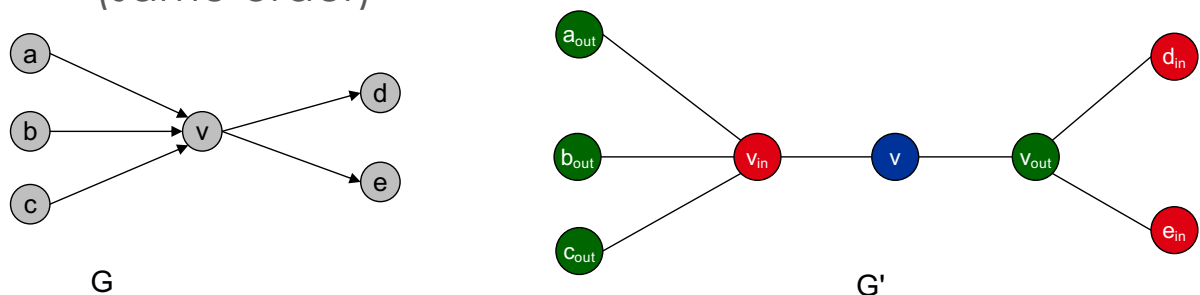


CSC611/Lecture 17

57

$\text{DIR-HAM-CYCLE} \leq_p \text{HAM-CYCLE}$

- 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)

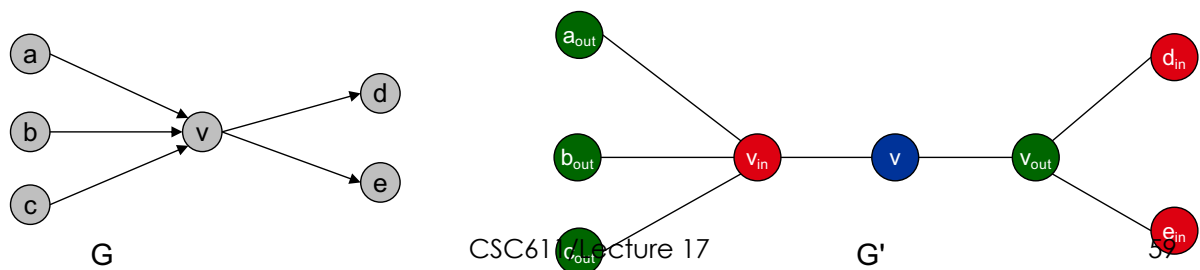


CSC611/Lecture 17

58

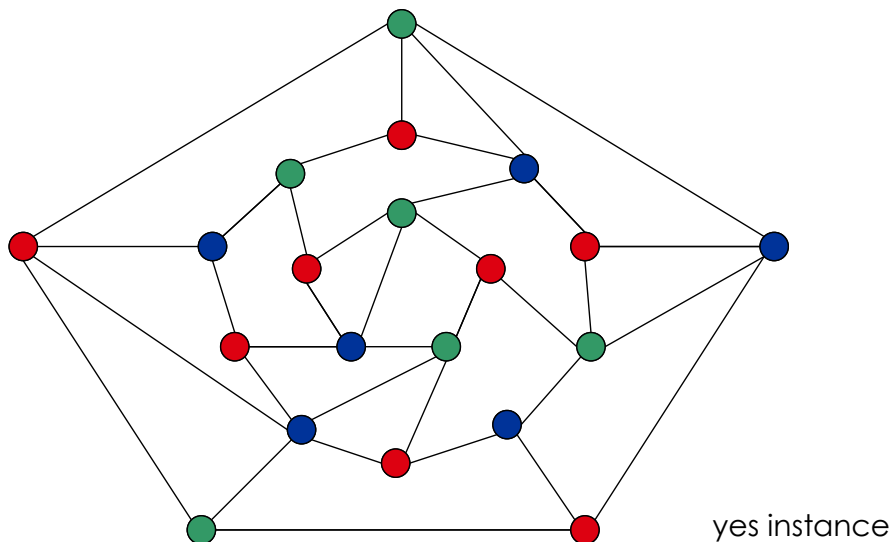
DIR-HAM-CYCLE \leq_p HAM-CYCLE

- Claim: G has a Hamiltonian cycle iff G' does.
- Proof: " \Leftarrow "
 - Suppose G' has an undirected Hamiltonian cycle Γ'
 - Γ' must visit nodes in G' using one of following 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-Colorability

- Given an undirected graph G does there exist a way to color the nodes red, green, and blue so that no adjacent nodes have the same color?



Register Allocation

- Register allocation
 - Assign program variables to machine register so that no more than k registers are used and no two program variables that are needed at the same time are assigned to the same register
- Interference graph
 - Nodes are program variables names, edge between u and v if there exists an operation where both u and v are "live" at the same time.
- Observation [Chaitin 1982]
 - Can solve register allocation problem iff interference graph is k -colorable
- Fact
 - $3\text{-COLOR} \leq_p k\text{-REGISTER-ALLOCATION}$ for any constant $k \geq 3$

CSC611/Lecture 17

61

$3\text{-CNF} \leq_p 3\text{-COLOR}$

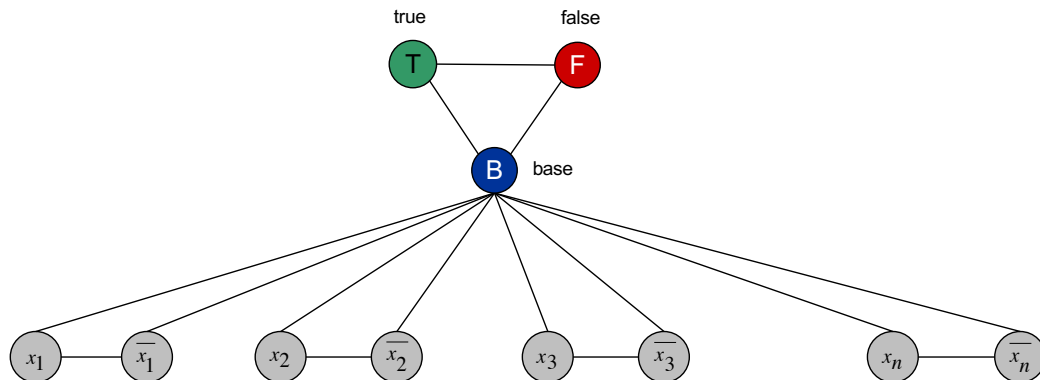
- Given 3-CNF instance Φ , we construct an instance of 3-COLOR that is 3-colorable iff Φ is satisfiable
- Construction
 - For each literal, create a node
 - Create 3 new nodes T, F, B ; connect them in a triangle, and connect each literal to B
 - Connect each literal to its negation
 - For each clause, add a 6-node subgraph

CSC611/Lecture 17

62

$3\text{-CNF} \leq_p 3\text{-COLOR}$

- For each literal, create a node
- Create 3 new nodes T, F, B; connect them in a triangle, and connect each literal to B
- Connect each literal to its negation

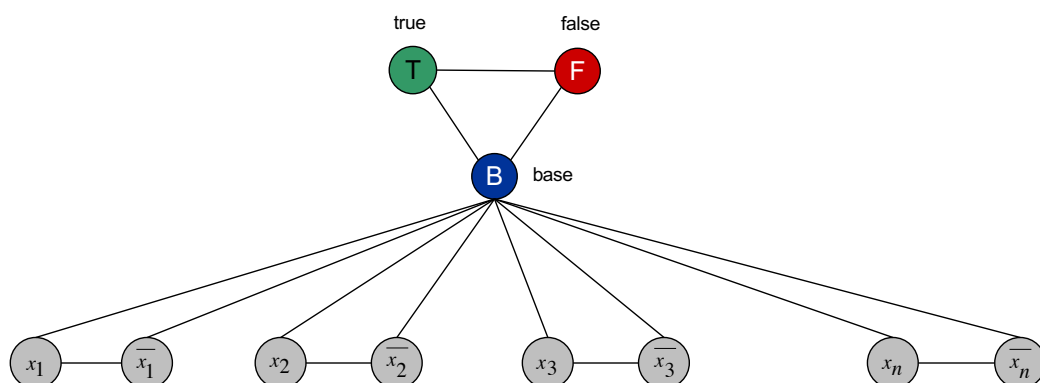


CSC611/Lecture 17

63

$3\text{-CNF} \leq_p 3\text{-COLOR}$

- Any 3-coloring implicitly determines a truth assignment for variables in 3-CNF
 - Nodes T, F, B must get different colors
 - For x_i and $\neg x_i$, one will take T color one F color

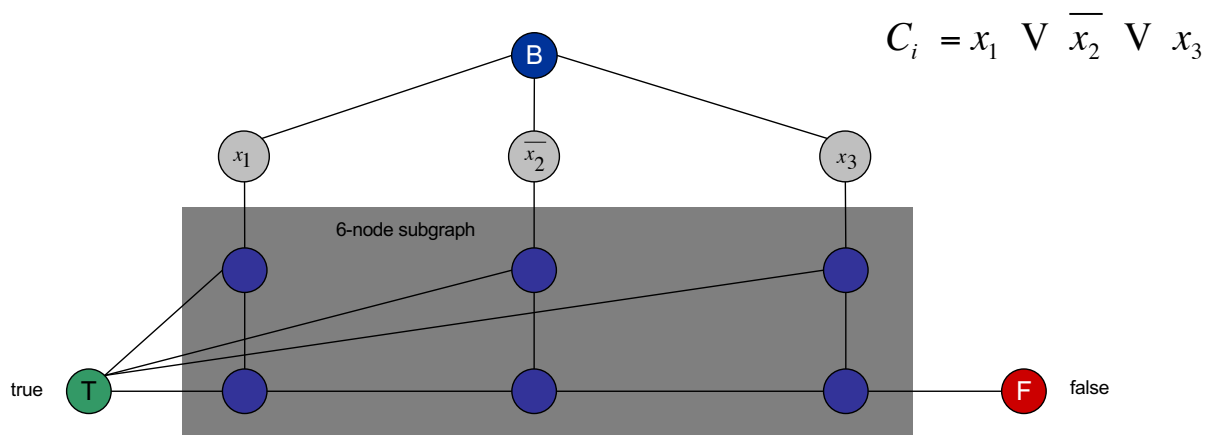


CSC611/Lecture 17

64

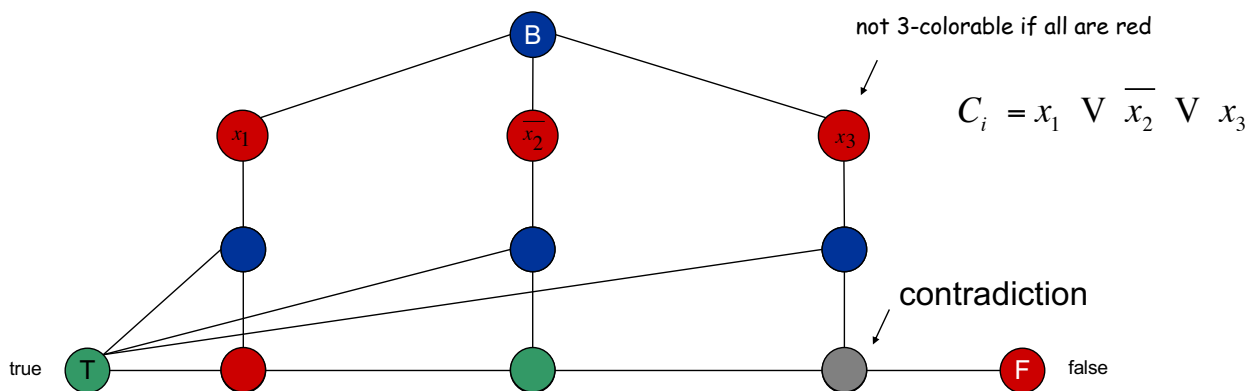
3-CNF \leq_p 3-COLOR

- Must ensure that only satisfying assignments can result in 3-coloring of the full graph
 - For each clause, add a 6-node subgraph



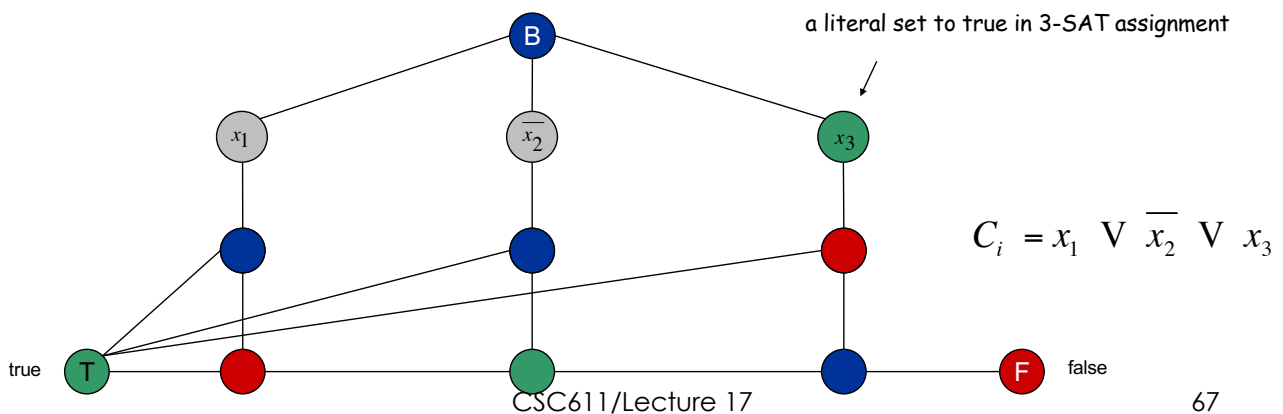
3-CNF \leq_p 3-COLOR

- Proof “ \Rightarrow ” Suppose graph is 3-colorable
 - Proof by contradiction: assume that all three literals get a False color



3-CNF \leq_p 3-COLOR

- Proof " \Leftarrow " Suppose 3-CNF formula Φ is satisfiable
 - Color all true literals T
 - Color node below green node F, and node below B
 - Color remaining middle row nodes B
 - Color remaining bottom nodes T or F as forced



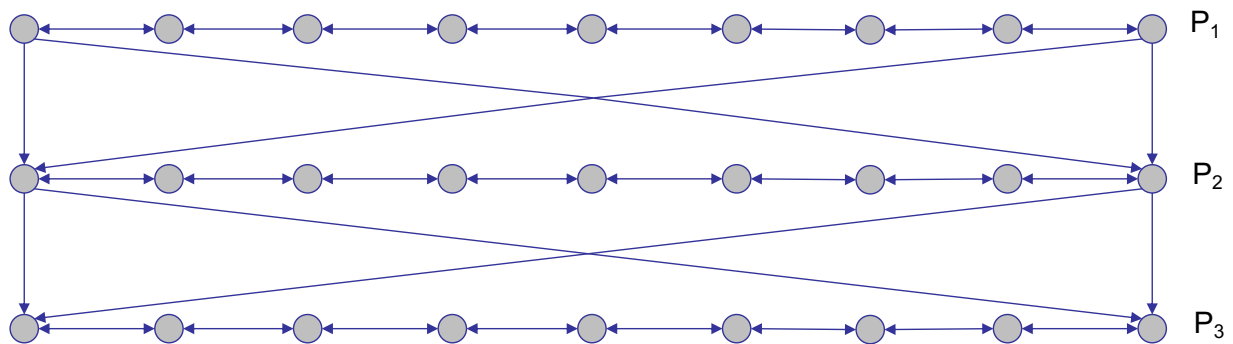
67

Directed Hamiltonian Cycle

- Given a digraph $G = (V, E)$, does there exist a simple directed cycle Γ that contains every node in V ?
- Idea:
 - Given an instance Φ of 3-CNF, we construct an instance of DIR-HAM-CYCLE that has a Hamiltonian cycle iff Φ is satisfiable
- Construction
 - Create a graph that has 2^n Hamiltonian cycles which correspond in a natural way to 2^n possible truth assignments

3-CNF \leq_p DIR-HAM-CYCLE

- Construction: given 3-CNF instance Φ with n variables x_i and k clauses C_1, \dots, C_k
 - Construct n paths P_1, \dots, P_n , with P_i containing $v_{i1}, v_{i2}, \dots, v_{ib}$
 - There are edges between adjacent vertices on path in each direction
 - Hook the paths together with edges

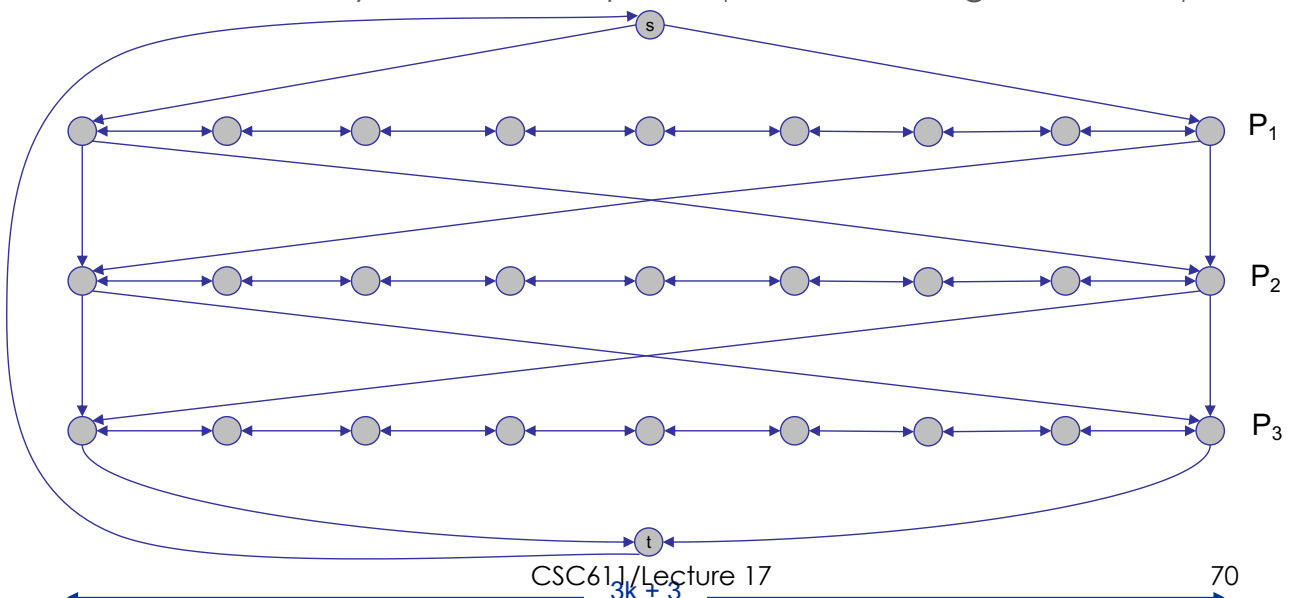


CSC611/Lecture 17

69

3-CNF \leq_p DIR-HAM-CYCLE

- Construction (continued)
 - Add two vertices s and t and connect them with edges
 - Add edge from t to s
 - Intuition: cycle traverses path P_i from left to right \Leftrightarrow set $x_i = 1$



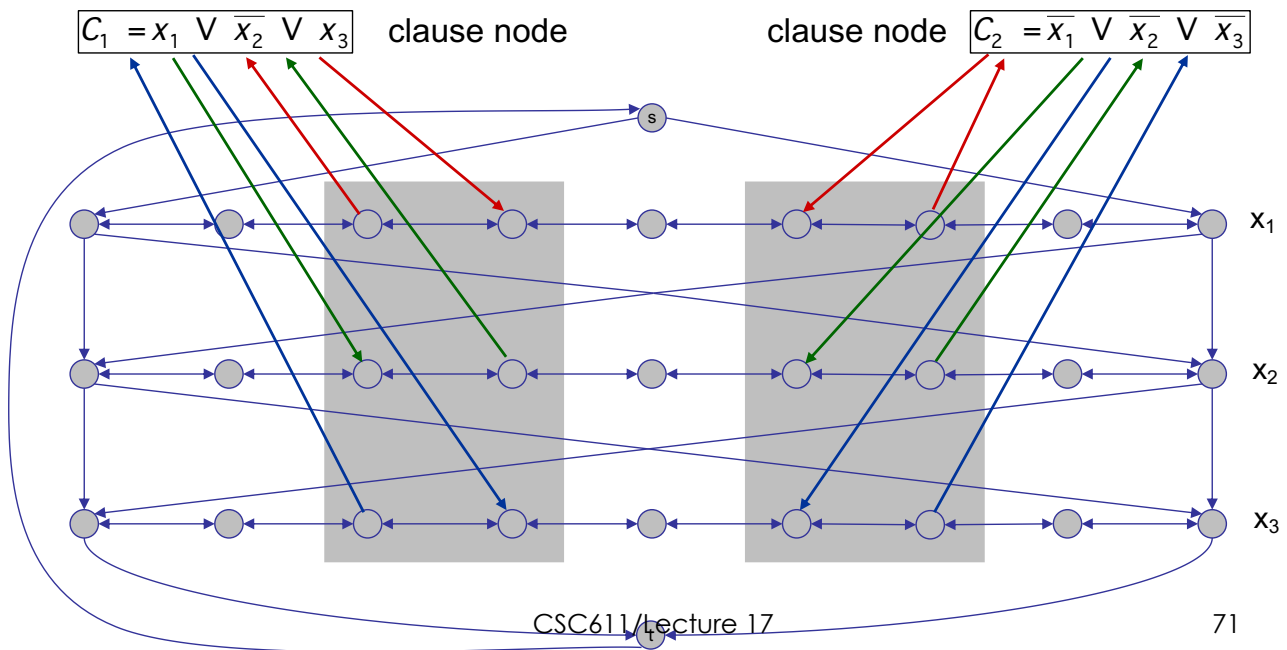
CSC611/Lecture 17

$3k + 3$

70

3-CNF \leq_p DIR-HAM-CYCLE

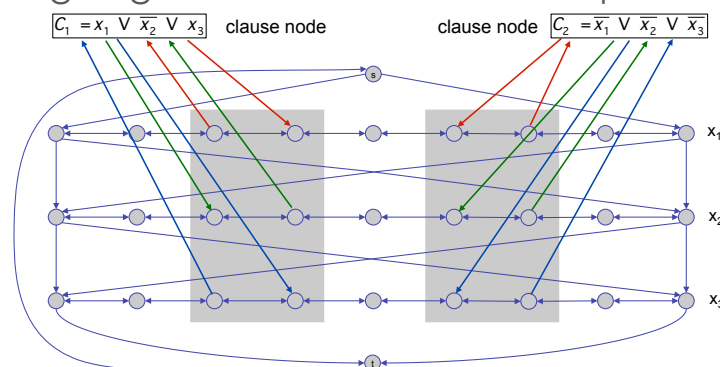
- Construction (continued)
 - For each clause: add a node and 6 edges



71

3-CNF \leq_p DIR-HAM-CYCLE

- Claim: Φ is satisfiable iff G has a Hamiltonian cycle
- Proof " \Rightarrow " Suppose 3-CNF 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-CNF \leq_p DIR-HAM-CYCLE

- Claim: Φ is satisfiable iff G has a Hamiltonian cycle Γ
- Proof " \Leftarrow " Suppose G has a Hamiltonian cycle Γ
 - If Γ enters clause node C_j , it must depart on mate edge
 - Nodes before and after C_j are connected by an edge e in G
 - Removing C_j from cycle, replace it with edge $e \Rightarrow$ Hamiltonian cycle on $G - \{C_j\}$
 - Continuing in this way, \Rightarrow Hamiltonian cycle Γ' in $G - \{C_1, C_2, \dots, C_k\}$
 - Set $x_i^* = 1$ iff Γ' traverses row i left to right, otherwise set to 0
 - Since Γ visits each clause node C_j , at least one of the paths is traversed in "correct" direction, and each clause is satisfied

