Lecture 21: NP-Completeness I

Michael Dinitz

November 9, 2021

601.433/633 Introduction to Algorithms

Introduction

Last few weeks: slower and slower algorithms for harder and harder problems

- From $\mathbf{O(m+n)}$ time algorithms for BFS/DFS/topological sort/SCCs, to $\mathbf{O(m^2n)}$ for max flow
- Today: start of two lectures on NP-completeness: the (or at least one) line between tractability and intractability

Definition

An algorithm runs in *polynomial time* if its (worst-case) running time is $\mathbf{O(n^c)}$ for some constant $\mathbf{c} \geq \mathbf{0}$, where $\mathbf{n}$ is the size of the input.

Think of polynomial time as "fast", super-polynomial time as "slow"

Question: When do polynomial-time algorithms exist?

Decision Problems

Definition

A *decision problem* is a computational problem in which the output is either YES or NO.

Examples:

- Max-Flow: Input is $G = (V, E), c : E \to \mathbb{R}_{\geq 0}, s, t \in V, k \in \mathbb{R}^+$. Output YES if there is an (s, t)-flow of value at least k, otherwise output NO.
- Shortest $s - t$ path: Input is $G = (V, E), \ell : E \to \mathbb{R}, s, t \in V, k \in \mathbb{R}$. Output YES if $d(s, t) \leq k$, otherwise output NO.

Some problems naturally decision, others naturally optimization, but can turn any optimization problem into a decision problem.

- If can solve decision, can almost always solve optimization.

Note: Can divide instances (inputs) of any decision problem into YES-instances and NO-instances

P

Definition

P is the set of decision problems that can be solved in polynomial time.

Note: *problems* are in **P**, not *algorithms*

Question: Are all problems in **P**?
Answer: No!

- By *time hierarchy theorem* there are problems that require super-polynomial time!
- Undecidability: there are problems which cannot be solved by *any* algorithm at all!

Verification

Different Setting: If in addition to the input we're given a purported solution, can we check that this solution is valid/feasible (in polynomial time)?

- Max-Flow: given $\mathbf{f : E \to \mathbb{R}_{\geq 0}}$, check that value $\geq \mathbf{k}$, flow conservation at all nodes other than $\mathbf{s, t}$, and capacity constraints obeyed

Definition (3-Coloring)

Input: Undirected graph $\mathbf{G = (V, E)}$
Output: YES if $\exists$ *coloring* $\mathbf{f : V \to \{R, G, B\}}$ such that $\mathbf{f(u) \neq f(v)}$ for all $\mathbf{\{u, v\} \in E}$. NO otherwise

Verification: Given $\mathbf{f}$,

- Check that $\mathbf{f(u) \in \{R, G, B\}}$ for all $\mathbf{u \in V}$, and
- Check each edge $\mathbf{\{u, v\}}$ to make sure that $\mathbf{f(u) \neq f(v)}$

NP

NP: decision problems where solutions can be *verified* in polynomial time.

Definition

A decision problem $\mathbf{Q}$ is in **NP** (*nondeterministic polynomial time*) if there exists a polynomial time algorithm $\mathbf{V(I, X)}$ (called the *verifier*) such that

1. If $\mathbf{I}$ is a YES-instance of $\mathbf{Q}$, then there is some $\mathbf{X}$ (usually called the *witness*, *proof*, or *solution*) with size polynomial in $|\mathbf{I}|$ so that $\mathbf{V(I, X)}$ = YES.
2. If $\mathbf{I}$ is a NO-instance of $\mathbf{Q}$, then $\mathbf{V(I, X)}$ = NO for all $\mathbf{X}$.

Examples:

- 3-coloring: Witness $\mathbf{X}$ is a coloring $\mathbf{f : V \to \{R, B, G\}}$, verifier checks each edge $\mathbf{\{u, v\}}$ to make sure $\mathbf{f(u) \neq f(v)}$
  - If $\mathbf{I}$ is a YES instance, then there is a coloring so verifier will return YES
  - If $\mathbf{I}$ is a NO instance, then no valid coloring exists. Whatever $\mathbf{X}$ is, verifier returns NO.
- Max-Flow: Witness $\mathbf{X}$ is a flow $\mathbf{f : E \to \mathbb{R}_{\geq 0}}$, verifier checks that it's feasible of value $\geq \mathbf{k}$
  - If $\mathbf{I}$ is a YES instance, then there is a feasible flow of value at least $\mathbf{k}$ so verifier (on this flow) will return YES

P vs NP

Theorem

$\mathbf{P} \subseteq \mathbf{NP}$

Proof.

Let $\mathbf{Q} \in \mathbf{P}$.
$\mathbf{V(I, X)}$: Ignore $\mathbf{X}$, solve on instance $\mathbf{I}$. □

Question: Does $\mathbf{P} = \mathbf{NP}$, i.e., is $\mathbf{NP} \subseteq \mathbf{P}$?

- *Almost* everyone thinks no, but we don't know for sure!
- Not even particularly close to a proof.
- Think about what $\mathbf{P} = \mathbf{NP}$ would mean...

Reductions

Question: How could we prove that **P** = **NP** or **P** ≠ **NP**?

- **P** = **NP**: Need to show that *every* problem in **NP** is also in **P**!
- **P** ≠ **NP**: Need to prove that *some* problem in **NP** not in **P**. What is the "hardest" problem in **NP**?

Definition

Problem **A** is *polytime reducible* to problem **B** (written $\mathbf{A} \leq_{\mathbf{p}} \mathbf{B}$) if, given a polynomial-time algorithm for **B**, we can use it to produce a polynomial-time algorithm for **A**.

Means that **B** is "at least as hard" as **A**: if **B** is in **P**, then so is **A**.

- So "hardest" problems in **NP** are problems that many other problems reduce to.

Many-One (Karp) Reductions

Almost always (and always in this course), use a special type of reduction.

Definition

A *Many-one* or *Karp* reduction from $\mathbf{A}$ to $\mathbf{B}$ is a function $\mathbf{f}$ which takes arbitrary instances of $\mathbf{A}$ and transforms them into instances of $\mathbf{B}$ so that

1. If $\mathbf{x}$ is a YES-instance of $\mathbf{A}$ then $\mathbf{f(x)}$ is a YES-instance of $\mathbf{B}$.
2. If $\mathbf{x}$ is a NO-instance of $\mathbf{A}$ then $\mathbf{f(x)}$ is a NO-instance $\mathbf{B}$.
3. $\mathbf{f}$ can be computed in polynomial time.

So given instance $\mathbf{x}$ of $\mathbf{A}$, compute $\mathbf{f(x)}$ and use polytime algorithm for $\mathbf{B}$ on $\mathbf{f(x)}$

- Polytime, since $\mathbf{f}$ in polytime and algorithm for $\mathbf{B}$ in polytime
- Correct by first two properties of many-one reduction.

NP-Completeness

So what is "hardest problem" in **NP**?

Definition

Problem **Q** is **NP**-*hard* if $\mathbf{Q'} \leq_{\mathbf{p}} \mathbf{Q}$ for all problems **Q′** in **NP**.

Definition

Problem **Q** is **NP**-*complete* if it is **NP**-hard and in **NP**.

So suppose **Q** is **NP**-complete.

- To prove $\mathbf{P} \neq \mathbf{NP}$: Hardest problem in **NP**! If anything in **NP** is not in **P**, then **Q** is not in **P**
- To prove $\mathbf{P} = \mathbf{NP}$: Just need to prove that $\mathbf{Q} \in \mathbf{P}$.

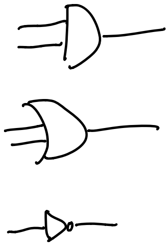
Is anything **NP**-complete?

Circuit-SAT

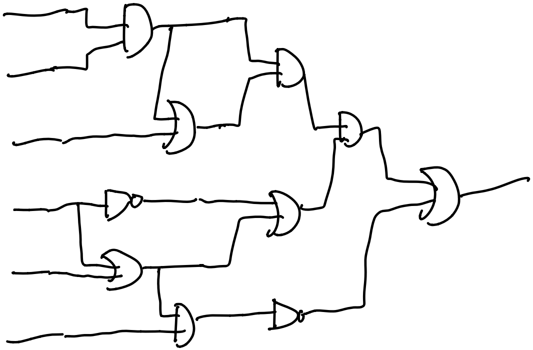
Definition

Circuit-SAT: Given a boolean circuit with a single output and no loops (some inputs might be hardwired), is there a way of setting the inputs so that the output of the circuit is **1**?

Gates: AND, OR, NOT



Arbitrary fan-out



Circuit-SAT

Theorem

*Circuit-SAT is **NP**-complete.*

Sketch of proof here. See book for details.

Lemma

*Circuit-SAT is in **NP**.*

Proof.

Witness is a T/F (or 1/0) assignment to inputs. Verifier simulates circuit on assignment, checks that it outputs **1**.

- If input is a YES instance then there is some assignment so circuit outputs **1**. When verifier run on that assignment, returns YES.
- In input is a NO instance then in every assignment circuit outputs **0**. So verifier returns NO on every witness.

□

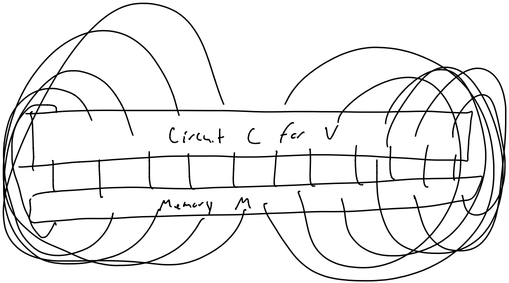
Circuit-SAT is **NP**-hard

Let **A** ∈ **NP**. Want to show **A** $\leq_p$ Circuit-SAT (construct a many-one reduction).

Where to start? What do we know about **A**?

- In **NP**, so has verifier algorithm **V**
- **V** algorithm runs on a computer (or Turing machine)!

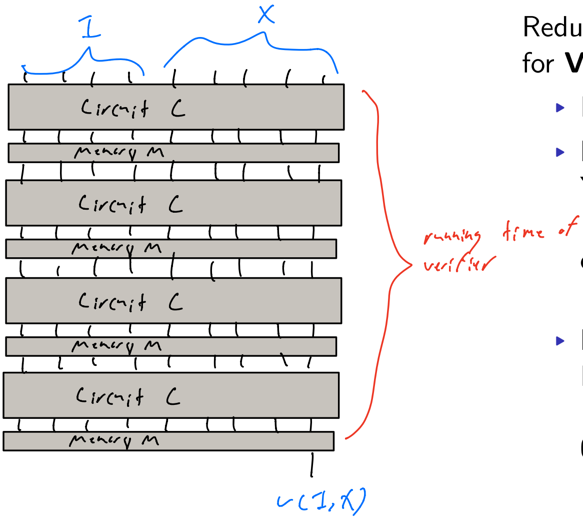
Computer: memory + circuit for modifying memory!



Not a boolean circuit in Circuit-SAT sense: loops (feedback)

Fix: "Unroll" circuit using fact that **V** runs in polynomial time

Reduction



Reduction: given instance **I** of **A**, construct this circuit for **V**, hardwire **I**. Combined circuit **f(I)**

- Polytime since **V** runs in polytime
- If **I** YES of **A**: there is some **X** so that $\mathbf{V(I,X)} =$ YES
  $\implies$ some **X** so that when **X** input to **f(I)**, outputs **1**
  $\implies$ **f(I)** YES instance of Circuit-SAT.
- If **I** NO of **A**: For every **X**, know that $\mathbf{V(I,X)} =$ NO
  $\implies$ for every **X**, when **X** input to **f(I)**, outputs **0**
  $\implies$ **f(I)** NO instance of Circuit-SAT