Computational Complexity Theory

Lecture 2: Class P and NP

Department of Computer Science,
Indian Institute of Science

Recap: Turing Machines

- An algorithm is a set of instructions or rules.
- To understand the performance of an algorithm we need a model of computation. Turing machine is one such *natural* model (introduced by Alan Turing in 1936).
- A TM consists of:
  - Memory tape(s)
  - A finite set of rules
- Turing machines ⟷ A mathematical way to describe algorithms.

Recap: Turing Machines

- Definition. A k-tape Turing Machine M is described by a tuple (Γ, Q, δ) such that
- M has k memory tapes (input/work/output tapes) with *heads*;
- Γ is a finite set of alphabets. (Every memory cell contains an element of Γ)
- Q is a finite set of states. (special states: q_{start} , q_{halt})
- δ is a function from $Q \times \Gamma^k$ to $Q \times \Gamma^k \times \{L,S,R\}^k$

known as transition function; it captures the dynamics of M

Recap: TM Computation

- Start configuration.
  - All tapes other than the input tape contain blank symbols.
  - The input tape contains the input string.
  - All the head positions are at the start of the tapes.
  - The machine is in the start state q_{start} .
- Computation.
  - A **step of computation** is performed by applying δ.
- Halting.
  - Once the machine enters q_{halt} it stops computation.

Recap: TM Running time

- Let $f: \{0,1\}^* \rightarrow \{0,1\}^*$ and $T: \mathbb{N} \rightarrow \mathbb{N}$ and M be a Turing machine.

- Definition. We say M ***computes*** f if on every x in $\{0,1\}^*$, M halts with f(x) on its output tape beginning from the start configuration with x on its input tape.

- Definition. M computes f *in T(|x|)* ***time***, if for every x in $\{0,1\}^*$, M halts within T(|x|) steps of computation and outputs f(x).

Recap: Turing Machines that halt

- In this course, we would be dealing with
  - Turing machines that halt on every input.
  - Computational problems that can be solved by Turing machines.

- Can every computational problem be solved using Turing machines?

Recap: Uncomputability

- There are problems for which there exists *no* TM that halts on every input instances of the problem and outputs the correct answer.

  - Input: A system of polynomial equations in many variables with integer coefficients.
  - Output: Check if the system has integer solutions .
  - Question: Is there an algorithm to solve this problem?

- Theorem. There doesn't exist any algorithm (realizable by a TM) to solve this problem. (Davis, Putnam, Robinson, Matiyasevich 1970)

Recap: Why Turing Machines?

- TMs are natural and intuitive.

- Church-Turing thesis: *"Every physically realizable computation device – whether it's based on silicon, DNA, neurons or some other alien technology – can be simulated by a Turing machine".*

  --- [quoted from Arora-Barak's book]

- Several other computational models can be simulated by TMs.

Recap: Why Turing Machines?

- TMs are natural and intuitive.

- Strong Church-Turing thesis: *"Every physically realizable computation device – whether it's based on silicon, DNA, neurons or some other alien technology – can be simulated efficiently by a Turing machine".*

Possible exception: Quantum machines!

Recap: Time Constructible functions

- Time constructible functions. A function $T: N \rightarrow N$ is *time constructible* if $T(n) \geq n$ and there's a TM that computes the function that maps x to $T(|x|)$ in $O(T(|x|))$ time.

  in binary

- Examples: $T(n) = n^2$, or 2^n, or $n \log n$

Recap: Robustness of TM

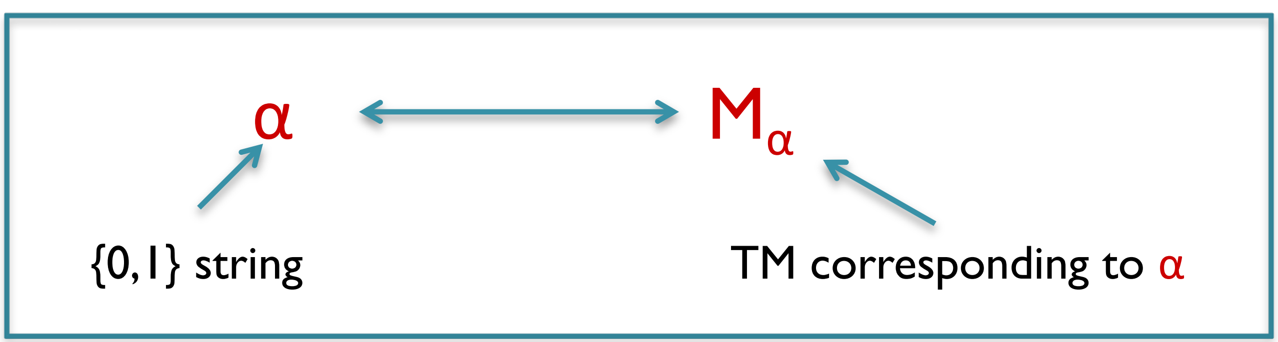
- Let f: $\{0,1\}^* \rightarrow \{0,1\}^*$ and T: $N \rightarrow N$ be a time constructible function.

- Binary alphabets suffice.
  - If a TM M computes f in T(n) time using Γ as the alphabet set, then there's another TM M' that computes f in time $4.\log |\Gamma| .T(n)$ using {0, 1, blank} as the alphabet set.

- A single tape suffices.
  - If a TM M computes f in T(n) time using k tapes then there's another TM M' that computes f in time $5k . T(n)^2$ using a single tape that is used for input, work and output.

Recap: TM as strings

- Every TM can be represented by a finite string over {0,1}.
- Every string over {0,1} represents some TM.
- Every TM has infinitely many string representations.



Recap: TM as strings

- Every TM can be represented by a finite string over {0,1}.
- Every string over {0,1} represents some TM.
- Every TM has infinitely many string representations.
- A TM (i.e., its string representation) can be given as an input to another TM !!

Recap: Universal Turing Machines

- Theorem. There exists a TM U that on every input x, α in $\{0,1\}^*$ outputs $M_\alpha(x)$.

- Further, if M_α halts within T steps then U halts within $C. T. \log T$ steps, where C is a constant that depends only on M_α 's alphabet size, number of states and number of tapes.

- Physical realization of UTMs are modern day electronic computers.

Complexity class P

Decision Problems

- In the initial part of this course, we'll focus primarily on decision problems.

Decision Problems

- In the initial part of this course, we'll focus primarily on decision problems.
- Decision problems can be naturally identified with Boolean functions, i.e., functions from $\{0,1\}^*$ to $\{0,1\}$.

Decision Problems

- In the initial part of this course, we'll focus primarily on decision problems.
- Decision problems can be naturally identified with Boolean functions, i.e., functions from $\{0,1\}^*$ to $\{0,1\}$.
- Boolean functions can be naturally identified with sets of $\{0,1\}$ strings, also called languages.

Decision Problems

Decision problems ⟺ Boolean functions ⟺ Languages

- Definition. We say a TM M *decides* a language $L \subseteq \{0,1\}^*$ if M computes f_L, where $f_L(x) = 1$ if and only if $x \in L$.

The characteristic function of L .

Complexity Class P

- Let $T: N \rightarrow N$ be some function.
- Definition: A language L is in $DTIME(T(n))$ if there's a TM that decides L in time $O(T(n))$.
- Defintion: Class $P = \bigcup_{c > 0} DTIME(n^c)$.

Complexity Class P

- Let $T: N \rightarrow N$ be some function.
- Definition: A language L is in $DTIME(T(n))$ if there's a TM that decides L in time $O(T(n))$.
- Defintion: Class $P = \bigcup_{c > 0} DTIME(n^c)$.

Deterministic polynomial-time

Complexity Class P : Examples

- Cycle detection *(DFS)*
  - Check if a given graph has a cycle.

Complexity Class P : Examples

- Cycle detection

- Solvabililty of a system of linear equations *(Gaussian elimination)*
  - Given a system of linear equations over $\mathbb{Q}$ check if there exists a common solution to all the linear equations.

Complexity Class P : Examples

- Cycle detection
- Solvabililty of a system of linear equations
- Perfect matching *(Edmonds 1965)* (birth of class P)
  - Check if a given graph has a perfect matching

Complexity Class P : Examples

- Cycle detection
- Solvabililty of a system of linear equations
- Perfect matching
- Planarity testing *(Hopcroft & Tarjan 1974)*
  - Check if a given graph is planar

Complexity Class P : Examples

- Cycle detection
- Solvabililty of a system of linear equations
- Perfect matching
- Planarity testing
- Primality testing *(Agrawal, Kayal & Saxena 2002)*
  - Check if a number is prime

Polynomial-time Turing Machines

- Definition. A TM M is a *polynimial-time* TM if there's a polynomial function $q: \mathbb{N} \rightarrow \mathbb{N}$ such that for every input $x \in \{0,1\}^*$, M halts within $q(|x|)$ steps.

Polynomial function. $q(n) = O(n^c)$ for some constant c.

Class (functional) P

- What if a problem is not a decision problem? Like the task of adding two integers.

Class (functional) P

- What if a problem is not a decision problem? Like the task of adding two integers.
- One way is to focus on the i-th bit of the output and make it a decision problem.

(Is the i-th bit, on input x, 1?)

Class (functional) P

- What if a problem is not a decision problem? Like the task of adding two integers.
- One way is to focus on the i-th bit of the output and make it a decision problem.

- Alternatively, we define a class called functional P or FP.

Class (functional) P

- What if a problem is not a decision problem? Like the task of adding two integers.
- One way is to focus on the i-th bit of the output and make it a decision problem.

- We say that a problem or a function $f: \{0,1\}^* \longrightarrow \{0,1\}^*$ is in FP (functional P) if there's a polynomial-time TM that computes f.

Complexity Class FP : Examples

- Greatest Common Divisor *(Euclid ~300 BC)*
    - Given two integers a and b, find their gcd.

Complexity Class FP : Examples

- Greatest Common Divisor
- Counting paths in a DAG *(homework)*
  - Find the number of paths between two vertices in a directed acyclic graph.

Complexity Class FP : Examples

- Greatest Common Divisor
- Counting paths in a DAG
- Maximum matching *(Edmonds 1965)*
  - Find a maximum matching in a given graph

Complexity Class FP : Examples

- Greatest Common Divisor
- Counting paths in a DAG
- Maximum matching
- Linear Programming *(Khachiyan 1979, Karmarkar 1984)*
    - Optimize a linear objective function subject to linear (in)equality constraints

Complexity Class FP : Examples

- Greatest Common Divisor
- Counting paths in a DAG
- Maximum matching
- Linear Programming *(Khachiyan 1979, Karmarkar 1984)*
  - Optimize a linear objective function subject to linear (in)equality constraints

Not known if LP has a *strongly polynomial-time* algorithm.

Homework: Read about the differences between *strongly poly-time*, *weakly poly-time* and *pseudo poly-time* algorithms.

Complexity Class FP : Examples

- Greatest Common Divisor
- Counting paths in a DAG
- Maximum matching
- Linear Programming
- Factoring Polynomials *(Lenstra, Lenstra, Lovasz 1982)*
  - Compute the irreducible factors of a univariate polynomial over $\mathbb{Q}$

Complexity class NP

Complexity Class NP

- Solving a problem is generally *harder* than verifying a given solution to the problem.

- Class NP captures the set of decision problems whose solutions are *efficiently verifiable*.

Complexity Class NP

- Solving a problem is generally *harder* than verifying a given solution to the problem.

- Class NP captures the set of decision problems whose solutions are *efficiently verifiable*.

<u>Nondeterministic polynomial-time</u>

Complexity Class NP

- Definition. A language $L \subseteq \{0,1\}^*$ is in NP if there's a polynomial function $p: \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M (called the *verifier*) such that for every x,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

Complexity Class NP

- Definition. A language $L \subseteq \{0,1\}^*$ is in NP if there's a polynomial function $p: \mathbb{N} \rightarrow \mathbb{N}$ and a polynomial-time TM M (called the *verifier*) such that for every x,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

u is called a *certificate or witness* for x (w.r.t L and M), if $x \in L$.

Complexity Class NP

- Definition. A language $L \subseteq \{0,1\}^*$ is in NP if there's a polynomial function $p: \mathbb{N} \rightarrow \mathbb{N}$ and a polynomial-time TM M (called the *verifier*) such that for every x,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

- It follows that verifier M **cannot be fooled** !

Complexity Class NP

- Definition. A language $L \subseteq \{0,1\}^*$ is in NP if there's a polynomial function $p: \mathbb{N} \rightarrow \mathbb{N}$ and a polynomial-time TM M (called the *verifier*) such that for every x,

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

- Class NP contains those problems (languages) which have such efficient verifiers.

Class NP : Examples

- Vertex cover
  - Given a graph G and an integer k, check if G has a vertex cover of size k.

Class NP : Examples

- Vertex cover
- 0/1 integer programming
  - Given a system of linear (in)equalities with integer coefficients, check if there's a 0-1 assignment to the variables that satisfy all the (in)equalities.

Class NP : Examples

- Vertex cover
- 0/1 integer programming
- Integer factoring
  - Given two numbers n and U, check if n has a prime factor less than or equal to U.

Class NP : Examples

- Vertex cover
- 0/1 integer programming
- Integer factoring
- Graph isomorphism
  - Given two graphs, check if they are isomorphic.

Class NP : Examples

- 2-Diophantine solvability
  - Given three integers a, b and c, check if the equation $ax^2 + by + c = 0$ has a solution (x, y), where both x and y are positive integers.

[Homework]: Show that the above problem is in NP.

Hint: If (x, y) is a solution, then so is $(x + b, y - a(2x + b))$.

Is P = NP ?

- Obviously, $P \subseteq NP$.

- Whether or not $P = NP$ is an outstanding open question in mathematics and TCS!

Is P = NP ?

- Obviously, $P \subseteq NP$.
- Whether or not $P = NP$ is an outstanding open question in mathematics and TCS!
- Solving a problem does seem harder than verifying its solution, so most people believe that $P \neq NP$.

Is P = NP ?

- Obviously, $P \subseteq NP$.

- Whether or not $P = NP$ is an outstanding open question in mathematics and TCS!

- $P = NP$ has many weird consequences, and if true, will pose a serious threat to secure and efficient cryptography (and e-commerce).

Is P = NP ?

- Obviously, $P \subseteq NP$.
- Whether or not $P = NP$ is an outstanding open question in mathematics and TCS!
- Mathematics has gained much from attempts to prove such "negative" statements—Galois theory, Godel's incompleteness, Fermat's Last Theorem, Turing's undecidability, Continuum hypothesis etc.

Is P = NP ?

- Obviously, $P \subseteq NP$.
- Whether or not $P = NP$ is an outstanding open question in mathematics and TCS!
- Complexity theory has several of such intriguing unsolved questions.

The history and status of the P versus NP question

-- survey by Michael Sipser (1992)