NP and NP Completeness

Lecture 23

NOV 29, 2018
Part I

NP
**P and NP and Turing Machines**

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

- Many natural problems we would like to solve are in **NP**.
- Every problem in **NP** has an exponential time algorithm
- \( P \subseteq NP \)
- Some problems in **NP** are in **P** (example, shortest path problem)

**Big Question**: Does every problem in **NP** have an efficient algorithm? Same as asking whether \( P = NP \).
Problems with no known polynomial time algorithms

<table>
<thead>
<tr>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Independent Set</td>
</tr>
<tr>
<td>2 Vertex Cover</td>
</tr>
<tr>
<td>3 Set Cover</td>
</tr>
<tr>
<td>4 SAT</td>
</tr>
<tr>
<td>5 3SAT</td>
</tr>
</tbody>
</table>

There are of course undecidable problems (no algorithm at all!) but many problems that we want to solve are of similar flavor to the above.

Question: What is common to above problems?
Above problems share the following feature:

**Checkability**

For any YES instance $I_X$ of $X$ there is a proof/certificate/solution that is of length $\text{poly}(|I_X|)$ such that given a proof one can efficiently check that $I_X$ is indeed a YES instance.
Above problems share the following feature:

**Checkability**

For any YES instance $I_X$ of $X$ there is a proof/certificate/solution that is of length $\text{poly}(|I_X|)$ such that given a proof one can efficiently check that $I_X$ is indeed a YES instance.

Examples:

1. **SAT** formula $\varphi$: proof is a satisfying assignment.
2. **Independent Set** in graph $G$ and $k$: a subset $S$ of vertices.
3. **Homework**
Given $n \times n$ sudoku puzzle, does it have a solution?
Certifiers

**Definition**

An algorithm \( C(\cdot, \cdot) \) is a **certifier** for problem \( X \) if the following two conditions hold:

- For every \( s \in X \) there is some string \( t \) such that \( C(s, t) = "yes" \)
- If \( s \not\in X \), \( C(s, t) = "no" \) for every \( t \).

The string \( t \) is called a **certificate** or **proof** for \( s \).
Definition (Efficient Certifier.)

A certifier $C$ is an **efficient certifier** for problem $X$ if there is a polynomial $p(\cdot)$ such that the following conditions hold:

- For every $s \in X$ there is some string $t$ such that $C(s, t) =$ "yes" and $|t| \leq p(|s|)$.
- If $s \notin X$, $C(s, t) =$ "no" for every $t$.
- $C(\cdot, \cdot)$ runs in polynomial time.
Example: Independent Set

1. **Problem:** Does $G = (V, E)$ have an independent set of size $\geq k$?

2. **Certificate:** Set $S \subseteq V$.

3. **Certifier:** Check $|S| \geq k$ and no pair of vertices in $S$ is connected by an edge.
Example: Vertex Cover

1. **Problem:** Does $G$ have a vertex cover of size $\leq k$?
2. **Certificate:** $S \subseteq V$.
3. **Certifier:** Check $|S| \leq k$ and that for every edge at least one endpoint is in $S$. 
Example: SAT

Problem: Does formula $\varphi$ have a satisfying truth assignment?

Certificate: Assignment $a$ of 0/1 values to each variable.

Certifier: Check each clause under $a$ and say “yes” if all clauses are true.
Example: Composites

Problem: **Composite**

**Instance:** A number $s$.

**Question:** Is the number $s$ a composite?

1. **Problem: Composite.**
   1. **Certificate:** A factor $t \leq s$ such that $t \neq 1$ and $t \neq s$.
   2. **Certifier:** Check that $t$ divides $s$. 
Example: NFA Universality

**Problem:** NFA Universality

**Instance:** Description of a NFA $M$.

**Question:** Is $L(M) = \Sigma^*$, that is, does $M$ accept all strings?

### Problem: NFA Universality.

1. **Certificate:** A DFA $M'$ equivalent to $M$
2. **Certifier:** Check that $L(M') = \Sigma^*$
Example: NFA Universality

**Problem: NFA Universality**

**Instance:** Description of a NFA $M$.

**Question:** Is $L(M) = \Sigma^*$, that is, does $M$ accept all strings?

1. **Problem:** NFA Universality.
   
   1. **Certificate:** A DFA $M'$ equivalent to $M$
   
   2. **Certifier:** Check that $L(M') = \Sigma^*$

Certifier is efficient but certificate is not necessarily short! We do not know if the problem is in $NP$. 
Example: A String Problem

Problem: PCP

Instance: Two sets of binary strings $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$

Question: Are there indices $i_1, i_2, \ldots, i_k$ such that $\alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_k} = \beta_{i_1} \beta_{i_2} \ldots \beta_{i_k}$

1. Problem: PCP
   1. Certificate: A sequence of indices $i_1, i_2, \ldots, i_k$
   2. Certifier: Check that $\alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_k} = \beta_{i_1} \beta_{i_2} \ldots \beta_{i_k}$
Example: A String Problem

**Problem:** PCP

**Instance:** Two sets of binary strings $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$

**Question:** Are there indices $i_1, i_2, \ldots, i_k$ such that $\alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_k} = \beta_{i_1} \beta_{i_2} \ldots \beta_{i_k}$

1. **Problem:** PCP
   1. **Certificate:** A sequence of indices $i_1, i_2, \ldots, i_k$
   2. **Certifier:** Check that $\alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_k} = \beta_{i_1} \beta_{i_2} \ldots \beta_{i_k}$

PCP = Posts Correspondence Problem and it is undecidable! Implies no finite bound on length of certificate!
Nondeterministic Polynomial Time

Definition

Nondeterministic Polynomial Time (denoted by $\text{NP}$) is the class of all problems that have efficient certifiers.
Nondeterministic Polynomial Time

**Definition**

Nondeterministic Polynomial Time (denoted by \textbf{NP}) is the class of all problems that have efficient certifiers.

**Example**

Independent Set, Vertex Cover, Set Cover, SAT, 3SAT, and Composite are all examples of problems in \textbf{NP}.
Why is it called…

Nondeterministic Polynomial Time

A certifier is an algorithm \( C(I, c) \) with two inputs:

1. \( I \): instance.
2. \( c \): proof/certificate that the instance is indeed a YES instance of the given problem.

One can think about \( C \) as an algorithm for the original problem, if:

1. Given \( I \), the algorithm guesses (non-deterministically, and who knows how) a certificate \( c \).
2. The algorithm now verifies the certificate \( c \) for the instance \( I \).

\( \text{NP} \) can be equivalently described using Turing machines.
Asymmetry in Definition of NP

Note that only YES instances have a short proof/certificate. NO instances need not have a short certificate.

Example

SAT formula $\varphi$. No easy way to prove that $\varphi$ is NOT satisfiable!

More on this and co-NP later on.
Proposition

\[ P \subseteq NP. \]
Proposition

\[ P \subseteq NP. \]

For a problem in \( P \) no need for a certificate!

Proof.

Consider problem \( X \in P \) with algorithm \( A \). Need to demonstrate that \( X \) has an efficient certifier:

1. Certifier \( C \) on input \( s, t \), runs \( A(s) \) and returns the answer.
2. \( C \) runs in polynomial time.
3. If \( s \in X \), then for every \( t \), \( C(s, t) = "yes" \).
4. If \( s \not\in X \), then for every \( t \), \( C(s, t) = "no" \).
Exponential Time

**Definition**

*Exponential Time* (denoted $\text{EXP}$) is the collection of all problems that have an algorithm which on input $s$ runs in exponential time, i.e., $O(2^{\text{poly}(|s|)})$. 

Example:

- $O(2^n)$
- $O(2^{n \log n})$
- $O(2^{n^3})$
- ...
Exponential Time

Definition

**Exponential Time** (denoted EXP) is the collection of all problems that have an algorithm which on input $s$ runs in exponential time, i.e., $O(2^{\text{poly}(|s|)})$.

Example: $O(2^n)$, $O(2^{n \log n})$, $O(2^{n^3})$, ...
Proposition

\( \text{NP} \subseteq \text{EXP} \).

Proof.

Let \( X \in \text{NP} \) with certifier \( C \). Need to design an exponential time algorithm for \( X \).

1. For every \( t \), with \( |t| \leq p(|s|) \) run \( C(s,t) \); answer “yes” if any one of these calls returns “yes”.

2. The above algorithm correctly solves \( X \) (exercise).

3. Algorithm runs in \( O(q(|s| + |p(s)||2^{p(|s|)}) \), where \( q \) is the running time of \( C \).
Examples

1. **SAT**: try all possible truth assignment to variables.
2. **Independent Set**: try all possible subsets of vertices.
3. **Vertex Cover**: try all possible subsets of vertices.
Is $\textbf{NP}$ efficiently solvable?

We know $\textbf{P} \subseteq \textbf{NP} \subseteq \textbf{EXP}$. 
Is NP efficiently solvable?

We know $P \subseteq NP \subseteq EXP$.

**Big Question**

Is there a problem in NP that does not belong to P? Is $P = NP$?
If $P = NP \ldots$

Or: If pigs could fly then life would be sweet.

1. Many important optimization problems can be solved efficiently.
If $P = NP \ldots$

Or: If pigs could fly then life would be sweet.

1. Many important optimization problems can be solved efficiently.
2. The RSA cryptosystem can be broken.
If $P = NP$.

Or: If pigs could fly then life would be sweet.

1. Many important optimization problems can be solved efficiently.
2. The RSA cryptosystem can be broken.
3. No security on the web.
If $P = NP$ ... 
Or: If pigs could fly then life would be sweet.

1. Many important optimization problems can be solved efficiently.
2. The RSA cryptosystem can be broken.
3. No security on the web.
4. No e-commerce ...
If $P = \text{NP}$...

Or: If pigs could fly then life would be sweet.

1. Many important optimization problems can be solved efficiently.
2. The RSA cryptosystem can be broken.
3. No security on the web.
4. No e-commerce . . .
5. Creativity can be automated! Proofs for mathematical statement can be found by computers automatically (if short ones exist).
If $P = \text{NP}$ this implies that...

(A) Vertex Cover can be solved in polynomial time.
(B) $P = \text{EXP}$.
(C) $\text{EXP} \subseteq P$.
(D) All of the above.
### Status

Relationship between $\text{P}$ and $\text{NP}$ remains one of the most important open problems in mathematics/computer science.

**Consensus:** Most people feel/believe $\text{P} \neq \text{NP}$.

Resolving $\text{P}$ versus $\text{NP}$ is a Clay Millennium Prize Problem. You can win a million dollars in addition to a Turing award and major fame!
“Hardest” Problems

Question

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

Towards a definition

1. Hardest problem must be in NP.
2. 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. $X \in \text{NP}$, and
2. (Hardness) For any $Y \in \text{NP}$, $Y \leq_p X$. 

Chandra Chekuri (UIUC)
Proposition

Suppose $X$ is NP-Complete. Then $X$ can be solved in polynomial time if and only if $P = NP$.

Proof.

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

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

$\Leftarrow$ Since $P = NP$, and $X \in NP$, we have a polynomial time algorithm for $X$. 

Chandra Chekuri (UIUC)
A problem $X$ is said to be **NP-Hard** if

1. **(Hardness)** For any $Y \in \text{NP}$, we have that $Y \leq_P X$.

An **NP-Hard** problem need not be in **NP**!

**Example:** Halting problem is **NP-Hard** (why?) but not **NP-Complete**.
If $X$ is NP-Complete

1. Since we believe $P \neq NP$,
2. and solving $X$ implies $P = NP$.

$X$ is unlikely to be efficiently solvable.
Consequences of proving \textit{NP-Completeness}

If $X$ is \textbf{NP-Complete}

1. Since we believe $P \neq NP$,
2. 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 $X$. 
Consequences of proving \textbf{NP-Completeness}

If $X$ is \textbf{NP-Complete}

1. Since we believe $P \neq NP$,  
2. 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 $X$.  

Consequences of proving **NP-Completeness**

**If** $X$ **is NP-Complete**

1. Since we believe $P \neq NP$,
2. 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 $X$.
(This is proof by mob opinion — take with a grain of salt.)
NP-Complete Problems

Question
Are there any problems that are NP-Complete?

Answer
Yes! Many, many problems are NP-Complete.
Theorem (Cook-Levin)

**SAT is NP-Complete.**
Cook-Levin Theorem

Theorem (Cook-Levin)

\textbf{SAT is NP-Complete.}

Need to show

1. SAT is in NP.
2. every NP problem \( X \) reduces to SAT.

Will see proof in next lecture.

Steve Cook won the Turing award for his theorem.
To prove $X$ is NP-Complete, show

1. Show that $X$ is in NP.
2. Give a polynomial-time reduction from a known NP-Complete problem such as SAT to $X$
To prove \( X \) is \textbf{NP-Complete}, show

1. Show that \( X \) is in \textbf{NP}.
2. Give a polynomial-time reduction \textit{from} a known \textbf{NP-Complete} problem such as \textbf{SAT} \textit{to} \( X \).

\( \text{SAT} \leq_P X \) implies that every \textbf{NP} problem \( Y \leq_P X \). Why?
To prove $X$ is NP-Complete, show

1. Show that $X$ is in NP.
2. Give a polynomial-time reduction from a known NP-Complete problem such as SAT to $X$

$\text{SAT} \leq_p X$ implies that every NP problem $Y \leq_p X$. Why?

Transitivity of reductions:

$Y \leq_p \text{SAT}$ and $\text{SAT} \leq_p X$ and hence $Y \leq_p X$. 
3-SAT is NP-Complete

- **3-SAT** is in **NP**
- **SAT \( \leq_p 3\text{-SAT} \)** as we saw
NP-Completeness via Reductions

1. **SAT** is **NP-Complete** due to Cook-Levin theorem
2. **SAT** $\leq_P 3$-**SAT**
3. **3-SAT** $\leq_P$ **Independent Set**
4. **Independent Set** $\leq_P$ **Vertex Cover**
5. **Independent Set** $\leq_P$ **Clique**
6. **3-SAT** $\leq_P$ **3-Color**
7. **3-SAT** $\leq_P$ **Hamiltonian Cycle**

Hundreds and thousands of different problems from many areas of science and engineering have been shown to be NP-Complete. A surprisingly frequent phenomenon!
SAT is **NP-Complete** due to Cook-Levin theorem

2. **SAT \( \leq_P 3\text{-SAT} \)**

3. **3\text{-SAT} \( \leq_P \) Independent Set**

4. **Independent Set \( \leq_P \) Vertex Cover**

5. **Independent Set \( \leq_P \) Clique**

6. **3\text{-SAT} \( \leq_P \) 3\text{-Color}**

7. **3\text{-SAT} \( \leq_P \) Hamiltonian Cycle**

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

A surprisingly frequent phenomenon!
Part III

Reducing 3-SAT to Independent Set
Problem: Independent Set

**Instance:** A graph $G$, integer $k$.

**Question:** Is there an independent set in $G$ of size $k$?
3SAT \leq_P \text{ Independent Set}

The reduction 3SAT \leq_P \text{ Independent Set}

**Input:** Given a 3CNF formula $\varphi$

**Goal:** Construct a graph $G_\varphi$ and number $k$ such that $G_\varphi$ has an independent set of size $k$ if and only if $\varphi$ is satisfiable.
The reduction \(3\text{SAT} \leq_p \text{Independent Set}\)

**Input:** Given a 3CNF formula \(\varphi\)

**Goal:** Construct a graph \(G_\varphi\) and number \(k\) such that \(G_\varphi\) has an independent set of size \(k\) if and only if \(\varphi\) is satisfiable.

\(G_\varphi\) should be constructable in time polynomial in size of \(\varphi\).
3SAT $\leq_P$ Independent Set

The reduction 3SAT $\leq_P$ Independent Set

**Input:** Given a 3CNF formula $\varphi$

**Goal:** Construct a graph $G_\varphi$ and number $k$ such that $G_\varphi$ has an independent set of size $k$ if and only if $\varphi$ is satisfiable. $G_\varphi$ should be constructable in time polynomial in size of $\varphi$.

**Importance of reduction:** Although 3SAT is much more expressive, it can be reduced to a seemingly specialized Independent Set problem.

**Notice:** We handle only 3CNF formulas – reduction would not work for other kinds of boolean formulas.
Interpreting 3SAT

There are two ways to think about 3SAT:

1. Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.

2. Pick a literal from each clause and find a truth assignment to make all of them true. You will fail if two of the literals you pick are in conflict, i.e., you pick \( x_i \) and \( \neg x_i \).

We will take the second view of 3SAT to construct the reduction.
Interpreting \textbf{3SAT}

There are two ways to think about \textbf{3SAT}

1. Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.

2. Pick a literal from each clause and find a truth assignment to make all of them true. You will fail if two of the literals you pick are in conflict, i.e., you pick \(x_i\) and \(\neg x_i\).
Interpreting 3SAT

There are two ways to think about 3SAT

1. Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.

2. Pick a literal from each clause and find a truth assignment to make all of them true.
Interpreting 3SAT

There are two ways to think about 3SAT

1. Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.

2. Pick a literal from each clause and find a truth assignment to make all of them true. You will fail if two of the literals you pick are in conflict, i.e., you pick $x_i$ and $\neg x_i$

We will take the second view of 3SAT to construct the reduction.
The Reduction

$G_\varphi$ will have one vertex for each literal in a clause.

Figure: Graph for

$\varphi = (\neg x_1 \vee x_2 \vee x_3) \land (x_1 \vee \neg x_2 \vee x_3) \land (\neg x_1 \vee x_2 \vee x_4)$
The Reduction

1. $G_\varphi$ will have one vertex for each literal in a clause
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true

Figure: Graph for

$\varphi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_4)$
The Reduction

1. $G_\varphi$ will have one vertex for each literal in a clause.
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true.

Figure: Graph for

$$\varphi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_4)$$
The Reduction

1. $G_\varphi$ will have one vertex for each literal in a clause
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true
3. Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict

Figure: Graph for

$$\varphi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_4)$$

Values:
- $x_1 = 0$
- $x_2 = 1$
- $x_3 = 1$
- $x_4 = 0$
The Reduction

1. $G_\varphi$ will have one vertex for each literal in a clause
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true
3. Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict
4. Take $k$ to be the number of clauses

Figure: Graph for

$$\varphi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_4)$$
Correctness

Proposition

\( \varphi \) is satisfiable iff \( G_{\varphi} \) has an independent set of size \( k \) (\( = \) number of clauses in \( \varphi \)).

Proof.

\( \Rightarrow \) Let \( a \) be the truth assignment satisfying \( \varphi \)
Correctness

Proposition

ϕ is satisfiable iff \( G_\varphi \) has an independent set of size \( k \) (\( = \) number of clauses in \( \varphi \)).

Proof.

\( \Rightarrow \) Let \( a \) be the truth assignment satisfying \( \varphi \)

1. Pick one of the vertices, corresponding to true literals under \( a \), from each triangle. This is an independent set of the appropriate size. Why?
Correctness (contd)

Proposition

ϕ is satisfiable iff \( G_\varphi \) has an independent set of size \( k \) (\( = \) number of clauses in \( \varphi \)).

Proof.

\( \Leftarrow \) Let \( S \) be an independent set of size \( k \)

1. \( S \) must contain exactly one vertex from each clause
2. \( S \) cannot contain vertices labeled by conflicting literals
3. Thus, it is possible to obtain a truth assignment that makes in the literals in \( S \) true; such an assignment satisfies one literal in every clause