CS 473: Algorithms

Chandra Chekuri
chekuri@cs.illinois.edu
3228 Siebel Center

University of Illinois, Urbana-Champaign

Fall 2010
P: set of decision problems that have polynomial time algorithms

NP: set of decision problems that have polynomial time non-deterministic algorithms
P and NP and Turing Machines

- **P**: set of decision problems that have polynomial time algorithms
- **NP**: set of decision problems that have polynomial time non-deterministic algorithms

**Question**: What is an algorithm?
P and NP and Turing Machines

- **P**: set of decision problems that have polynomial time algorithms
- **NP**: set of decision problems that have polynomial time non-deterministic algorithms

**Question**: What is an algorithm? Depends on the model of computation!
P and NP and Turing Machines

- **P**: set of decision problems that have polynomial time algorithms
- **NP**: set of decision problems that have polynomial time non-deterministic algorithms

**Question**: What is an algorithm? Depends on the model of computation!

What is our model of computation?
P and NP and Turing Machines

- **P**: set of decision problems that have polynomial time algorithms
- **NP**: set of decision problems that have polynomial time non-deterministic algorithms

**Question**: What is an algorithm? Depends on the model of computation!

What is our model of computation?

Formally speaking our model of computation is Turing Machines.
Turing Machines: Recap

- Infinite tape
- Finite state control
- Input at beginning of tape
- Special tape letter “blank” □
- Head can move only one cell to left or right
A TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$:

- $Q$ is set of states in finite control
- $q_0$ start state, $q_{\text{accept}}$ is accept state, $q_{\text{reject}}$ is reject state
- $\Sigma$ is input alphabet, $\Gamma$ is tape alphabet (includes $\square$)
- $\delta : Q \times \Gamma \rightarrow \{L, R\} \times \Gamma \times Q$ is transition function
  - $\delta(q, a) = (q', b, L)$ means that $M$ in state $q$ and head seeing $a$ on tape will move to state $q'$ while replacing $a$ on tape with $b$ and head moves left.

$L(M)$: language accepted by $M$ is set of all input strings $s$ on which $M$ when started in $q_0$ on tape cell 1 and $s$ on tape halts in $q_{\text{accept}}$. 
### Definition

$M$ is a polynomial time TM if there is some polynomial $p(\cdot)$ such that on all inputs $w$, $M$ halts in $p(|w|)$ steps.

### Definition

$L$ is a language in $P$ iff there is a polynomial time TM $M$ such that $L = L(M)$. 
**Definition**

$L$ is an NP language iff there is a *non-deterministic* polynomial time TM $M$ such that $L = L(M)$. 

*Non-deterministic TM:* each step has a choice of moves $δ: Q × Γ → P(Q × Γ × \{L, R\})$.

Example: $δ(q, a) = \{(q_1, b, L), (q_2, c, R), (q_3, a, R)\}$ means $M$ can non-deterministically choose one of the three possible moves from $(q, a)$. 

$L(M)$: set of all strings $s$ on which there exists some sequence of valid choices at each step that lead from $q_0$ to $q_{accept}$.
NP via TMs

**Definition**

$L$ is an NP language iff there is a *non-deterministic* polynomial time TM $M$ such that $L = L(M)$.

Non-deterministic TM: each step has a choice of moves

- $\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$.
- Example: $\delta(q, a) = \{(q_1, b, L), (q_2, c, R), (q_3, a, R)\}$ means that $M$ can non-deterministically choose one of the three possible moves from $(q, a)$.

- $L(M)$: set of all strings $s$ on which there exists some sequence of valid choices at each step that lead from $q_0$ to $q_{\text{accept}}$.
Non-deterministic TMs vs Certifiers

Two definition of NP:

- $L$ is in NP iff $L$ has a polynomial time certifier $C(\cdot, \cdot)$.
- $L$ is in NP iff $L$ is decided by a non-deterministic polynomial time TM $M$.

Claim: Two definitions are equivalent. Why?
Non-deterministic TMs vs Certifiers

Two definitions of NP:
- $L$ is in NP iff $L$ has a polynomial time certifier $C(\cdot, \cdot)$.
- $L$ is in NP iff $L$ is decided by a non-deterministic polynomial time TM $M$.

Claim: Two definitions are equivalent. Why?

Informal proof idea: the certificate $t$ for $C$ corresponds to non-deterministic choices of $M$ and vice-versa. In other words, $L$ is in NP iff $L$ is accepted by a NTM which first guesses a proof $t$ of length poly in input $|s|$ and then acts as a deterministic TM.
Non-determinism, guessing and verification

- A non-deterministic machine has choices at each step and accepts a string if there exists a set of choices which lead to a final state.

- Equivalently the choices can be thought of as guessing a solution and then verifying that solution. In this view all the choices are made apriori and hence the verification can be deterministic. The “guess” is the “proof” and the “verifier” is the “certifier”.

- We reemphasize the asymmetry inherent in the definition of non-determinism. Strings in the language can be easily verified. No easy way to verify that a string is not in the language.
Algorithms: TMs vs RAM Model

Why do we use TMs some times and RAM Model other times?

- TMs are very simple: no complicated instruction set, no jumps/pointers, no explicit loops etc.
  - Simplicity is useful in proofs
  - The “right” formal bare-bones model when dealing with subtleties
- RAM model is a closer approximation to the running time/space usage of realistic computers for reasonable problem sizes
  - Not appropriate for certain kinds of formal proofs when algorithms can take super-polynomial time and space
“Hardest” Problems

Question

What is the hardest problem in $NP$? How do we define it?
“Hardest” Problems

**Question**
What is the hardest problem in $NP$? How do we define it?

**Towards a definition**
“Hardest” Problems

Question

What is the hardest problem in $NP$? How do we define it?

Towards a definition

- Hardest problem must be in $NP$
“Hardest” Problems

Question
What is the hardest problem in $NP$? How do we define it?

Towards a definition
- Hardest problem must be in $NP$
- Hardest problem must be at least as “difficult” as every other problem in $NP$
NP-Complete Problems

Definition

A problem $X$ is said to be $NP$-complete if

1. $X \in NP$
2. (Hardness) For any $Y \in NP$, $Y \leq_P X$
Solving NP-Complete Problems

**Proposition**

Suppose $X$ is NP-complete. Then $X$ can be solved in polynomial time iff $P = NP$.
Solving $NP$-Complete Problems

**Proposition**

*Suppose $X$ is $NP$-complete. Then $X$ can be solved in polynomial time iff $P = NP$*

**Proof.**
Solving $NP$-Complete Problems

**Proposition**

Suppose $X$ is $NP$-complete. Then $X$ can be solved in polynomial time iff $P = NP$

**Proof.**

$\Rightarrow$ Suppose $X$ can be solved in polynomial time
Solving $NP$-Complete Problems

**Proposition**

Suppose $X$ is $NP$-complete. Then $X$ can be solved in polynomial time iff $P = NP$

**Proof.**

$\Rightarrow$ Suppose $X$ can be solved in polynomial time

- Let $Y \in NP$. We know $Y \leq_P X$
Proposition

Suppose X is NP-complete. Then X can be solved in polynomial time iff P = NP

Proof.

⇒ Suppose X can be solved in polynomial time

- Let Y ∈ NP. We know Y ≤_P X
- We showed that if Y ≤_P X and X can be solved in polynomial time, then Y can be solved in polynomial time
Proposition

Suppose $X$ is $NP$-complete. Then $X$ can be solved in polynomial time iff $P = NP$.

Proof.

⇒ Suppose $X$ can be solved in polynomial time.
   • Let $Y \in NP$. We know $Y \leq_P X$.
   • We showed that if $Y \leq_P X$ and $X$ can be solved in polynomial time, then $Y$ can be solved in polynomial time.
   • Thus, every problem $Y \in NP$ is such that $Y \in P$; $NP \subseteq P$.
Proposition

Suppose $X$ is NP-complete. Then $X$ can be solved in polynomial time iff $P = NP$

Proof.

$\Rightarrow$ Suppose $X$ can be solved in polynomial time

- Let $Y \in NP$. We know $Y \leq_P X$
- We showed that if $Y \leq_P X$ and $X$ can be solved in polynomial time, then $Y$ can be solved in polynomial time
- Thus, every problem $Y \in NP$ is such that $Y \in P$; $NP \subseteq P$
- Since $P \subseteq NP$, we have $P = NP$
Proposition

Suppose $X$ is $NP$-complete. Then $X$ can be solved in polynomial time iff $P = NP$

Proof.

⇒ Suppose $X$ can be solved in polynomial time

- Let $Y \in NP$. We know $Y \leq_P X$
- We showed that if $Y \leq_P X$ and $X$ can be solved in polynomial time, then $Y$ can be solved in polynomial time
- Thus, every problem $Y \in NP$ is such that $Y \in P$; $NP \subseteq P$
- Since $P \subseteq NP$, we have $P = NP$

⇐ Since $P = NP$, and $X \in NP$, we have a polynomial time algorithm for $X$
NP-Hard Problems

**Definition**

A problem $X$ is said to be $NP$-hard if

$\bullet$ *(Hardness)* For any $Y \in NP$, $Y \leq_P X$

An $NP$-hard problem need not be in $NP$!

**Example:** Halting problem is $NP$-hard (why?) but not $NP$-complete.
Consequences of proving $NP$-completeness

If $X$ is $NP$-complete

- Since we believe $P \neq NP$,
Consequences of proving $NP$-completeness

If $X$ is $NP$-complete

- Since we believe $P \neq NP$,
- and solving $X$ implies $P = NP$
Consequences of proving $NP$-completeness

If $X$ is $NP$-complete

- Since we believe $P \neq NP$,
- and solving $X$ implies $P = NP$

$X$ is unlikely to be efficiently solvable
Consequences of proving $NP$-completeness

If $X$ is $NP$-complete

- Since we believe $P \neq NP$,
- and solving $X$ implies $P = NP$

$X$ is unlikely to be efficiently solvable
At the very least, many smart people before you have failed to find an efficient algorithm for $X$
NP-Complete Problems

**Question**

Are there any problems that are \( NP \)-complete?

**Answer**

Yes! Many, many problems are \( NP \)-complete.
Circuits

Definition

A circuit is a directed *acyclic* graph with

- **Input** vertices (without incoming edges) labelled with 0, 1 or a distinct variable

![Circuit Diagram]

Inputs: 1, ?, ?, 0, ?
A circuit is a directed *acyclic* graph with

- **Input** vertices (without incoming edges) labelled with 0, 1 or a distinct variable
- Every other vertex is labelled \( \lor, \land \) or \( \neg \)

Inputs: 1, ?, ?, 0, ?
Circuits

**Definition**

A circuit is a directed *acyclic* graph with

- **Input** vertices (without incoming edges) labelled with 0, 1 or a distinct variable
- Every other vertex is labelled \( \lor, \land \) or \( \neg \)
- Single node output vertex with no outgoing edges
Cook-Levin Theorem

Circuit Satisfaction

Given a circuit as input, is there an assignment to the input variables that causes the output to get value 1?
Cook-Levin Theorem

Circuit Satisfaction

Given a circuit as input, is there an assignment to the input variables that causes the output to get value 1?

Theorem (Cook-Levin)

*Circuit Satisfaction is NP-complete*
Cook-Levin Theorem

Circuit Satisfaction
Given a circuit as input, is there an assignment to the input variables that causes the output to get value 1?

Theorem (Cook-Levin)
Circuit Satisfaction is NP-complete

Need to show
- Circuit Satisfaction is in NP
- every NP problem $X$ reduces to Circuit Satisfaction
Circuit Satisfaction is in $NP$.

- Certificate:
- Certifier:
Circuit Satisfaction is in \( NP \).

- **Certificate**: assignment to input variables
- **Certifier**: evaluate the value of each gate in a topological sort of DAG and check the output gate value
Circuit Satisfaction is NP-hard: Idea

Need to show that every NP problem $X$ reduces to Circuit-SAT.
Circuit Satisfaction is NP-hard: Idea

Need to show that every NP problem $X$ reduces to Circuit-SAT.

What does it mean that $X \in NP$?
Circuit Satisfaction is NP-hard: Idea

Need to show that every NP problem \( X \) reduces to Circuit-SAT.

What does it mean that \( X \in NP \)?

\( X \in NP \) implies that there are polynomials \( p() \) and \( q() \) and certifier/verifier program \( C \) such that for every string \( s \) the following is true:
Circuit Satisfaction is NP-hard: Idea

Need to show that every NP problem $X$ reduces to Circuit-SAT.

What does it mean that $X \in NP$?

$X \in NP$ implies that there are polynomials $p()$ and $q()$ and certifier/verifier program $C$ such that for every string $s$ the following is true:

- If $s$ is a YES instance ($s \in X$) then there is a proof $t$ of length $p(|s|)$ such that $C(s, t)$ says YES.
- If $s$ is a NO instance ($s \notin X$) then for every string $t$ of length at $p(|s|)$, $C(s, t)$ says NO.
- $C(s, t)$ runs in time $q(|s| + |t|)$ time (hence polynomial time)
Reducing $X$ to Circuit-SAT

$X$ is in NP means we have access to $p(), q(), C(\cdot, \cdot)$. 
Reducing $X$ to Circuit-SAT

$X$ is in NP means we have access to $p()$, $q()$, $C(\cdot, \cdot)$. What is $C(\cdot, \cdot)$? It is a program or equivalently a Turing Machine!
Reducing $X$ to Circuit-SAT

$X$ is in NP means we have access to $p()$, $q()$, $C(\cdot, \cdot)$. What is $C(\cdot, \cdot)$? It is a program or equivalently a Turing Machine! How are $p()$ and $q()$ given?
Reducing X to Circuit-SAT

\(X\) is in NP means we have access to \(p(), q(), C(\cdot, \cdot)\).

What is \(C(\cdot, \cdot)\)? It is a program or equivalently a Turing Machine!

How are \(p()\) and \(q()\) given? As numbers.

Example: if 3 is given then \(p() = n^3\).

Thus an NP problem is essentially a three tuple \(< p, q, C >\) where \(C\) is either a program or TM.
Reducing $X$ to Circuit-SAT

Thus an NP problem is essentially a three tuple $< p, q, C >$ where $C$ is either a program or TM.

**Problem X:** Given string $s$, is $s \in X$?
Reducing $X$ to Circuit-SAT

Thus an NP problem is essentially a three tuple $<p, q, C>$ where $C$ is either a program or TM.

Problem $X$: Given string $s$, is $s \in X$?

Same as the following: is there a proof $t$ of length $p(|s|)$ such that $C(s, t)$ says YES.

How do we reduce $X$ to Circuit-SAT?
Reducing X to Circuit-SAT

Thus an NP problem is essentially a three tuple \(< p, q, C >\) where
\(C\) is either a program or TM.

Problem X: Given string \(s\), is \(s \in X\)?

Same as the following: is there a proof \(t\) of length \(p(|s|)\) such that
\(C(s, t)\) says YES.

How do we reduce \(X\) to Circuit-SAT? Need an algorithm \(A\) that
Reducing $X$ to Circuit-SAT

Thus an NP problem is essentially a three tuple $< p, q, C >$ where $C$ is either a program or TM.

**Problem X:** Given string $s$, is $s \in X$?

Same as the following: is there a proof $t$ of length $p(|s|)$ such that $C(s, t)$ says YES.

How do we reduce $X$ to Circuit-SAT? Need an algorithm $A$ that
- takes $s$ (and $< p, q, C >$) and creates a circuit $G$ in polynomial time in $|s|$ (note that $< p, q, C >$ are fixed).
Reducing X to Circuit-SAT

Thus an NP problem is essentially a three tuple $< p, q, C >$ where $C$ is either a program or TM.

**Problem X:** Given string $s$, is $s \in X$?

Same as the following: is there a proof $t$ of length $p(|s|)$ such that $C(s, t)$ says YES.

How do we reduce $X$ to Circuit-SAT? Need an algorithm $A$ that

- takes $s$ (and $< p, q, C >$) and creates a circuit $G$ in polynomial time in $|s|$ (note that $< p, q, C >$ are fixed).
- $G$ is satisfiable if and only if there is a proof $t$ such that $C(s, t)$ says YES.
Reducing $X$ to Circuit-SAT

How do we reduce $X$ to Circuit-SAT?

Need an algorithm $A$ that takes $s$ (and $p$, $q$, $C$) and creates a circuit $G$ in polynomial time in $|s|$ (note that $p$, $q$, $C$ are fixed).

$G$ is satisfiable if and only if there is a proof $t$ such that $C(s, t)$ says YES

Simple but Big Idea: Programs are essentially the same as Circuits!

Convert $C(s, t)$ into a circuit $G$ with $t$ as unknown inputs (rest is known including $s$)

We know that $|t| = p(|s|)$ so express boolean string $t$ as $p(|s|)$ variables $t_1, t_2, \ldots, t_k$ where $k = p(|s|)$.

Asking if there is a proof $t$ that makes $C(s, t)$ say YES is same as whether there is an assignment of values to “unknown” variables $t_1, t_2, \ldots, t_k$ that will make $G$ evaluate to true/YES.
Reducing $X$ to Circuit-SAT

How do we reduce $X$ to Circuit-SAT? Need an algorithm $\mathcal{A}$ that
Reducing $X$ to Circuit-SAT

How do we reduce $X$ to Circuit-SAT? Need an algorithm $A$ that

- takes $s$ (and $< p, q, C >$) and creates a circuit $G$ in polynomial time in $|s|$ (note that $< p, q, C >$ are fixed).
- $G$ is satisfiable if and only if there is a proof $t$ such that $C(s, t)$ says YES.
Reducing $X$ to Circuit-SAT

How do we reduce $X$ to Circuit-SAT? Need an algorithm $A$ that
- takes $s$ (and $<p, q, C>$) and creates a circuit $G$ in polynomial time in $|s|$ (note that $<p, q, C>$ are fixed).
- $G$ is satisfiable if and only if there is a proof $t$ such that $C(s, t)$ says YES

Simple but Big Idea: Programs are essentially the same as Circuits!
Reducing X to Circuit-SAT

How do we reduce X to Circuit-SAT? Need an algorithm $A$ that

- takes $s$ (and $< p, q, C >$) and creates a circuit $G$ in polynomial time in $|s|$ (note that $< p, q, C >$ are fixed).
- $G$ is satisfiable if and only if there is a proof $t$ such that $C(s, t)$ says YES

**Simple but Big Idea:** Programs are essentially the same as Circuits!

- Convert $C(s, t)$ into a circuit $G$ with $t$ as unknown inputs (rest is known including $s$)
Reducing $X$ to Circuit-SAT

How do we reduce $X$ to Circuit-SAT? Need an algorithm $A$ that

- takes $s$ (and $< p, q, C >$) and creates a circuit $G$ in polynomial time in $|s|$ (note that $< p, q, C >$ are fixed).
- $G$ is satisfiable if and only if there is a proof $t$ such that $C(s, t)$ says YES

Simple but Big Idea: Programs are essentially the same as Circuits!

- Convert $C(s, t)$ into a circuit $G$ with $t$ as unknown inputs (rest is known including $s$)
- We know that $|t| = p(|s|)$ so express boolean string $t$ as $p(|s|)$ variables $t_1, t_2, \ldots, t_k$ where $k = p(|s|)$.
Reducing $X$ to Circuit-SAT

How do we reduce $X$ to Circuit-SAT? Need an algorithm $A$ that
- takes $s$ (and $<p, q, C>$) and creates a circuit $G$ in polynomial time in $|s|$ (note that $<p, q, C>$ are fixed).
- $G$ is satisfiable if and only if there is a proof $t$ such that $C(s, t)$ says YES

Simple but Big Idea: Programs are essentially the same as Circuits!
- Convert $C(s, t)$ into a circuit $G$ with $t$ as unknown inputs (rest is known including $s$)
- We know that $|t| = p(|s|)$ so express boolean string $t$ as $p(|s|)$ variables $t_1, t_2, \ldots, t_k$ where $k = p(|s|)$.
- Asking if there is a proof $t$ that makes $C(s, t)$ say YES is same as whether there is an assignment of values to “unknown” variables $t_1, t_2, \ldots, t_k$ that will make $G$ evaluate to true/YES.
Example: Independent Set

- **Problem:** Does $G = (V, E)$ have an independent set of size $\geq k$?
  - **Certificate:** Set $S \subseteq V$
  - **Certifier:** Check $|S| \geq k$ and no pair of vertices in $S$ is connected by an edge
Example: Independent Set

- **Problem:** Does \( G = (V, E) \) have an independent set of size \( \geq k \)?
  - **Certificate:** Set \( S \subseteq V \)
  - **Certifier:** Check \( |S| \geq k \) and no pair of vertices in \( S \) is connected by an edge

Formally why is Independent Set in NP?
Example: Independent Set

Formally why is Independent Set in NP?

- **Input:**
  \[
  \langle n, y_{1,1}, y_{1,2}, \ldots, y_{1,n}, y_{2,1}, \ldots, y_{2,n}, \ldots, y_{n,1}, \ldots, y_{n,n}, k \rangle
  \]
  encodes \(\langle G, k \rangle\).
  
  - \(n\) is number of vertices in \(G\)
  - \(y_{i,j}\) is a bit which is 1 if edge \((i, j)\) is in \(G\) and 0 otherwise (adjacency matrix representation)
  - \(k\) is size of independent set

- **Certificate:** \(t = t_1 t_2 \ldots t_n\). Interpretation is that \(t_i\) is 1 if vertex \(i\) is in the independent set, 0 otherwise.
Certifier $C(s, t)$ for Independent Set:

if $(t_1 + t_2 + \ldots + t_n < k)$ then
  return NO
else
  for each $(i, j)$ do
    if $(t_i \wedge t_j \wedge y_{i,j})$ then
      return NO
  return YES
Example: Independent Set

Figure: Graph $G$ with $k = 2$
Circuit from Certifier
Consider “program” $A$ that takes $f(|s|)$ steps on input string $s$.

**Question:** What computer is the program running on and what does *step* mean?
Programs, Turing Machines and Circuits

Consider “program” $A$ that takes $f(|s|)$ steps on input string $s$.

**Question:** What computer is the program running on and what does *step* mean?

Real computers difficult to reason with mathematically because
- instruction set is too rich
- pointers and control flow jumps in one step
- assumption that pointer to code fits in one word

Turing Machines
- simpler model of computation to reason with
- can simulate real computers with *polynomial* slow down
- all moves are *local* (head moves only one cell)
Certifiers that at TMs

Assume $C(\cdot, \cdot)$ is a (deterministic) Turing Machine $M$

**Problem:** Given $M$, input $s$, $p$, $q$ decide if there is a proof $t$ of length $p(|s|)$ such that $M$ on $s$, $t$ will halt in $q(|s|)$ time and say YES.

There is an algorithm $A$ that can reduce above problem to Circuit-SAT mechanically as follows.

- $A$ first computes $p(|s|)$ and $q(|s|)$.
- Knows that $M$ can use at most $q(|s|)$ memory/tape cells
- Knows that $M$ can run for at most $q(|s|)$ time
- Simulates the evolution of the state of $M$ and memory over time using a big circuit
Simulation of Computation via Circuit

- Think of $M$’s state at time $\ell$ as a string $x^{\ell} = x_1 x_2 \ldots x_k$ where each $x_i \in \{0, 1, B\} \times Q \cup \{q_{-1}\}$.
- At time 0 the state of $M$ consists of input string $s$ a guess $t$ (unknown variables) of length $p(|s|)$ and rest $q(|s|)$ blank symbols.
- At time $q(|s|)$ we wish to know if $M$ stops in $q_{\text{accept}}$ with say all blanks on the tape.
- We write a circuit $C_\ell$ which captures the transition of $M$ from time $\ell$ to time $\ell + 1$.
- Composition of the circuits for all times 0 to $q(|s|)$ gives a big (still poly) sized circuit $C$.
- The final output of $C$ should be true if and only if the entire state of $M$ at the end leads to an accept state.
NP-hardness of Circuit Satisfaction

Key Ideas in reduction:

- Use TMs as the code for certifier for simplicity
- Since $p()$ and $q()$ are known to $A$, it can set up all required memory and time steps in advance
- Simulate computation of the TM from one time to the next as a circuit that only looks at three adjacent cells at a time
**NP-hardness of Circuit Satisfaction**

Key Ideas in reduction:

- Use TMs as the code for certifier for simplicity
- Since \( p() \) and \( q() \) are known to \( A \), it can set up all required memory and time steps in advance
- Simulate computation of the TM from one time to the next as a circuit that only looks at three adjacent cells at a time

**Note:** Above reduction can be done to SAT as well. Reduction to SAT was the original proof of Steve Cook.
SAT is \( NP \)-complete

- We have seen that SAT \( \in \) \( NP \)
- To show \( NP \)-hardness, we will reduce Circuit Satisfiability (CIR-SAT) to SAT
CIR-SAT $\leq_P$ SAT

Reduction
CIR-SAT $\leq_P$ SAT

**Reduction**

- For each gate (vertex) $v$ in the circuit, create a variable $x_v$
**Reduction**

- For each gate (vertex) $v$ in the circuit, create a variable $x_v$
- **Case $\neg$:** $v$ is labelled $\neg$ and has one incoming edge from $u$ (so $x_v = \neg x_u$).
CIR-SAT $\leq_P$ SAT

Reduction

- For each gate (vertex) $v$ in the circuit, create a variable $x_v$
- **Case $\neg$:** $v$ is labelled $\neg$ and has one incoming edge from $u$ (so $x_v = \neg x_u$). In SAT, add clauses ($x_u \lor x_v$), ($\neg x_u \lor \neg x_v$)
CIR-SAT $\leq_P$ SAT

Reduction
- For each gate (vertex) $v$ in the circuit, create a variable $x_v$
- **Case $\neg$:** $v$ is labelled $\neg$ and has one incoming edge from $u$ (so $x_v = \neg x_u$). In SAT, add clauses $(x_u \lor x_v)$, $(\neg x_u \lor \neg x_v)$
- **Case $\lor$:** So $x_v = x_u \lor x_w$.
CIR-SAT $\leq_P$ SAT

**Reduction**

- For each gate (vertex) $v$ in the circuit, create a variable $x_v$
- **Case $\neg$:** $v$ is labelled $\neg$ and has one incoming edge from $u$ (so $x_v = \neg x_u$). In SAT, add clauses $(x_u \lor x_v)$, $(\neg x_u \lor \neg x_v)$
- **Case $\lor$:** So $x_v = x_u \lor x_w$. In SAT, add clauses $(x_v \lor \neg x_u)$, $(x_v \lor \neg x_w)$, and $(\neg x_v \lor x_u \lor x_w)$

If $v$ is an input gate with a fixed value then we do the following. If $x_v = 1$ add clause $x_v$. If $x_v = 0$ add clause $\neg x_v$. Add the clause $x_v$ where $v$ is the variable for the output gate.
Reduction

- For each gate (vertex) \( v \) in the circuit, create a variable \( x_v \).
- **Case \( \neg \):** \( v \) is labelled \( \neg \) and has one incoming edge from \( u \) (so \( x_v = \neg x_u \)). In SAT, add clauses \( (x_u \lor x_v), (\neg x_u \lor \neg x_v) \).
- **Case \( \lor \):** So \( x_v = x_u \lor x_w \). In SAT, add clauses \( (x_v \lor \neg x_u) \), \( (x_v \lor \neg x_w) \), and \( (\neg x_v \lor x_u \lor x_w) \).
- **Case \( \land \):** So \( x_v = x_u \land x_w \).
**CIR-SAT \( \leq_P \) SAT**

### Reduction

- For each gate (vertex) \( v \) in the circuit, create a variable \( x_v \)

- **Case \( \neg \):** \( v \) is labelled \( \neg \) and has one incoming edge from \( u \) (so \( x_v = \neg x_u \)). In SAT, add clauses \((x_u \lor x_v), (\neg x_u \lor \neg x_v)\)

- **Case \( \lor \):** So \( x_v = x_u \lor x_w \). In SAT, add clauses \((x_v \lor \neg x_u), (x_v \lor \neg x_w), \text{ and } (\neg x_v \lor x_u \lor x_w)\)

- **Case \( \land \):** So \( x_v = x_u \land x_w \). In SAT, add clauses \((\neg x_v \lor x_u), (\neg x_v \lor x_w), \text{ and } (x_v \lor \neg x_u \lor \neg x_w)\)
CIR-SAT $\leq_P$ SAT

Reduction

- For each gate (vertex) $v$ in the circuit, create a variable $x_v$
- **Case $\neg$:** $v$ is labelled $\neg$ and has one incoming edge from $u$ (so $x_v = \neg x_u$). In SAT, add clauses $(x_u \lor x_v)$, $(\neg x_u \lor \neg x_v)$
- **Case $\lor$:** So $x_v = x_u \lor x_w$. In SAT, add clauses $(x_v \lor \neg x_u)$, $(x_v \lor \neg x_w)$, and $(\neg x_v \lor x_u \lor x_w)$
- **Case $\land$:** So $x_v = x_u \land x_w$. In SAT, add clauses $(\neg x_v \lor x_u)$, $(\neg x_v \lor x_w)$, and $(x_v \lor \neg x_u \lor \neg x_w)$
- If $v$ is an input gate with a fixed value then we do the following. If $x_v = 1$ add clause $x_v$. If $x_v = 0$ add clause $\neg x_v$
CIR-SAT \leq_P SAT

Reduction

- For each gate (vertex) \( v \) in the circuit, create a variable \( x_v \)
- **Case \( \neg \):** \( v \) is labelled \( \neg \) and has one incoming edge from \( u \) (so \( x_v = \neg x_u \)). In SAT, add clauses \( (x_u \lor x_v), (\neg x_u \lor \neg x_v) \)
- **Case \( \lor \):** So \( x_v = x_u \lor x_w \). In SAT, add clauses \( (x_v \lor \neg x_u), (x_v \lor \neg x_w) \), and \( (\neg x_v \lor x_u \lor x_w) \)
- **Case \( \land \):** So \( x_v = x_u \land x_w \). In SAT, add clauses \( (\neg x_v \lor x_u), (\neg x_v \lor x_w) \), and \( (x_v \lor \neg x_u \lor \neg x_w) \)
- If \( v \) is an input gate with a fixed value then we do the following. If \( x_v = 1 \) add clause \( x_v \). If \( x_v = 0 \) add clause \( \neg x_v \)
- Add the cluase \( x_v \) where \( v \) is the variable for the output gate
SAT formula from circuit

Variables $x_a, x_b, \ldots, x_k$, one for each gate

Relationship between variables given by gates and their input/outputs.

\[
\begin{align*}
  x_a &= 1 \\
  x_f &= x_a \land x_b \\
  x_i &= \neg x_f \\
  x_j &= x_g \land x_h \\
  x_k &= x_i \land x_j
\end{align*}
\]

\[
\begin{align*}
  x_d &= 0 \\
  x_g &= x_b \lor x_c \\
  x_h &= x_d \lor x_e
\end{align*}
\]
Example

Express equalities using CNF clauses

Final SAT formula $\varphi_C$: conjunction of all of above clauses and the clause $x_k$, the output of the final gate
\[ x_f = x_a \land x_b \]

\[ = \overline{\overline{(\overline{x_f} \Rightarrow (x_a \land x_b)) \land \overline{(x_a \land x_b)}} \Rightarrow x_f} \]

\[ = (\overline{x_f} \lor (x_a \land x_b)) \land (\overline{x_f} \lor (x_a \land x_b) \lor x_f) \]

\[ = (\overline{x_f} \lor x_a) \land (\overline{x_f} \lor x_b) \land (\overline{x_f} \lor x_a \lor x_b) \]
Correctness of Reduction

Need to show circuit $C$ is satisfiable iff $\varphi_C$ is satisfiable

$\Rightarrow$ Consider a satisfying assignment $a$ for $C$
Correctness of Reduction

Need to show circuit $C$ is satisfiable iff $\varphi_C$ is satisfiable

⇒ Consider a satisfying assignment $a$ for $C$
  - Find values of all gates in $C$ under $a$
Correctness of Reduction

Need to show circuit $C$ is satisfiable iff $\varphi_C$ is satisfiable

⇒ Consider a satisfying assignment $a$ for $C$
  - Find values of all gates in $C$ under $a$
  - Give value of gate $v$ to variable $x_v$; call this assignment $a'$
Correctness of Reduction

Need to show circuit $C$ is satisfiable iff $\varphi_C$ is satisfiable

$\Rightarrow$ Consider a satisfying assignment $a$ for $C$

- Find values of all gates in $C$ under $a$
- Give value of gate $v$ to variable $x_v$; call this assignment $a'$
- $a'$ satisfies $\varphi_C$ (exercise)
Correctness of Reduction

Need to show circuit $C$ is satisfiable iff $\varphi_C$ is satisfiable

$\Rightarrow$ Consider a satisfying assignment $a$ for $C$
- Find values of all gates in $C$ under $a$
- Give value of gate $v$ to variable $x_v$; call this assignment $a'$
- $a'$ satisfies $\varphi_C$ (exercise)

$\Leftarrow$ Consider a satisfying assignment $a$ for $\varphi_C$
Correctness of Reduction

Need to show circuit $C$ is satisfiable iff $\varphi_C$ is satisfiable

$\Rightarrow$ Consider a satisfying assignment $a$ for $C$
  - Find values of all gates in $C$ under $a$
  - Give value of gate $v$ to variable $x_v$; call this assignment $a'$
  - $a'$ satisfies $\varphi_C$ (exercise)

$\Leftarrow$ Consider a satisfying assignment $a$ for $\varphi_C$
  - Let $a'$ be the restriction of $a$ to only the input variables
Correctness of Reduction

Need to show circuit $C$ is satisfiable iff $\varphi_C$ is satisfiable

$\Rightarrow$ Consider a satisfying assignment $a$ for $C$
  - Find values of all gates in $C$ under $a$
  - Give value of gate $v$ to variable $x_v$; call this assignment $a'$
  - $a'$ satisfies $\varphi_C$ (exercise)

$\Leftarrow$ Consider a satisfying assignment $a$ for $\varphi_C$
  - Let $a'$ be the restriction of $a$ to only the input variables
  - Value of gate $v$ under $a'$ is the same as value of $x_v$ in $a$
Correctness of Reduction

Need to show circuit $C$ is satisfiable iff $\varphi_C$ is satisfiable

$\Rightarrow$ Consider a satisfying assignment $a$ for $C$
- Find values of all gates in $C$ under $a$
- Give value of gate $v$ to variable $x_v$; call this assignment $a'$
- $a'$ satisfies $\varphi_C$ (exercise)

$\Leftarrow$ Consider a satisfying assignment $a$ for $\varphi_C$
- Let $a'$ be the restriction of $a$ to only the input variables
- Value of gate $v$ under $a'$ is the same as value of $x_v$ in $a$
- Thus, $a'$ satisfies $C$
Proving that a problem X is NP-complete

To prove X is NP-complete, show

- Show X is in \( NP \).
  - certificate/proof of polynomial size in input
  - polynomial time certifier \( C(s, t) \)

- Reduction from a known NP-complete problem such as CIR-SAT or SAT to X
Proving that a problem $X$ is NP-complete

To prove $X$ is NP-complete, show

- Show $X$ is in $NP$.
  - certificate/proof of polynomial size in input
  - polynomial time certifier $C(s, t)$
- Reduction from a known NP-complete problem such as CIR-SAT or SAT to $X$

$SAT \leq_P X$ implies that every NP problem $Y \leq_P X$. Why?
Proving that a problem $X$ is NP-complete

To prove $X$ is NP-complete, show

- Show $X$ is in $NP$.
  - certificate/proof of polynomial size in input
  - polynomial time certifier $C(s, t)$
- Reduction from a known NP-complete problem such as CIR-SAT or SAT to $X$

$SAT \leq_P X$ implies that every NP problem $Y \leq_P X$. Why?

Transitivity of reductions:

$Y \leq_P SAT$ and $SAT \leq_P X$ and hence $Y \leq_P X$. 
NP-Completeness via Reductions

- CIR-SAT is NP-complete
- CIR-SAT $\leq_P$ SAT and SAT is in NP and hence SAT is NP-complete
- SAT $\leq_P$ 3-SAT and hence 3-SAT is NP-complete
- 3-SAT $\leq_P$ Independent Set (which is in NP) and hence Independent Set is NP-complete
- Vertex Cover is NP-complete
- Clique is NP-complete

Hundreds and thousands of different problems from many areas of science and engineering have been shown to be NP-complete. A surprisingly frequent phenomenon!
NP-Completeness via Reductions

- CIR-SAT is NP-complete
- CIR-SAT $\leq_P$ SAT and SAT is in NP and hence SAT is NP-complete
- SAT $\leq_P$ 3-SAT and hence 3-SAT is NP-complete
- 3-SAT $\leq_P$ Independent Set (which is in NP) and hence Independent Set is NP-complete
- Vertex Cover is NP-complete
- Clique is NP-complete

Hundreds and thousands of different problems from many areas of science and engineering have been shown to be NP-complete.

A surprisingly frequent phenomenon!
$P \neq \text{NP}$