BU CS 332 – Theory of Computation

Lecture 22:

- Nondeterministic time, NP
- NP-completeness

Reading:

Sipser Ch 7.3-7.4

Mark Bun

April 14, 2021

Big-Oh, formally

$f(n) = O(g(n))$ means:

There exist constants $c > 0, n_0 > 0$ such that

$f(n) \leq cg(n)$ for every $n \geq n_0$

Example: Show that $7n^2 \cdot 3^n = 2^{O(n)}$

Big-Oh, formally

$f(n) = O(g(n))$ means:

There exist constants $c > 0, n_0 > 0$ such that

$f(n) \leq cg(n)$ for every $n \geq n_0$

Example: Show that $7n^2 \cdot 3^n = 2^{O(n)}$

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

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

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

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

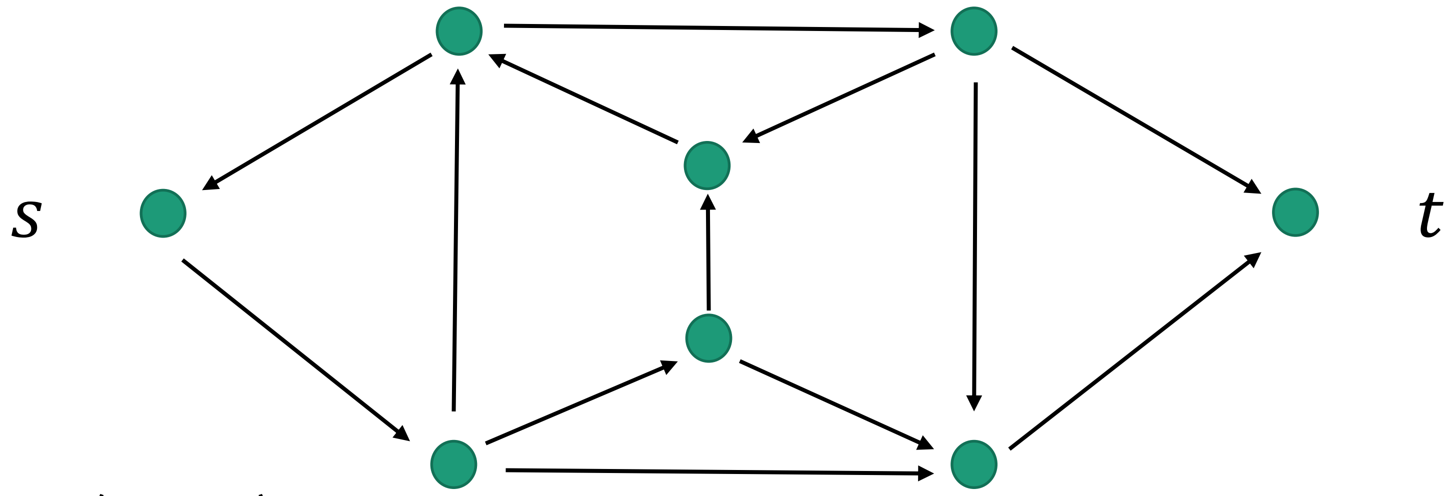
Complexity class NP

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

$$\mathrm{NP} = \bigcup_{k=1}^{\infty} \mathrm{NTIME}(n^k)$$

$HAMPATH \in \mathrm{NP}$

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



On input $\langle G, s, t\rangle$:

1. **Nondeterministically** guess a sequence of vertices
2. Check that the guess forms a Hamiltonian path from s to t

An alternative characterization of $\mathbf{NP}$

"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 $1, \dots, k$)

1. Check that $c_1, c_2, \dots, c_k$ is a permutation: Every number $1, \dots, k$ 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 $L \in \mathrm{NP}$ iff there is a polynomial-time verifier for L

Proof: $\Leftarrow$ Let L have a poly-time verifier $V(\langle w, c \rangle)$

Idea: Design NTM N for L that nondeterministically guesses a certificate

NP is the class of languages with polynomial-time verifiers

$\Rightarrow$ Let L be decided by an NTM N making up to b nondeterministic choices in each step

Idea: Design verifier V for L where certificate is sequence of “good” nondeterministic choices

Alternative proof of $\mathrm{NP} \subseteq \mathrm{EXP}$

One can prove $\mathrm{NP} \subseteq \mathrm{EXP}$ as follows. Let V be a verifier for a language L running in time $T(n)$. We can construct a $2^{O(T(n))}$ time algorithm for L as follows.

a) On input $\langle w, c \rangle$, run V on $\langle w, c \rangle$ and output the result

b) On input w, run V on all possible $\langle w, c \rangle$, where c is a certificate. Accept if any run accepts.

c) On input w, run V on all possible $\langle w, c \rangle$, where c is a certificate of length at most $T(|w|)$. Accept if any run accepts.

d) On input w, run V on all possible $\langle x, c \rangle$, where x is a string of length $|w|$ and c is a certificate of length at most $T(|w|)$. Accept if any run accepts.

WARNING: Don’t mix-and-match the NTM and verifier interpretations of $\mathbf{NP}$

To show a language L is in NP, do exactly one:

1) Exhibit a poly-time NTM for L

   N = “On input x:

   <Do some nondeterministic stuff>…”

OR

2) Exhibit a poly-time (deterministic) verifier for L

   V = “On input x and certificate c:

   <Do some deterministic stuff>…”

Examples of $\mathbf{NP}$ 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: $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT)
- Boolean formula: Expression made of Boolean variables and operations. Ex: $(x_1 \vee \overline{x_2}) \wedge x_3$
- An assignment of 0s and 1s to the variables satisfies a formula φ if it makes the formula evaluate to 1
- A formula φ is satisfiable if there exists an assignment that satisfies it

Examples of NP languages: SAT

Ex: $(x_1 \vee \overline{x_2}) \wedge x_3$ Satisfiable?

Ex: $(x_1 \vee x_2) \wedge \overline{x_1} \wedge \overline{x_2}$ Satisfiable?

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

Claim: $SAT \in \mathrm{NP}$

Examples of $\mathbf{NP}$ languages: Traveling Salesperson

"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 $D: \{1, \dots, m\}^2 \to \mathbb{N}$ giving the distance between each pair of cities
- A distance bound B

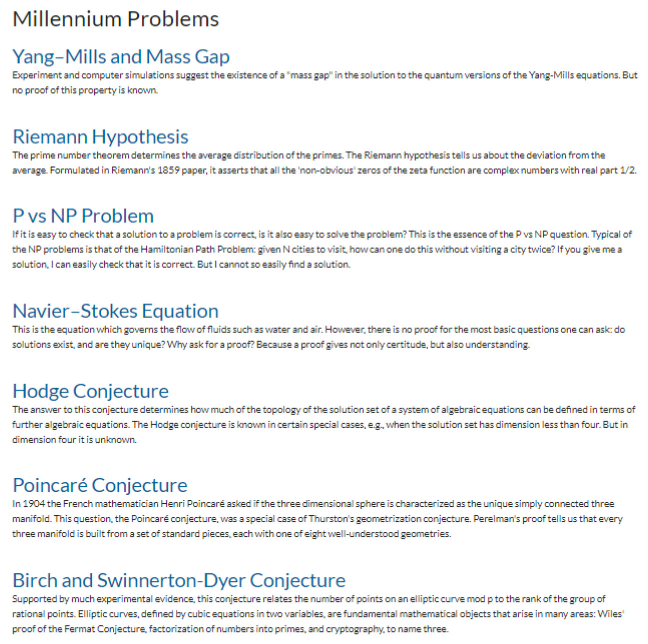
$$TSP = \{\langle m, D, B\rangle | \exists \text{ a tour visiting every city with length} \leq B\}$$

P vs. NP

Question: Does $\mathrm{P} = \mathrm{NP}$?

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

A central problem in mathematics and computer science



Millennium Problems

Yang–Mills and Mass Gap

Experiment and computer simulations suggest the existence of a "mass gap" in the solution to the quantum versions 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. Formulated in Riemann's 1859 paper, it asserts that all the 'non-obvious' zeros of the zeta function are complex numbers with real part 1/2.

P vs NP Problem

If it is easy to check that a solution to a problem is correct, is it 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 Hamiltonian Path Problem: given N cities to visit, how can one do this without visiting a city twice? If you give me a solution, I can easily check that it is correct. But I cannot so easily find a solution.

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 one can ask: do solutions exist, and are they unique? Why ask for a proof? Because a proof gives not only certitude, but also understanding.

Hodge Conjecture

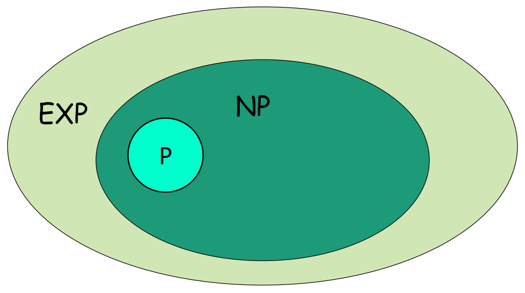
The answer to this conjecture determines how much of the topology of the solution set of a system of algebraic equations can be defined in terms of further algebraic equations. The Hodge conjecture is known in certain special cases, e.g., when the solution set has dimension less than four. But in dimension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

Poincaré Conjecture

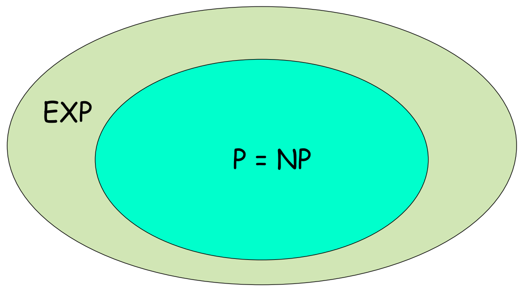
In 1904 the French mathematician Henri Poincaré asked if the three dimensional sphere is characterized as the unique simply connected three manifold. This question, the Poincaré conjecture, was a special case of Thurston's geometrization conjecture. Perelman's proof tells us that every three manifold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

Birch and Swinnerton-Dyer Conjecture

Supported by much experimental evidence, this conjecture relates the number of points on an elliptic curve mod p 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 three.



If $\mathrm{P} \neq \mathrm{NP}$



If $\mathrm{P} = \mathrm{NP}$

A world where $\mathrm{P} = \mathrm{NP}$

- 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 $\mathrm{P} = \mathrm{NP}$

- 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: $\mathrm{P} \neq \mathrm{NP}$

NP-Completeness

Understanding the P vs. NP question

Believe $\mathrm{P} \neq \mathrm{NP}$, but very far from proving it

Question 1: How can studying specific computational problems help us get a handle on resolving P vs. NP?

Question 2: What would $\mathrm{P} \neq \mathrm{NP}$ allow us to conclude about specific problems we care about?

Idea: Identify the "hardest" problems in NP

Find $L \in \mathrm{NP}$ such that $L \in \mathrm{P}$ iff $\mathrm{P} = \mathrm{NP}$

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

$$A \leq_{\mathrm{m}} B$$

if there is a computable function $f: \Sigma^* \to \Sigma^*$ such that for all strings $w \in \Sigma^*$, we have $w \in A \Leftrightarrow f(w) \in B$

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 reducible to language B, written

$$A \leq_{\mathrm{p}} B$$

if there is a polynomial-time computable function $f: \Sigma^* \to \Sigma^*$ such that for all strings $w \in \Sigma^*$, we have $w \in A \Leftrightarrow f(w) \in B$

Implications of poly-time reducibility

Theorem: If $A \leq_{\mathrm{p}} B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$

Proof: Let M decide B in poly time, and let f be a poly-time reduction from A to B. The following TM decides A in poly time:

Is NP closed under poly-time reductions?

If $A \leq_{\mathrm{p}} B$ and B is in NP, does that mean A is also in NP?

a) Yes, the same proof works using NTMs instead of TMs
b) No, because the new machine is an NTM instead of a deterministic TM
c) No, because the new NTM may not run in polynomial time
d) No, because the new NTM may accept some inputs it should reject
e) No, because the new NTM may reject some inputs it should accept

NP-completeness

Definition: A language B is NP-complete if

1) $B \in \mathrm{NP}$, and
2) Every language $A \in \mathrm{NP}$ is poly-time reducible to B, i.e., $A \leq_{\mathrm{p}} B$ ("B is NP-hard")

Implications of NP-completeness

Theorem: Suppose B is NP-complete.

Then $B \in \mathrm{P}$ iff $\mathrm{P} = \mathrm{NP}$

Proof:

Implications of NP-completeness

Theorem: Suppose B is NP-complete.

Then $B \in \mathrm{P}$ iff $\mathrm{P} = \mathrm{NP}$

Consequences of B being NP-complete:

1) If you want to show $\mathrm{P} = \mathrm{NP}$, you just have to show $B \in \mathrm{P}$
2) If you want to show $\mathrm{P} \neq \mathrm{NP}$, you just have to show $B \notin \mathrm{P}$
3) If you already believe $\mathrm{P} \neq \mathrm{NP}$, then you believe $B \notin \mathrm{P}$