

CPSC 365 / ECON 365:
Algorithms

Lecture 23: Complexity 5

Andre Wibisono

Yale University

April 19, 2022

Last time

- P vs NP
- SAT is NP-Complete [Cook-Levin]
- 2-SAT in P vs.
3-SAT is NP-Complete

Today: 3-COLORING is NP complete

Notes:

- PS 6 due today
- PS 7 out, due Tue Apr 26
- Midterm 2: Thu Apr 28, in class, 2:30 - 3:45 pm
 - Can bring 1 page (double-sided) of **written** notes

Plan

Review: SAT

Graph Coloring

Satisfiability

Important results:

- SAT is NP-Complete [Cook-Levin]
- $\text{SAT} \leq_P \text{3-SAT}$
 \Rightarrow 3-SAT is NP-Complete
- 2-SAT is in P

P vs NP

Recall problem classes:

- P = set of problems we can solve efficiently (polynomial time)
 - There is a poly-time algorithm $A(s) = \text{Yes}$ iff $s \in X$
- NP = set of problems we can check efficiently
 - There is a poly-time certifier B such that:
 - if $s \in X$, then $\exists t$ poly size such that $B(s, t) = \text{Yes}$;
 - if $s \notin X$, then $\forall t, B(s, t) = \text{False}$.
- $P \subseteq NP$

NP-Complete

A problem X is **NP-Complete** if:

- X is in NP; and
- X is **NP-hard**:

$$\forall Y \in \text{NP}, Y \leq_P X$$

- **NP-complete** problems are the *hardest* problems in NP.
- Recall **polynomial-time reduction**:

$$Y \leq_P X$$

means we can solve Y using polynomial-time computation and polynomial number of calls to an algorithm solving X .

NP-Complete

A problem X is **NP-Complete** if:

- X is in NP; and
- X is **NP-hard**:

$$\forall Y \in \text{NP}, Y \leq_P X$$

Lemma

If X is **NP-complete** and $X \in P$, then $P = \text{NP}$.

Lemma

If X is **NP-complete** and $X \leq_P Z$ for some $Z \in \text{NP}$, then Z is **NP-complete**.

SAT is NP-Complete

Theorem (Cook-Levin)

SAT is NP-complete.

Proof sketch: SAT is in NP. Want to show $\forall X \in \text{NP}, X \leq_P \text{SAT}$.

- Given $X \in \text{NP}$ and input s , will construct a Boolean formula f such that: $s \in X$ iff f is satisfiable.
- Since $X \in \text{NP}$, there is an efficient certifier B such that $s \in X$ iff $B(s, t) = \text{Yes}$ for some poly-size t .
- Can build a poly-size *circuit* K such that: $B(s, t) = \text{Yes}$ iff K is satisfiable (can set input to make output True).
- Can translate circuit into satisfiability of Boolean formula f .

□

Proof reference:

- [Kleinberg & Tardos, Theorem 8.13]: CIRCUIT-SAT is NP-complete
- [Sipser, "Introduction to the Theory of Computation", Theorem 7.37]

Satisfiability

- SAT is NP-Complete [Cook-Levin]
- $\text{SAT} \leq_P \text{3-SAT}$
 \Rightarrow 3-SAT is NP-Complete
- 2-SAT is in P

SAT to 3-SAT

Theorem

$SAT \leq_P 3\text{-SAT}$

Proof:

We are given a SAT instance, i.e. a CNF formula

$$f = C_1 \wedge \cdots \wedge C_m$$

where each clause $C \in \{C_1, \dots, C_m\}$ is of the form

$$C = t_1 \vee \cdots \vee t_k$$

where each term t_i is either a variable x_j or its negation \bar{x}_j .

Will construct 3-CNF f' st: f' is satisfiable iff f is satisfiable.

SAT to 3-SAT

It suffices to show how to convert each clause $C = t_1 \vee \cdots \vee t_k$ into a 3-SAT formula that preserves satisfiability.

1. If $k = 1$: $C = t_1$
 - (If we're allowed to repeat terms, then we can do $t_1 \vee t_1 \vee t_1$)
 - If each term in a clause must be distinct: Introduce new variables a, b , and consider:

$$C' = (t_1 \vee a \vee b) \wedge (t_1 \vee \bar{a} \vee b) \wedge (t_1 \vee a \vee \bar{b}) \wedge (t_1 \vee \bar{a} \vee \bar{b})$$

Observe: C' is a 3-CNF, and for any values of a, b :

$$C' \text{ is True} \quad \text{iff} \quad t_1 \text{ is True}$$

SAT to 3-SAT

2. If $k = 2$: $C = t_1 \vee t_2$

- Introduce a new variable a , and consider:

$$C' = (t_1 \vee t_2 \vee a) \wedge (t_1 \vee t_2 \vee \bar{a})$$

C' is a 3-CNF, and C' satisfiable if and only if C is satisfiable.

- Note: Logically, $(x \vee a) \wedge (x \vee \bar{a}) \equiv x$

3. If $k = 3$: $C = t_1 \vee t_2 \vee t_3$

- Already a 3-CNF, do nothing.

SAT to 3-SAT

4. If $k \geq 4$: $C = t_1 \vee t_2 \vee t_3 \vee \cdots \vee t_k$

- Introduce new variables u_1, \dots, u_{k-3}
- Replace C by a 3-CNF formula with $k - 2$ clauses:

$$C' = (t_1 \vee t_2 \vee u_1) \wedge (\bar{u}_1 \vee t_3 \vee u_2) \wedge \cdots \wedge (\bar{u}_{k-3} \vee t_{k-1} \vee t_k)$$

- Claim: C' is satisfiable if and only if C is satisfiable.
 - \Rightarrow Suppose C is False, which means each t_i is False. We claim there is no way to make C' True. This is because if we want C' True, then $u_1 = \text{True}$, which implies $u_2 = \text{True}$, \dots , which implies $u_{k-3} = \text{True}$, but then the last clause is False.
 - \Leftarrow Suppose C is True, which means at least one t_i is True. We can satisfy C' as follows: Set $u_1 = \cdots = u_{i-2} = \text{True}$, and set $u_{i-1} = \cdots = u_{k-3} = \text{False}$. (The first $i - 2$ clauses have $u_j = \text{True}$; the $(i - 1)$ -st clause has t_i ; the rest has $\bar{u}_j = \text{True}$.)

Final 3-CNF C' has $\leq mn$ clauses over $\leq 2n$ variables.



Satisfiability

- SAT is NP-Complete [Cook-Levin]
- $\text{SAT} \leq_P \text{3-SAT}$
 - \Rightarrow 3-SAT is NP-Complete
 - Recall:
 $\text{3-SAT} \leq_P \text{Independent-Set} \leq_P \text{Vertex-Cover} \leq_P \text{Set-Cover}$
 - \Rightarrow Ind-Set, Vertex-Cover, Set-Cover, ... are NP-Complete
- 2-SAT is in P

2-SAT

Theorem

2-SAT is in P.

Proof: We describe an algorithm to solve any instance of 2-SAT.

- The algorithm runs in polynomial time (in number of clauses and variables), and returns a satisfying assignment if one exists, or returns None otherwise.

Input: $f = C_1 \wedge C_2 \wedge \dots \wedge C_m$ where each $C_i = t_i \vee u_i$ for some terms t_i, u_i , over variables x_1, \dots, x_n .

2-SAT

$$f = C_1 \wedge C_2 \wedge \cdots \wedge C_m \text{ where } C_m = t_i \vee u_i$$

Algorithm:

1. Create graph $G = (V, E)$ with $2n$ vertices and $2m$ edges:
 - For each variable x_i , create two vertices labeled $[x_i]$ and $[\bar{x}_i]$.
 - For each clause $C = t \vee u$, add two edges:
 - from $[\bar{t}] \rightarrow u$; and
 - from $[\bar{u}] \rightarrow t$.
 - Edge \equiv implication: If source = True, then target = True.
 - If f is satisfied, $C = t \vee u$ is True, then at least one of t, u must be True.
 - If t is False, then $u = \text{True}$ ($[\bar{t}] \rightarrow [u]$)
 - If u is False, then $t = \text{True}$ ($[\bar{u}] \rightarrow [t]$)
 - If f is satisfied, any path in G is a valid logical implication.
 - Cannot have a contradiction: $\text{True} \rightarrow \cdots \rightarrow \text{False}$
 - But note that $\text{False} \rightarrow \text{True}$ is okay

2-SAT

Algorithm:

2. For each variable x_i , check if there is a **bad path** from x_i .
 - A **bad path** is a path $[x_i] \rightarrow \dots \rightarrow [\bar{x}_i] \rightarrow \dots \rightarrow [x_i]$.
 - How to check? Explore via BFS / DFS.
 - If there is a **bad path**, then there is no satisfying assignment.
 - Because if there is one, a **bad path** will cause a contradiction.
 - Either x_i or \bar{x}_i is True, and will get $\text{True} \rightarrow \dots \rightarrow \text{False}$.
 - If there is a **bad path** from any x_i , return No.
Else, a satisfying assignment exists, and we can find greedily.

2-SAT

Algorithm:

3. Assume there is no bad path. Assign variables:

- Choose an unassigned term t st there is no path from $[t]$ to $[\bar{t}]$.
- Label term $[t] = \text{True}$, as well as *any* node reachable from $[t]$.
 - If $t = x_i$, then set $[x_i] = \text{True}$ (and set $[\bar{x}_i] = \text{False}$).
 - If $t = \bar{x}_i$, then set $[x_i] = \text{False}$ (and set $[\bar{x}_i] = \text{True}$).
 - When we set a term to be True, also set any node reachable from it to be True.
- Repeat until we finish assigning all terms.
- Claim (prove): This gives a consistent satisfying assignment.
 - e.g. we cannot reach $[t] \rightarrow \dots \rightarrow [u] \rightarrow \dots \rightarrow [\bar{u}] \rightarrow \dots$

□

Satisfiability

- SAT is NP-Complete [Cook-Levin]
- $\text{SAT} \leq_P \text{3-SAT}$
 \Rightarrow 3-SAT is NP-Complete
- 2-SAT is in P

Plan

Review: SAT

Graph Coloring

Graph Coloring

Given an undirected graph $G = (V, E)$.

A k -coloring is an assignment $c: V \rightarrow \{1, \dots, k\}$ such that for each edge $(u, v) \in E$, $c(u)$ is different from $c(v)$.

k -COLORING:

- Input: An undirected graph $G = (V, E)$ and an integer $k > 0$.
- Output: Yes if a k -coloring of G exists, else No.

How difficult?

- 2-COLORING is
- 3-COLORING is

Graph Coloring

Given an undirected graph $G = (V, E)$.

A k -coloring is an assignment $c: V \rightarrow \{1, \dots, k\}$ such that for each edge $(u, v) \in E$, $c(u)$ is different from $c(v)$.

k -COLORING:

- Input: An undirected graph $G = (V, E)$ and an integer $k > 0$.
- Output: Yes if a k -coloring of G exists, else No.

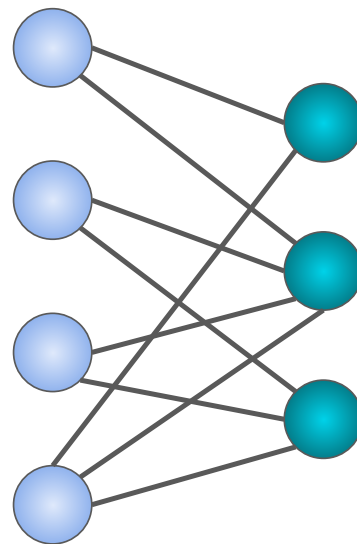
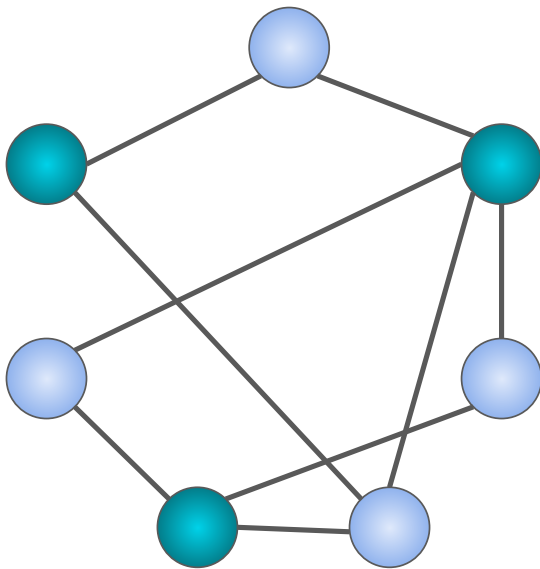
How difficult?

- 2-COLORING is in P
- 3-COLORING is NP-complete

2-Coloring

Can we check if a graph has a 2-coloring?

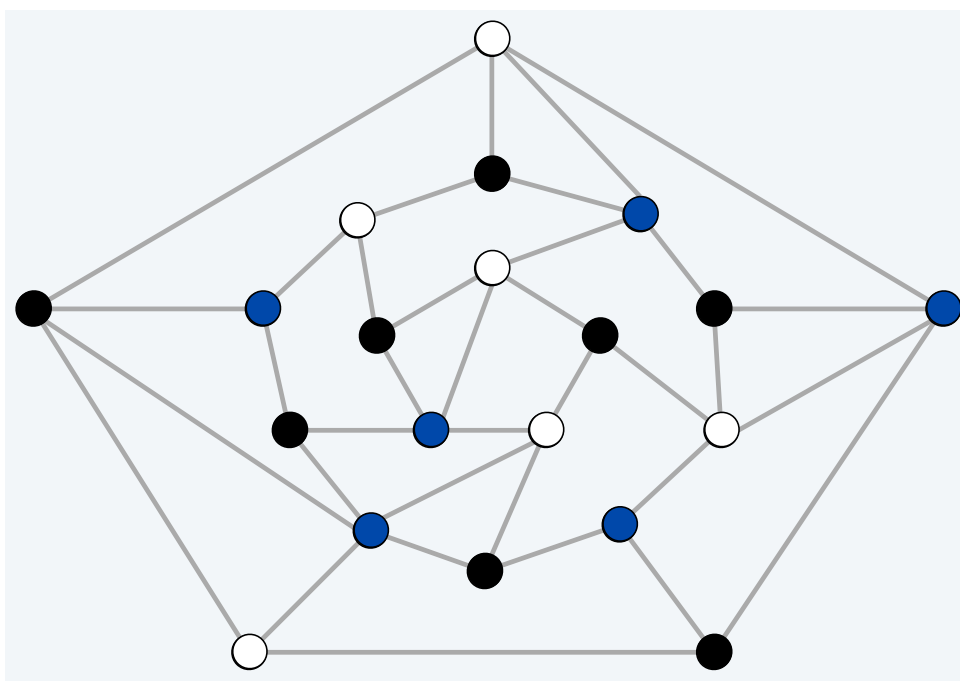
- \Leftrightarrow check if a graph is bipartite
- Can solve in linear time via BFS / DFS



3-Coloring

Can we check if a graph has a 3-coloring?

- NP-complete!



3-Coloring

Theorem

$3\text{-SAT} \leq_P 3\text{-COLORING}$

Proof: Given a 3-CNF formula f , we construct a graph $G = (V, E)$ such that f is satisfiable if and only if G is 3-colorable.

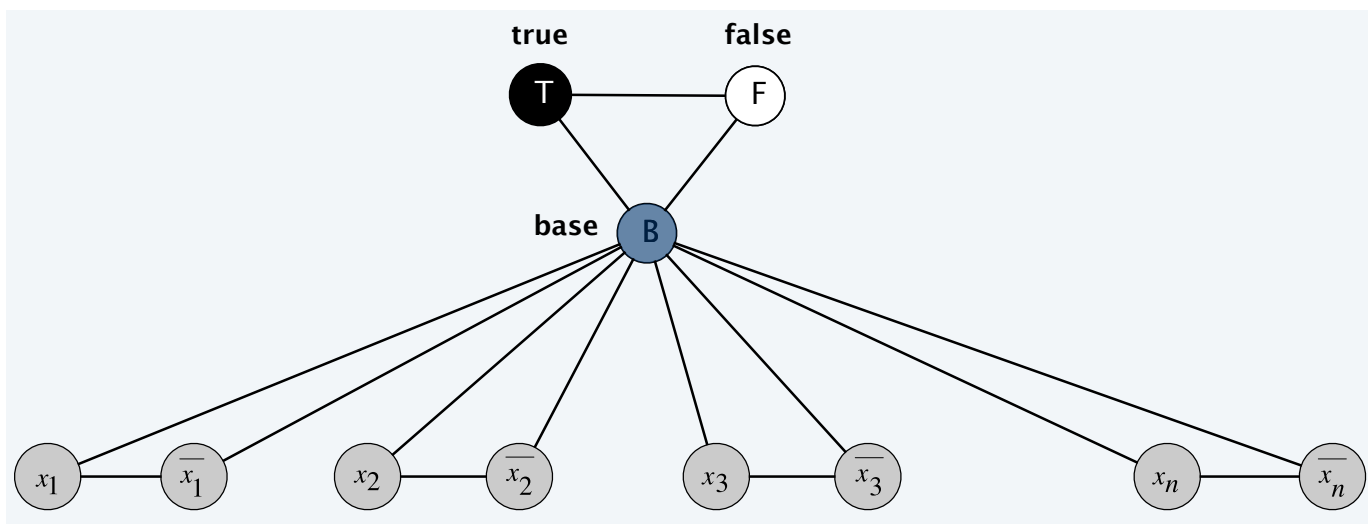
Idea:

- Define nodes and colors to encode logical terms.
- Use gadgets to encode each clause as a subgraph.

3-Coloring

Construct graph $G = (V, E)$ where:

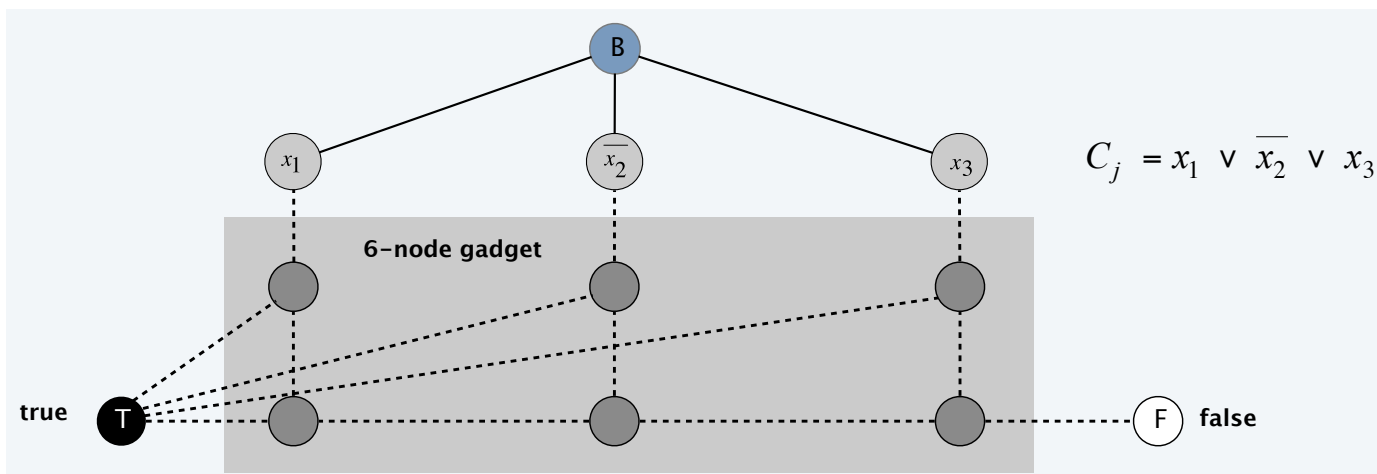
- For each variable x_i , create two nodes $[x_i]$ and $[\bar{x}_i]$.
Connect $[x_i]$ and $[\bar{x}_i]$ by an edge.
- Create 3 nodes: [T], [F], and [B]. Connect them in a triangle.
- Connect each term to [B].
⇒ In a 3-Coloring, one of x_i, \bar{x}_i is colored [T] and the other [F]
- For each clause C_j , add a *gadget* of 6 nodes



3-Coloring

Gadget to encode clause, e.g. $C = x_1 \vee \bar{x}_2 \vee x_3$:

- Add 6 nodes:
Top row connected to terms in C , to [T], and to bottom row.
Bottom row connected to top row, and chain from [T] to [F].
- Claim: Graph is 3-Colorable iff C is satisfiable



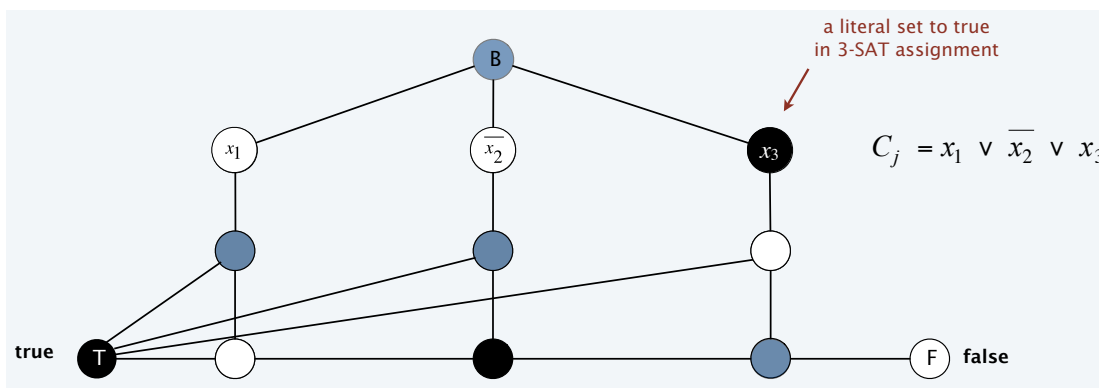
- By chaining the gadgets for each clause:
Graph is 3-Colorable iff formula f is satisfiable.

3-Coloring

Claim: Graph is 3-Colorable iff $C = x_1 \vee \bar{x}_2 \vee x_3$ is satisfiable.

Proof: \Leftarrow : Suppose C is satisfiable. In the satisfying assignment:

- Color each true term with T and each false term with F.
- Since C is true, at least one term must be T. Choose one such term:
 - Color node below that F
 - and color node below that B.
- Color the remaining nodes in the second row B.
 - Color node below that T/F as required by 3-coloring.
- Observe: This gives a 3-coloring.



3-Coloring

\Rightarrow : Conversely, suppose graph is 3-colorable.

- WLOG let [T] be colored T, [B] colored B, [F] colored F.
- Set all terms colored T to be True, and all F terms to be False.
 - Each term is colored T or F, and consistent with its negation.
- We claim this is a satisfying assignment: Each clause is True.
 - By contradiction, suppose clause is False, so all terms are F.
 - Then middle row must be all B.
 - Contradiction in bottom row: not a 3-coloring. □

