Computational Complexity Theory

Lecture 3: Class NP, Karp reductions, NP-completeness

Department of Computer Science,
Indian Institute of Science

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

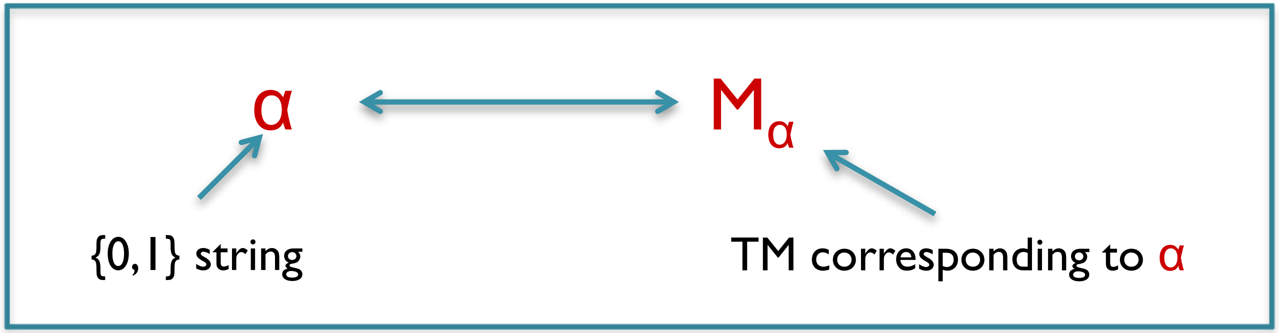
- Examples: $T(n) = n^2$, or 2^n, or $n \log n$

Recap: TM Robustness

- 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 $5k \cdot T(n)^2$ 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.

Recap: Decision Problems

Decision problems ⟺ Boolean functions ⟺ Languages

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

The characteristic function of L .

Recap: 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
- Solvabililty of a system of linear equations
- Perfect matching
- Planarity testing
- Primality testing

Recap: 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.

Recap: 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 $f: \{0,1\}^* \rightarrow \{0,1\}^*$ is in FP (functional P) if there's a polynomial-time TM that computes f.

Complexity Class FP : Examples

- Greatest Common Divisor
- Counting paths in a DAG
- Maximum matching
- Linear Programming
- Factoring Polynomials

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*.

Nondeterministic polynomial-time

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,

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

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

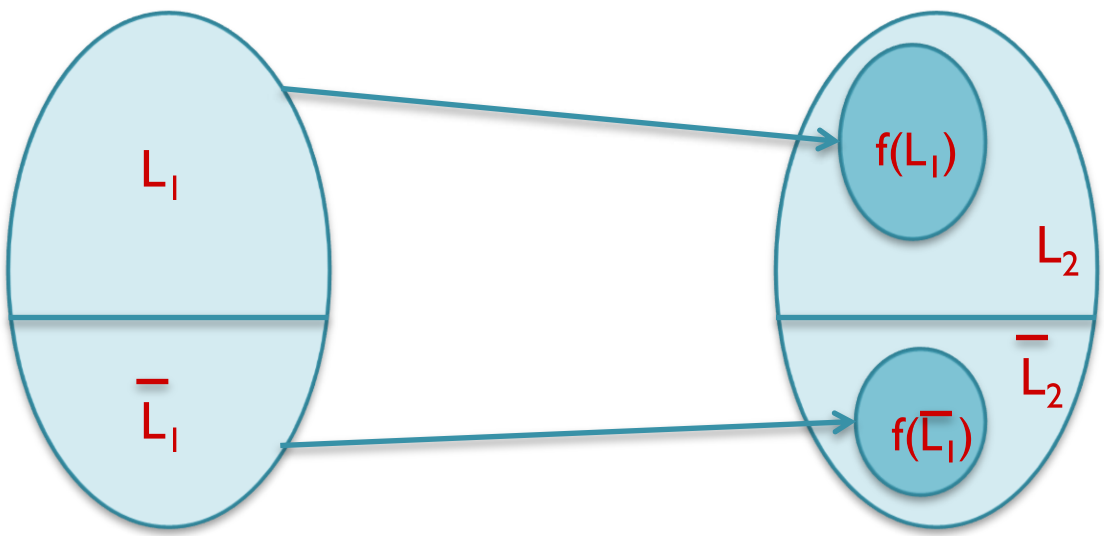
-- survey by Michael Sipser (1992)

Reductions

Polynomial-time reduction

- Definition. We say a language $L_1 \subseteq \{0,1\}^*$ is *polynomial-time (Karp) reducible* to a language $L_2 \subseteq \{0,1\}^*$ if there's a polynomial-time computable function f s.t.

$$x \in L_1 \iff f(x) \in L_2$$



Polynomial-time reduction

- Definition. We say a language $L_1 \subseteq \{0,1\}^*$ is *polynomial-time (Karp) reducible* to a language $L_2 \subseteq \{0,1\}^*$ if there's a polynomial time computable function f s.t.

$$x \in L_1 \iff f(x) \in L_2$$

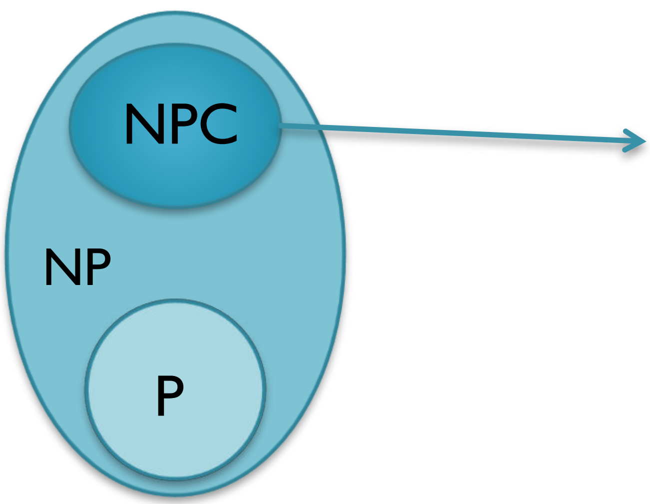
- Notation. $L_1 \le_p L_2$

- Observe. If $L_1 \le_p L_2$ and $L_2 \le_p L_3$ then $L_1 \le_p L_3$.

NP-completeness

- **Definition.** A language L' is *NP-hard* if for every L in NP, L $\leq_P$ L'. Further, L' is *NP-complete* if L' is in NP and is NP-hard.

- **Observe.** If L' is NP-hard and L' is in P then P = NP. If L' is NP-complete then L' in P if and only if P = NP.



Hardest problems inside NP in the sense that if one NPC problem is in P then all problems in NP is in P.

NP-completeness

- Definition. A language L' is *NP-hard* if for every L in NP, $L \leq_p L'$. Further, L' is *NP-complete* if L' is in NP and is NP-hard.

- Observe. If L' is NP-hard and L' is in P then P = NP. If L' is NP-complete then L' in P if and only if P = NP.

- [Homework]. Let $L_1 \subseteq \{0,1\}^*$ be any language and L_2 be a language in NP. If $L_1 \leq_p L_2$ then L_1 is also in NP.

Few words on reductions

- As to how we define a reduction from one language to the other (or one function to the other) is usually guided by a *question on whether two complexity classes are different or identical*.

- For polynomial-time reductions, the question is whether or not P equals NP.

- Reductions help us define *complete problems* (the 'hardest' problems in a class) which in turn help us compare the complexity classes under consideration.

Class NP : Examples

- Vertex cover (NP-complete)
- 0/1 integer programming (NP-complete)
- 3-coloring planar graphs (NP-complete)
- 2-Diophantine solvability (NP-complete)
- Integer factoring (unlikely to be NP-complete)
- Graph isomorphism (Quasi-P) *Babai 2015*

How to show existence of an NPC problem?

- Let L' = { $(\alpha, x, 1^m, 1^t)$: there exists a $u \in \{0,1\}^m$ s.t. M_α accepts (x, u) in t steps }

- Observation. L' is NP-complete.

- The language L' involves Turing machine in its definition. Next, we'll see an example of an NP-complete problem that is arguably more natural.