Last Lecture!

UMB CS 622

NP

Wednesday May 8, 2024

Who doesn't like niche NP jokes?

AN ENGINEER, A PHYSICIST, AND A MATHEMATICIAN ARE ROOMMATES AND ARE MOVING TO A NEW PLACE.

AS THE MOVER PULLS UP, THE MATHEMATICIAN WORRIES THERE ISN'T ENOUGH ROOM.

THE MOVER REASSURES THEM I'VE BEEN AT THIS 30 YEARS, I CAN LOOK AT ANY AMOUNT OF STUFF AND INSTANTLY TELL YA IF IT CAN FIT IN THE MOVING BINS.

THE ENGINEER SAYS... IT'S OBVIOUS. IT CAN FIT. ANYTHING THAT DOESN'T GO IN THE BINS CAN BE TAPE TO THE ROOF.

THE PHYSICIST SAYS... IT'S OBVIOUS. IT CAN FIT. IF IT WERE THE DENSITY OF A NEUTRON STAR, OUR STUFF WOULD BE THE SIZE OF A BASEBALL.

THE MATHEMATICIAN SAYS... PLEASE DON'T HACK MY EMAIL.
Announcements

• HW 11 extended
  • Due Wed 5/8 12pm noon
  • Due Fri 5/10 12pm noon

• HW 12 out
  • Due Wed 5/15 12pm noon (no late submissions accepted)

Lecture Participation

Q1 Which of the following are ways to show that a language is in NP?
1 Point
(select all that apply)
- create a deterministic poly time decider
- create a non-deterministic poly time decider
- create a deterministic poly time verifier
- create a non-deterministic poly time verifier
Last Time: Poly Time Complexity Class (P)

\[ P = \bigcup_{k} \text{TIME}(n^k). \]

- Corresponds to “realistically” solvable problems:
  - Problems in P
    - = “solvable” or “tractable”
  - Problems outside P
    - = “unsolvable” or “intractable”
Last Time: 3 Problems in $\mathbf{P}$

- **A Graph Problem:**
  \[
  \text{PATH} = \{ (G, s, t) \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t \}
  \]

- **A Number Problem:**
  \[
  \text{RELPRIME} = \{ (x, y) \mid x \text{ and } y \text{ are relatively prime} \}
  \]

- **A CFL Problem:**
  Every context-free language is a member of $\mathbf{P}$
Search vs Verification

• **Search** problems are often **unsolvable**
• But, **verification** of a search result is usually **solvable**

**Examples**

• **Factoring**
  • **Unsolvable:** Find factors of 8633
  • Must “try all” possibilities
  • **Solvable:** Verify 89 and 97 are factors of 8633
  • Just do multiplication

• **Passwords**
  • **Unsolvable:** Find my umb.edu password
  • **Solvable:** Verify whether my umb.edu password is ...
  • “correct horse battery staple”
The **PATH** Problem

\[ \text{PATH} = \{ (G, s, t) | G \text{ is a directed graph that has a directed path from } s \text{ to } t \} \]

- It’s a **search** problem:
  - **Exponential time** (brute force) algorithm \( (n^n) \):
    - Check all \( n^n \) possible paths and see if any connects \( s \) and \( t \)
  - **Polynomial time** algorithm:
    - Do a breadth-first search (roughly), marking “seen” nodes as we go \( (n = \# \text{ nodes}) \)

**Proof**

A polynomial time algorithm \( M \) for **PATH** operates as follows.

\[
\begin{align*}
M &= \text{“On input } (G, s, t), \text{ where } G \text{ is a directed graph with nodes } s \text{ and } t:\n1. & \quad \text{Place a mark on node } s.
2. & \quad \text{Repeat the following until no additional nodes are marked:}
3. & \quad \text{Scan all the edges of } G. \text{ If an edge } (a, b) \text{ is found going from a marked node } a \text{ to an unmarked node } b, \text{ mark node } b.
4. & \quad \text{If } t \text{ is marked, accept. Otherwise, reject.”}
\end{align*}
\]

\( O(n^3) \)
Verifying a \textit{PATH}

\[ PATH = \{ (G, s, t) \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t \} \]

The verification problem:

- \textbf{Given} some path \( p \) in \( G \), check that it is a path from \( s \) to \( t \)

- Let \( m = \) length of longest possible path = \# edges

\textbf{Verifier} \( V \) = On input \( <G, s, t, p> \), where \( p \) is some set of edges:

1. Check some edge in \( p \) has “from” node \( s \); mark and set it as “current” edge
   - Max steps = \( O(m) \)

2. \textbf{Loop:} While there remains unmarked edges in \( p \):
   1. Find the “next” edge in \( p \), whose “from” node is the “to” node of “current” edge
   2. If found, then mark that edge and set it as “current”; else reject
      - Each loop iteration: \( O(m) \)
      - \# loops: \( O(m) \)
      - Total looping time = \( O(m^2) \)

3. Check “current” edge has “to” node \( t \); if yes accept, else reject
   - Total time = \( O(m) + O(m^2) = O(m^2) \) = polynomial in \( m \)

\textbf{NOTE:} extra argument \( p \), “Verifying” an answer requires having a potential answer to check!

\textbf{PATH can be verified in polynomial time}
Verifiers, Formally

\[ \text{PATH} = \{ \langle G, s, t \rangle | G \text{ is a directed graph that has a directed path from } s \text{ to } t \} \]

A verifier for a language \( A \) is an algorithm \( V \), where
\[ A = \{ w | V \text{ accepts } \langle w, c \rangle \text{ for some string } c \} \]

We measure the time of a verifier only in terms of the length of \( w \), so a polynomial time verifier runs in polynomial time in the length of \( w \). A language \( A \) is polynomially verifiable if it has a polynomial time verifier.

- **NOTE:** a certificate \( c \) must be at most length \( n^k \), where \( n = \text{length of } w \)
- Why? Because it takes time \( n^k \) to read it

So \( \text{PATH} \) is polynomially verifiable
The *HAMPATH* Problem

- A Hamiltonian path goes through every node in the graph

- The **Search** problem:
  - **Exponential time** (brute force) algorithm:
    - Check all possible paths and see if any connect \( s \) and \( t \) using all nodes
  - **Polynomial time algorithm:** ???
    - We don’t know if there is one!!!

- The **Verification** problem:
  - Still \( O(m^2)! \) (same verifier for *PATH*)
  - *HAMPATH* is polynomially verifiable, but not polynomially decidable
The class **NP**

**DEFINITION**

**NP** is the class of languages that have polynomial time verifiers.

- *PATH is in NP*, and *P*
- *HAMPATH is in NP*, but it’s **unknown** whether it’s in *P***
**NP** = Nondeterministic polynomial time

**Theorem**

A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.

⇒ If a language is in NP, then it has a non-deterministic poly time decider

- **We know:** If a lang L is in NP, then it has a poly time verifier V
- **Need to:** create NTM deciding L:

  On input \( w = \)
  - Nondeterministically run \( V \) with \( w \) and all possible poly length certificates \( c \)

⇐ If a language has a non-deterministic poly time decider, then it is in NP

- **We know:** \( L \) has NTM decider \( N \),
- **Need to:** show \( L \) is in NP, i.e., create polytime verifier \( V \):

  On input \( \langle w, c \rangle = \)
  - Potentially exponential slowdown?
  - Convert \( N \) to deterministic TM, and run it on \( w \), but take only one computation path
  - Let certificate \( c \) dictate which computation path to follow

**NOTE:** a verifier cert is usually a potential “answer”, but does not have to be (like here)

Certificate \( c \) specifies a path

Deterministic (verifier) TMs cannot “call” non-deterministic TMs

Because Converting NTM to deterministic is exponentially slower!
\[ \text{NP} \]

\[ \text{NTIME}(t(n)) = \{ L \mid L \text{ is a language decided by an } O(t(n)) \text{ time} \]
\[ \text{nondeterministic Turing machine} \} \]

\[ \text{NP} = \bigcup_k \text{NTIME}(n^k) \]

\[ \text{NP} = \text{Nondeterministic polynomial time} \]
**NP vs P**

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

\[ P = \bigcup_k \text{TIME}(n^k). \]

**P = Deterministic polynomial time**

\[ \text{NTIME}(t(n)) = \{ L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine} \}. \]

NP = \bigcup_k \text{NTIME}(n^k)

**NP = Nondeterministic polynomial time**

Also, **NP = Deterministic polynomial time verification**
More \textbf{NP} Problems

- $\text{CLIQUE} = \{\langle G, k \rangle | G \text{ is an undirected graph with a } k\text{-clique}\}$
  - A clique is a subgraph where every two nodes are connected
  - A $k$-clique contains $k$ nodes

- $\text{SUBSET-SUM} = \{\langle S, t \rangle | S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \sum y_i = t\}$
Theorem: **CLIQUE** is in NP

**PROOF IDEA** The clique is the certificate.

**PROOF** The following is a verifier $V$ for **CLIQUE**.

$V =$ “On input $(G, k, c)$:

1. Test whether $c$ is a subgraph with $k$ nodes in $G$.
2. Test whether $G$ contains all edges connecting nodes in $c$.
3. If both pass, accept; otherwise, reject.”

Let $n = \# \text{ nodes in } G$

$c$ is at most $n$

For each: node in $c$, check whether it’s in $G$: $O(n)$

For each: pair of nodes in $c$, check whether there’s an edge in $G$: $O(n^2)$

A **verifier** for a language $A$ is an algorithm $V$, where $A = \{w | V \text{ accepts } (w, c) \text{ for some string } c\}$.

We measure the time of a verifier only in terms of the length of $w$, so a **polynomial time verifier** runs in polynomial time in the length of $w$. A language $A$ is **polynomially verifiable** if it has a polynomial time verifier.

**NP** is the class of languages that have polynomial time verifiers.
Proof 2: **CLIQUE** is in NP

**CLIQUE** = \{\langle G, k \rangle | G \text{ is an undirected graph with a } k\text{-clique}\}

\[ N = \text{"On input } \langle G, k \rangle \text{, where } G \text{ is a graph:} \]
1. Nondeterministically select a subset \( c \) of \( k \) nodes of \( G \).
2. Test whether \( G \) contains all edges connecting nodes in \( c \).
3. If yes, accept; otherwise, reject."

Checking whether a subgraph is clique: \( O(n^2) \)

To prove a lang \( L \) is in NP, create either a:
1. **Deterministic** poly time verifier
2. **Nondeterministic** poly time decider

How to prove a language is in \( \text{NP} \):
Proof technique #2: create an NTM

A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.

Don’t forget to count the steps
More **NP** Problems

- **CLIQUE** = \{\langle G, k\rangle | G \text{ is an undirected graph with a } k\text{-clique}\}
  - A clique is a subgraph where every two nodes are connected
  - A \(k\)-clique contains \(k\) nodes

- **SUBSET-SUM** = \{\langle S, t\rangle | S = \{x_1, \ldots, x_k\}, and for some \(y_1, \ldots, y_l \subseteq \{x_1, \ldots, x_k\}\), we have \(\Sigma y_i = t\)\}
  - Some subset of a set of numbers \(S\) must sum to some total \(t\)
  - e.g., \(\{4, 11, 16, 21, 27\}, 25\) \(\in\) **SUBSET-SUM**
Theorem: \( 	ext{SUBSET-SUM} \) is in NP

\[
\text{SUBSET-SUM} = \{ \langle S, t \rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t \}
\]

**Proof Idea:** The subset is the certificate.

To prove a language is in NP, create either:
1. Deterministic poly time verifier
2. Nondeterministic poly time decider

**Proof**

The following is a verifier \( V \) for \( 	ext{SUBSET-SUM} \).

\( V = \) “On input \( \langle S, t \rangle, c \):

1. Test whether \( c \) is a collection of numbers that sum to \( t \).
2. Test whether \( S \) contains all the numbers in \( c \).
3. If both pass, accept; otherwise, reject.”

Don’t forget to compute run time! Does this run in poly time?
Proof 2: \( \text{SUBSET-SUM} \) is in NP

\[ \text{SUBSET-SUM} = \{ \langle S, t \rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t \} \]

To prove a lang is in \( \text{NP} \), create either:
1. Deterministic poly time verifier
2. Nondeterministic poly time decider

Don’t forget to compute run time! Does this run in poly time?

**ALTERNATIVE PROOF** We can also prove this theorem by giving a nondeterministic polynomial time Turing machine for \( \text{SUBSET-SUM} \) as follows.

\( N = \) “On input \( \langle S, t \rangle \):
1. Nondeterministically select a subset \( c \) of the numbers in \( S \).
2. Test whether \( c \) is a collection of numbers that sum to \( t \).
3. If the test passes, accept; otherwise, reject.”

Nondeterministically runs the verifier on each possible subset in parallel
COMPOSITES = \{ x \mid x = pq, \text{ for integers } p, q > 1 \}

• A composite number is not prime

• COMPOSITES is polynomially verifiable
  • i.e., it’s in NP
  • i.e., factorability is in NP

• A certificate could be:
  • Some factor that is not 1

• Checking existence of factors (or not, i.e., testing primality) ...
  • ... is also poly time
  • But only discovered recently (2002)!
HW Question: Does $P = NP$?

How do you prove an algorithm doesn’t have a poly time algorithm? (in general it’s hard to prove that something doesn’t exist)
Implications if $P = NP$

- Problems with “brute force” (“try all”) solutions now have efficient solutions

- I.e., “unsolvable” problems are “solvable”

- **BAD:**
  - Cryptography needs unsolvable problems
  - Near perfect AI learning, recognition

- **GOOD:** Optimization problems are solved
  - Optimal resource allocation could fix all the world’s (food, energy, space ...) problems?
Progress on whether $P = NP$?

- Some, but still not close

- One important concept discovered:
  - NP-Completeness
**NP-Completeness**

**Definition**
A language $B$ is **NP-complete** if it satisfies two conditions:

1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$.

• How does this help the $P = NP$ problem?

**Theorem**
If $B$ is NP-complete and $B \in P$, then $P = NP$. 

**Must look at all langs, can’t just look at a single lang**

**easy**

**hard??**

**What’s this?**
**Flashback: Mapping Reducibility**

Language $A$ is **mapping reducible** to language $B$, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$  

The function $f$ is called the **reduction** from $A$ to $B$.

**IMPORTANT:** “if and only if” ...

To show mapping reducibility:
1. create **computable fn**
2. and then show **forward direction**
3. and **reverse direction** (or contrapositive of forward direction)

$A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ accepts } w \}$  

$HALT_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \}$

... means $\overline{A} \leq_m \overline{B}$

A function $f : \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.
Polynomial Time Mapping Reducibility

Language \( A \) is **mapping reducible** to language \( B \) if there is a computable function \( f : \Sigma^* \rightarrow \Sigma^* \),

\[ w \in A \iff f(w) \in B. \]

The function \( f \) is called the **reduction** from \( A \) to \( B \).

Language \( A \) is **polynomial time mapping reducible**, or simply **polynomial time reducible**, to language \( B \), written \( A \leq_P B \), if a polynomial time computable function \( f : \Sigma^* \rightarrow \Sigma^* \) exists, where for every \( w \),

\[ w \in A \iff f(w) \in B. \]

The function \( f \) is called the **polynomial time reduction** of \( A \) to \( B \).

To show polynomial time mapping reducibility:
1. create computable fn
2. show computable fn runs in poly time
3. then show forward direction
4. and show reverse direction (or contrapositive of reverse direction)

Don’t forget: “if and only if” ...

A function \( f : \Sigma^* \rightarrow \Sigma^* \) is a **computable function** if some Turing machine \( M \), on every input \( w \), halts with just \( f(w) \) on its tape.
Flashback: If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.

**Proof.** We let $M$ be the decider for $B$ and $f$ be the reduction from $A$ to $B$. We describe a decider $N$ for $A$ as follows.

$$N = \text{“On input } w:\text{ 1. Compute } f(w).\text{ 2. Run } M \text{ on input } f(w) \text{ and output whatever } M \text{ outputs.”}$$

Language $A$ is mapping reducible to language $B$, written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$  

The function $f$ is called the reduction from $A$ to $B$. 

This proof only works because of the if-and-only-if requirement.
Thm: If $A \leq_m B$ and $B \in \mathsf{P}$ is decidable, then $A \in \mathsf{P}$ is decidable.

**PROOF** We let $M$ be the decider for $B$ and $f$ be the reduction from $A$ to $B$. We describe a decider $N$ for $A$ as follows.

$N = \text{"On input } w:\)
1. Compute $f(w)$.
2. Run $M$ on input $f(w)$ and output whatever $M$ outputs.”

Language $A$ is *mapping reducible* to language $B$, written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$  

The function $f$ is called the *reduction* from $A$ to $B$. 
Thm: If $A \leq_m B$ and $B \in \mathbb{P}$ is decidable, then $A \in \mathbb{P}$ is decidable.

**Proof**: We let $M$ be the decider for $B$ and $f$ be the reduction from $A$ to $B$. We describe a decider $N$ for $A$ as follows.

$N = \text{"On input } w:\"
1. Compute $f(w)$.
2. Run $M$ on input $f(w)$ and output whatever $M$ outputs."

Language $A$ is mapping reducible to language $B$, written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

\[ w \in A \iff f(w) \in B. \]

The function $f$ is called the reduction from $A$ to $B$. 
To prove $P = \text{NP}$, must show:

1. every language in $P$ is in $\text{NP}$
   - Trivially true (why?)

2. every language in $\text{NP}$ is in $P$
   - Given a language $A \in \text{NP}$ ...
   - ... can poly time mapping reduce $A$ to $B$
     - because $B$ is $\text{NP}$-Complete
   - Then $A$ also $\in P$ ...
     - Because $A \leq_P B$ and $B \in P$, then $A \in P$

Next: How to do poly time mapping reducibility

Thus, if a language $B$ is $\text{NP}$-complete and in $P$, then $P = \text{NP}$
Next Time: 3SAT is polynomial time reducible to CLIQUE.
Lecture Participation 5/8

On gradescope