BU CS 332 – Theory of Computation

Lecture 20:
  • More on NP

Reading:
  Sipser Ch 7.3-7.5

Mark Bun
April 13, 2020
Goals of complexity theory

Ultimate goal: Classify problems according to their feasibility and inherent computational difficulty

\( P \approx \) Decision problems which can be solved efficiently

Can we exhibit general classes of problems which are either in \( P \) or provably not in \( P \)?

Some problems provably require exponential time! (Chapter 9)

NP: A fundamental and practically important class of problems which have defied classification, but nevertheless exhibits important structure (NP completeness)
Nondeterministic Time and NP
Nondeterministic time

Let $f : \mathbb{N} \to \mathbb{N}$

A NTM $M$ runs in time $f(n)$ if on every input $w \in \Sigma^n$, $M$ halts on $w$ within at most $f(n)$ steps on every computational branch.
Deterministic vs. nondeterministic time

Deterministic vs. Nondeterministic

- Deterministic
  - accept or reject
  - \( t(n) \)

- Nondeterministic
  - reject
  - accept
  - reject

4/13/2020
CS332 - Theory of Computation
5
Deterministic vs. nondeterministic time

Theorem: Let $t(n) \geq n$ be a function. Every NTM running in time $t(n)$ has an equivalent single-tape TM running in time $2^{O(t(n))}$

Proof: Simulate NTM by 3-tape TM

Input $w$ to $N$ (read-only)

Simulation tape (run $N$ on $w$ using nondeterministic choices from tape 3)

Address in computation tree
Deterministic vs. nondeterministic time

**Theorem:** Let $t(n) \geq n$ be a function. Every NTM running in time $t(n)$ has an equivalent single-tape TM running in time $2^{O(t(n))}$

**Proof:** Simulate NTM by 3-tape TM

- # leaves:
- # nodes:

Running time:

To simulate one root-to-node path:

Total time:
Deterministic vs. nondeterministic time

Theorem: Let $t(n) \geq n$ be a function. Every NTM running in time $t(n)$ has an equivalent single-tape TM running in time $2^{O(t(n))}$

Proof: Simulate NTM by 3-tape TM in time $2^{O(t(n))}$

We know that a 3-tape TM can be simulated by a single-tape TM with quadratic overhead, hence we get running time

\[
(2^{O(t(n))})^2 = 2^{2\cdot O(t(n))} = 2^{O(t(n))}
\]
Difference in time complexity

Extended Church-Turing Thesis:
At most polynomial difference in running time between all (reasonable) deterministic models

At most exponential difference in running time between deterministic and nondeterministic models
Nondeterministic time

Let $f : \mathbb{N} \to \mathbb{N}$

A NTM $M$ runs in time $f(n)$ if on every input $w \in \Sigma^n$, $M$ halts on $w$ within at most $f(n)$ steps on every computational branch.

$\text{NTIME}(f(n))$ is a class (i.e., set) of languages:

A language $A \in \text{NTIME}(f(n))$ if there exists an NTM $M$ that

1) Decides $A$, and
2) Runs in time $O(f(n))$
NTIME explicitly

A language $A \in \text{NTIME}(f(n))$ if there exists an NTM $M$ such that, on every input $w \in \Sigma^*$

1. Every computational branch of $M$ halts in either the accept or reject state within $f(|w|)$ steps

2. $w \in A$ iff there exists an accepting computational branch of $M$ on input $w$

3. $w \notin A$ iff every computational branch of $M$ rejects on input $w$ (or dies with no applicable transitions)
Complexity class NP

**Definition:** NP is the class of languages decidable in polynomial time on a nondeterministic TM

\[
NP = \bigcup_{k=1}^{\infty} \text{NTIME}(n^k)
\]
Hamiltonian Path

\[ HAMPATH = \{ \langle G, s, t \rangle \mid G \text{ is a directed graph and there is a path from } s \text{ to } t \text{ that passes through every vertex exactly once} \} \]
**HAMPATH ∈ NP**

The following nondeterministic algorithm accepts in time $O(n^{1.5})$, where $n = |\langle G, s, t \rangle|$, adjacency matrix encoding

On input $\langle G, s, t \rangle$: (Vertices of $G$ are numbers 1, ..., $k$)

1. **Nondeterministically** guess a sequence $c_1, c_2, ..., c_k$ of numbers 1, ..., $k$

2. Check that $c_1, c_2, ..., c_k$ is a permutation: Every number 1, ..., $k$ appears exactly once

3. Check that $c_1 = s$, $c_k = t$, and there is an edge from every $c_i$ to $c_{i+1}$

4. **Accept** if all checks pass, otherwise, **reject**.
An alternative characterization of \textbf{NP}

“Languages with polynomial-time verifiers”

A verifier for a language \( L \) is a deterministic algorithm \( V \) such that \( w \in L \) iff there exists a string \( c \) such that \( V(\langle w, c \rangle) \) accepts

Running time of a verifier is only measured in terms of \(|w|\)

\( V \) is a polynomial-time verifier if it runs in time polynomial in \(|w|\) on every input \( \langle w, c \rangle \)

(Without loss of generality, \(|c|\) is polynomial in \(|w|\), i.e.,
\(|c| = O(|w|^k)\) for some constant \(k\))
**HAMPATH** has a polynomial-time verifier

Certificate $c$:

Verifier $V$:

On input $\langle G, s, t; c \rangle$: (Vertices of $G$ are numbers 1, $\ldots$, $k$)

1. Check that $c_1, c_2, \ldots, c_k$ is a permutation: Every number 1, $\ldots$, $k$ appears exactly once

2. Check that $c_1 = s$, $c_k = t$, and there is an edge from every $c_i$ to $c_{i+1}$

3. **Accept** if all checks pass, otherwise, **reject**.
NP is the class of languages with polynomial-time verifiers

**Theorem:** A language $L \in \text{NP}$ iff there is a polynomial-time verifier for $L$

**Proof:**
Examples of **NP** languages: SAT

“Is there an assignment to the variables in a logical formula that make it evaluate to true?”

- **Boolean variable**: Variable that can take on the value true/false (encoded as 0/1)
- **Boolean operations**: $\land$ (AND), $\lor$ (OR), $\neg$ (NOT)
- **Boolean formula**: Expression made of Boolean variables and operations. *Ex*: $(x_1 \lor \overline{x_2}) \land x_3$
- An **assignment** of 0s and 1s to the variables satisfies a formula $\varphi$ if it makes the formula evaluate to 1
- A formula $\varphi$ is **satisfiable** if there exists an assignment that satisfies it
Examples of **NP** languages: **SAT**

**Example 1:** \((x_1 \lor \overline{x_2}) \land x_3\)  
Satisfiable?

**Example 2:** \((x_1 \lor x_2) \land (x_1 \lor \overline{x_2}) \land \overline{x_2}\)  
Satisfiable?

\[
SAT = \{\langle \varphi \rangle | \varphi \text{ is a satisfiable formula}\}
\]

**Claim:** \(SAT \in NP\)
Examples of NP languages: TSP

“Given a list of cities and distances between them, is there a ‘short’ tour of all of the cities?”

More precisely: Given

• A number of cities $m$

• A function $D: \{1, \ldots, m\}^2 \rightarrow \mathbb{N}$ giving the distance between each pair of cities

• A distance bound $B$

\[
TSP = \{(m, D, B) | \exists \text{ a tour visiting every city with length } \leq B\}
\]
**P vs. NP**

**Question:** Does $P = NP$?

Philosophically: Can every problem with an efficiently verifiable solution also be solved efficiently?

EXCEPT the table and schematic diagrams:

**Millennium Problems**

- **Yang-Mills and Mass Gap**
  - Experimental and computer simulations suggest the existence of a “mass gap” in the solution to the quantum version of the Yang-Mills equations. But no proof of this property is known.

- **Riemann Hypothesis**
  - The prime number theorem determines the average distribution of the primes. The Riemann hypothesis tells us about the deviation from the average. Proposed in Riemann’s 1859 paper, it asserts that all the non-trivial zeros of the zeta function have real part equal to 1/2.

- **P vs NP Problem**
  - It is easy to check that a solution to a problem is correct. It is also easy to solve the problem! This is the essence of the P vs NP question. Typical of the NP problems is that of the Traveling Salesman Problem: given $n$ cities and the distance between each pair, can you find an itinerary with total distance less than a given threshold? If you give me a route, I can easily check that it is correct. But I cannot so easily find a route.

- **Navier–Stokes Equation**
  - This is the equation which governs the flow of fluids such as water and air. However, there is no proof for the most basic questions regarding solutions to the Navier–Stokes equation.

- **Hodge Conjecture**
  - The answer to this conjecture determines how much of the topology of the solution set of a system of algebraic equations can be defined in terms of further algebraic equations. The Hodge conjecture is known in certain special cases, e.g., when the solution set has dimension less than four but in dimension four it fails in general.

- **Poincaré Conjecture**
  - In 1904 the French mathematician Henri Poincaré asked if the three-dimensional sphere is characterized as the unique simply connected three manifold. This question, the Poincaré conjecture, was a special case of Thurston’s geometrization conjecture. Perelman’s proof tells us that every three-manifold is either a finite union of standard pieces, each with one of eight well-understood geometries.

- **Birch and Swinnerton-Dyer Conjecture**
  - Proposed by Birch and Swinnerton-Dyer in 1960s, this conjecture relates the number of points on an elliptic curve modulo to the rank of the group of rational points. Elliptic curves, defined by cubic equations in two variables, are fundamental mathematical objects that arise in many areas. Wiles’ proof of the Fermat Conjecture, factorization of numbers into primes, and cryptography, to name a few.
A world where $P = NP$

- Many important decision problems can be solved in polynomial time ($HAMPATH, SAT, TSP$, etc.)

- Many search problems can be solved in polynomial time (e.g., given a natural number, find a prime factorization)

- Many optimization problems can be solved in polynomial time (e.g., find the lowest energy conformation of a protein)
A world where $P = NP$

- Secure cryptography becomes impossible
  An NP search problem: Given a ciphertext $C$, find a plaintext $m$ and encryption key $k$ that would encrypt to $C$

- AI / machine learning become easy: Identifying a consistent classification rule is an NP search problem

- Finding mathematical proofs becomes easy: NP search problem: Given a mathematical statement $S$ and length bound $k$, is there a proof of $S$ with length at most $k$?

General consensus: $P \neq NP$
NP Completeness
What about a world where $P \neq NP$

Believe this to be true, but very far from proving it

$P \neq NP$ implies that there is a problem in $NP$ which cannot be solved in polynomial time, but it might not be a useful one

**Question:** What would $P \neq NP$ allow us to conclude about problems we care about?

**Idea:** Identify the “hardest” problems in $NP$

Find $L \in NP$ such that $L \in P \iff P = NP$
Recall: Mapping reducibility

Definition:
A function \( f : \Sigma^* \to \Sigma^* \) is computable if there is a TM \( M \) which, given as input any \( w \in \Sigma^* \), halts with only \( f(w) \) on its tape.

Definition:
Language \( A \) is mapping reducible to language \( B \), written \( A \leq_m B \) if there is a computable function \( f : \Sigma^* \to \Sigma^* \) such that for all strings \( w \in \Sigma^* \), we have \( w \in A \iff f(w) \in B \).
Polynomial-time reducibility

Definition:
A function $f : \Sigma^* \to \Sigma^*$ is polynomial-time computable if there is a polynomial-time TM $M$ which, given as input any $w \in \Sigma^*$, halts with only $f(w)$ on its tape.

Definition:
Language $A$ is polynomial-time mapping reducible to language $B$, written
\[ A \leq_p B \]
if there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that for all strings $w \in \Sigma^*$, we have $w \in A \iff f(w) \in B$
Implications of poly-time reducibility

Theorem: If $A \leq_p B$ and $B \in P$, then $A \in P$

Proof: