Foundations of Algorithm SCS1308

Dr. Dinuni Fernando PhD

Senior Lecturer

How to prove problem is NP-Complete

1. The problem is in NP

This means any solution to the problem can be verified in polynomial time

2. The problem is NP-hard

This means every problem in NP can be reduced to this problem in polynomial time.

Steps to Prove NP-Completeness

1. Show the Problem is in NP

Identify the certificate (solution) for the problem.

Demonstrate that this certificate can be verified in polynomial time.

2. Show the Problem is NP-Hard

Choose a known NP-complete problem (e.g., **3-SAT**, **Hamiltonian Cycle**, **Vertex Cover**, etc.).

Provide a **polynomial-time reduction** from the known NP-complete problem to the target problem.

Converting to decision problems

- Optimization problems can be converted to decision problems (typically) by adding a bound B on the value to optimize, and asking the question:

  - Is there a solution whose value is at most B? (for a minimization problem)
  - Is there a solution whose value is at least B? (for a maximization problem)

An optimization problem: traveling salesman

- Given:
  - A finite set $C = \{c_1,\ldots,c_m\}$ of cities and
  - A distance function $d(c_i, c_j)$ of non-negative numbers
- Find the length of the **minimum** distance tour which visits every city exactly once and comes back to the starting city

A decision problem for traveling salesman

- Given a finite set C = $\{c_1,\ldots,c_m\}$ of cities, a distance function $d(c_i, c_j)$ of nonnegative numbers and a bound B

- Is there a tour of all the cities (in which each city is visited exactly once) with total length **at most B**?

- There is no known polynomial bound algorithm for TS.

Relation between an optimization problem and the decision problem

- If we have a solution to the optimization problem, we can compare the solution to the bound and answer “yes” or “no”

- Therefore, if the optimization problem is tractable so is the decision problem

- If the decision problem is “hard” the optimization problem is also “hard”
    - If the optimization is easy, then the decision problem is easy

The class P

- **P is the class of decision problems that are polynomial bound**

- Is the following problem in P?
  - Given a weighted graph G, is there a spanning tree of weight at most B?

- The decision versions of problems such as shortest distance path and minimum spanning tree belong to P
  - Simply compute an MST and compare its weight to B

The goal of verification algorithms

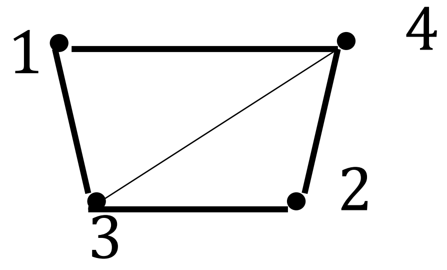
- The goal of a verification algorithm is to verify a “yes” answer to a decision problem’s input (i.e., if the answer is “yes” the verification algorithm verifies this answer)

- The inputs to the verification algorithm are:
  - the original input (problem instance) and
  - a *certificate* (possible solution)

Verification Algorithms

- A *verification algorithm* takes a problem instance x and *answers "yes"*, if there **exists** a certificate y such that the answer for x with certificate y is "yes"

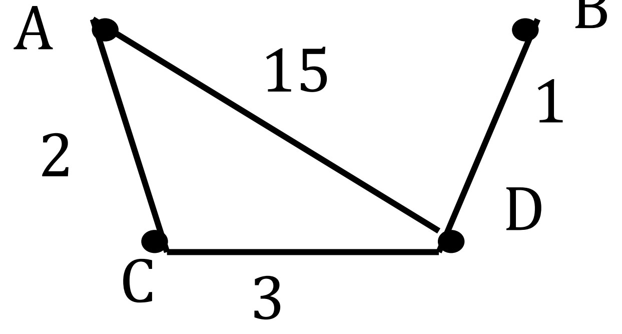
- Consider HAMILTONIAN-CYCLE
- A problem *instance* x lists the vertices and edges of G: ({1,2,3,4}, {(3,2), (2,4), (3,4), (4,1), (1, 3)})
- There **exists** a certificate y = (3, 2, 4, 1, 3) for which the verification algorithm answers "yes"



The problem PATH

- PATH denotes the decision problem version of shortest path.
- PATH: Given a graph G, a start vertex u, and an end vertex v. Does there exist a path in G, from u to v of length at most k?

- The instance is: G=({A, B, C, D}, {(A, C,2), (A, D, 15), (C,D, 3), (D, B, 1)} k=6
- A certificate y=(A, C, D, B)



A verification algorithm for PATH

- Verification algorithm:
  - Given the problem instance x and a certificate *y*
    - Check that *y* is indeed a path from *u* to *v*.
    - Verify that the length of *y* is at most *k*

- Is the verification algorithm for PATH polynomial bound?
- Is the size of y polynomial in the size of x?

Example: A verification algorithm for TS (Traveling Salesman)

- Given a problem instance x for TS and a certificate *y*
  - Check that *y* is indeed a cycle that includes every vertex exactly once except for the starting node
  - Verify that the length of the cycle is at most B

- Is the size of y polynomial in the size of x?
- Is the verification algorithm polynomial?

The class NP
(Non-deterministic Polynomial)

- **NP is the class of decision problems for which there is a polynomial bound verification algorithm**

- It can be shown that:
  - All decision problems in P are also in NP.
  - Decision problems such as traveling salesman, knapsack, bin packing, are also in NP.

The relation between P and NP

- P | NP
- It is not known whether P = NP or P ≠ NP
- Problems in P can be *solved* “quickly”
- Problems in NP can be *verified* “quickly”
- It is easier to verify a solution than solving a problem
- Some researchers believe that P and NP are not the same class (But no one has proved whether or not this is true)

Polynomial reductions

- **Motivation**: The definition of NP-completeness uses the notion of *polynomial reductions* of one problem A to another problem B, written as

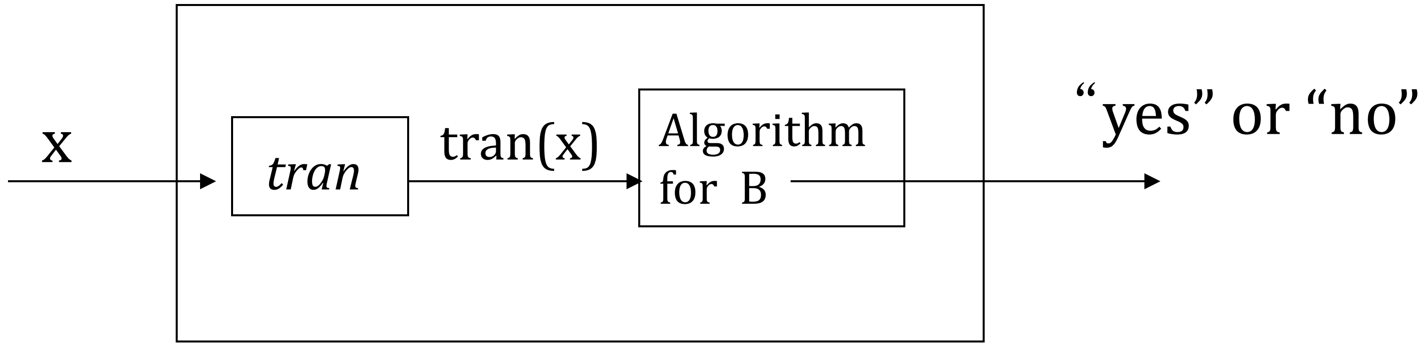
$$A \propto B$$

- Let *tran* be a function that converts any input x for decision problem A into input tran(x) for decision problem B

Polynomial reductions

tran is a *polynomial reduction* from A to B if:

1. *tran* can be computed in polynomial bound time
2. The answer to A for input x is *yes* if and only if the answer to B for input *tran*(x) is *yes*.



Algorithm for A

Two simple problems

- A: Given n Boolean variables with values $x_1,\dots,x_n$, does at least one variable have the value True?
- B: Given n integers $i_1,\dots,i_n$ is $max\{i_1,\dots,i_n\}>0$?

Algorithm for B :

Check the integers one after the other.
If one is positive, stop and answer "*yes*"
If none is positive, stop and answer "*no*".

Example:

n=4.

Given integers: -1, 0, 3, and 20.

Algorithm for B answers "*yes*".

Given integers: -1, 0, 0, and 0.

Algorithm for B answers "*no*".

Is there a transformation?

- Can we transform an instance of A into an instance of B?
- Yes.

```
tran(x)
        for ( j =1; j =< n; j ++)
                if ($x_j$ == true)
                   $i_j$=1
                else   // $x_j$=false
                   $i_j$= 0
```

T(false, false, true, false)= 0,0,1,0

- Is this transformation polynomial bound? yes

Does it satisfy all the requirements?

- Can we show that when the answer for an instance $x_1,\ldots,x_n$ of A is "*yes*" the answer for the transformed instance *tran*($x_1,\ldots,x_n$)= $i_1,\ldots,i_n$ of B is also "*yes*"?
- If the answer for the given instance $x_1,\ldots,x_n$ of A is "*yes*", there is some x_j=true.
- The transformation assigns i_j=1.
- Therefore the answer for problem B is also "*yes*"

The other direction

- Can we also show that when the answer for problem B with input *tran*($x_1,\ldots,x_n$)= $i_1,\ldots,i_n$ is "*yes*", the answer for the instance $x_1,\ldots,x_n$ of A is also "*yes*"?
- If the answer for problem B is "*yes*", it means that there is an $i_j > 0$ in the transformed instance.
- i_j is either 0 or 1 in the transformed instance. If $i_j=1$, x_j=true.
- So the answer for A is also "*yes*"

Polynomial reductions

Theorem:

If A$\propto$ B and B is in P, then A is in P

If A is not in P then B is also not in P

Halting problem is not NP hard!

- Halting problem : Turing showed Halting problem is undecidable. No algorithm that can solve this problem for all inputs P and x.
- How to prove :
- NP hardness : a problem X is NP-hard if every problem in NP can be reduced to X in polynomial time.
- For any problem in NP, there exists a non-deterministic Turing machine that verifies solutions in polynomial time.

Not NP Complete

- For a problem to be in NP-complete
    1. Problem is in NP.
    2. Problem is NP-hard.

Halting problem is not in NP hard because :

1. Problems in NP require a solution that can be verified in polynomial time.
2. For halting problem there is no algorithm that can verify whether a program P halts or not.

Therefore, halting problem is not in NP, cannot be NP-complete.

NP-completeness and Reducibility

- The existence of NP-complete problems leads us to *suspect* that P == NP.
- If HAMILTONIAN CYCLE, which is an NP-complete problem, can be solved in polynomial time, every problem in NP can be solved in polynomial time. This means every problem in NP is polynomial bound and, therefore, P=NP.
- If HAMILTONIAN CYCLE could not be solved in polynomial time, every NP-complete problem cannot be solved in polynomial time. Thus NP $\neq$ P

Revisit the SAT problem

- First, Conjunctive Normal Form (CNF) will be defined

- Second, satisfiability (SAT) problem will be defined

- Finally, we will show a polynomial bounded verification algorithm for the problem.

Reduction of 3SAT to Clique

- This shows clique is NP complete since 3SAT is NP complete.
- 3SAT problem – Boolean formula in CNF where each clause contains exactly 3 literals.

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

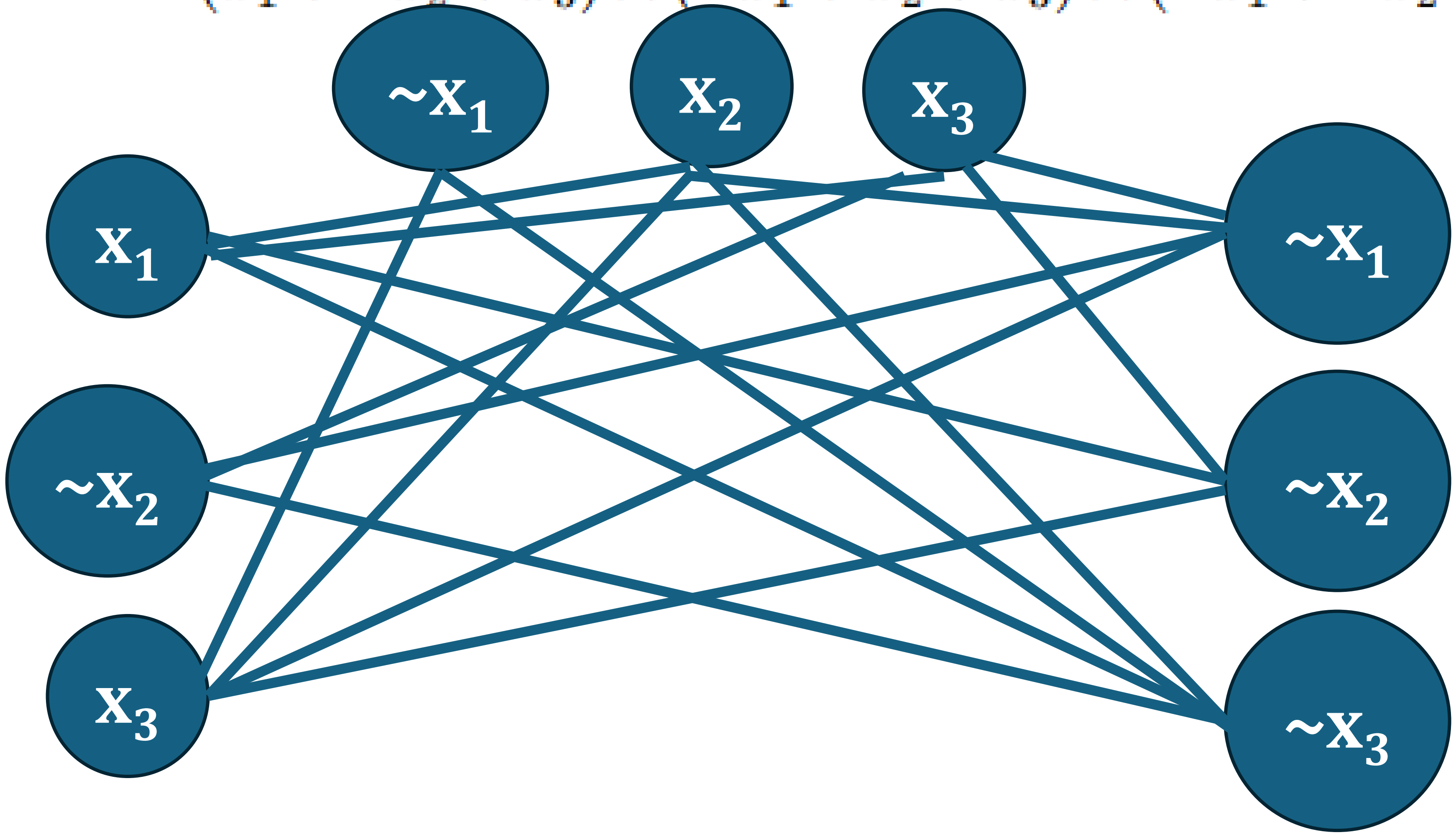
- Goal is to determine if there exists an assignment of variables x_1, x_2, x_3 such that F evaluates to be true.
- Clique problem – G = (V,E) and integer k, determine whether G contains a clique of size k. A clique is a subset of vertices $v' \subseteq v$ such that every pair of vertices in v' is connected by an edge.

Reduction of 3SAT to Clique

- Inputs to reduction : 3 SAT formula F can have m clauses and each clause c_i contains exactly 3 literals.
- Construct graph G = for each literal l in each clause c_i create a vertex, 3 vertices for each clause and 3m vertices in total.
- Add edge between two vertices iff:
  - If they are from different clause
  - If they are not contradictory
- Set k = m such that no. of clauses in 3SAT. Goal is to find a clique of size m in G.

Reduction of 3SAT to Clique

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

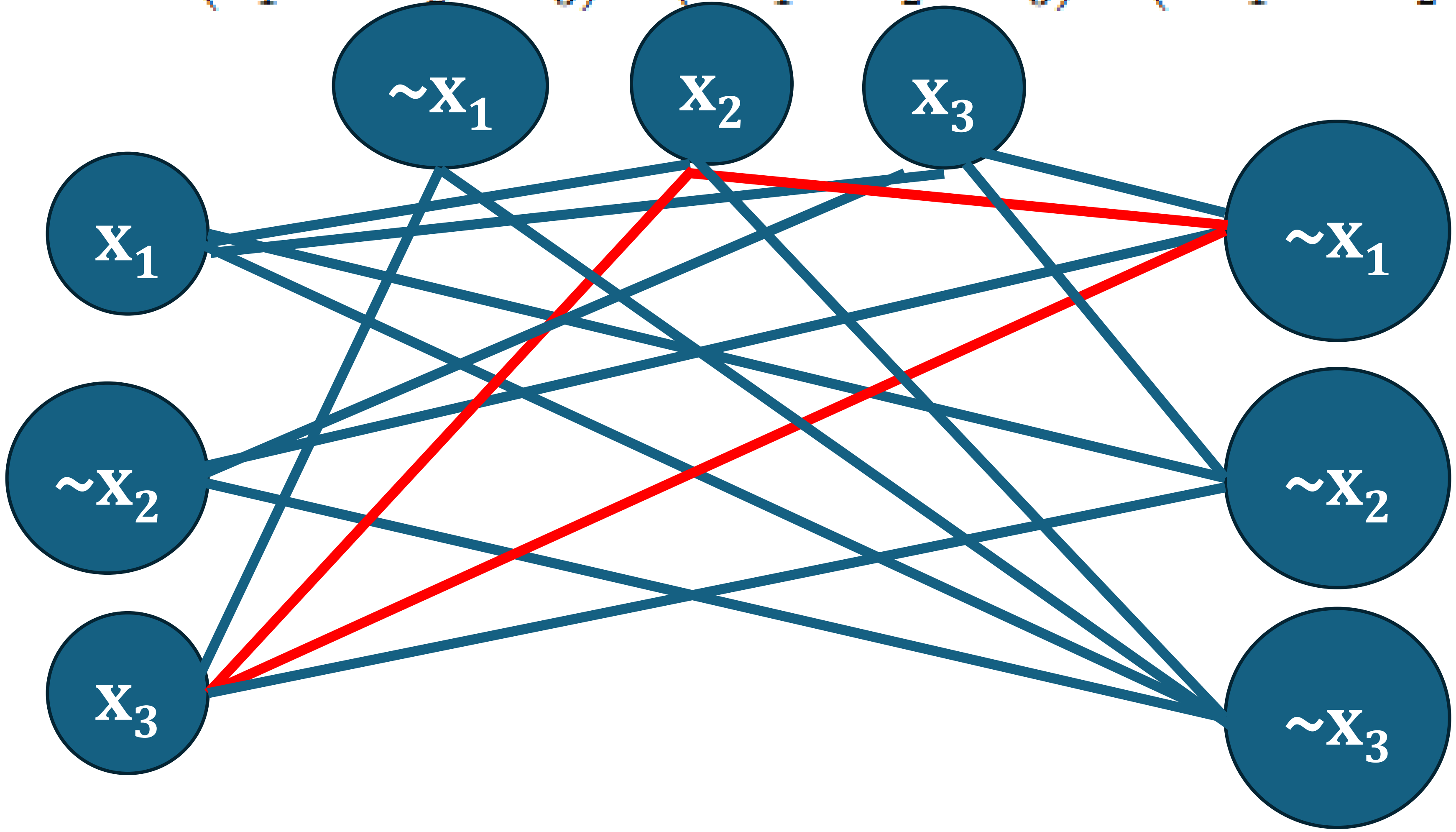


Reduction of 3SAT to Clique

- Construction of the graph can be performed in polynomial time.
- If F is satisfiable, let A be a satisfying assignment, where we select from each clause a literal that is true in A, such that we construct a set S where |S| = k since there are no two literals from same clause.
- All of them simultaneously true for all corresponding nodes and they are connected to each other forming k clique. In our case its 3 clique.

Reduction of 3SAT to Clique

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



Reduction of 3SAT to Clique

- There exist a clique such that corresponding assignment of variables satisfies all clauses in F.
- Conversely If F is satisfiable, there will be a clique of size 3 in G
- X_1 = False
- X_2 = True
- X_3 = True

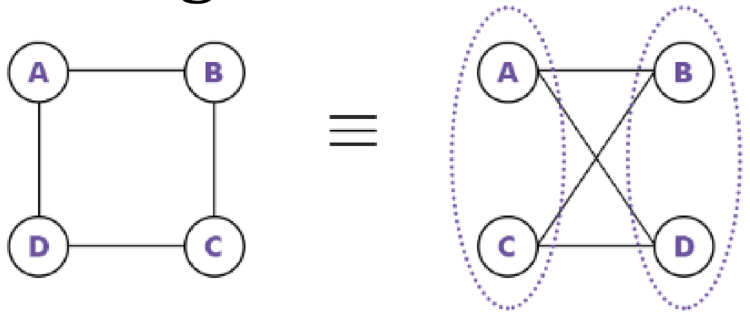
$$\underset{T}{F} = (x_1 \vee \neg x_2 \vee \underset{T}{x_3}) \wedge (\neg x_1 \vee \underset{T}{x_2} \vee x_3) \wedge (\underset{T}{\neg x_1} \vee \neg x_2 \vee \neg x_3)$$

So F is True.

Decision Problems

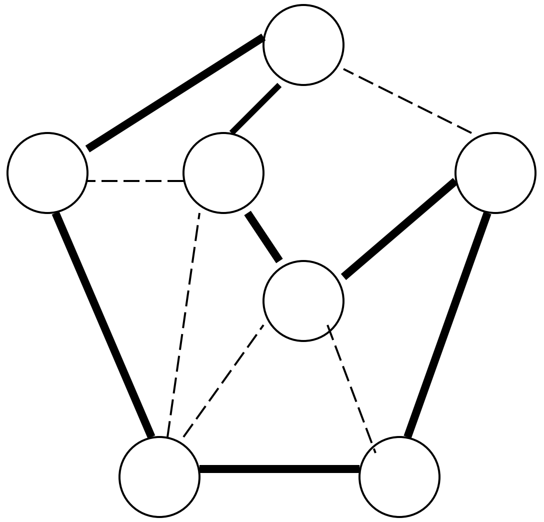
- A *decision* problem answers *yes or no* for a given input

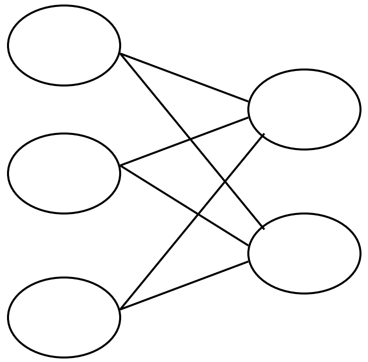
- Examples:
  - Given a graph *G*, is there a path from *s* to *t* of length at most *k*?
  - Does graph *G* contain a Hamiltonian cycle?
  - Given a graph *G*, is it bipartite?
  - For a 0-1 knapsack problem, is there a solution whose benefit is \$100 or more?



A decision problem: HAMILTONIAN-CYCLE

- A *Hamiltonian cycle* of a graph G is a cycle that visits each vertex of the graph (except for the starting node) exactly once.
- Problem: Given a graph G, does G have a Hamiltonian cycle?





Hamiltonian cycle is in NP

- Certificate : ordered list of vertices
- Verifier
  - First check if there are duplicated vertices
  - Then check all consecutive pairs are joined by some edges belong to the graph.
  - Polynomial time

Reduce 3SAT to Hamiltonian cycle

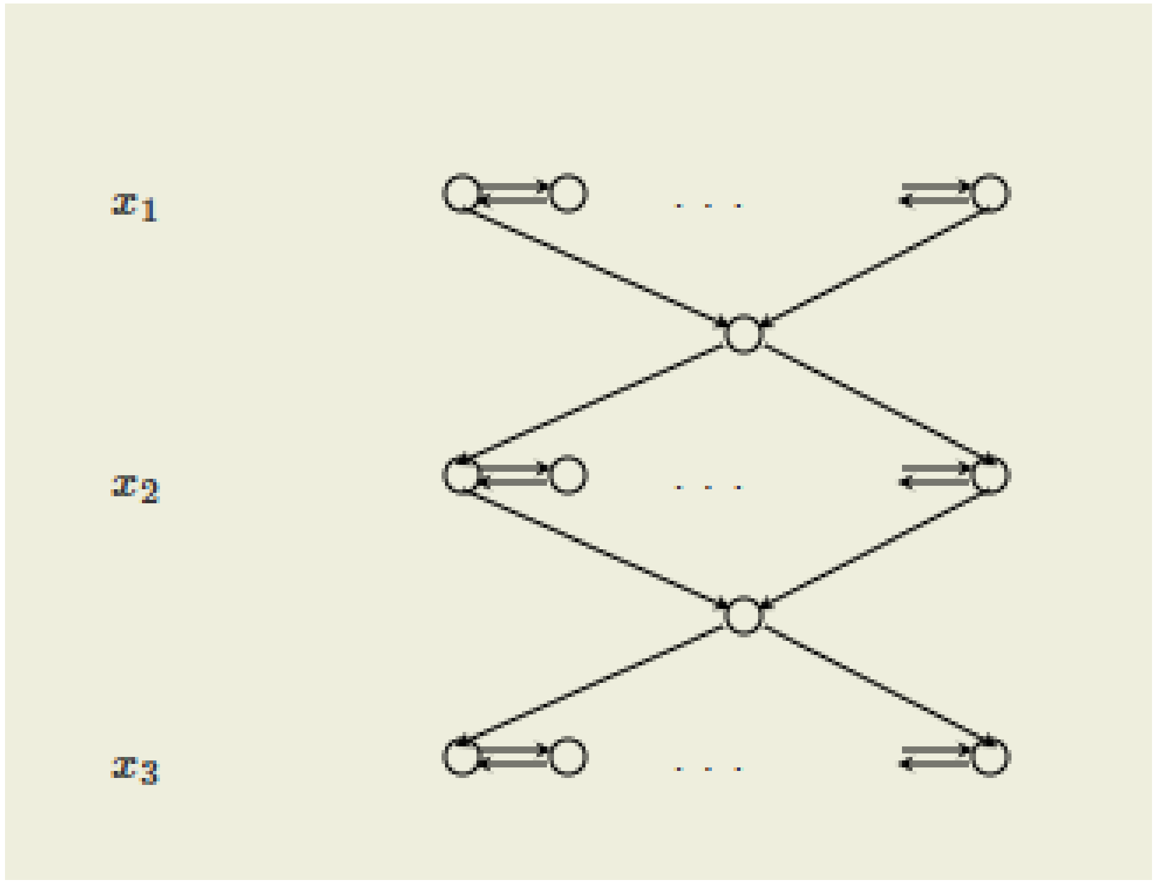
- 3SAT
  - Input variables $x_1, x_2, x_3, \ldots, x_n$
  - Clauses $c_1, c_2, \ldots,$ cm
  - Example $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$
  - Yes instance if some assignment of variables can satisfy the formula.
- HamCycle
  - Input : directed graph G = (V,E)
  - Yes, if contains cycle such that visits each vertex exactly once.

Reduction Part I

- Each literal corresponds to a variable
- Each time we can go left or right
  - Left to right means x_1 is true
  - Right to left means x_1 is false
- 2^n different possible truth assignments correspond to 2^n distinct possible cycles of the form
- S-> down left/right -> right/left ->….->t->s

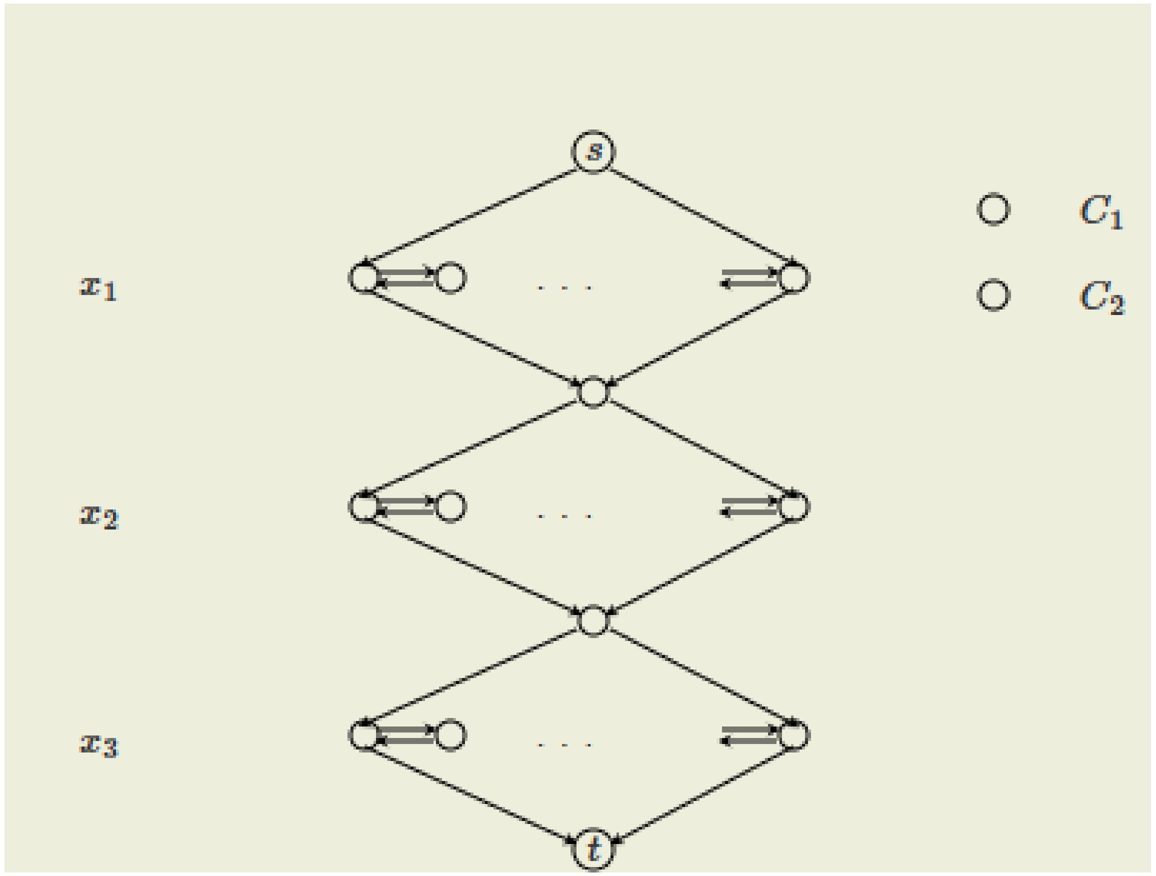
Graph construction

- Each variable from 3SAT represent by its own row of nodes (gadget).
- Row will be connected to left to right. Representing CNF expression with three variables x_1, x_2, x_3
- Very top node S (start) is added, bottom node t is added.



Graph construction

- Representing clauses using nodes.
- If there are 2 clauses c_1, c_2 we create 2 nodes for each clause.
- Q : How do we connect the clause nodes to the graph ?
- Before, we first need to find how many nodes need to be added for variable gadget in each row !



Graph construction : variable gadget

- 3SAT reduction to HamCycle requires 3k+3 nodes for each variable row. (k - # of clauses)

1. 3SAT formula connects to 3 literals (variable or their negation)
    - One node to “enter” the clause
    - One node to “exit” the clause

k clauses : there are 2x 3k = 6k total connections needed in the graph.

However :

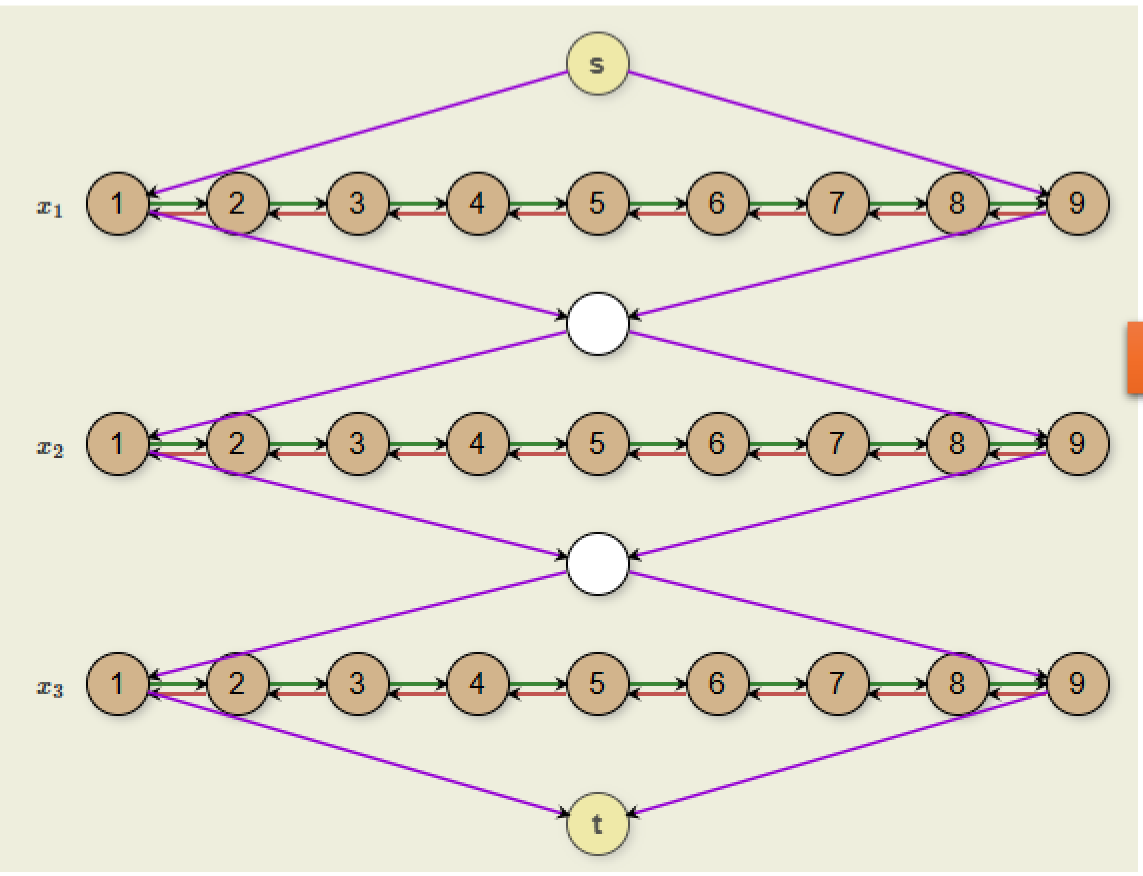
- Each variable gadget doesn’t handle clause connection all at once. It share responsibilities with row of nodes assigned to each variable.
- Each row for a variable handles 3 connections per clause, which result in 3k nodes for the clause connection in that row.

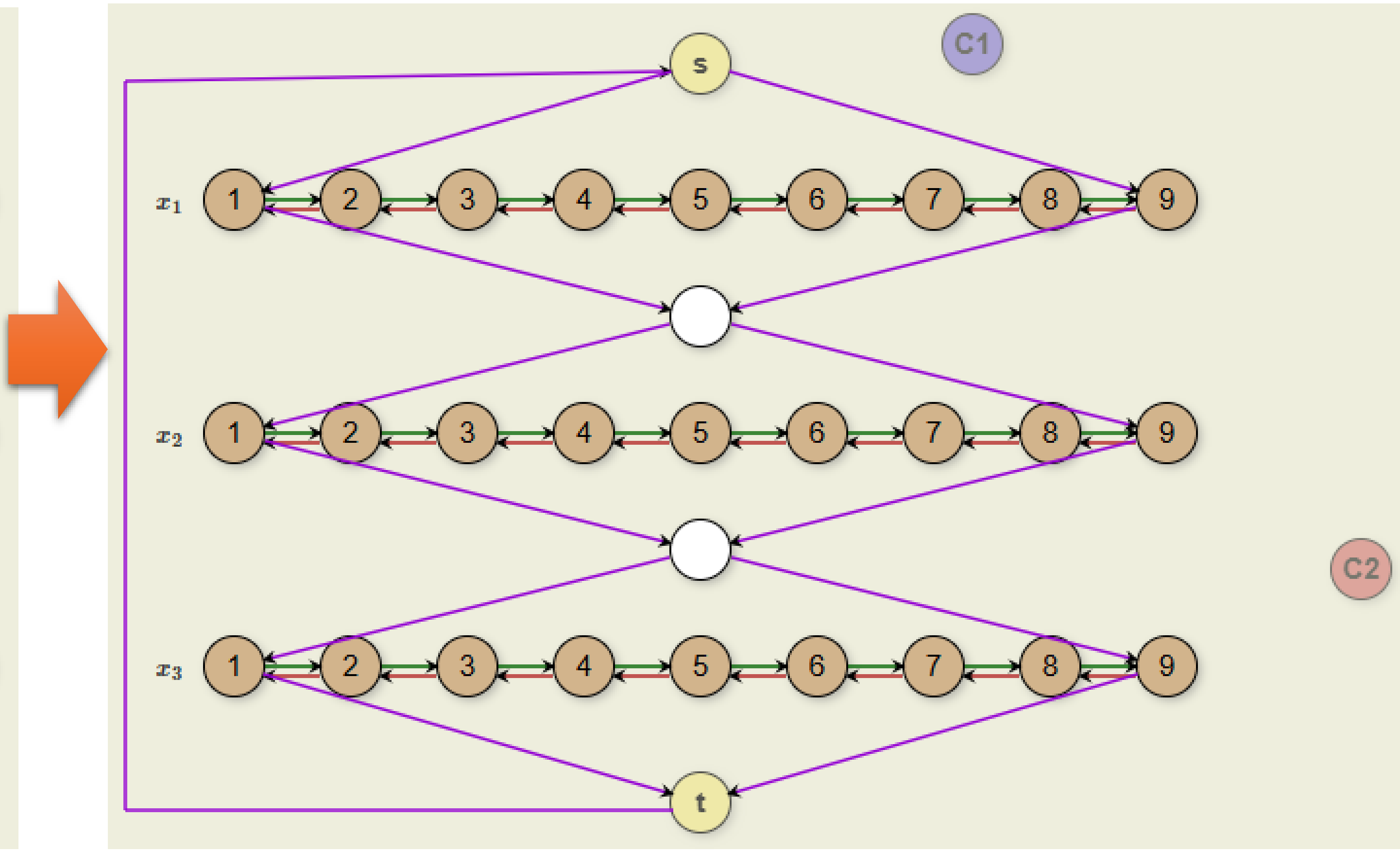
Graph construction : variable gadget

- Why 3 connections per row ?
- For each variable row
  - 1 connection for the positive literal
  - 1 connection for the negated literal
  - 1 auxiliary node or padding for traversing logical paths in gadget
- Additional 3 nodes (Entry, Exit and Control)
- Additional 3 nodes ensure that the Hamiltonian cycle
  - Enters the gadget correctly from other parts of the graph.
  - Traverses the True or False path for the variable
  - Exists the gadget correctly.

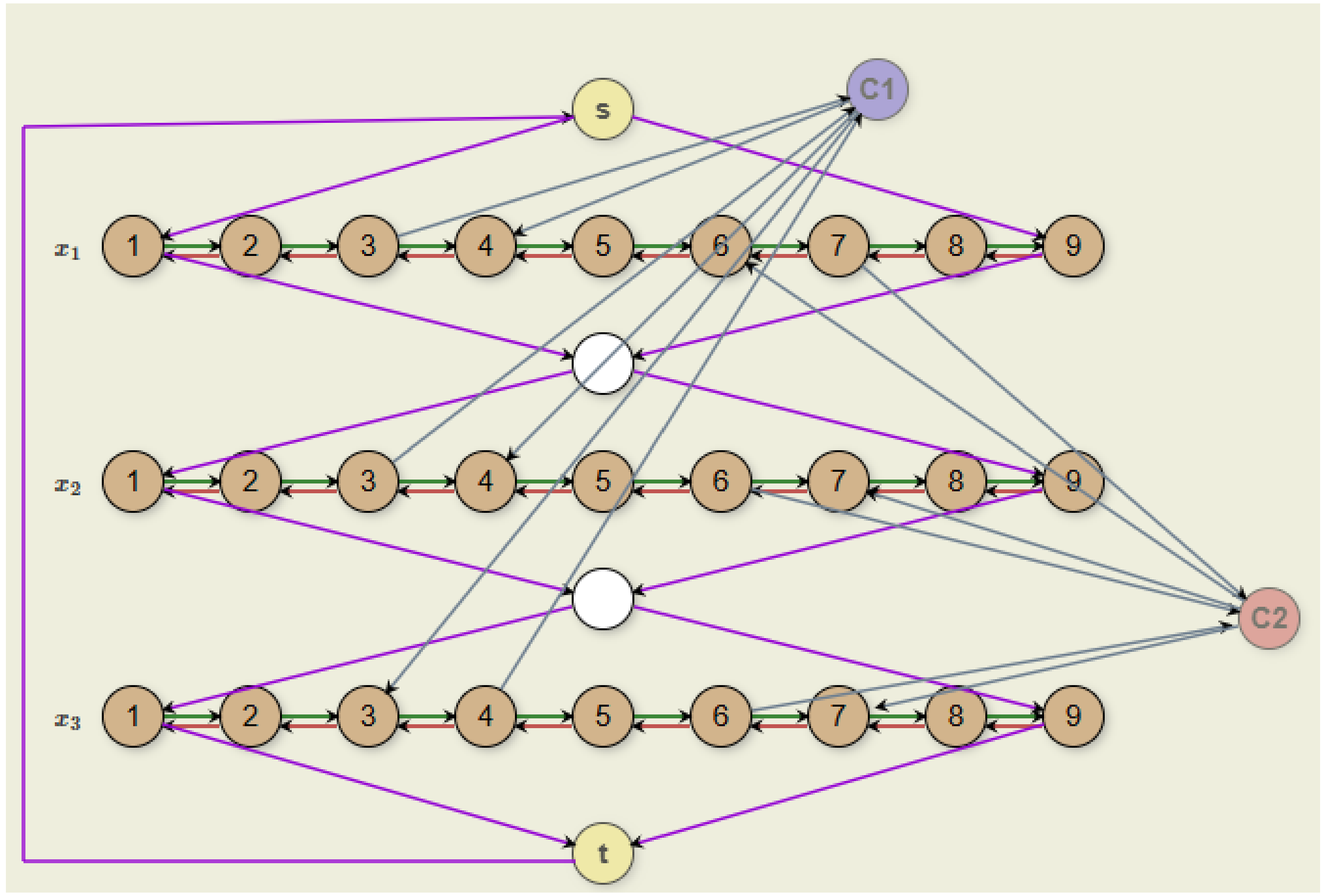
Graph construction

$(x_1 + x_2 + \overline{x_3}).(\overline{x_1} + x_2 + x_3).$





Completed graph construction



3SAT and Hamiltonian cycle

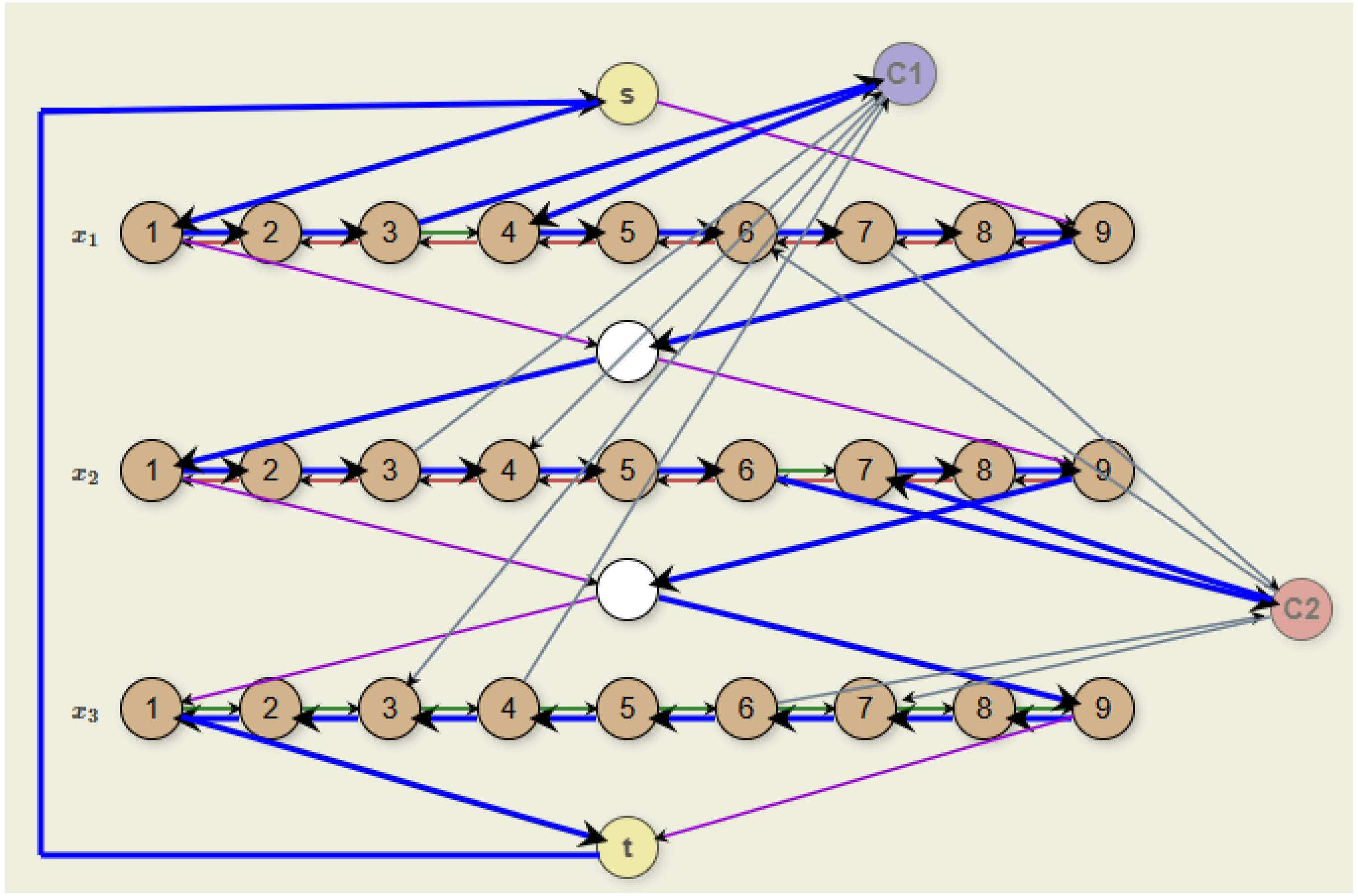
1. If there exist a Hamiltonian cycle H in the graph G

- If H traverses P1 from left to right, assign xi = True
- If H traverses P1 from right to left, assign xi = False
- The assignment obtained here satisfies the given 3CNF.

2. If there exists a satisfying assignment for the 3CNF.

- Select the path that traverses Pi from left to right if xi=true or right to left if xi = false.
- Connect source to p1,pn to target and pi to pi+1 appropriately to maintain the continuity of the path
- Connect the target to source to complete the cycle.

Hamiltonian cycle in the constructed graph



Proving Hamiltonian Cycle is NP-Complete

- **Step 1: Hamiltonian Cycle is in NP**
- **Certificate:** A sequence of vertices representing a Hamiltonian cycle (visiting each vertex exactly once and returning to the starting vertex).
- **Verification:**
  - Check if the sequence contains all vertices exactly once O(V)
  - Check if there is an edge between consecutive vertices in the sequence O(V)
  - Total verification time is $O(V^2)$, which is polynomial.
- **Conclusion:** Hamiltonian Cycle is in NP.

- **Step 2: Hamiltonian Cycle is NP-Hard**
- Reduce a known NP-complete problem (e.g., **3-SAT**) to Hamiltonian Cycle in polynomial time.
- Sketch of the reduction:
    - Construct a graph G from the 3-SAT formula such that:
        - Each clause is represented by a subgraph.
        - Each variable is represented by a subgraph.
        - Connections between variable and clause subgraphs enforce the logical constraints.
    - A Hamiltonian cycle in G exists if and only if the 3-SAT formula is satisfiable.
- Prove that the reduction process takes polynomial time.
- **Step 3: Conclude NP-Completeness**
- Since Hamiltonian Cycle is in NP and NP-hard, it is NP-complete.

Traveling Salesperson

- Reduce Hamiltonian Cycle to Traveling Salesperson

Traveling Salesman

- A *tour* is a Hamiltonian cycle in a graph. We want the minimum cost tour in a weighted graph.

- **TSP:**
  - **Input**: A graph G, weights c for edges and a positive integer k.
  - **Output**: YES iff G with weights c has a TS tour of cost at most k.

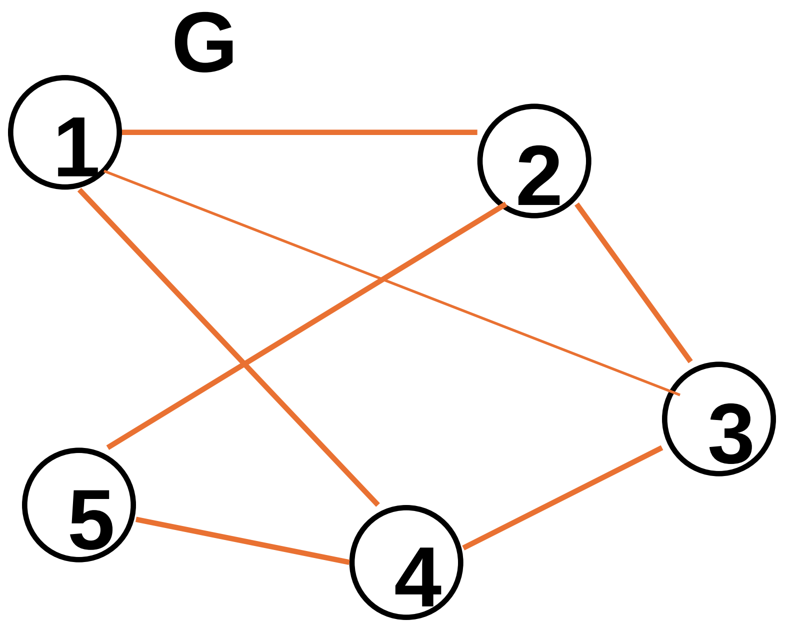
Traveling Salesman

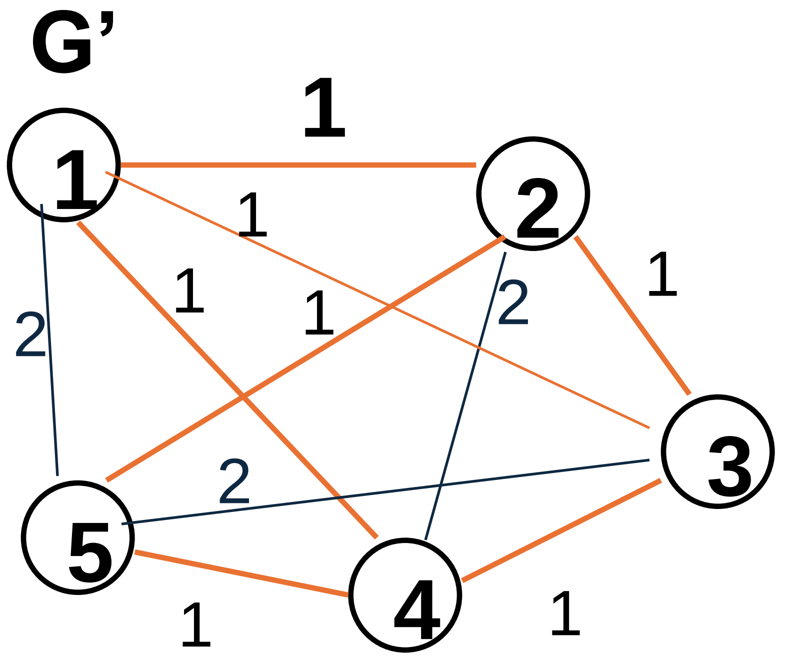
- **Theorem: TSP is NP-complete.**
- **Proof: Step 1: TSP is in NP**
    - The certificate is a representation of the tour, for example a permutation of the cities.
    - This certificate can be verified easily by checking that all cities are included exactly once and that the sum of the distances between all pairs of consecutive tour nodes is k or less.
    - This can be done in polynomial time, so TSP $\in$ NP.

The reduction

- Step 2: Select HAM-CYCLE (We will show that HAM-CYCLE $\propto$ TSP).
- Step 3: The reduction
  - Given an instance G of HAM-CYCLE, we construct a graph G' = (V, E'). G' is a complete graph and c(i,j) = 1 if (i,j) is an edge and 2 or infinity otherwise.
  - Find out if there is TSP with length n where n is the number of the vertices.

The reduction (example)





The reduction (step 4)

- If G has a Hamiltonian cycle h, each edge in h belongs to E and thus has no cost in G'. Thus h is a tour with cost n.

- If G' has a tour of cost n, the tour must have edges from E (since any edge not in E adds 2 to the cost). Thus, the tour must be a Hamiltonian cycle in G.

Vertex Cover

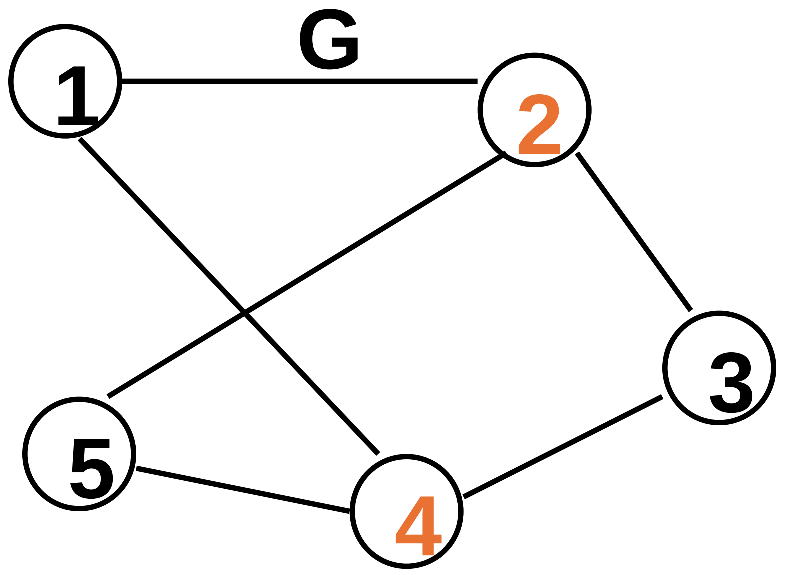
- Reduce clique to vertex cover

The vertex-cover problem

- **A *vertex cover* of an undirected graph is a set of vertices V' such that for every edge (u,v), either u or v or both are in V'. The problem is to find a cover of minimum size.**

- VERTEX-COVER
  - Input**: A graph G and a number k.**
  - Output**: YES iff G has a vertex cover of size k.**

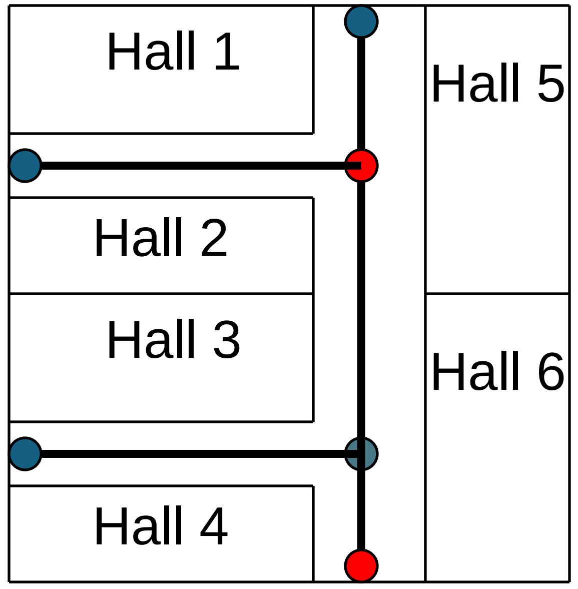
Example of a vertex cover problem

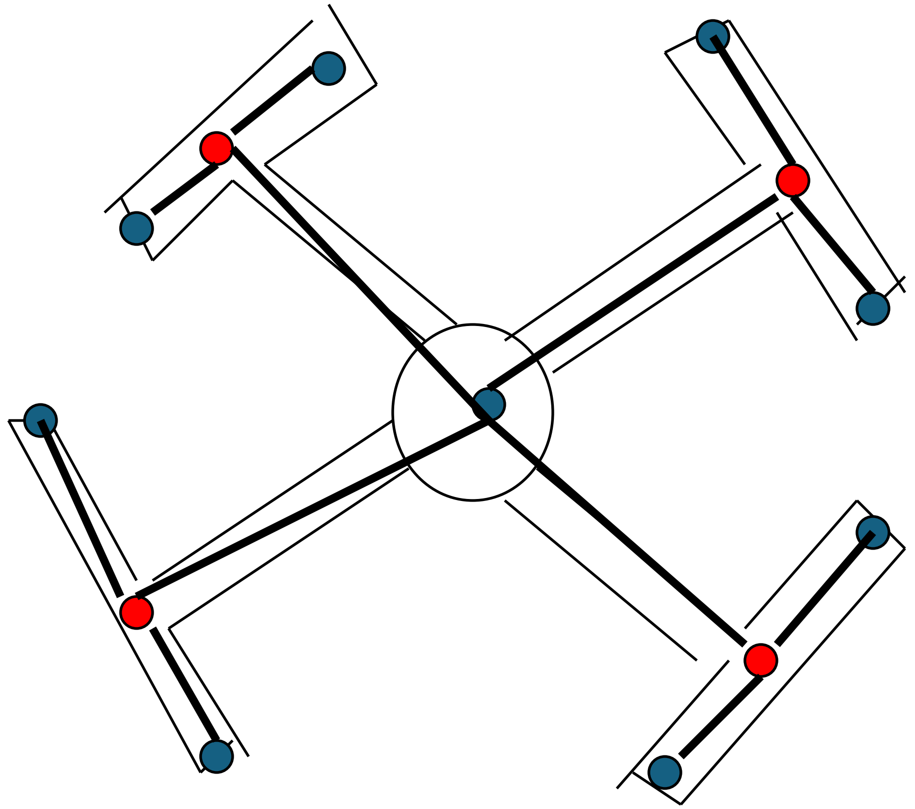


k=2

Application of vertex cover

- What is the fewest # of guards we need to place in a museum to cover all the corridors? An airport to cover all the main walkways





The vertex-cover problem

- **Theorem**: VERTEX-COVER is NP-complete.

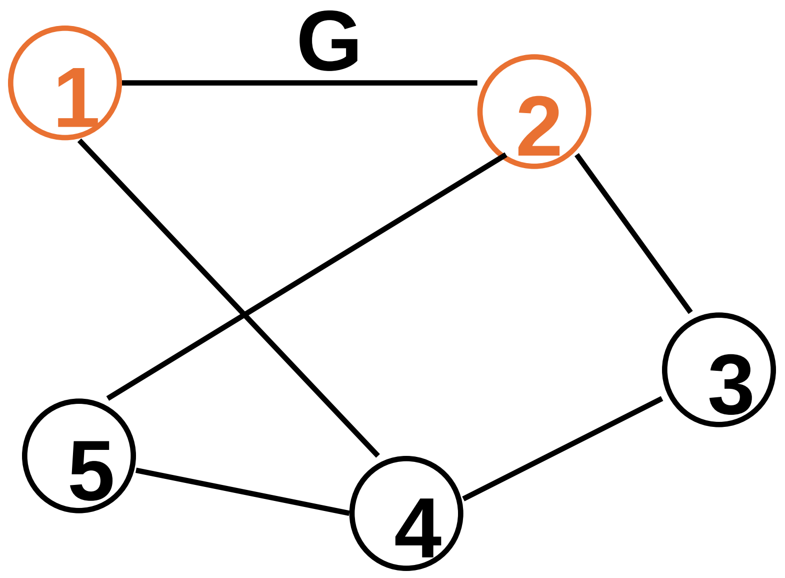
- **Proof**: **Step 1**. VERTEX-COVER $\in$ NP (obvious algorithm, given a subset of vertices).

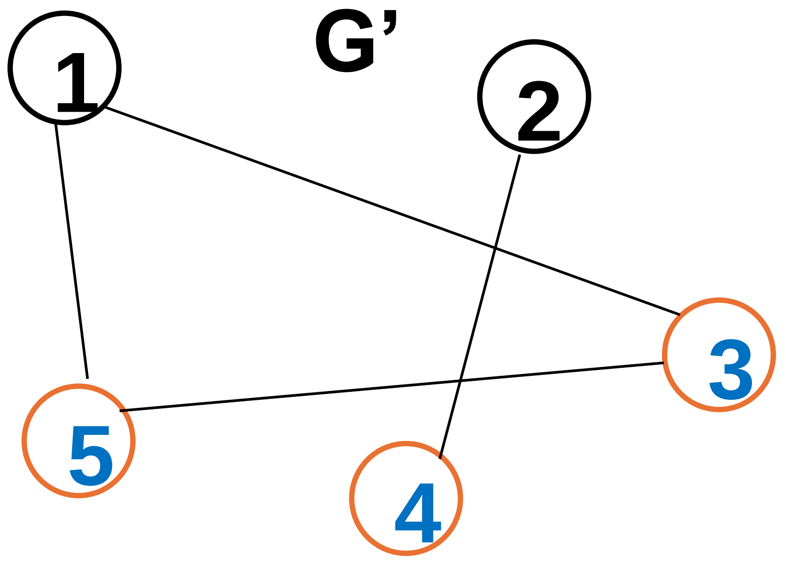
- **Step 2**. We select CLIQUE (will show that CLIQUE $\propto$ VERTEX-COVER)

The reduction

- **Step 3**. The mapping.
- Given an instance of the CLIQUE problem <G, k> we output an instance <G’, |V|-k> of the VERTEX-COVER problem.
- G’ has the same vertices as G and exactly those edges that are not in G.
- It is easy to show the reduction is polynomial (step 5)

Reduction Example



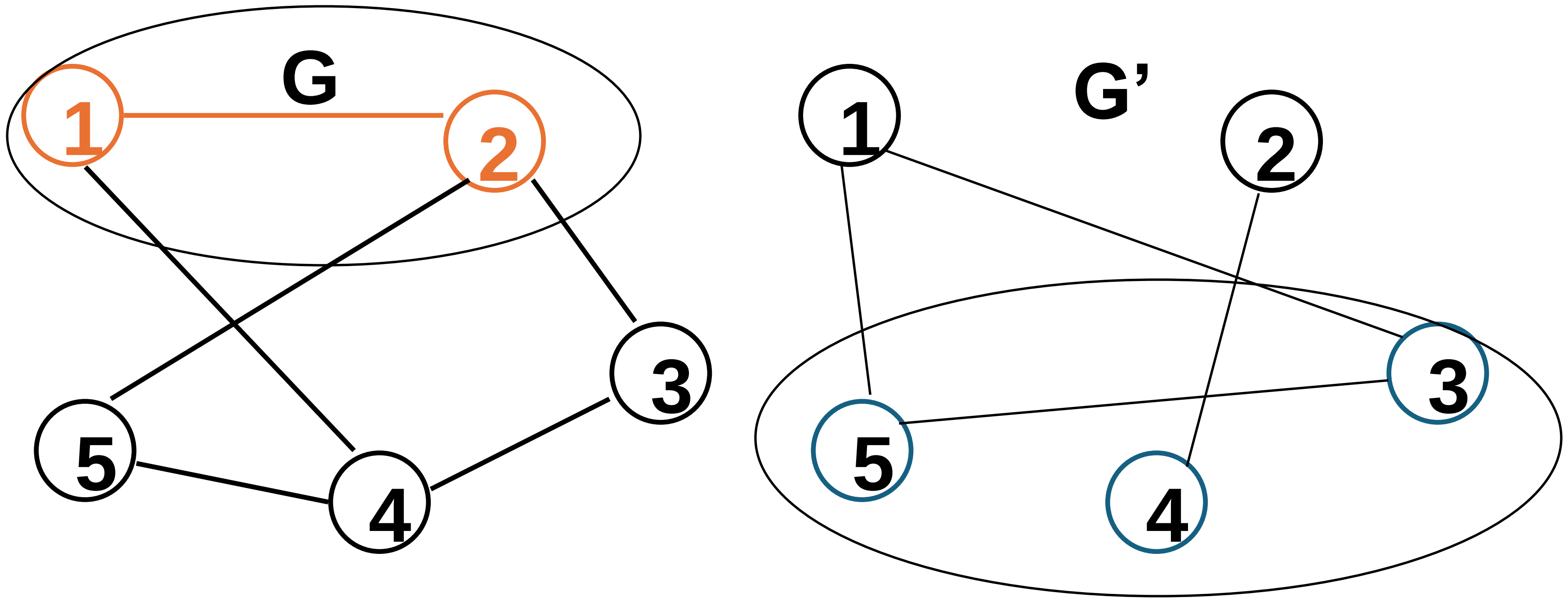


Clique {1,2} of size 2

Cover {3,4,5} of size 3

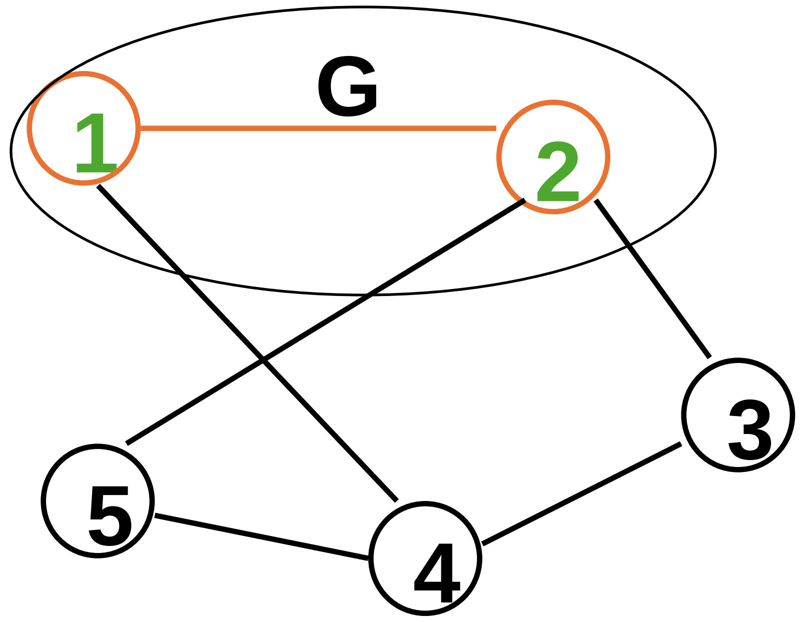
Step 4. Correctness of the reduction

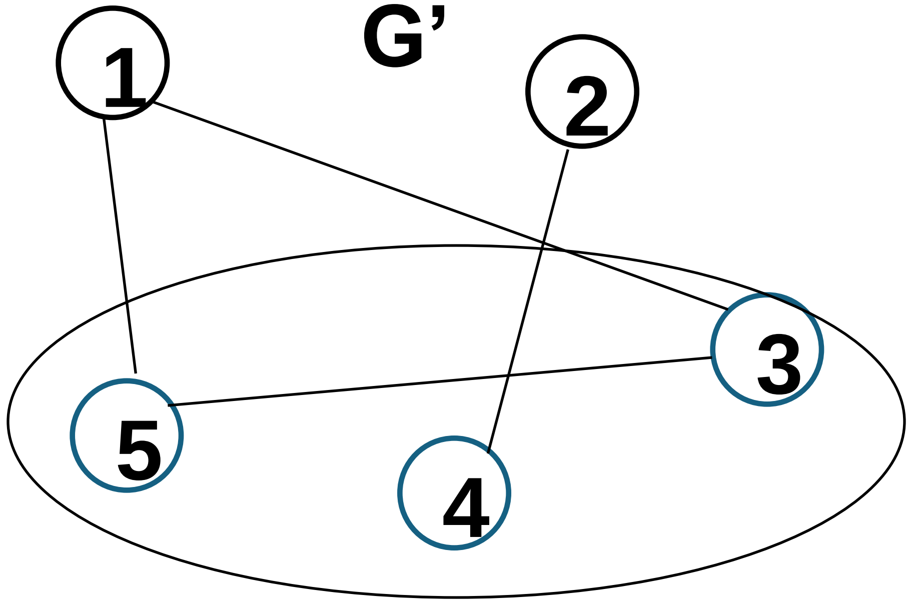
- Assume G has a clique C of size k.
- In G' there are no edges between any pair of vertices in C



Step 4 cont

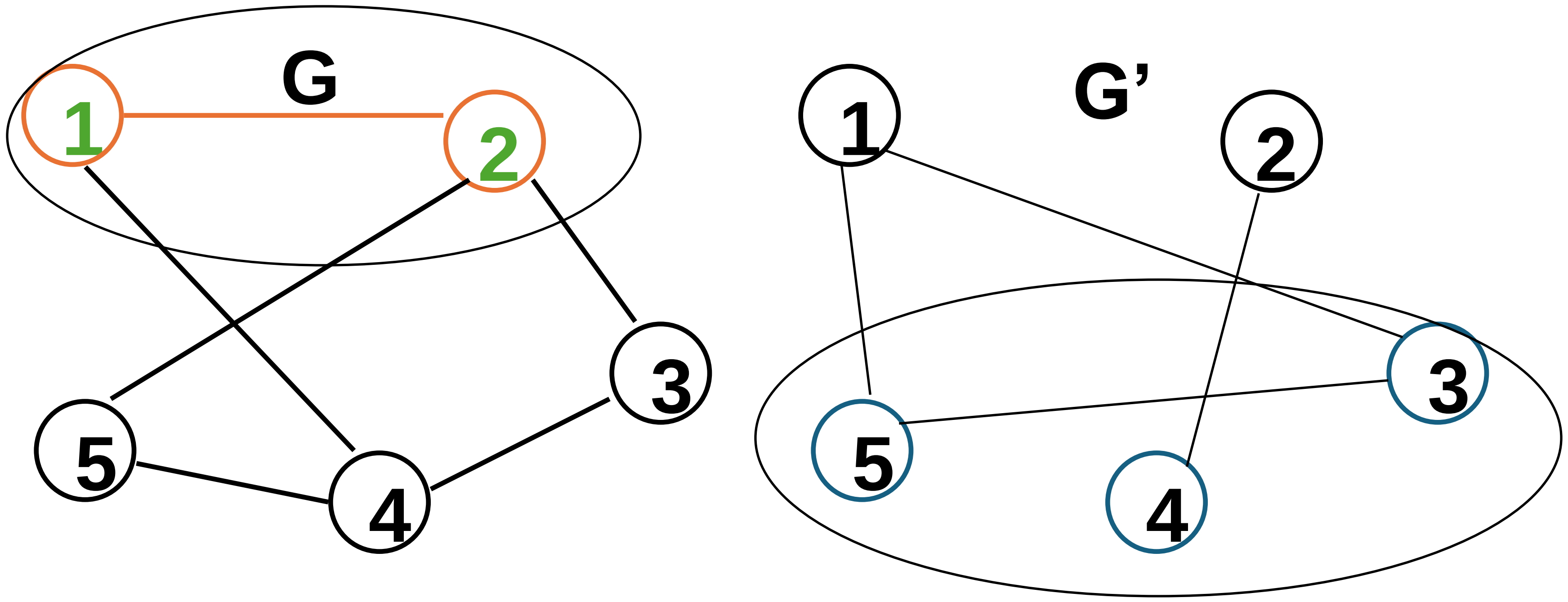
- So all edges in G’ are between a node in C and a node in V-C, or two nodes in V-C.
- So V-C is a vertex cover for G’.



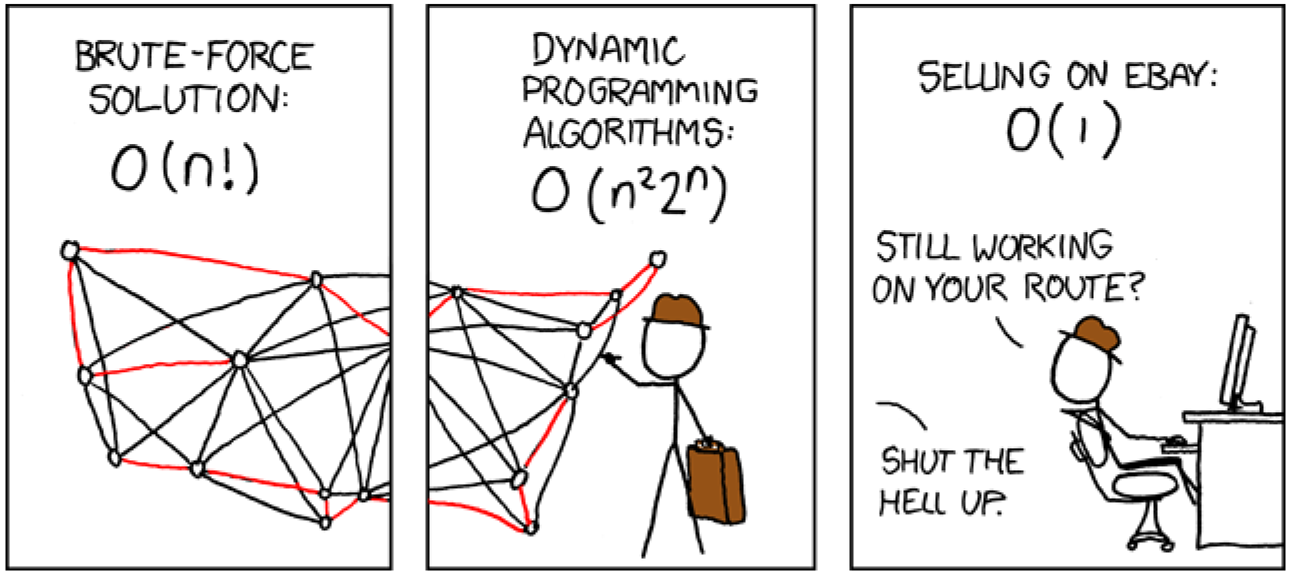


Step 4. Correctness of the reduction

- Assume G'=(V, E') has a vertex cover V' $\subseteq$ V, where |V'| = |V|-k.
- Thus for all u, v $\in$ V-V' (not in the cover), (u,v) $\notin$ E' and thus (u,v) $\in$ E
- V-V' is thus a clique.



Questions?


BRUTE-FORCE SOLUTION:
O(n!)
DYNAMIC PROGRAMMING ALGORITHMS:
O(n²2ⁿ)
SELLING ON EBAY:
O(1)
STILL WORKING ON YOUR ROUTE?
SHUT THE HELL UP.