

# Design and Analysis of Algorithms

---

## 4. NP and Computational Intractability

**Mingyu XIAO**

School of Computer Science and Engineering  
University of Electronic Science and Technology of China

# Algorithm Design Patterns and Anti-Patterns

## Algorithm design patterns.

- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Duality.
- Reductions.

## Ex.

$O(n \log n)$  interval scheduling.

$O(n \log n)$  FFT.

$O(n^2)$  edit distance.

$O(n^3)$  bipartite matching.

## Algorithm design anti-patterns.

- NP-completeness.
- PSPACE-completeness.
- Undecidability.

$O(n^c)$  algorithm unlikely.

$O(n^c)$  certification algorithm unlikely.

No algorithm possible.

## 4.1 Polynomial-Time Reductions

---

# Classify Problems According to Computational Requirements

Q. Which problems will we be able to solve in practice?

**A working definition.** [von Neumann 1953, Godel 1956, Cobham 1964, Edmonds 1965, Rabin 1966]

Those with polynomial-time algorithms.

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

# Classify Problems

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

**Frustrating news.** Huge number of fundamental problems have defined classification for decades.

**This chapter.** Show that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one **really hard** problem.

# Polynomial-Time Reduction

don't confuse with reduces from

**Reduction.** Problem X **polynomially reduces to** problem Y if arbitrary instances of problem X can be solved using:

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

**Notation.**  $X \leq_p Y$ .

computational model supplemented by special piece of hardware that solves instances of Y in a single step

**Questions.**

- Any requirement on the input size of Y?

# Polynomial-Time Reduction

**Purpose.** Classify problems according to **relative** difficulty.

**Design algorithms.** If  $X \leq_p Y$  and  $Y$  can be solved in polynomial-time, then  $X$  can also be solved in polynomial time.

**Establish intractability.** If  $X \leq_p Y$  and  $X$  cannot be solved in polynomial-time, then  $Y$  cannot be solved in polynomial time.

**Establish equivalence.** If  $X \leq_p Y$  and  $Y \leq_p X$ , we use notation  $X \equiv_p Y$ .

  
up to cost of reduction

# Reduction By Simple Equivalence

---

Basic reduction strategies.

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

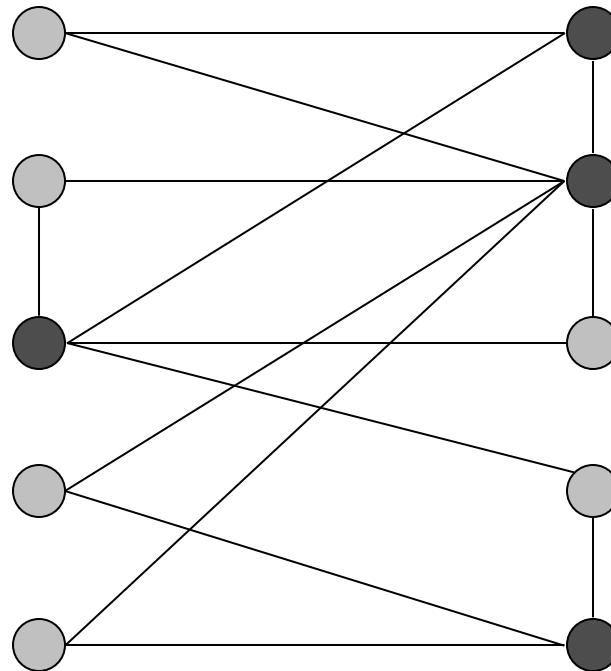


# Independent Set

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

**Ex.** Is there an independent set of size  $\geq 6$ ? Yes.

**Ex.** Is there an independent set of size  $\geq 7$ ? No.



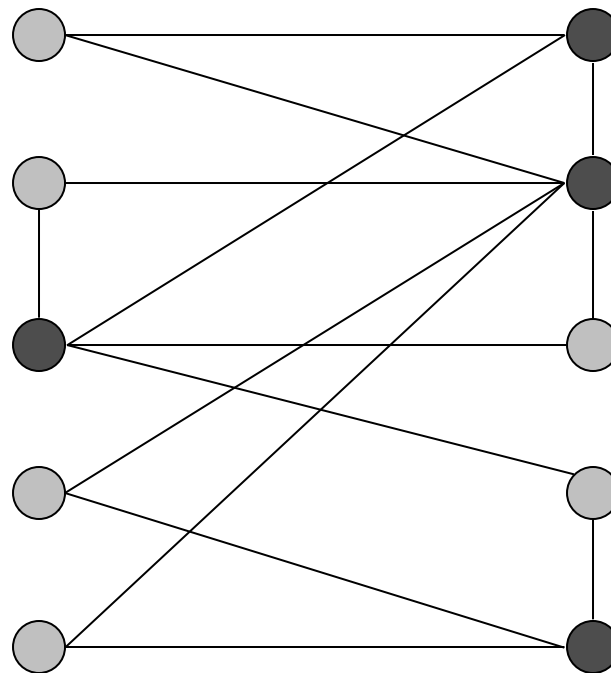
● independent set

# Vertex Cover

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

**Ex.** Is there a vertex cover of size  $\leq 4$ ? Yes.

**Ex.** Is there a vertex cover of size  $\leq 3$ ? No.

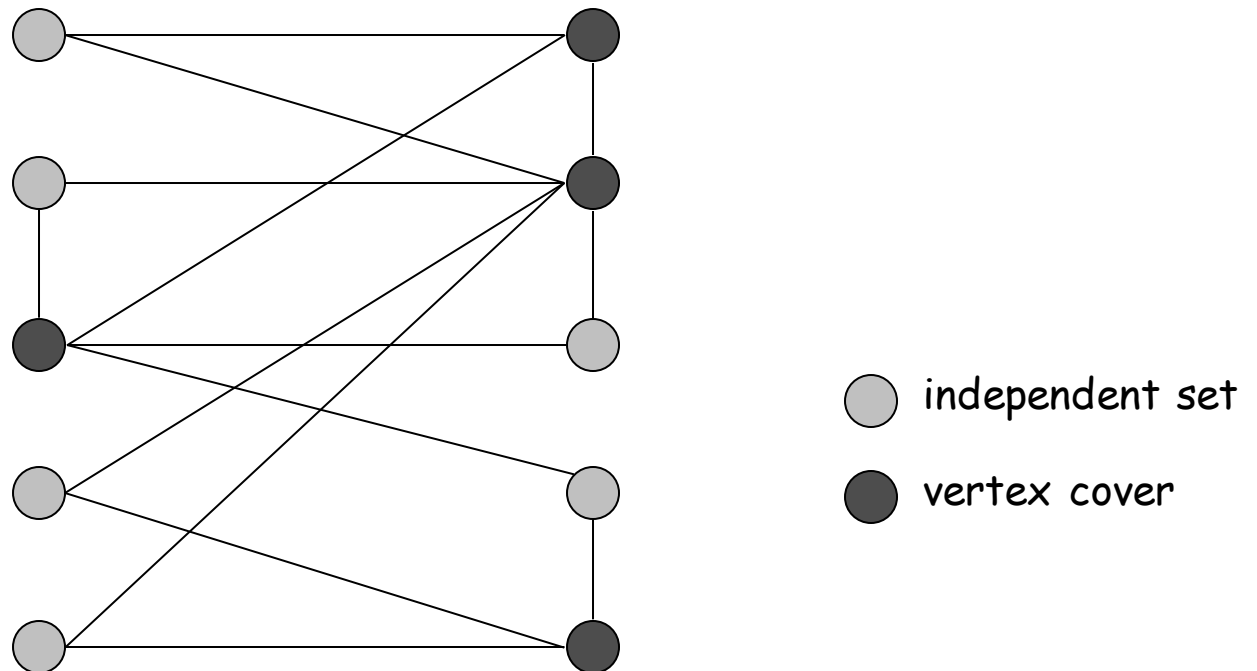


● vertex cover

# Vertex Cover and Independent Set

**Claim.** VERTEX-COVER  $\equiv_p$  INDEPENDENT-SET.

**Pf.** We show  $S$  is an independent set iff  $V - S$  is a vertex cover.



# Vertex Cover and Independent Set

**Claim.** VERTEX-COVER  $\equiv_p$  INDEPENDENT-SET.

**Pf.** We show  $S$  is an independent set iff  $V - S$  is a vertex cover.

$\Rightarrow$

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

$\Leftarrow$

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

# Reduction from Special Case to General Case

---

Basic reduction strategies.

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

# Set Cover

**SET COVER:** Given a set  $U$  of elements, a collection  $S_1, S_2, \dots, S_m$  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 we would like our system to 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.

Ex:

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

$$k = 2$$

$$S_1 = \{3, 7\}$$

$$S_4 = \{2, 4\}$$

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

$$S_5 = \{5\}$$

$$S_3 = \{1\}$$

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

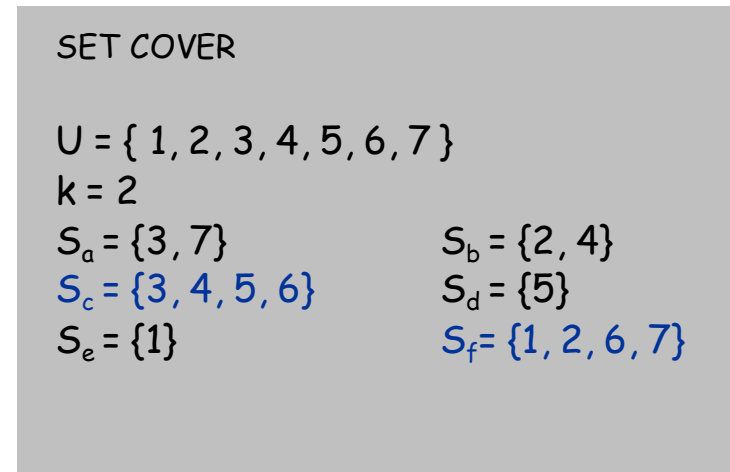
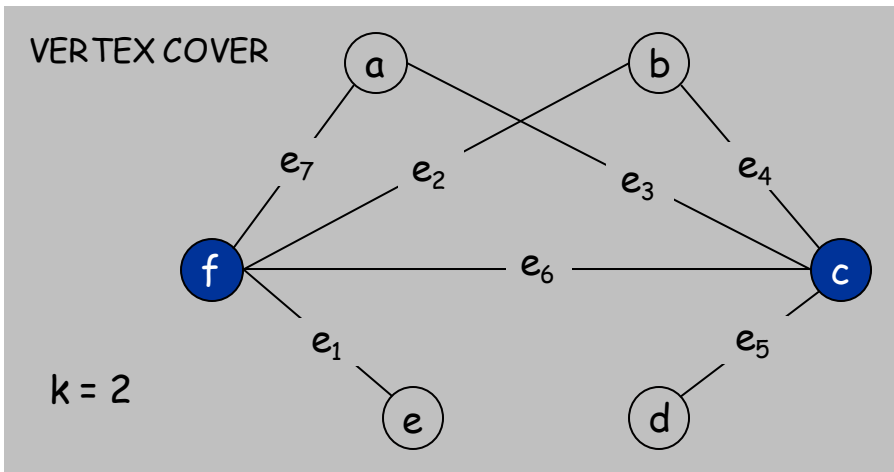
# Vertex Cover Reduces to Set Cover

**Claim.** VERTEX-COVER  $\leq_p$  SET-COVER.

**Pf.** 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  $\leq k$ . ▪



# Polynomial-Time Reduction

## Basic strategies.

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



## 4.2 Reductions via "Gadgets"

---

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction via "gadgets."

# Satisfiability

**Literal:** A Boolean variable or its negation.

$$x_i \text{ or } \overline{x_i}$$

**Clause:** A disjunction of literals.


$$C_j = x_1 \vee \overline{x_2} \vee x_3$$

**Conjunctive normal form:** A propositional formula  $\Phi$  that is the conjunction of clauses.

$$\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$$

**SAT:** Given CNF formula  $\Phi$ , does it have a satisfying truth assignment?

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

 each corresponds to a different variable

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

**Yes:**  $x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false}.$

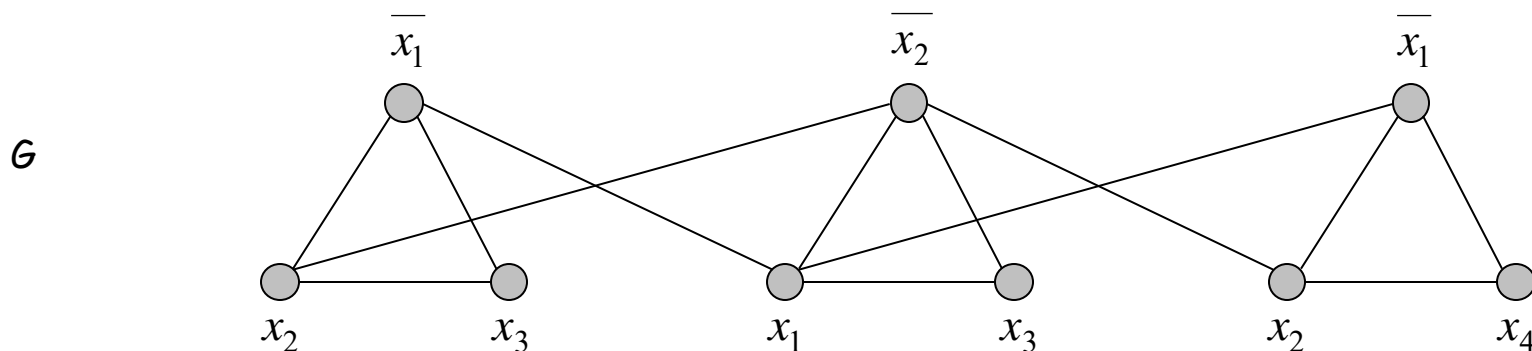
### 3 Satisfiability Reduces to Independent Set

**Claim.**  $3\text{-SAT} \leq_p \text{INDEPENDENT-SET}$ .

**Pf.** Given an instance  $\Phi$  of 3-SAT, we construct an instance  $(G, k)$  of INDEPENDENT-SET that has an independent set of size  $k$  iff  $\Phi$  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.



$k = 3$

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

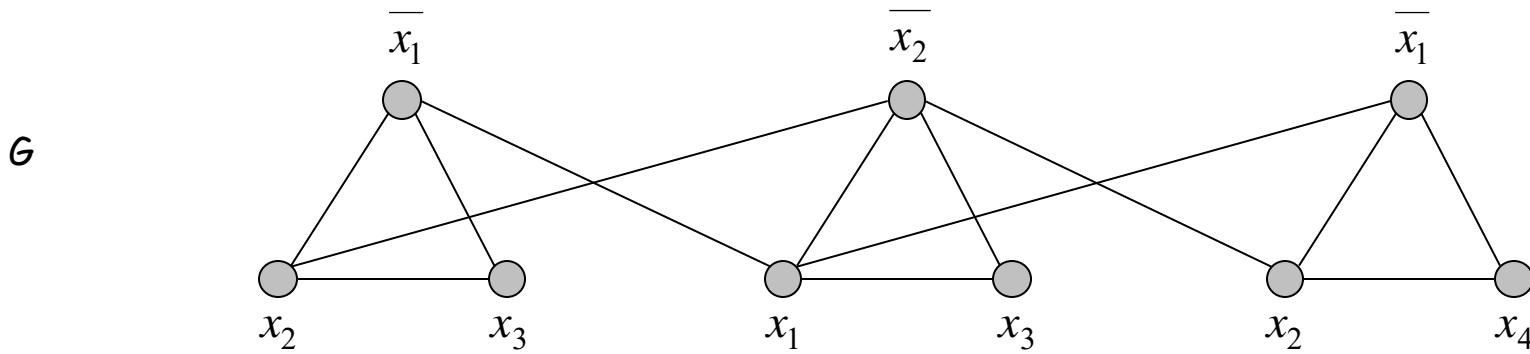
### 3 Satisfiability Reduces to Independent Set

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

**Pf.**  $\Rightarrow$  Let  $S$  be independent set of size  $k$ .

- $S$  must contain exactly one vertex in each triangle.
- Set these literals to true.  $\leftarrow$  and any other variables in a consistent way
- Truth assignment is consistent and all clauses are satisfied.

**Pf**  $\Leftarrow$  Given satisfying assignment, select one true literal from each triangle. This is an independent set of size  $k$ . ▀



$k = 3$

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

# Review

## Basic reduction strategies.

- Simple equivalence:  $\text{INDEPENDENT-SET} \equiv_p \text{VERTEX-COVER}$ .
- Special case to general case:  $\text{VERTEX-COVER} \leq_p \text{SET-COVER}$ .
- Encoding with gadgets:  $3\text{-SAT} \leq_p \text{INDEPENDENT-SET}$ .

**Transitivity.** If  $X \leq_p Y$  and  $Y \leq_p Z$ , then  $X \leq_p Z$ .

**Pf idea.** Compose the two algorithms.

**Ex:**  $3\text{-SAT} \leq_p \text{INDEPENDENT-SET} \leq_p \text{VERTEX-COVER} \leq_p \text{SET-COVER}$ .

# Self-Reducibility

**Decision problem.** Does there **exist** a vertex cover of size  $\leq k$ ?


**Search problem.** **Find** vertex cover of minimum cardinality.

**Self-reducibility.** Search problem  $\leq_p$  decision version.

- Applies to all (NP-complete) problems in this chapter.
- Justifies our focus on decision problems.

**Ex:** to find min cardinality vertex cover.

- (Binary) search for cardinality  $k^*$  of min vertex cover.
- Find a vertex  $v$  such that  $G - \{v\}$  has a vertex cover of size  $\leq k^* - 1$ .
  - any vertex in any min vertex cover will have this property
- Include  $v$  in the vertex cover.
- Recursively find a min vertex cover in  $G - \{v\}$ .

  
delete  $v$  and all incident edges

## 4.3 Definition of NP

---

# Decision Problems

## Decision problem.

- $X$  is a set of strings.
- Instance: string  $s$ .
- Algorithm  $A$  solves problem  $X$ :  $A(s) = \text{yes}$  iff  $s \in X$ .

**Polynomial time.** Algorithm  $A$  runs in poly-time if for every string  $s$ ,  $A(s)$  terminates in at most  $p(|s|)$  "steps", where  $p(\cdot)$  is some polynomial.

↑  
length of  $s$

**PRIMES:**  $X = \{ 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \dots \}$

**Algorithm.** [Agrawal-Kayal-Saxena, 2002]  $p(|s|) = |s|^8$ .



# Definition of P

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

Problem	Description	Algorithm	Yes	No
MULTIPLE	Is $x$ a multiple of $y$ ?	Grade school division	51, 17	51, 16
RELPRIME	Are $x$ and $y$ relatively prime?	Euclid (300 BCE)	34, 39	34, 51
PRIMES	Is $x$ prime?	AKS (2002)	53	51
EDIT-DISTANCE	Is the edit distance between $x$ and $y$ less than 5?	Dynamic programming	niether neither	acgggt ttttta
LSOLVE	Is there a vector $x$ that satisfies $Ax = b$ ?	Gauss-Edmonds elimination	$\left[ \begin{array}{ccc c} 0 & 1 & 1 & 4 \\ 2 & 4 & -2 & 2 \\ 0 & 3 & 15 & 36 \end{array} \right]$	$\left[ \begin{array}{ccc c} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{array} \right]$

# NP

## Certification algorithm intuition.

- Certifier views things from "managerial" viewpoint.
- Certifier doesn't determine whether  $s \in X$  on its own; rather, it checks a proposed proof  $t$  that  $s \in X$ .

**Def.** Algorithm  $C(s, t)$  is a **certifier** for problem  $X$  if for every string  $s$ ,  $s \in X$  iff there exists a string  $t$  such that  $C(s, t) = \text{yes}$ .

↑  
"certificate" or "witness"

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

↑  
 $C(s, t)$  is a poly-time algorithm and  
 $|t| \leq p(|s|)$  for some polynomial  $p(\cdot)$ .

**Remark.** NP stands for **nondeterministic** polynomial-time.

# Certifiers and Certificates: Composite

**COMPOSITES.** Given an integer  $s$ , is  $s$  composite?

**Certificate.** A nontrivial factor  $t$  of  $s$ . Note that such a certificate exists iff  $s$  is composite. Moreover  $|t| \leq |s|$ .

**Certifier.**

```
boolean C(s, t) {  
    if (t ≤ 1 or t ≥ s)  
        return false  
    else if (s is a multiple of t)  
        return true  
    else  
        return false  
}
```

**Instance.**  $s = 437,669$ .

**Certificate.**  $t = 541$  or  $809$ .  $\longleftarrow 437,669 = 541 \times 809$

**Conclusion.** COMPOSITES is in NP.

## Certifiers and Certificates: 3-Satisfiability

**SAT.** Given a CNF formula  $\Phi$ , is there a satisfying assignment?

**Certificate.** An assignment of truth values to the  $n$  boolean variables.

**Certifier.** Check that each clause in  $\Phi$  has at least one true literal.

**Ex.**

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

instance  $s$

$$x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$$

certificate  $t$

**Conclusion.** SAT is in NP.

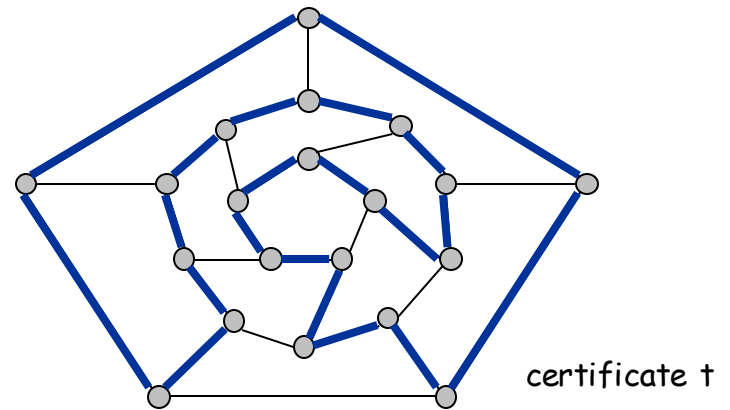
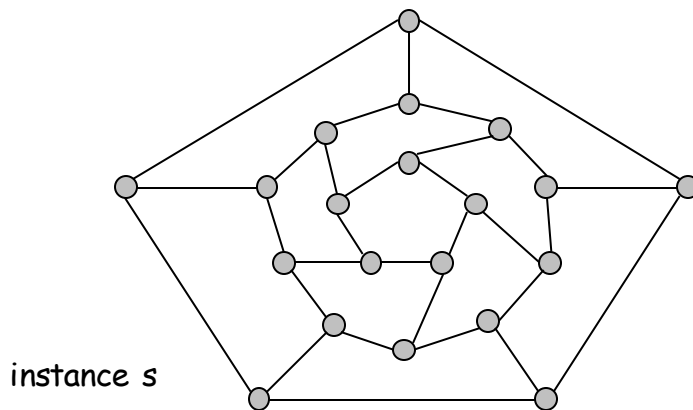
# Certifiers and Certificates: Hamiltonian Cycle

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

**Certificate.** A permutation of the  $n$  nodes.

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

**Conclusion.** HAM-CYCLE is in NP.



# P, NP, EXP

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

**EXP.** Decision problems for which there is an **exponential-time algorithm**.

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

**Claim.**  $P \subseteq NP$ .

**Pf.** Consider any problem  $X$  in  $P$ .

- By definition, there exists a poly-time algorithm  $A(s)$  that solves  $X$ .
- Certificate:  $t = \varepsilon$ , certifier  $C(s, t) = A(s)$ . ▪

**Claim.**  $NP \subseteq EXP$ .

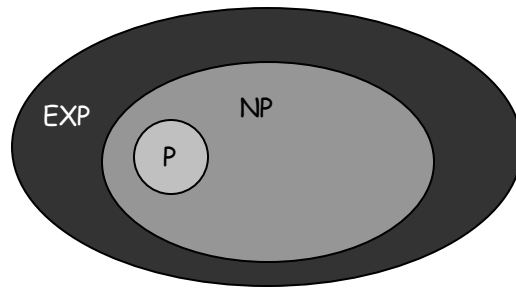
**Pf.** Consider any problem  $X$  in  $NP$ .

- By definition, there exists a poly-time certifier  $C(s, t)$  for  $X$ .
- To solve input  $s$ , run  $C(s, t)$  on all strings  $t$  with  $|t| \leq p(|s|)$ .
- Return **yes**, if  $C(s, t)$  returns **yes** for any of these. ▪

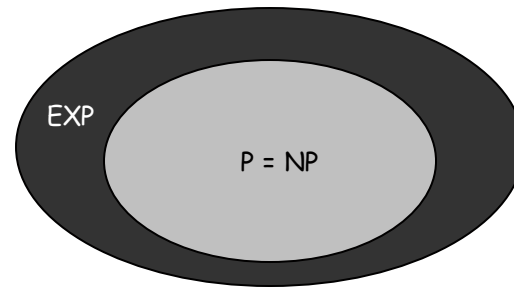
# The Main Question: P Versus NP

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

- Is the decision problem as easy as the certification problem?
- Clay \$1 million prize.



If  $P \neq NP$



If  $P = NP$

would break RSA cryptography  
(and potentially collapse economy)

If **yes**: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, ...

If **no**: No efficient algorithms possible for 3-COLOR, TSP, SAT, ...

Consensus opinion on  $P = NP$ ? Probably no.

## 4.4 NP-Completeness

---



# Polynomial Transformation

**Def.** Problem  $X$  **polynomial reduces** (Cook) to problem  $Y$  if arbitrary instances of problem  $X$  can be solved using:

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

**Def.** Problem  $X$  **polynomial transforms** (Karp) to problem  $Y$  if given any input  $x$  to  $X$ , we can construct an input  $y$  such that  $x$  is a  $_{yes}$  instance of  $X$  iff  $y$  is a  $_{yes}$  instance of  $Y$ .

↑  
we require  $|y|$  to be of size polynomial in  $|x|$

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

# NP-Complete

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

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

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

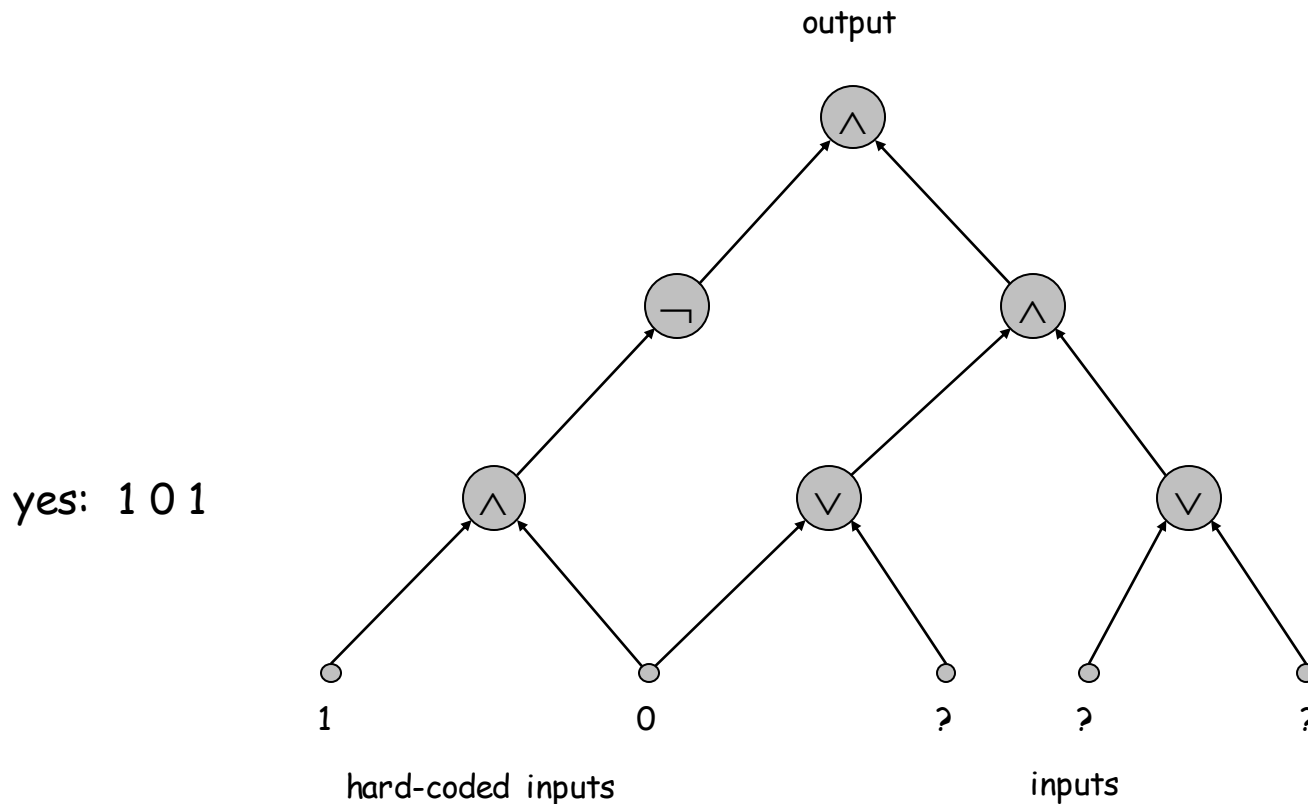
**Pf.**  $\Rightarrow$  Suppose  $Y$  can be solved in poly-time.

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

**Fundamental question.** Do there exist "natural" NP-complete problems?

# Circuit Satisfiability

**CIRCUIT-SAT.** Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?



# The "First" NP-Complete Problem

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

**Pf.** (sketch)

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

sketchy part of proof; fixing the number of bits is important, and reflects basic distinction between algorithms and circuits

- Consider some problem  $X$  in NP. It has a poly-time certifier  $C(s, t)$ . To determine whether  $s$  is in  $X$ , need to know if there exists a certificate  $t$  of length  $p(|s|)$  such that  $C(s, t) = \text{yes}$ .
- View  $C(s, t)$  as an algorithm on  $|s| + p(|s|)$  bits (input  $s$ , certificate  $t$ ) and convert it into a poly-size circuit  $K$ .
  - first  $|s|$  bits are hard-coded with  $s$
  - remaining  $p(|s|)$  bits represent bits of  $t$
- Circuit  $K$  is satisfiable iff  $C(s, t) = \text{yes}$ .

# Establishing NP-Completeness

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

**Recipe to establish NP-completeness of problem  $Y$ .**

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

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

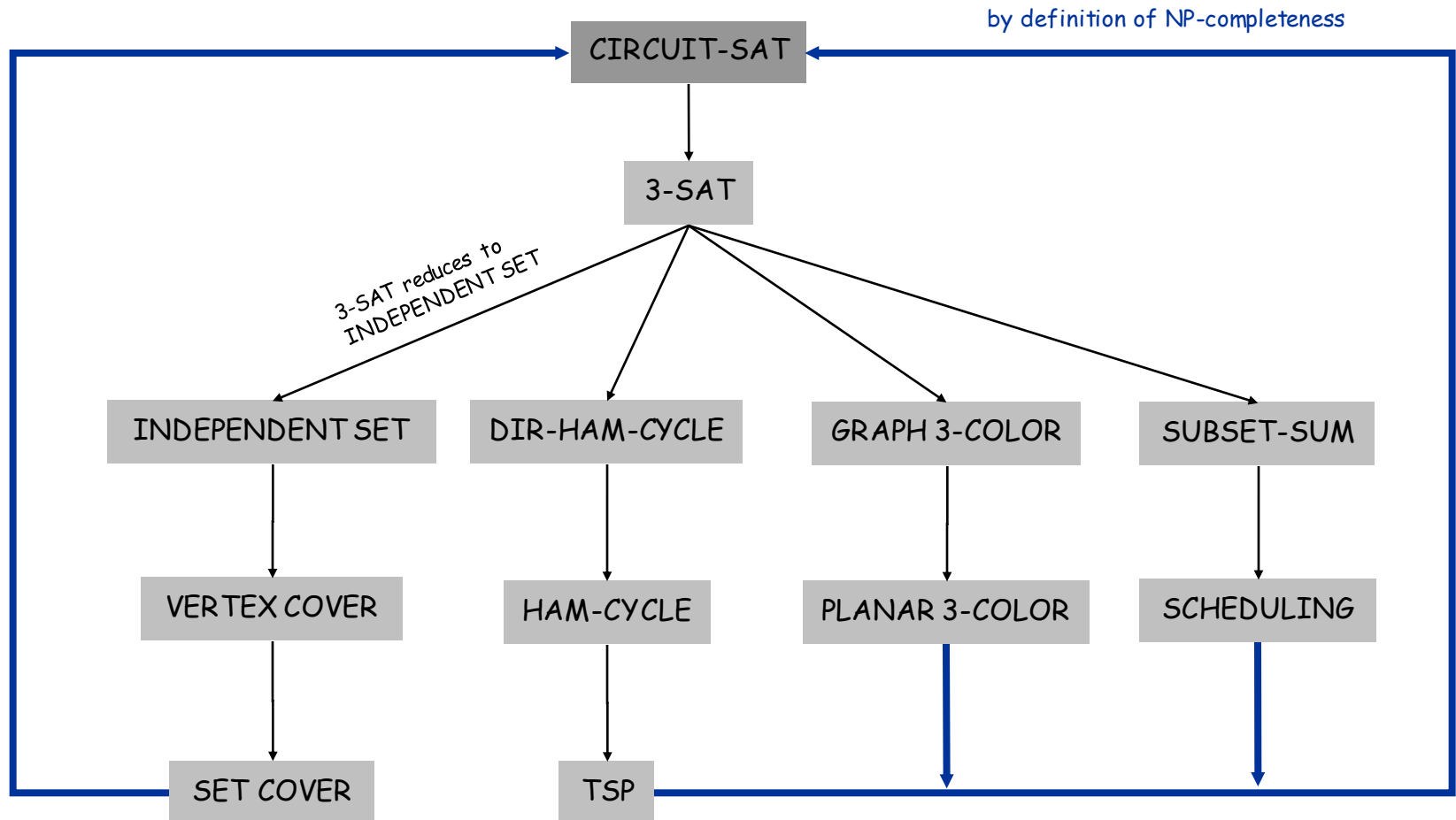
**Pf.** Let  $W$  be any problem in NP. Then  $W \leq_p X \leq_p Y$ .

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

$\uparrow$                        $\uparrow$   
by definition of      by assumption  
NP-complete

# NP-Completeness

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



# Some NP-Complete Problems

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

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

**Practice.** Most NP problems are either known to be in P or NP-complete.

**Notable exceptions.** Factoring, graph isomorphism, Nash equilibrium.

# Extent and Impact of NP-Completeness

Extent of NP-completeness. [Papadimitriou 1995]

- Prime intellectual export of CS to other disciplines.
- 6,000 citations per year (title, abstract, keywords).
  - more than "compiler", "operating system", "database"
- Broad applicability and classification power.
- "Captures vast domains of computational, scientific, mathematical endeavors, and seems to roughly delimit what mathematicians and scientists had been aspiring to compute feasibly."



# More Hard Computational Problems

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

## 4.5 Sequencing Problems

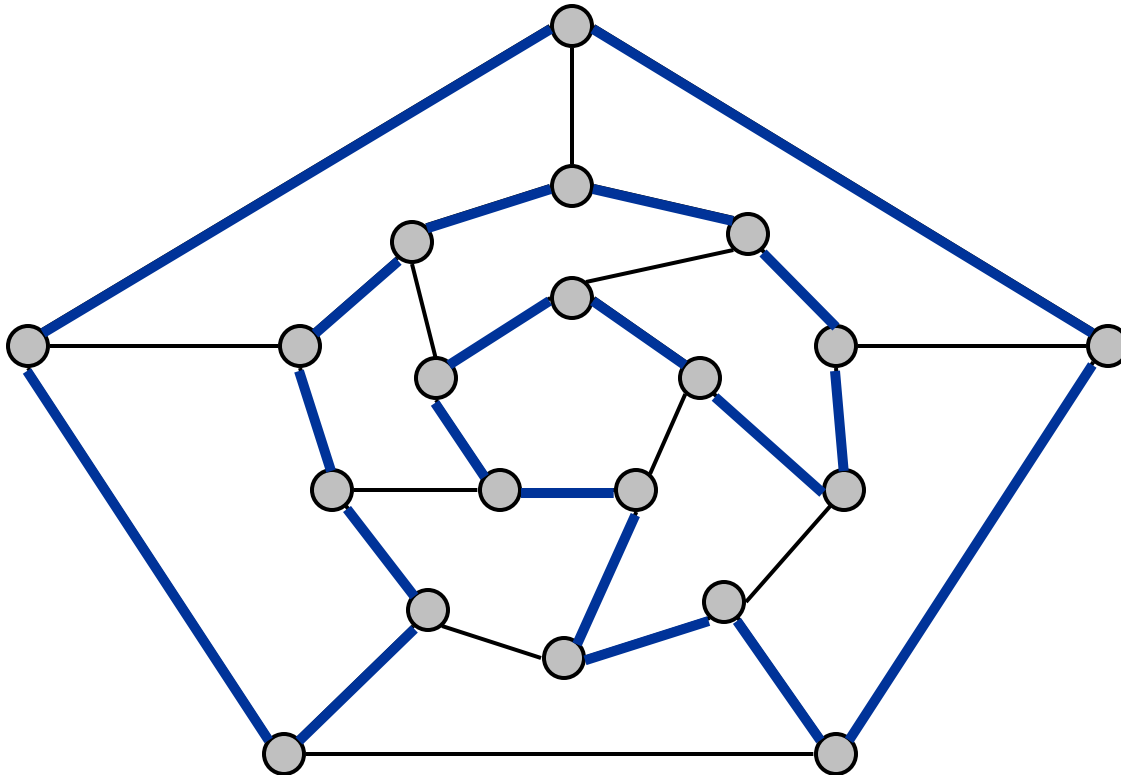
---

Basic genres.

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

# Hamiltonian Cycle

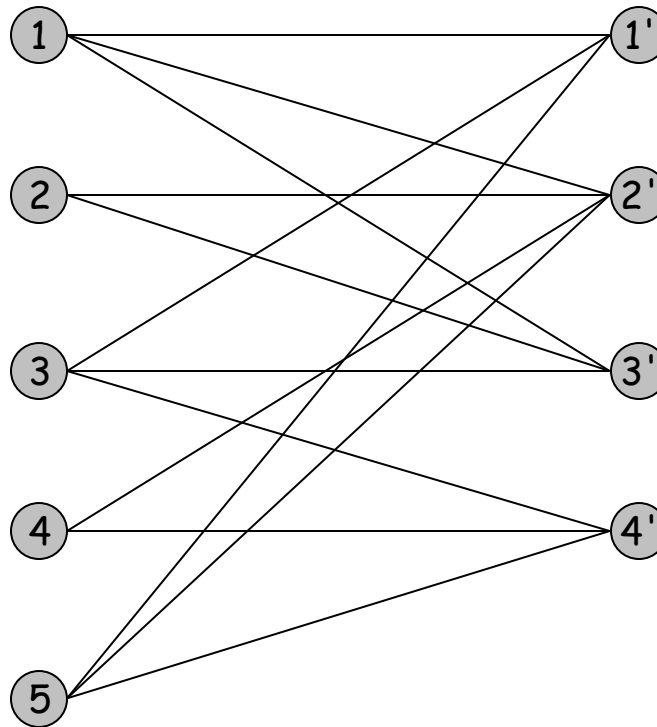
**HAM-CYCLE:** given an undirected graph  $G = (V, E)$ , does there exist a simple cycle  $\Gamma$  that contains every node in  $V$ .



YES: vertices and faces of a dodecahedron.

# Hamiltonian Cycle

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



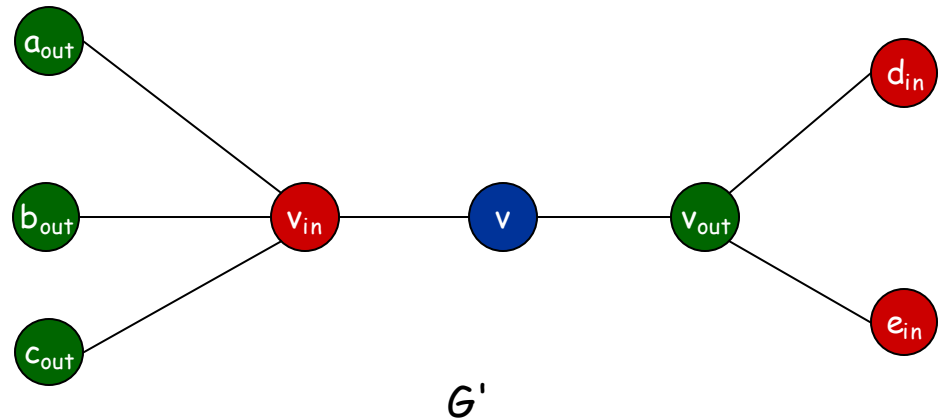
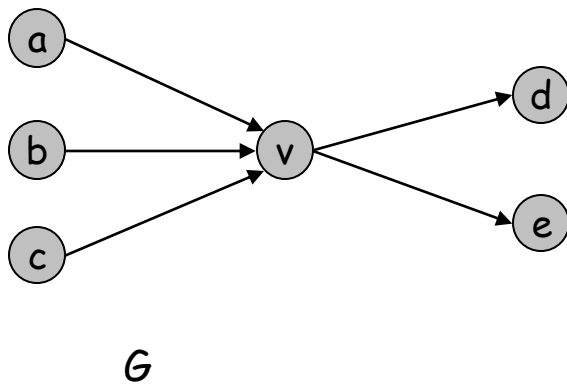
NO: bipartite graph with odd number of nodes.

# Directed Hamiltonian Cycle

**DIR-HAM-CYCLE:** given a **digraph**  $G = (V, E)$ , does there exist a simple directed cycle  $\Gamma$  that contains every node in  $V$ ?

**Claim.**  $\text{DIR-HAM-CYCLE} \leq_p \text{HAM-CYCLE}$ .

**Pf.** Given a directed graph  $G = (V, E)$ , construct an undirected graph  $G'$  with  $3n$  nodes.



# Directed Hamiltonian Cycle

**Claim.**  $G$  has a Hamiltonian cycle iff  $G'$  does.

**Pf.**  $\Rightarrow$

- Suppose  $G$  has a directed Hamiltonian cycle  $\Gamma$ .
- Then  $G'$  has an undirected Hamiltonian cycle (same order).

**Pf.**  $\Leftarrow$

- Suppose  $G'$  has an undirected Hamiltonian cycle  $\Gamma'$ .
- $\Gamma'$  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  $\Gamma'$  make up directed Hamiltonian cycle  $\Gamma$  in  $G$ , or reverse of one. ▪

## 3-SAT Reduces to Directed Hamiltonian Cycle

**Claim.**  $3\text{-SAT} \leq_p \text{DIR-HAM-CYCLE}$ .

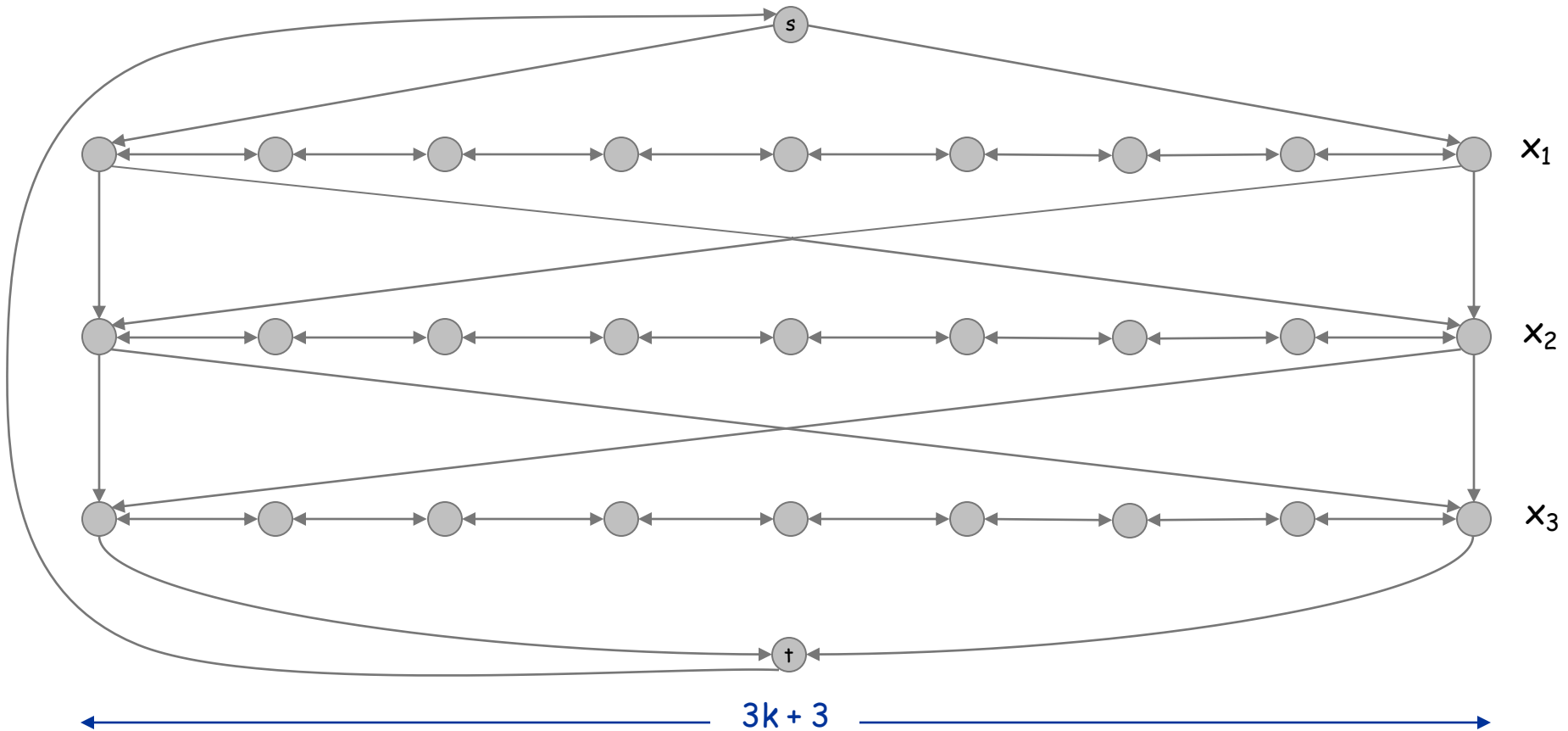
**Pf.** Given an instance  $\Phi$  of 3-SAT, we construct an instance of DIR-HAM-CYCLE that has a Hamiltonian cycle iff  $\Phi$  is satisfiable.

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

# 3-SAT Reduces to Directed Hamiltonian Cycle

**Construction.** Given 3-SAT instance  $\Phi$  with  $n$  variables  $x_i$  and  $k$  clauses.

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

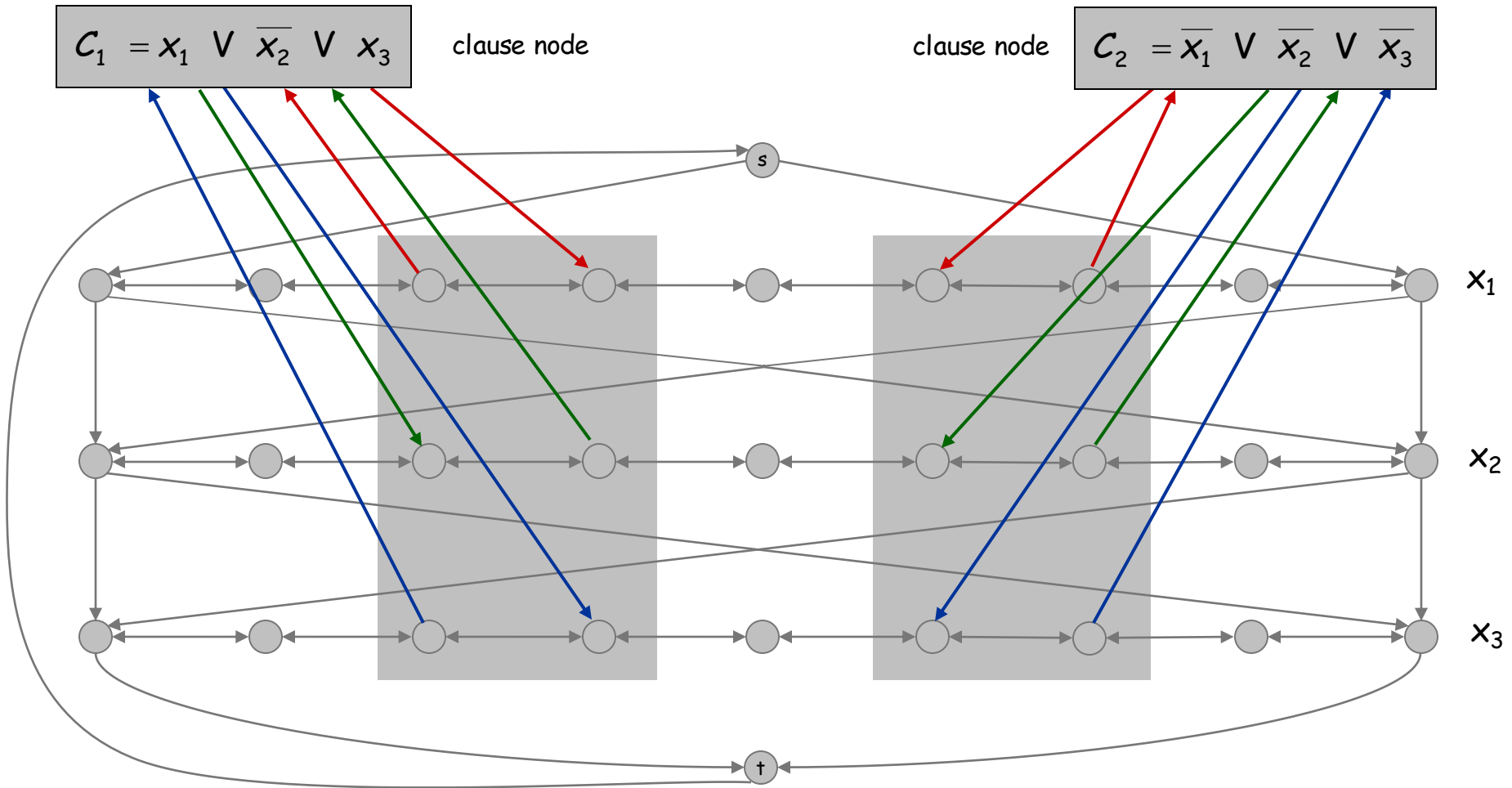




# 3-SAT Reduces to Directed Hamiltonian Cycle

**Construction.** Given 3-SAT instance  $\Phi$  with  $n$  variables  $x_i$  and  $k$  clauses.

- For each clause: add a node and 6 edges.



## 3-SAT Reduces to Directed Hamiltonian Cycle

**Claim.**  $\Phi$  is satisfiable iff  $G$  has a Hamiltonian cycle.

**Pf.**  $\Rightarrow$

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

## 3-SAT Reduces to Directed Hamiltonian Cycle

**Claim.**  $\Phi$  is satisfiable iff  $G$  has a Hamiltonian cycle.

**Pf.**  $\Leftarrow$

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

# Longest Path

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

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

**Claim.**  $3\text{-SAT} \leq_p \text{LONGEST-PATH}$ .

**Pf 1.** Redo proof for  $\text{DIR-HAM-CYCLE}$ , ignoring back-edge from  $t$  to  $s$ .

**Pf 2.** Show  $\text{HAM-CYCLE} \leq_p \text{LONGEST-PATH}$ .

# The Longest Path †

**Lyrics.** Copyright © 1988 by Daniel J. Barrett.

**Music.** Sung to the tune of *The Longest Time* by Billy Joel.



Woh-oh-oh-oh, find the longest path!  
Woh-oh-oh-oh, find the longest path!

If you said  $P$  is  $NP$  tonight,  
There would still be papers left to write,  
I have a weakness,  
I'm addicted to completeness,  
And I keep searching for the longest path.

The algorithm I would like to see  
Is of polynomial degree,  
But it's elusive:  
Nobody has found conclusive  
Evidence that we can find a longest path.

I have been hard working for so long.  
I swear it's right, and he marks it wrong.  
Some how I'll feel sorry when it's done:  
GPA 2.1  
Is more than I hope for.

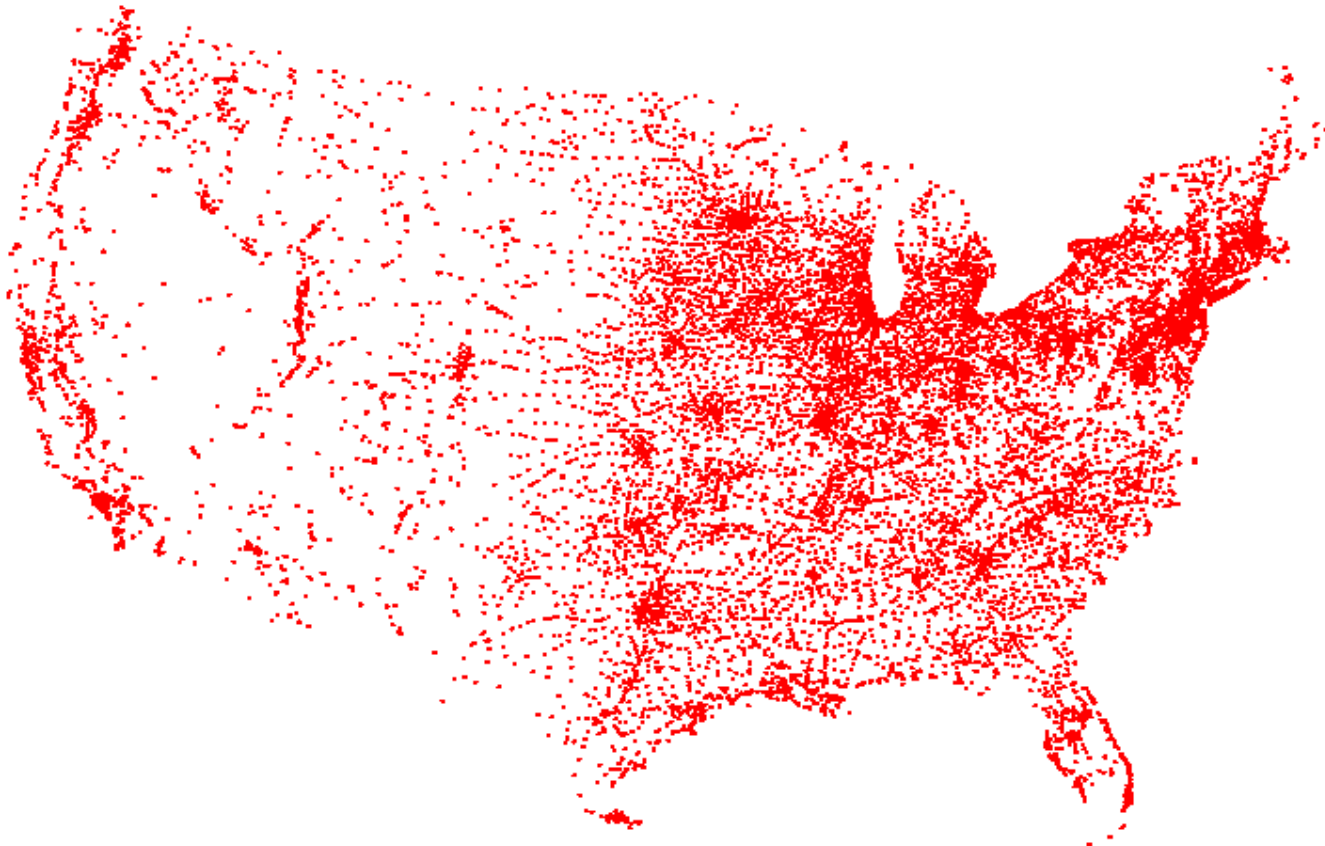
Garey, Johnson, Karp and other men (and women)  
Tried to make it order  $N \log N$ .  
Am I a mad fool  
If I spend my life in grad school,  
Forever following the longest path?

Woh-oh-oh-oh, find the longest path!  
Woh-oh-oh-oh, find the longest path!  
Woh-oh-oh-oh, find the longest path.

† Recorded by Dan Barrett while a grad student at Johns Hopkins during a difficult algorithms final.

# Traveling Salesperson Problem

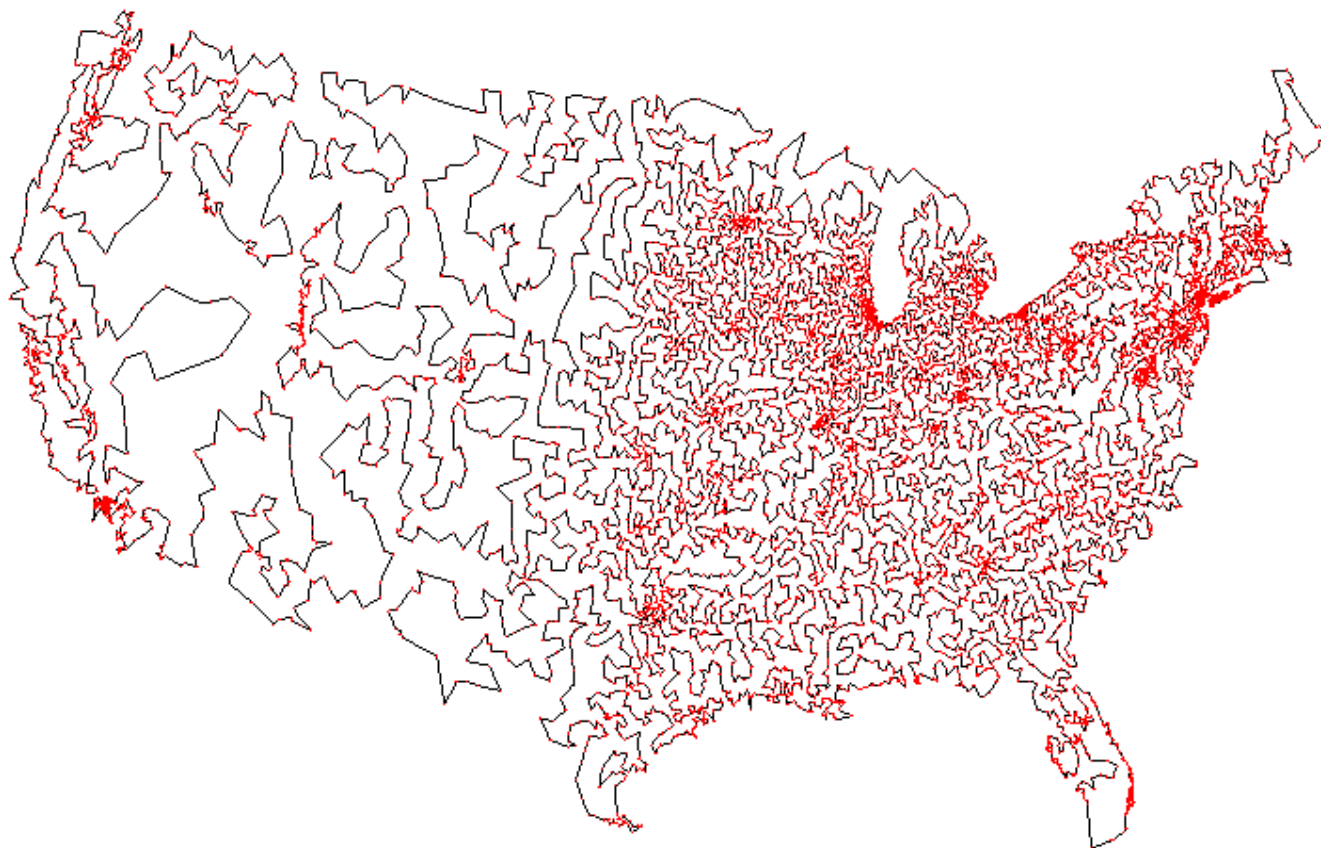
**TSP.** Given a set of  $n$  cities and a pairwise distance function  $d(u, v)$ , is there a tour of length  $\leq D$ ?



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

# Traveling Salesperson Problem

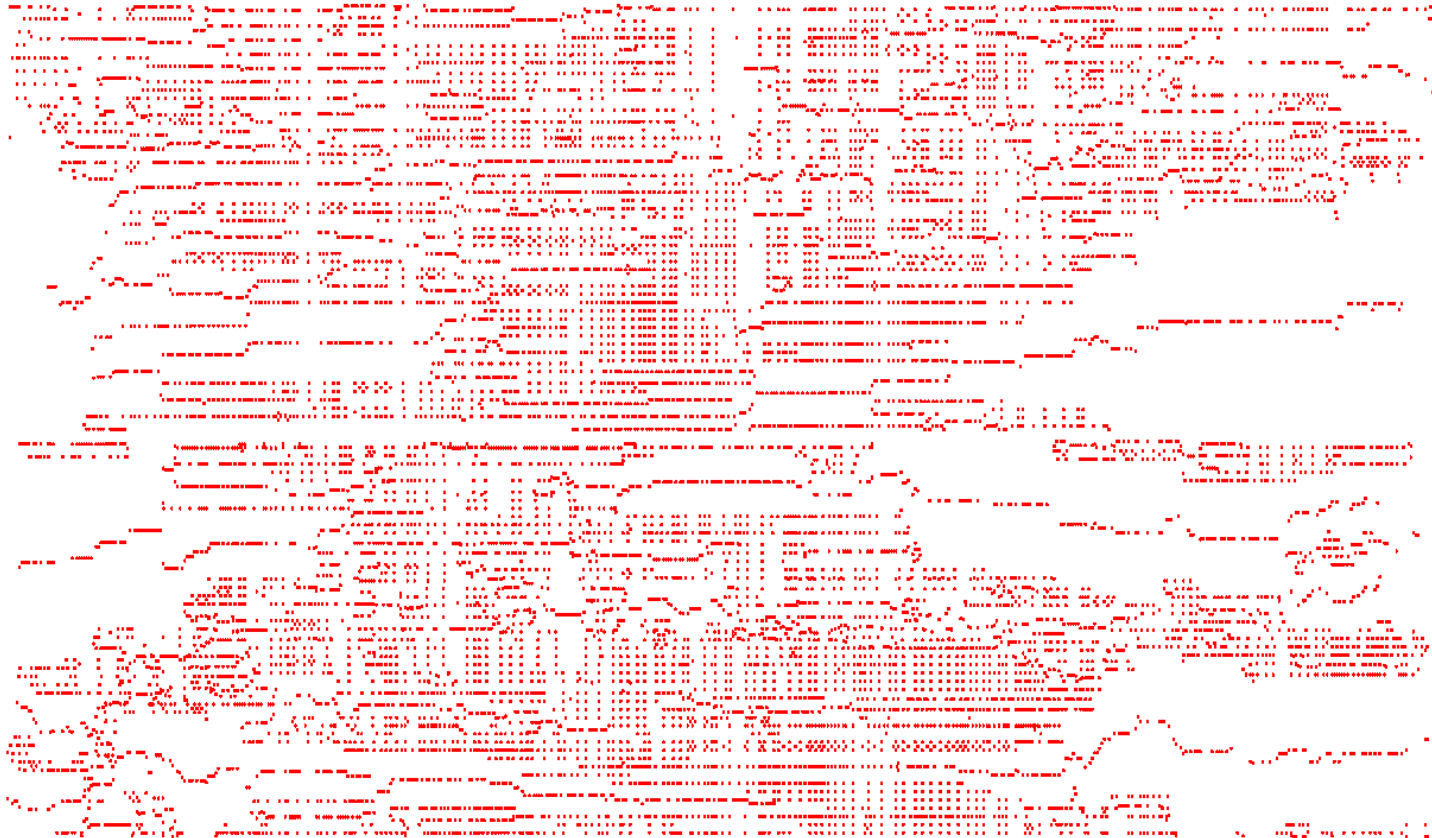
**TSP.** Given a set of  $n$  cities and a pairwise distance function  $d(u, v)$ , is there a tour of length  $\leq D$ ?



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

# Traveling Salesperson Problem

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

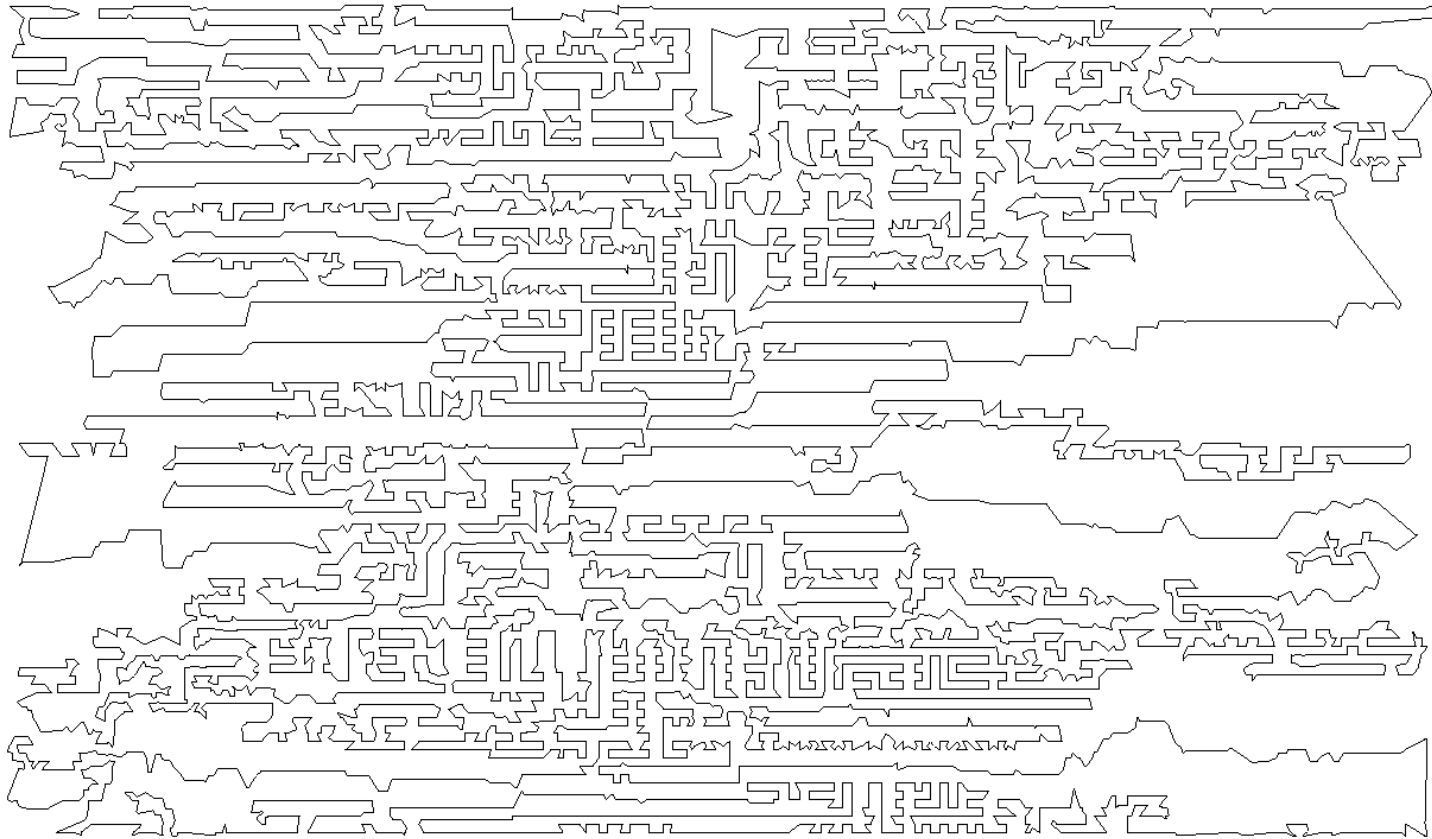


11,849 holes to drill in a programmed logic array  
Reference: <http://www.tsp.gatech.edu>



# Traveling Salesperson Problem

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



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

# Traveling Salesperson Problem

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

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

**Claim.**  $\text{HAM-CYCLE} \leq_p \text{TSP}$ .

**Pf.**

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

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

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

**Remark.** TSP instance in reduction satisfies  $\Delta$ -inequality.

## 4.6 Partitioning Problems

---

Basic genres.

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

## 3-Dimensional Matching

**3D-MATCHING.** Given  $n$  instructors,  $n$  courses, and  $n$  times, and a list of the possible courses and times each instructor is willing to teach, is it possible to make an assignment so that all courses are taught at different times?

Instructor	Course	Time
Wayne	COS 423	MW 11-12:20
Wayne	COS 423	TTh 11-12:20
Wayne	COS 226	TTh 11-12:20
Wayne	COS 126	TTh 11-12:20
Tardos	COS 523	TTh 3-4:20
Tardos	COS 423	TTh 11-12:20
Tardos	COS 423	TTh 3-4:20
Kleinberg	COS 226	TTh 3-4:20
Kleinberg	COS 226	MW 11-12:20
Kleinberg	COS 423	MW 11-12:20

## 3-Dimensional Matching

**3D-MATCHING.** Given disjoint sets  $X$ ,  $Y$ , and  $Z$ , each of size  $n$  and a set  $T \subseteq X \times Y \times Z$  of triples, does there exist a set of  $n$  triples in  $T$  such that each element of  $X \cup Y \cup Z$  is in exactly one of these triples?

**Claim.**  $3\text{-SAT} \leq_p 3\text{D-MATCHING}$ .

**Pf.** Given an instance  $\Phi$  of 3-SAT, we construct an instance of 3D-matching that has a perfect matching iff  $\Phi$  is satisfiable.

# 3-Dimensional Matching

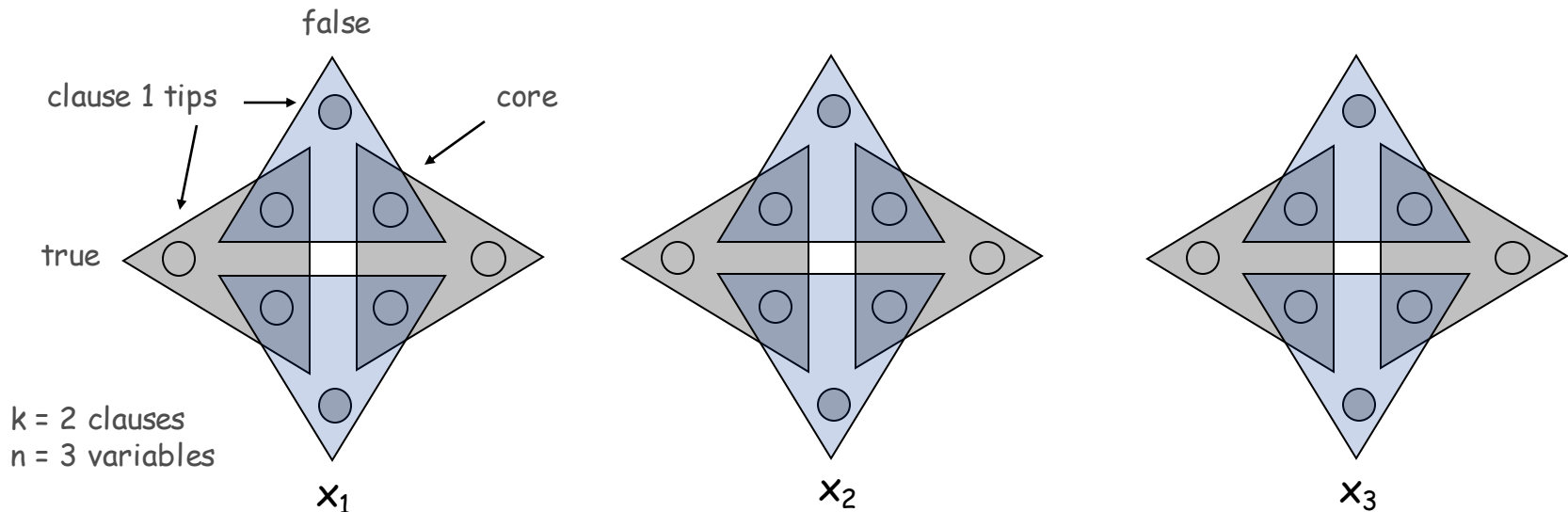
## Construction. (part 1)

- Create gadget for each variable  $x_i$  with  $2k$  core and tip elements.
- No other triples will use core elements.
- In gadget  $i$ , 3D-matching must use either both grey triples or both blue ones.

number of clauses

set  $x_i = \text{true}$

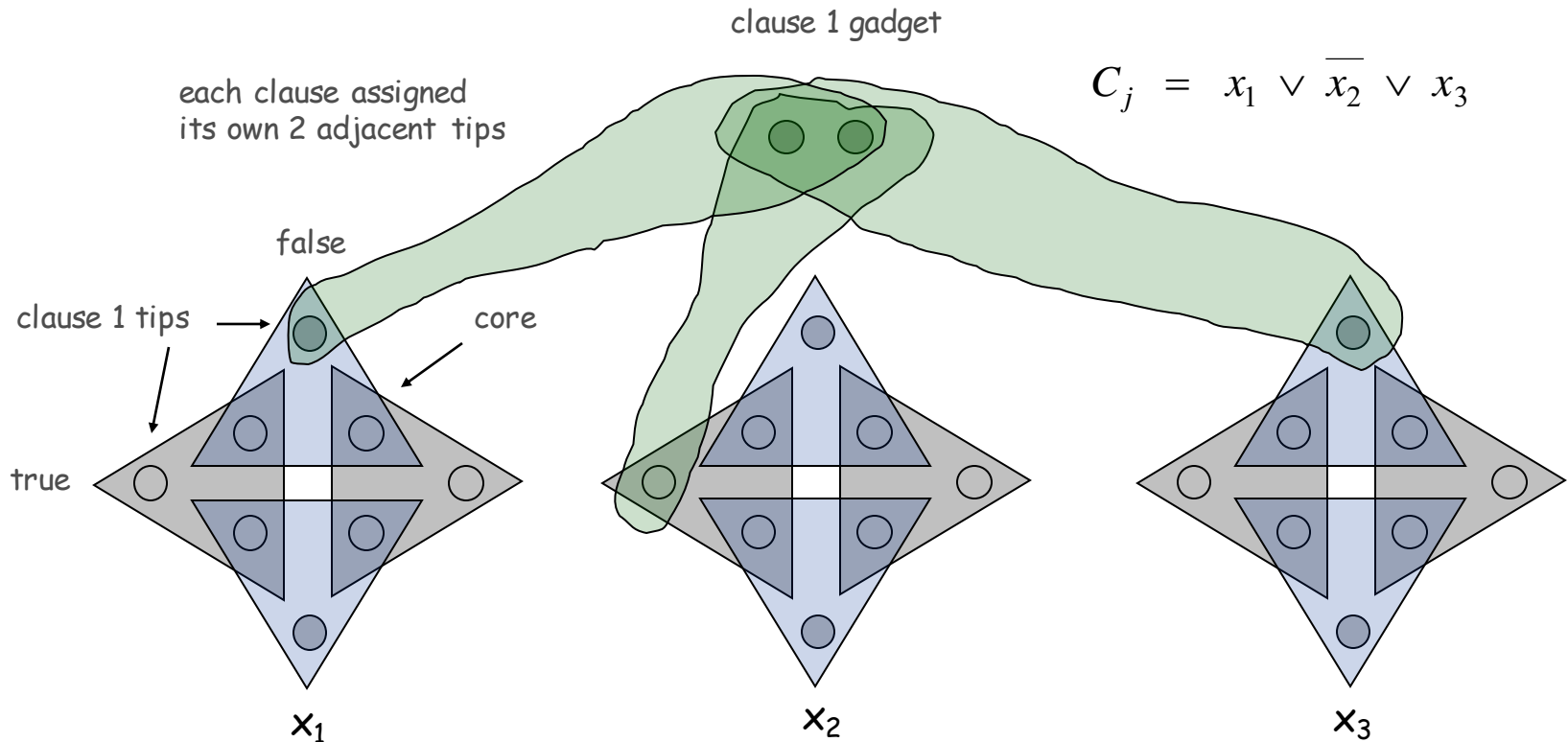
set  $x_i = \text{false}$



# 3-Dimensional Matching

## Construction. (part 2)

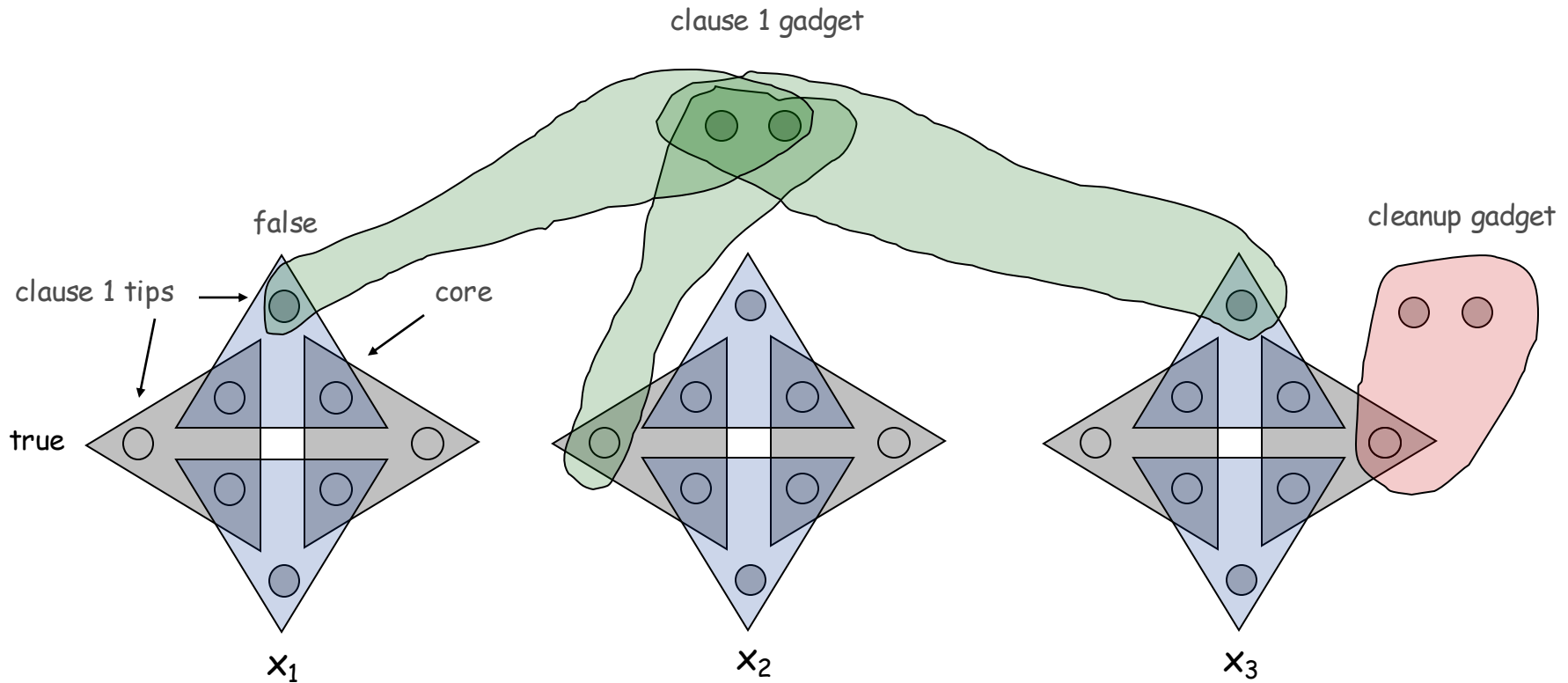
- For each clause  $C_j$  create two elements and three triples.
- Exactly one of these triples will be used in any 3D-matching.
- Ensures any 3D-matching uses either (i) grey core of  $x_1$  or (ii) blue core of  $x_2$  or (iii) grey core of  $x_3$ .



# 3-Dimensional Matching

## Construction. (part 3)

- For each tip, add a cleanup gadget.

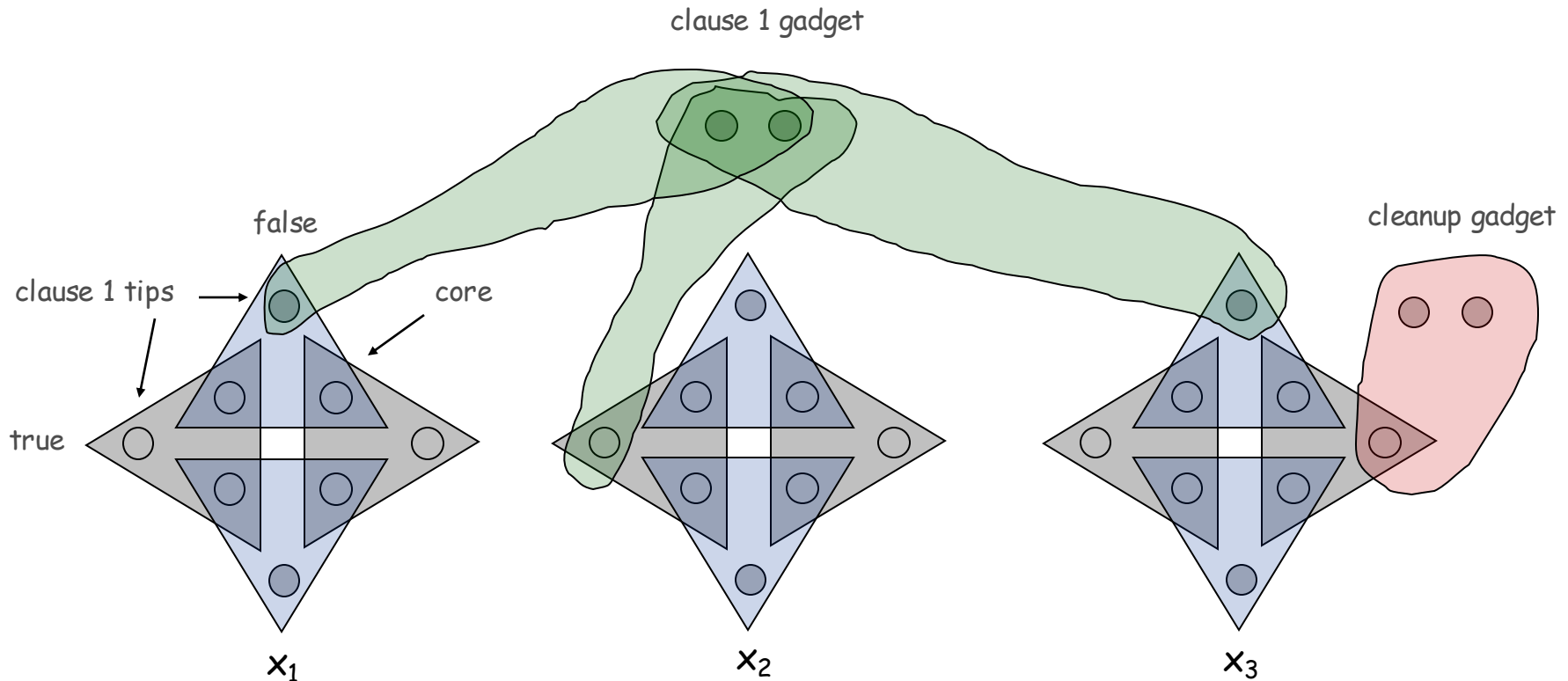




# 3-Dimensional Matching

**Claim.** Instance has a 3D-matching iff  $\Phi$  is satisfiable.

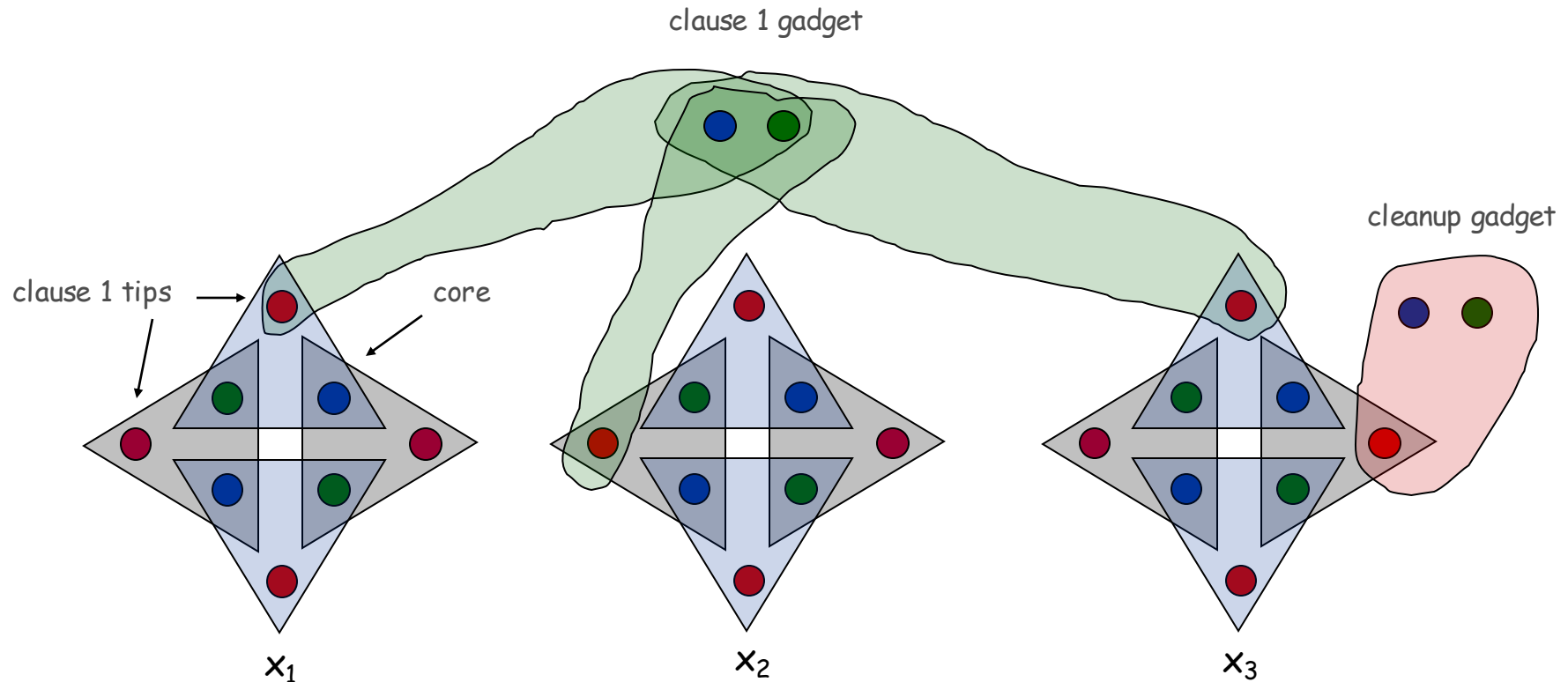
**Detail.** What are  $X$ ,  $Y$ , and  $Z$ ? Does each triple contain one element from each of  $X$ ,  $Y$ ,  $Z$ ?



# 3-Dimensional Matching

**Claim.** Instance has a 3D-matching iff  $\Phi$  is satisfiable.

**Detail.** What are  $X$ ,  $Y$ , and  $Z$ ? Does each triple contain one element from each of  $X$ ,  $Y$ ,  $Z$ ?



## 4.7 Graph Coloring

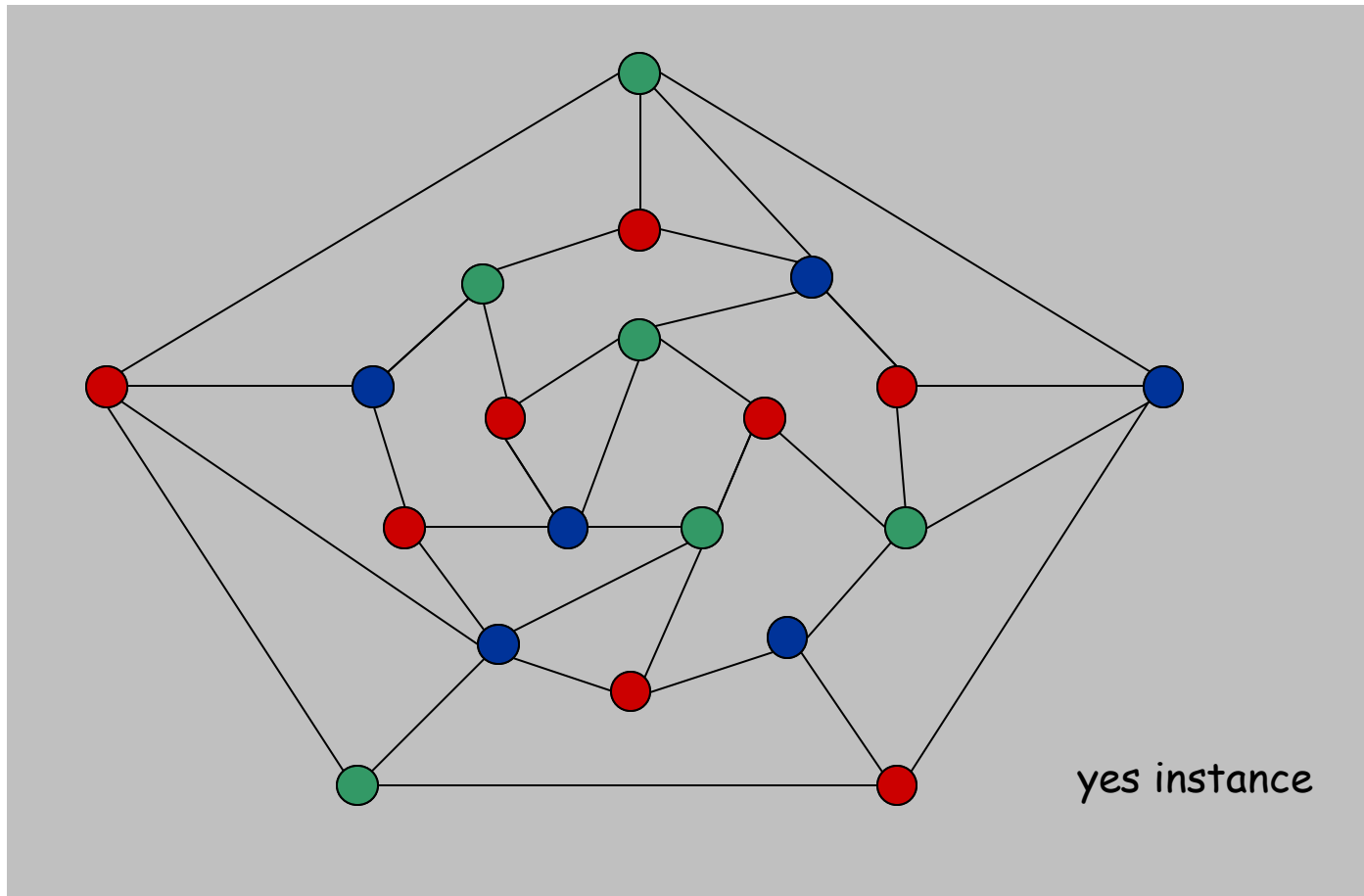
---

Basic genres.

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

# 3-Colorability

**3-COLOR:** 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  $k \geq 3$ .

# 3-Colorability

**Claim.**  $3\text{-SAT} \leq_p 3\text{-COLOR}$ .

**Pf.** Given 3-SAT instance  $\Phi$ , we construct an instance of 3-COLOR that is 3-colorable iff  $\Phi$  is satisfiable.

**Construction.**

- i. For each literal, create a node.
- ii. Create 3 new nodes T, F, B; connect them in a triangle, and connect each literal to B.
- iii. Connect each literal to its negation.
- iv. For each clause, add gadget of 6 nodes and 13 edges.

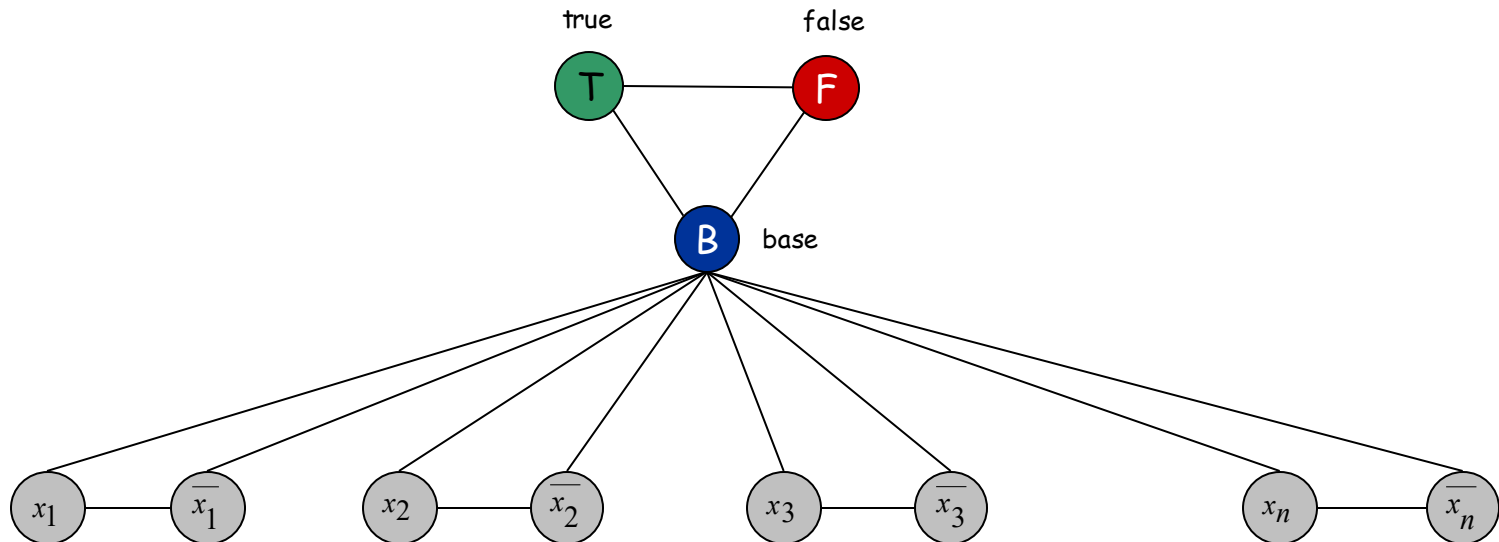
↑  
to be described next

# 3-Colorability

**Claim.** Graph is 3-colorable iff  $\Phi$  is satisfiable.

**Pf.**  $\Rightarrow$  Suppose graph is 3-colorable.

- Consider assignment that sets all T literals to true.
- (ii) ensures each literal is T or F.
- (iii) ensures a literal and its negation are opposites.

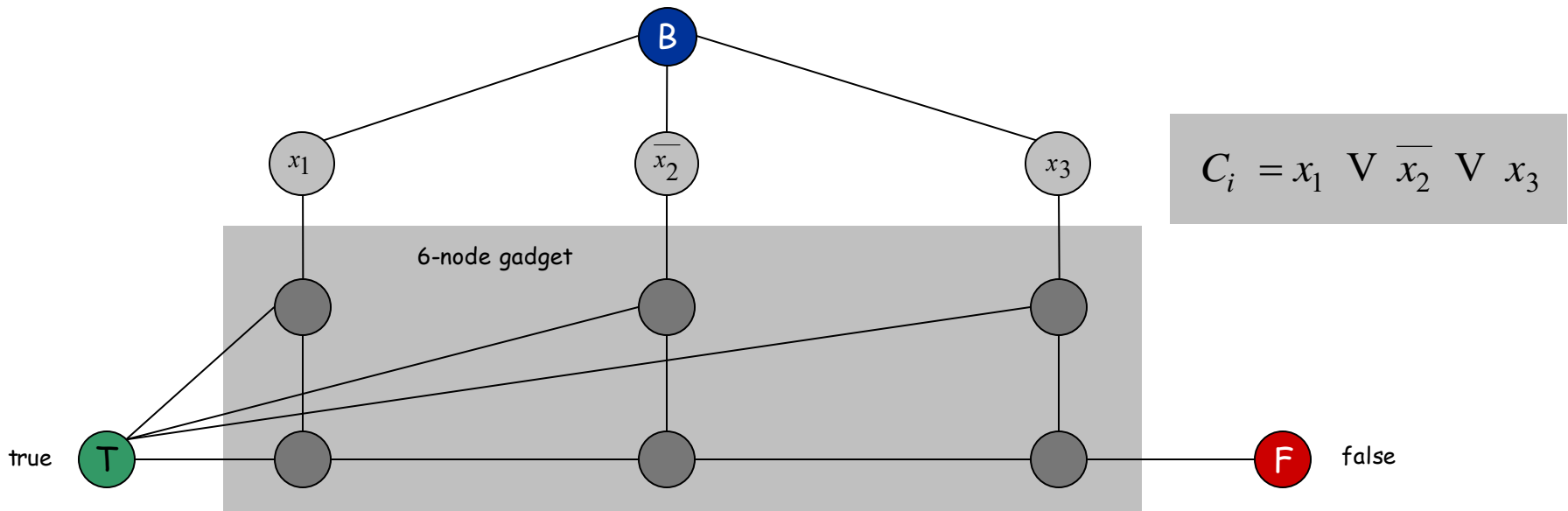


# 3-Colorability

**Claim.** Graph is 3-colorable iff  $\Phi$  is satisfiable.

**Pf.**  $\Rightarrow$  Suppose graph is 3-colorable.

- Consider assignment that sets all T literals to true.
- (ii) ensures each literal is T or F.
- (iii) ensures a literal and its negation are opposites.
- (iv) ensures at least one literal in each clause is T.



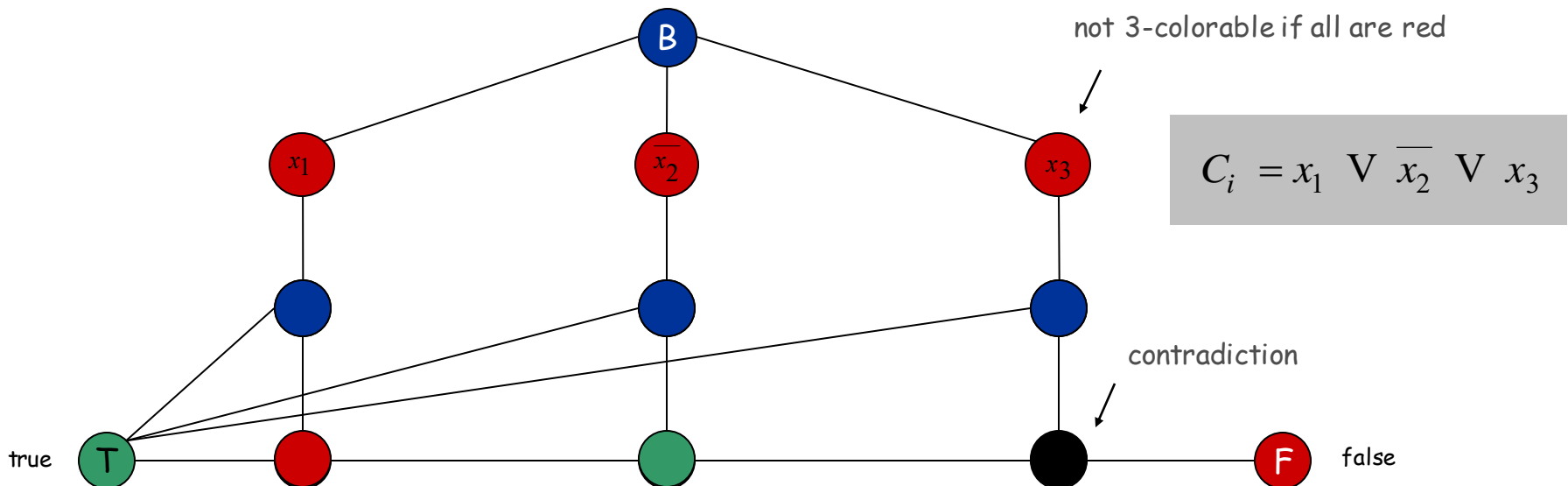


# 3-Colorability

**Claim.** Graph is 3-colorable iff  $\Phi$  is satisfiable.

**Pf.**  $\Rightarrow$  Suppose graph is 3-colorable.

- Consider assignment that sets all T literals to true.
- (ii) ensures each literal is T or F.
- (iii) ensures a literal and its negation are opposites.
- (iv) ensures at least one literal in each clause is T.

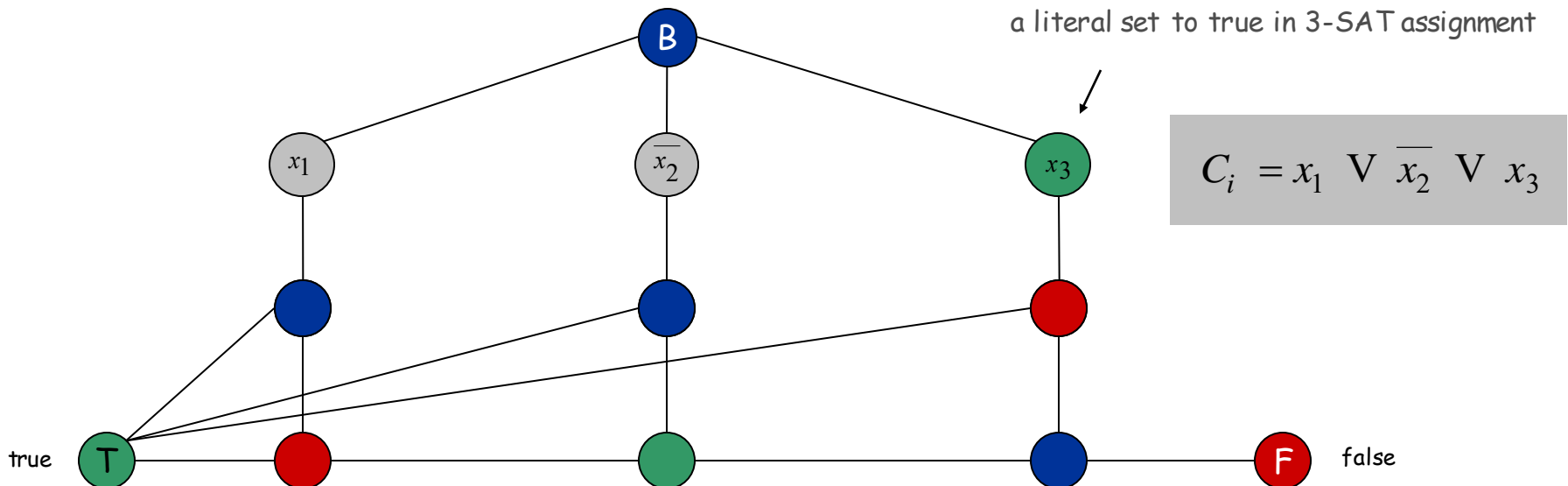


# 3-Colorability

**Claim.** Graph is 3-colorable iff  $\Phi$  is satisfiable.

**Pf.**  $\Leftarrow$  Suppose 3-SAT formula  $\Phi$  is satisfiable.

- Color all true literals T.
- Color node below green node F, and node below that B.
- Color remaining middle row nodes B.
- Color remaining bottom nodes T or F as forced. ▪



## 4.8 Numerical Problems

---

Basic genres.

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

# Subset Sum

**SUBSET-SUM.** Given natural numbers  $w_1, \dots, w_n$  and an integer  $W$ , is there a subset that adds up to exactly  $W$ ?

**Ex:**  $\{ 1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344 \}$ ,  $W = 3754$ .

**Yes.**  $1 + 16 + 64 + 256 + 1040 + 1093 + 1284 = 3754$ .

**Remark.** With arithmetic problems, input integers are encoded in binary. Polynomial reduction must be polynomial in **binary** encoding.

**Claim.**  $3\text{-SAT} \leq_p \text{SUBSET-SUM}$ .

**Pf.** Given an instance  $\Phi$  of 3-SAT, we construct an instance of SUBSET-SUM that has solution iff  $\Phi$  is satisfiable.

# Subset Sum

**Construction.** Given 3-SAT instance  $\Phi$  with  $n$  variables and  $k$  clauses, form  $2n + 2k$  decimal integers, each of  $n+k$  digits, as illustrated below.

**Claim.**  $\Phi$  is satisfiable iff there exists a subset that sums to  $W$ .

**Pf.** No carries possible.

$$C_1 = \bar{x} \vee y \vee z$$

$$C_2 = x \vee \bar{y} \vee z$$

$$C_3 = \bar{x} \vee \bar{y} \vee \bar{z}$$

dummies to get clause  
columns to sum to 4

	x	y	z	$C_1$	$C_2$	$C_3$	
x	1	0	0	0	1	0	100,010
$\neg x$	1	0	0	1	0	1	100,101
y	0	1	0	1	0	0	10,100
$\neg y$	0	1	0	0	1	1	10,011
z	0	0	1	1	1	0	1,110
$\neg z$	0	0	1	0	0	1	1,001
}	0	0	0	1	0	0	100
	0	0	0	2	0	0	200
	0	0	0	0	1	0	10
	0	0	0	0	2	0	20
	0	0	0	0	0	1	1
	0	0	0	0	0	2	2
	0	0	0	0	0	0	0
W	1	1	1	4	4	4	111,444