Lecture 35: NP Completeness

COSC 311 *Algorithms*, Fall 2022

Announcement

Job Candidate Talk **TOMORROW**

Lauren Biernacki, University of Michigan

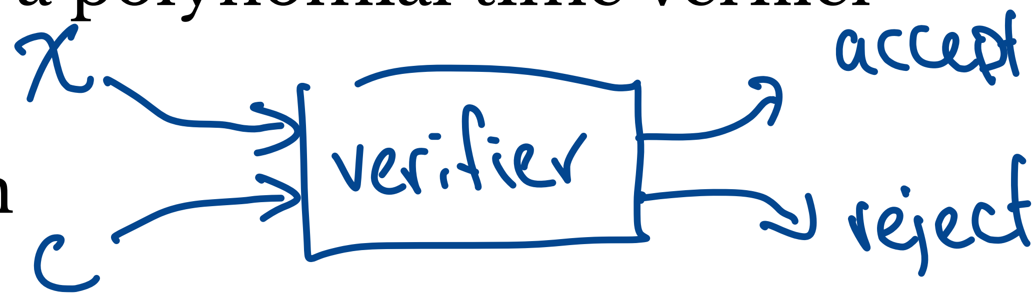
- 4:00 in SCCE A131
- Refreshments at 3:30 in SCCE C209

Last Time

Two Classes of Problems:

P: decision problems solvable in polynomial time

NP: decision problems with a polynomial time verifier



- verifier takes as input
  1. instance X of a problem
  2. a certificate C
- returns "accept"/"reject" subject to
  - *completeness* if X is "yes" instance, then some certificate is accepted
  - *soundness* if X is "no" instance, then no certificate is accepted

We Showed

Verifier ignores cert., solves problem directly

1. P ⊆ NP: *every* problem in P is in NP
2. IndpendentSet (IS) is in NP

   Input: Graph G, number k

   Output: "yes" $\iff$ G has indpendnet set of size k

   Certificate?: a set S of k vertices $v_1, v_2, \ldots, v_k$

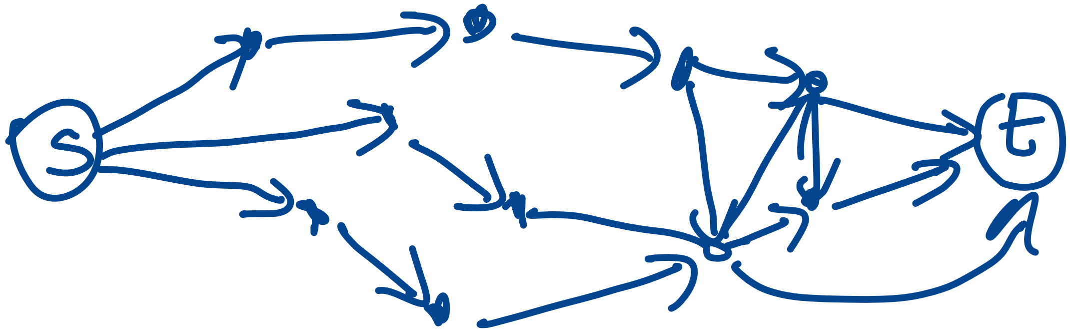
   Verification?:

   $\{v_1, \ldots, v_k\}$ is an IS

   check no edges among these vertices.

Examples

NoFlow



Input:

- directed graph $G = (V, E)$, source s, sink t, all edge capacities 1
- positive integer k

Output:

- "yes" if G does *not* admit a flow of value at least k
- "no" if G does admit a flow of value at least k

Question. Is NoFlow in NP? Yes — puzzle?

Give flow of val k ⇒ no instance

Find MaxFlow, is $< k$

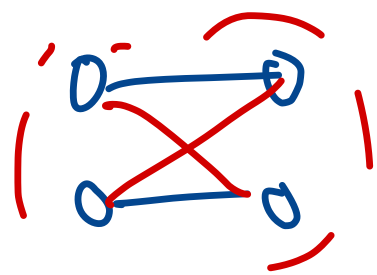
↳ Use: Ford-Fulkerson

NoFlow, Again?

What if we did not know that MaxFlow can be solved in polynomial time?

- How could we infer that NoFlow is in NP?

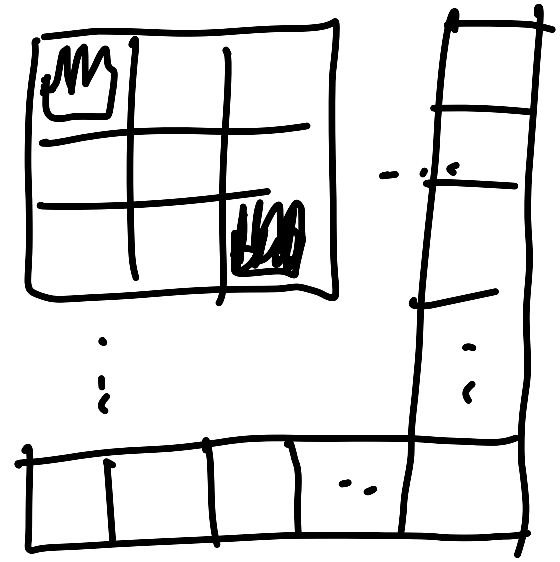
– enumerate all flows ?



MaxFlow = MinCut

certificate: an s-t cut (bottle neck) in network, "accept" if val cut < k.

GeneralizedChess



Input: $n \times n$ chessboard, configuartion

Output: "yes" $\iff$ player 1 can force a win

Question. Is GeneralizedChess in NP?

certificate: seq of moves $\rightsquigarrow$ checkmate

what about P2?

Fact. Solving Generalized Chess requires exponential time in n.

What if G.C. in NP? $\Rightarrow$ P $\neq$ NP

Boolean Formulae

- **variables** are Boolean variables, $x, y, z, \ldots$
- **logical connectives**
  - $\wedge$ = "and"
  - $\vee$ = "or"
  - $\neg$ = "not"
    - also $\bar{x} \equiv \neg x$

Example. $\varphi(x, y, z) = (x \wedge y) \vee (\bar{y} \wedge z)$.

- $\varphi(F, F, T) =$
- $\varphi(F, T, F) =$

BooleanSatisfiability T/F

Input: a Boolean formula $\varphi(x_1, x_2, \ldots, x_n)$

Output: "yes" $\iff$ φ has a satisfying assignment

Question. Is BooleanSatisfiability in NP?

Certificate:
values (T/F) for $x_1, x_2, \ldots, x_n$

Verify:
Plug in values to φ
and evaluate
"accept" if $\varphi(\ ) = $ TRUE

Reducibility in NP

Main Question. What are the hardest problems in NP?

Reducibility in NP

Main Question. What are the hardest problems in NP?

Sub-question. How are problems in NP related to each other?

- $\leq_P$ = polynomial-time reduction

Observation. If $A \leq_P B$ and $B \in \text{NP}$, then $A \in \text{NP}$

Why?

Want: Poly time verifier for A

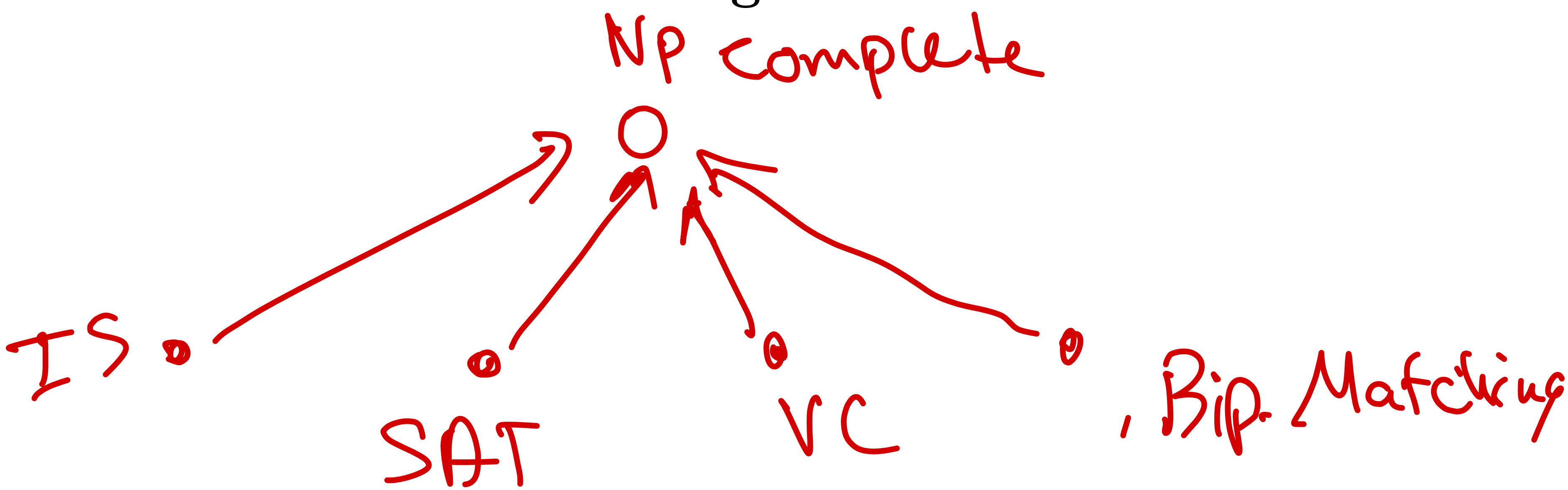
Have: (1) verifier for B
(2) reduction from A to B

A verifier: transform to B, then apply verifier for B

The Hardest Problems in NP

Definition. We say that a decision problem A is **NP complete** if for every problem $B \in \text{NP}$, we have $B \leq_P A$

- A is NP complete if every instance of every problem in NP can be reduced to solving an instance of A



The Hardest Problems in NP

Definition. We say that a decision problem A is **NP complete** if for every problem $B \in \mathrm{NP}$, we have $B \leq_P A$

- A is NP complete if every instance of every problem in NP can be reduced to solving an instance of A

Theorem [Cook 1971, Levin 1973]. There exists an NP complete problem.

NP and Verification

Observation. Every problem in NP has a polynomial time verifier

- suppose A a problem in NP
- verify is a verifier for A:
  - $\text{verify}(X, C) \mapsto$ "accept"/"reject"
- X is "yes" instance $\iff$ there exists a certificate C such that $\text{verify}(X, C) =$ "accept"
- solving A can be reduced to answering:
  - *"Is there a certificate C that is accepted by* $\text{verify}(X, C)$*?"*

Idea of Cook-Levin Proof

Suppose $A \in \mathrm{NP}$

- Given (1) verifier verify for A, (2) instance X of A
- Construct: a Boolean formula $\varphi(x_1, \ldots, x_n)$ such that φ is satisfiable $\iff$ there is a certificate C accepted by verify(X, C)
- certificates for verify$(X, \cdot)$ correspond to variable assignments for $\varphi(\cdot)$
- determining if there is a certificate C accepted by verify(X, C) is equivalent to determining if some assignment $x_1, \ldots, x_n$ satisfies $\varphi(x_1, \ldots, x_n)$.

Formal proof requires formal definition of algorithm (e.g., Turing machines) CS 401

Conclusion?

BooleanSatisfiability (SAT) is NP complete!

- every problem A in NP satisfies $A \leq_P \text{SAT}$
- an efficient algorithm for SAT would imply $\text{P} = \text{NP}$

Conclusion?

BooleanSatisfiability (SAT) is NP complete!

- every problem A in NP satisfies $A \leq_P$ SAT
- an efficient algorithm for SAT would imply $\mathrm{P} = \mathrm{NP}$

Question. Are other problems are other problems NP complete?

- How could we show a problem A is NP complete?

Simpler Boolean Formulae

Terminology:

- a **literal** is a variable or its negation: $x, \bar{x}$
- a **clause** is an expression of the from
  1. $(z_1 \wedge z_2 \wedge \cdots \wedge z_k)$ (conjuctive clause) where each z_i is a literal, or
  2. $(z_1 \vee z_2 \vee \cdots \vee z_k)$ (disjunctive clause) where each z_i is a literal
- a **conjunctive normal form** (**CNF**) expression is an expression of the form $C_1 \wedge C_2 \wedge \cdots \wedge C_\ell$ where each C_i is a disjunctive clause

Observation: a CNF formula evaluates to true $\iff$ all clauses evaluate to true

3-SAT

Definition. A **3-CNF formula** is a Boolean formula in conjunctive normal form such that every clause contains 3 literals.

Example.

$$\varphi(w, x, y, z) = (x \vee y \vee z) \wedge (y \vee \bar{z} \vee w) \wedge (\bar{x} \vee \bar{y} \vee \bar{w})$$

3-SAT:

- Input: a 3-CNF formula φ
- Output: "yes" $\iff$ φ is satisfiable

3-SAT is NP-Complete

Theorem (Tseytin 1970). Any Boolean formula φ can be efficiently (in polynomial time) transformed into a 3-CNF formula ψ such that:

1. if φ is satisfiable, then so is ψ
2. if φ is not satisfiable, then neither is ψ

3-SAT is NP-Complete

Theorem (Tseytin 1970). Any Boolean formula φ can be efficiently (in polynomial time) transformed into a 3-CNF formula ψ such that:

1. if φ is satisfiable, then so is ψ
2. if φ is not satisfiable, then neither is ψ

Consequences.

1. SAT $\leq_P$ 3-SAT
2. 3-SAT is NP complete

Relationships

IS is NP Complete

Theorem. IS in NP Complete.

Question. What do we need to show?

IS is NP Complete

Theorem. IS in NP Complete.

Question. What do we need to show?

Strategy. Reduction from 3-SAT

- show 3-SAT $\leq_P$ IS

Question. How to transform a 3-CNF φ into a graph G such that solving IS on G tells us whether φ is satisfiable?

Example

$$\varphi(w, x, y, z) = (x \vee y \vee z) \wedge (y \vee \bar{z} \vee w) \wedge (\bar{x} \vee \bar{y} \vee \bar{w})$$

Next Time

1. IS Completed
2. Coping with NP Completeness