CS 401

NP-Complete

Xiaorui Sun

Stuff

Teaching evaluation

- Extra 1% score for all the students if overall response rate >= 80%
- Additional to the final score cut
- May improve the final grade (if overall response rate >= 80%)
- Current response rate ~ 50%

Next lecture: Final exam review

NP Completeness

NP-complete: Hardest problems in NP

The "first" NP-complete problem: CIRCUIT-SAT

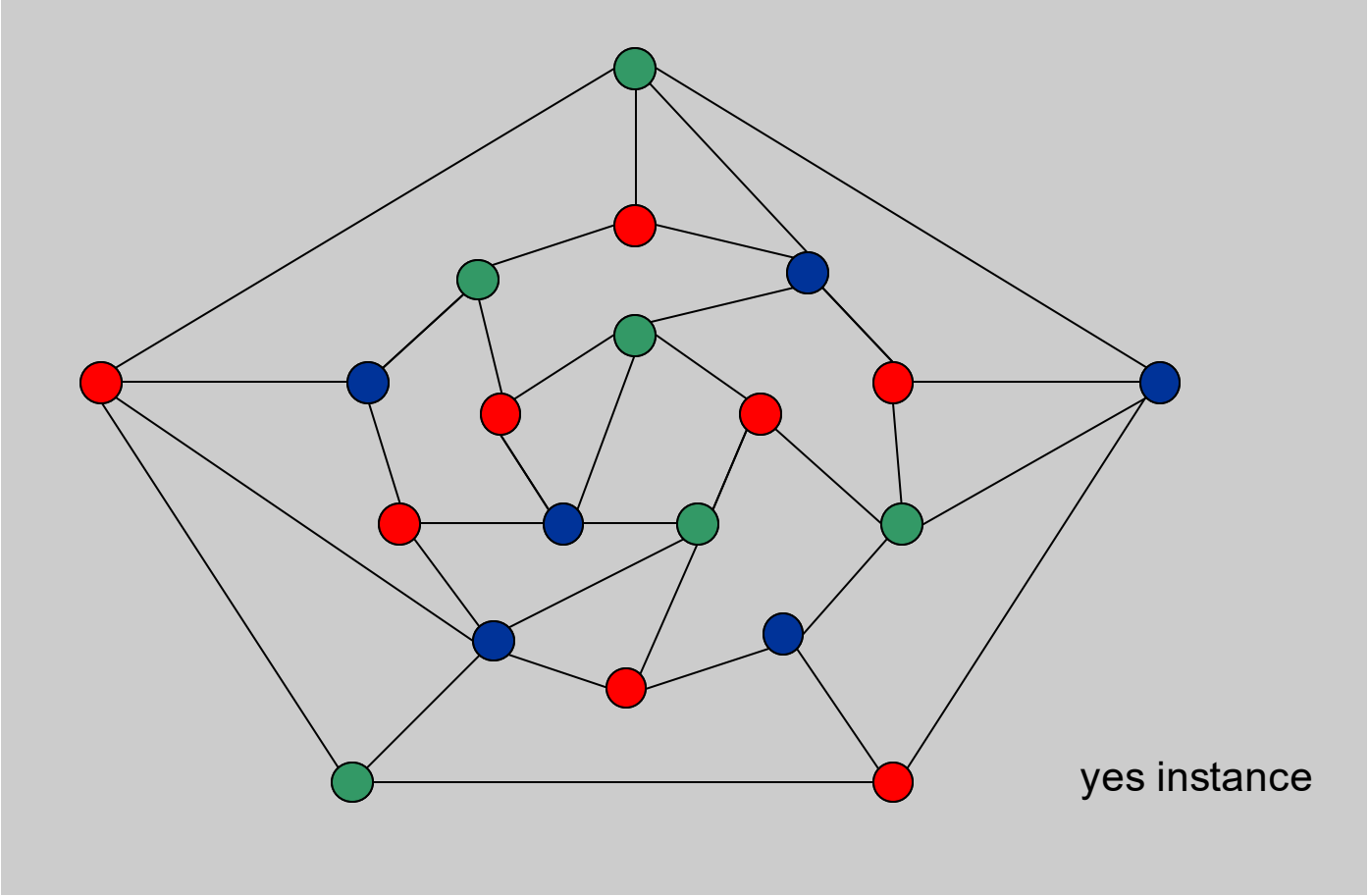
Recipe to establish NP-completeness of problem Y.

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

Example: 3-SAT is NP-complete by CIRCUIT-SAT $\leq_P$ 3-SAT

3-Colorability

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



3-Colorability

Claim. 3-SAT $\leq_P$ 3-COLOR.

Pf. Given 3-SAT instance Φ, we construct an instance of 3-COLOR that is 3-colorable iff Φ is satisfiable.

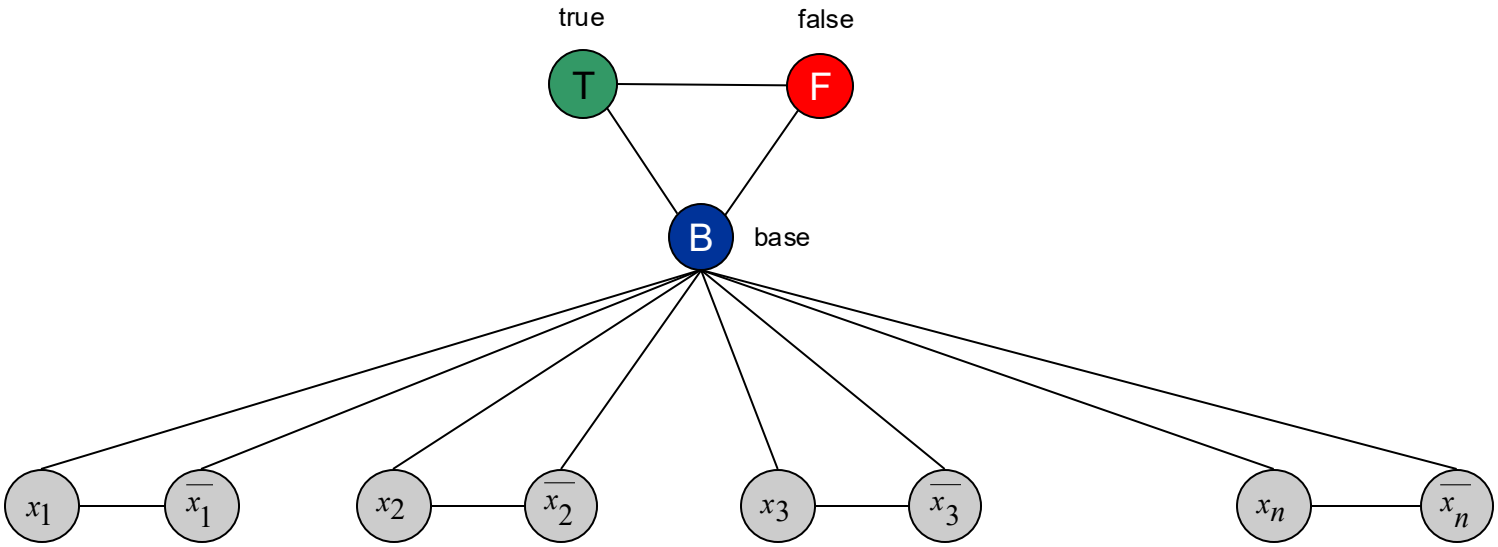
3-Colorability

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.



3-Colorability

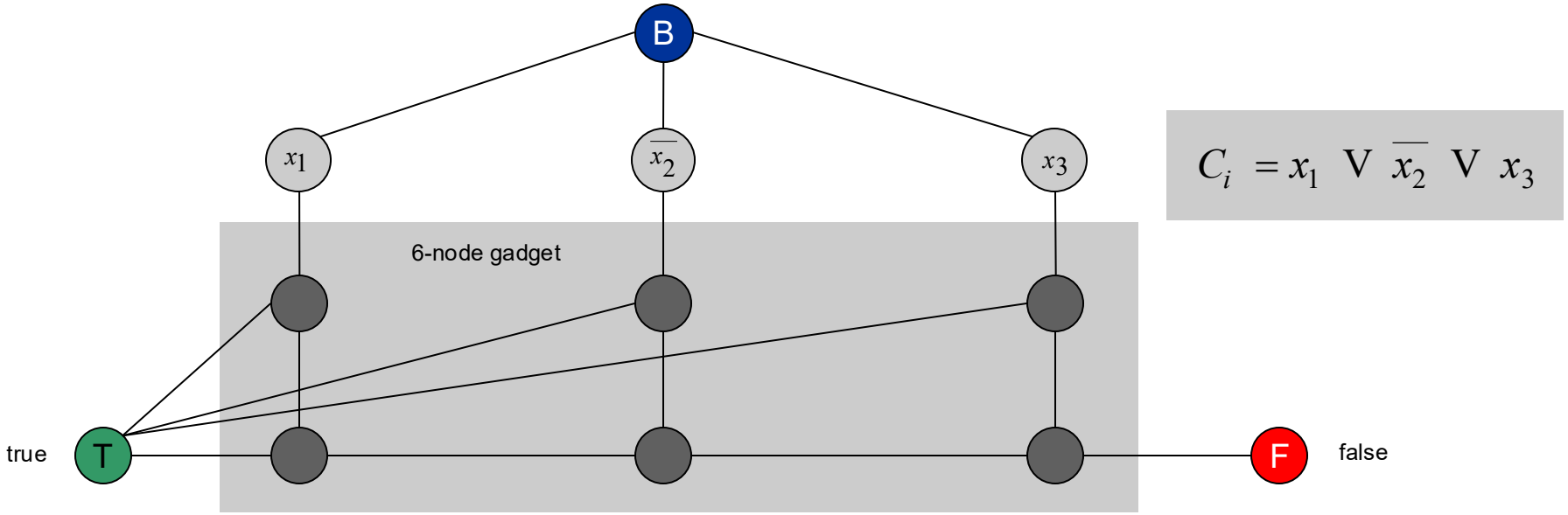
Construction.

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.

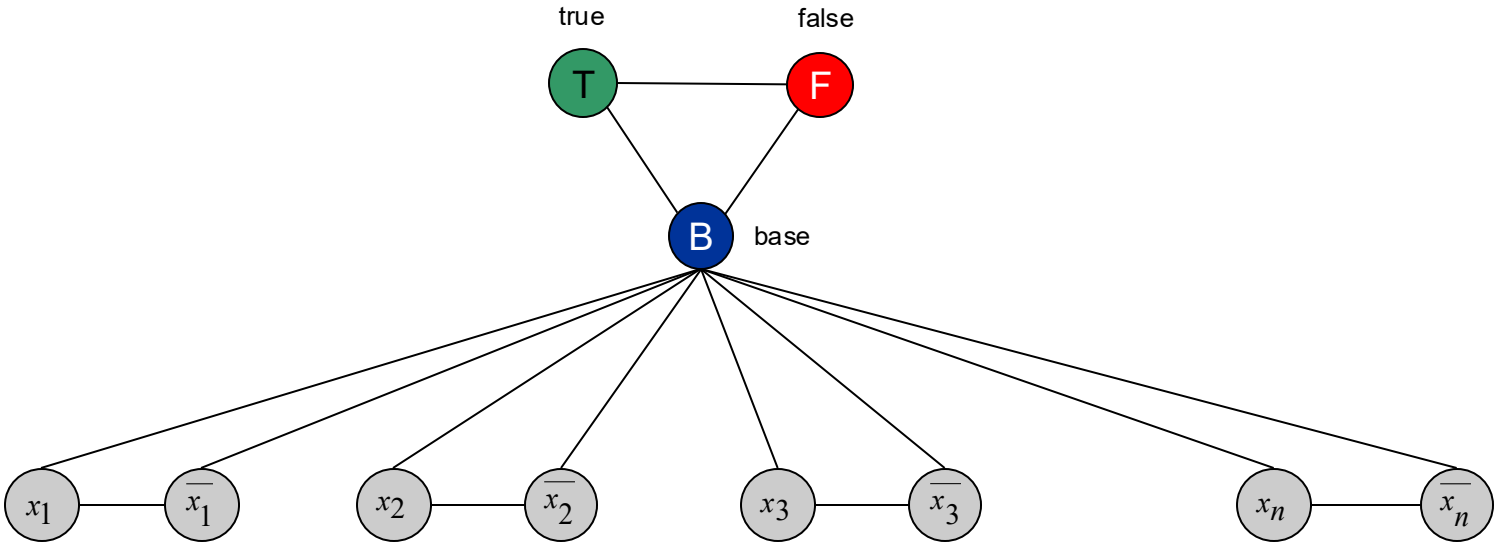


3-Colorability

Claim. Graph is 3-colorable iff Φ 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.

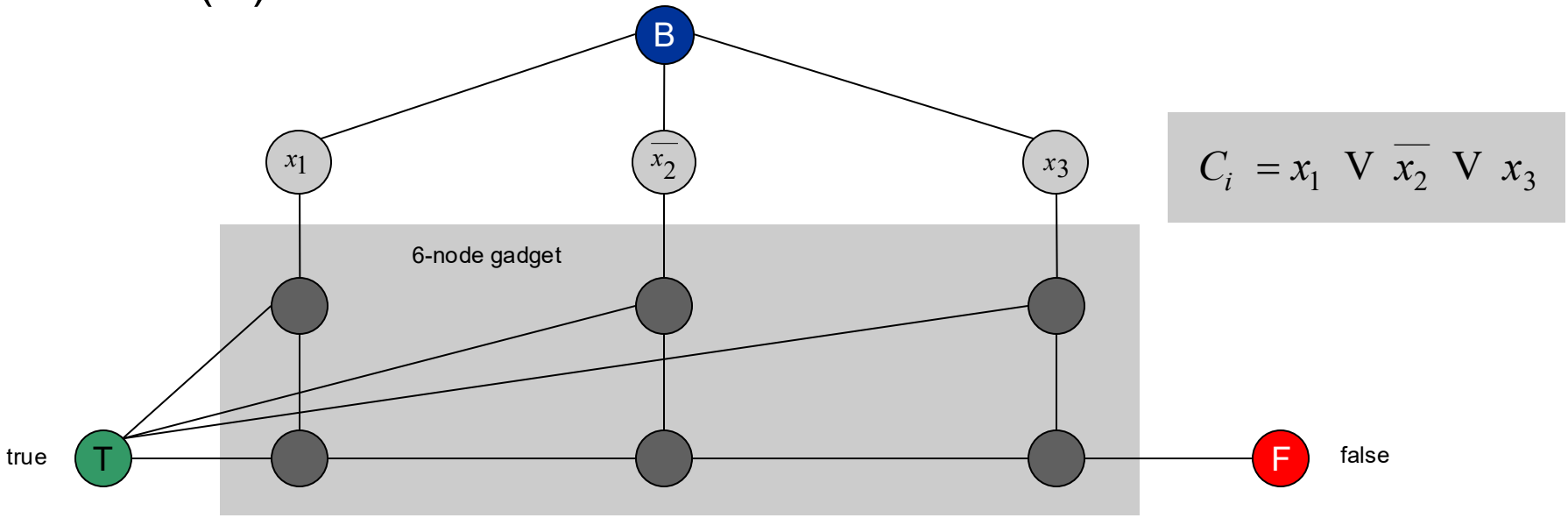


3-Colorability

Claim. Graph is 3-colorable iff Φ 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.

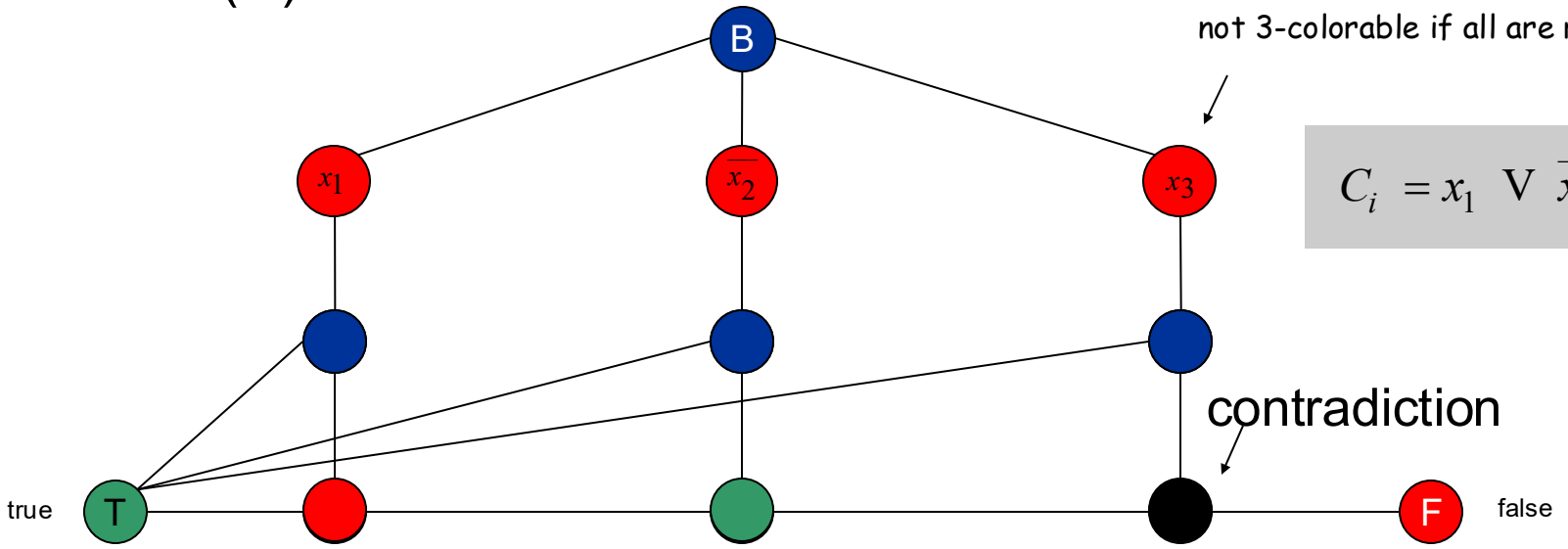


3-Colorability

Claim. Graph is 3-colorable iff Φ 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.

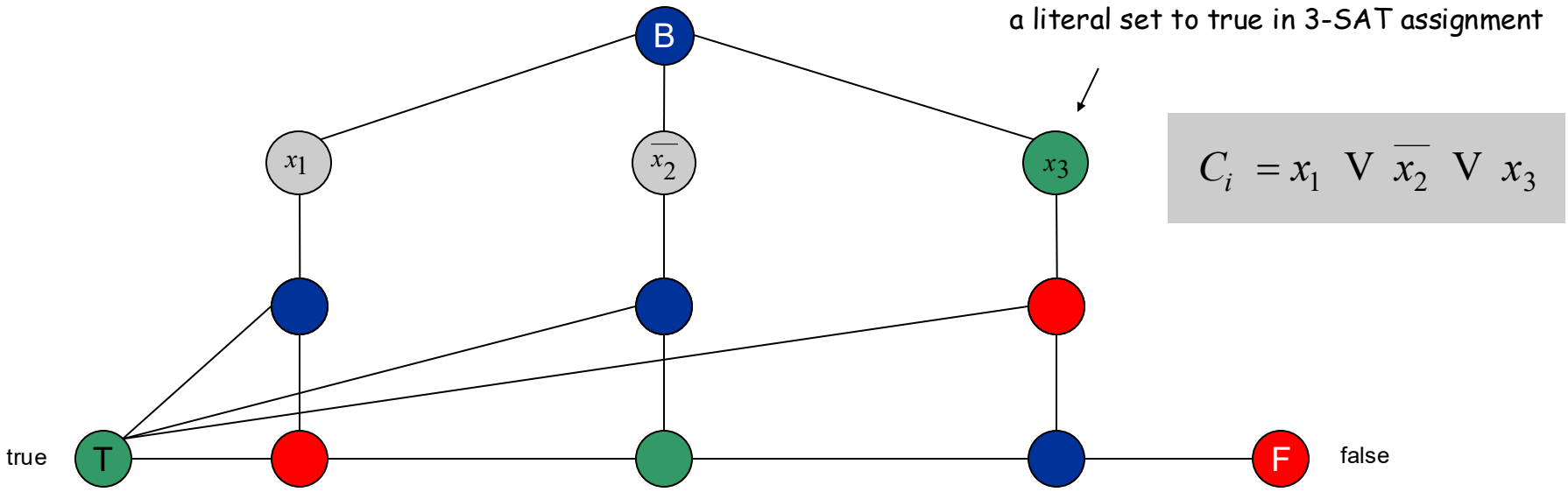


3-Colorability

Claim. Graph is 3-colorable iff Φ is satisfiable.

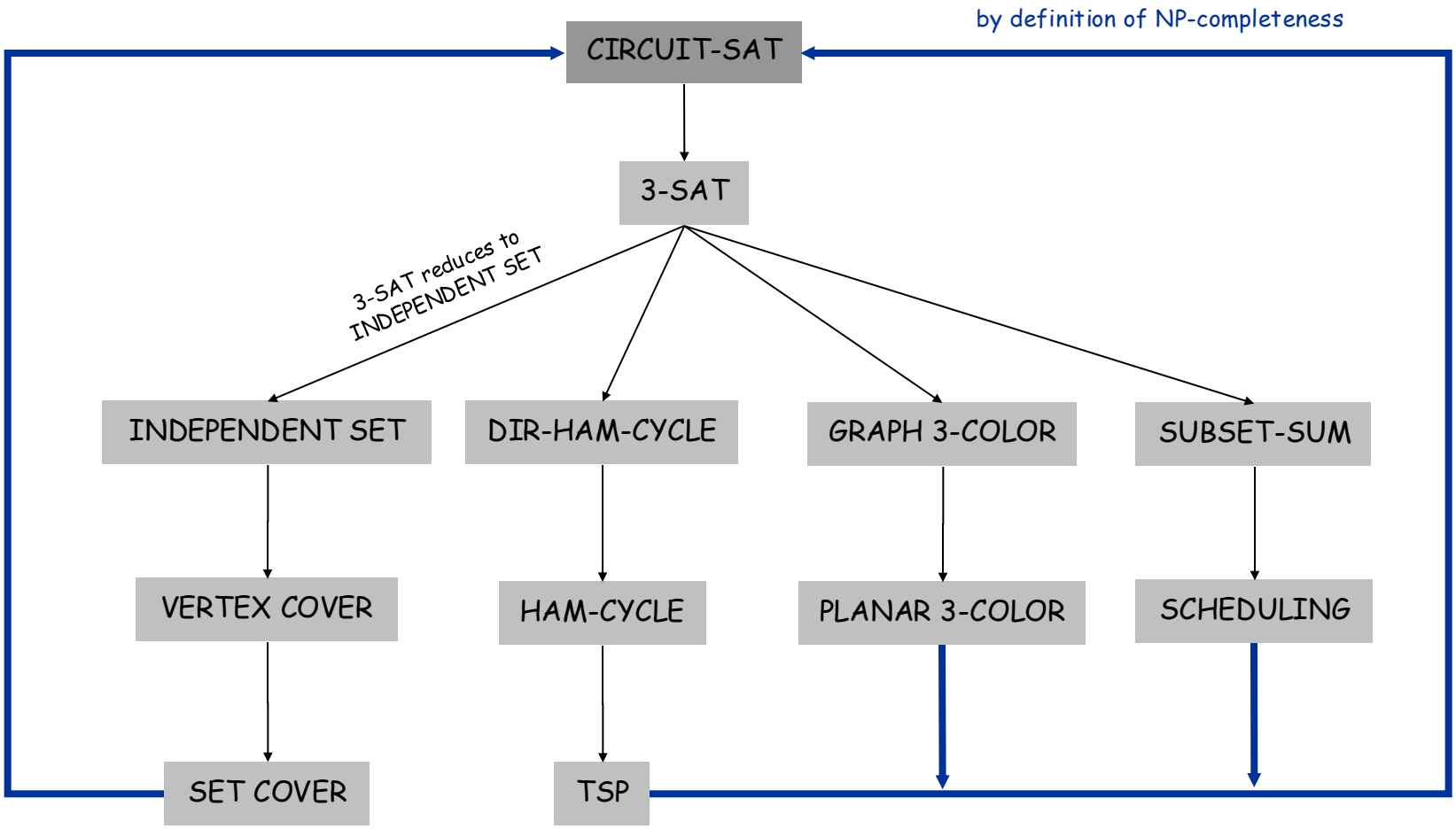
Pf. $\Leftarrow$ Suppose 3-SAT formula Φ is satisfiable.

- 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. ▪



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.

Are all the problems in NP either in P or NP-complete?

Answer: Most NP problems are either kn
complete.

NP-intermediate problems

Notable exceptions. Factoring, graph isomorphism.

Determine if a composite number c has a factor $\leq k$

NP-Hard

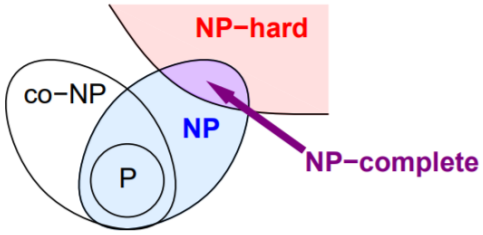
Not every computational problem is a decision problem

- E.g. compute edit distance of two strings

How do we compare hardness of general problems?

NP-hard: A problem B is NP-hard iff for any problem $A \in NP$, we have $A \leq_p B$

Remark: NP-hard problems may not necessarily belong to NP



More of what we *think* the world looks like.

Summary

We learned the crucial idea of polynomial-time reduction. This can be even used in algorithm design, e.g., we know how to solve max-flow so we reduce image segmentation to max-flow

Polynomial-time reductions are transitive relations

NP: Set of all decision problems for which there exists a poly-time certifier.

NP-complete problems are the hardest problem in NP

Advanced algorithm design techniques

Approximation algorithm

Many NP-hard problems have no efficient exact algorithm

Alternatively, people try to design efficient algorithms to solve problems approximately

For a maximization problem, we say an algorithm achieves α (for some $0 < \alpha < 1$) multiplicative approximation factor if for every input instance,

$$\mathrm{SOL} \geq \alpha \, \mathrm{OPT}$$

- SOL is the solution given by the algorithm
- OPT is the optimal solution

Knapsack is NP-complete

Def. $OPT(i, v)$ = min weight of a knapsack for which we can obtain a solution of value ≥ v using a subset of items $1,\ldots, i$.

$$OPT(i,v) = \begin{cases} 0 & \text{if } v \leq 0 \\ \infty & \text{if } i = 0 \text{ and } v > 0 \\ \min\{OPT(i-1,v),\ w_i + OPT(i-1, v-v_i)\} & \text{otherwise} \end{cases}$$

Theorem. Dynamic programming algorithm computes the optimal value in $O(n^2\, v_{\max})$ time, where $v_{\max}$ is the maximum of any value.

Knapsack problem: polynomial-time approximation scheme

Intuition for approximation algorithm.

- Round all values up to lie in smaller range.
- Run dynamic programming algorithm II on rounded/scaled instance.
- Return optimal items in rounded instance.

item	value	weight
1	934221	1
2	5956342	2
3	17810013	5
4	21217800	6
5	27343199	7

original instance (W = 11)

item	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

rounded instance (W = 11)

Knapsack problem: polynomial-time approximation scheme

Round up all values:

- $0 < \varepsilon \leq 1$ = precision parameter.
- $v_{\max}$ = largest value in original instance.
- θ = scaling factor = $\varepsilon\, v_{\max} / 2n$.

$$\bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta, \quad \hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil$$

Observation. Optimal solutions to problem with $\bar{v}$ are equivalent to optimal solutions to problem with $\hat{v}$.

Intuition. $\bar{v}$ close to v so optimal solution using $\bar{v}$ is nearly optimal; $\hat{v}$ small and integral so dynamic programming algorithm II is fast.

Theorem. For any $\varepsilon > 0$, the rounding algorithm computes a feasible solution whose value is within a $(1 - \varepsilon)$ factor of the optimum in $O(n^3 / \varepsilon)$ time.

Randomization

in practice, access to a pseudo-random number generator

Randomization. Allow fair coin flip in unit time.

Why randomize? Can lead to simplest, fastest, or only known algorithm for a particular problem.

Ex. Symmetry-breaking protocols, graph algorithms, quicksort, hashing, load balancing, closest pair, Monte Carlo integration, cryptography, ...

Maximum 3-satisfiability

exactly 3 literals per clause and each literal corresponds to a different variable

Maximum 3-satisfiability. Given a 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.

$$
\begin{aligned}
C_1 &= x_2 \vee \overline{x_3} \vee \overline{x_4} \\
C_2 &= x_2 \vee x_3 \vee \overline{x_4} \\
C_3 &= \overline{x_1} \vee x_2 \vee x_4 \\
C_4 &= \overline{x_1} \vee \overline{x_2} \vee x_3 \\
C_5 &= x_1 \vee \overline{x_2} \vee \overline{x_4}
\end{aligned}
$$

Remark. **NP**-hard optimization problem.

Simple idea. Flip a coin, and set each variable true with probability ½, independently for each variable.

Maximum 3-satisfiability: analysis

Claim. Given a 3-SAT formula with k clauses, the expected number of clauses satisfied by a random assignment is $7k / 8$.

Pf. Consider random variable $Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise.} \end{cases}$

- Let Z = number of clauses satisfied by random assignment.

$$E[Z] = \sum_{j=1}^{k} E[Z_j]$$

linearity of expectation

$$= \sum_{j=1}^{k} \Pr[\text{clause } C_j \text{ is satisfied}]$$

$$= \frac{7}{8} k$$

The probabilistic method

Corollary. For any instance of 3-SAT, there exists a truth assignment that satisfies at least a $7/8$ fraction of all clauses.

Q. Can we turn this idea into a $7/8$-approximation algorithm?

A. Yes (but a random variable can almost always be below its mean).

Lemma. The probability that a random assignment satisfies $\geq 7k / 8$ clauses is at least $1 / (8k)$.

Johnson's algorithm. Repeatedly generate random truth assignments until one of them satisfies $\geq 7k / 8$ clauses.

Theorem. Johnson's algorithm is a $7/8$-approximation algorithm.

What is next?

CS 402: Algorithms in Practice

- Hashing, sketching, local algorithms, approximate and randomized algorithms, linear programming based algorithms

CS 501: Computer Algorithms II (Graduate Algorithms Course)

- Instructor: Gyorgy Turan
- Advanced randomized and approximation algorithms

CS 505: Computability and Complexity Theory

- How to prove lower bounds on algorithms?

CS 506: An Introduction to Quantum Computing

- How to design algorithms for quantum computers?