CS340: Theory of Computation Sem I 2017-18

Lecture Notes 18: Introduction to Complexity Theory

Raghunath Tewari IIT Kanpur

We next shift our focus to the class of decidable languages and study them in more detail. Specifically, we consider restrictions on computing resources such as time and space, and study the resulting class of problems and their relations.

- From now on we will consider only halting Turing machines unless otherwise specified.
- We will assume that our TMs have 3 tapes - a read only input tape, a read/write work tape and a write only output tape.

1 Time and Space Complexity

Definition 1.1. Let M be a Turing machine.

- The *running time* of M (also known as *time complexity*) is a function $f : \mathbb{N} \longrightarrow \mathbb{N}$, such that for all inputs x with $|x| = n$, M halts on x within $f(n)$ number of steps.
- The *space required* by M (also known as *space complexity*) is a function $f : \mathbb{N} \longrightarrow \mathbb{N}$, such that for all inputs x with $|x| = n$, M's work tape uses at most $f(n)$ cells on input x.

Note 1. A few key points when dealing with time/space complexity.

- Generally we choose the best (smallest) possible function $f(n)$ when describing the time or space complexity of a TM.
- When considering time or space complexity of a TM we use the asymptotic notations and disregard multiplicative constants and lower order terms.

Lets take a small detour and refresh ourselves with the asymptotic notations for comparing functions.

1.1 Asymptotic Analysis of Functions

Let $f, g : \mathbb{N} \longrightarrow \mathbb{N}$.

- $f(n) \in O(g(n))$ if $\exists\ c > 0$ and $n_0 \geq 0$, such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$.
- $f(n) \in \Theta(g(n))$ if $\exists\ c_1, c_2 > 0$ and $n_0 \geq 0$, such that for all $n \geq n_0$, $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$.
- $f(n) \in o(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

1.2 Deterministic Time and Space Classes

Definition 1.2 (Deterministic Complexity Classes)**.**

- Let $t : \mathbb{N} \longrightarrow \mathbb{N}$.

$$\mathsf{TIME}(t(n)) = \{L \mid L = L(M) \text{ for some TM } M \text{ having } O(t(n)) \text{ time complexity}\}.$$

- Let $s : \mathbb{N} \longrightarrow \mathbb{N}$.

$$\mathsf{SPACE}(s(n)) = \{L \mid L = L(M) \text{ for some TM } M \text{ having } O(s(n)) \text{ space complexity}\}.$$

2 Time and Space Complexity of Nondeterministic TMs

Definition 2.1. Let N be a nondeterministic Turing machine.

- The *running time* of N (also known as *time complexity*) is a function $f : \mathbb{N} \longrightarrow \mathbb{N}$, such that for all inputs x, every computation path of N halts within $f(|x|)$ number of steps.
- The *space required* by M (also known as *space complexity*) is a function $f : \mathbb{N} \longrightarrow \mathbb{N}$, such that for all inputs x, every computation path of N uses at most $f(|x|)$ cells on its work tape.

Theorem 1. *Let $t(n) \geq n$. Then every $t(n)$ time TM having multiple work tapes has an equivalent $O((t(n))^2)$ time TM with a single work tape.*

Theorem 2. *Let $t(n) \geq n$. Then every $t(n)$ time NDTM has an equivalent $2^{O(t(n))}$ time deterministic TM.*

2.1 Nondeterministic Time and Space Classes

Definition 2.2 (Nondeterministic Complexity Classes)**.**

- Let $t : \mathbb{N} \longrightarrow \mathbb{N}$.

$$\mathsf{NTIME}(t(n)) = \{L \mid L = L(N) \text{ for some NDTM } N \text{ having } O(t(n)) \text{ time complexity}\}.$$

- Let $s : \mathbb{N} \longrightarrow \mathbb{N}$.

$$\mathsf{NSPACE}(s(n)) = \{L \mid L = L(N) \text{ for some NDTM } N \text{ having } O(s(n)) \text{ space complexity}\}.$$

3 The Complexity Classes P and NP

In this section we will define the complexity classes P and NP and see examples of problems in these classes.

3.1 The Class P

- The class of problems that can be solved in polynomial time by a deterministic TM.
- Formally,
$$\mathsf{P} = \bigcup_{k \geq 0} \mathsf{TIME}(n^k).$$
- Widely accepted as the class of problems with an efficient solution. Most algorithms that we have seen so far belong to P.
- Examples: Matrix multiplication, graph reachability, all CFLs (using dynamic programming), computing determinant of a matrix using the Gaussian elimination method.

Note 2. Time complexity of a TM (or algorithm) is a property of the design of the TM and not that of the problem. For the same problem we can have TMs of differing time complexity.

3.2 The Class NP

- The class of problems that can be solved in polynomial time by a nondeterministic TM.
- Formally,
$$\mathsf{NP} = \bigcup_{k \geq 0} \mathsf{NTIME}(n^k).$$
- Widely accepted as the class of problems whose solutions are efficiently verifiable. Based on this we have the following alternative definition of NP.

Definition 3.1 (Alternate Definition of NP)**.** $L \in \mathsf{NP}$ if there exists constants $c > 0$ and $k \geq 0$, and a polynomial time TM V (also known as the "verifier"), such that $\forall x \in \Sigma^*$,

$$x \in L \iff \exists y \in \Sigma^* \text{ such that } |y| \leq c \cdot |x|^k \text{ and } V(x, y) = 1.$$

- The string y is also known as the *certificate* or *proof* of the fact that x belongs to L.
- Its length is polynomially bounded by the length of x.
- If $x \in L$ then there exists *some* certificate which makes a verifier accept. Alternately, if $x \notin L$ then *every* certificate must make the verifier reject.
- The certificate can be any string as long as it helps the verifier make the correct decision.
- Generally we will use this definition to show that a language is in NP.

3.3 Examples of Problems in NP

1. $$\text{COMPOSITES} = \{p \mid p \text{ is a composite number}\}.$$

   Claim 3. COMPOSITES $\in$ NP.

   Certificate

   A positive integer q.

Verifier's Algorithm

Input: $\langle p, q \rangle$

(a) Check whether $1 < q < p$. If not, then *reject*.

(b) Check if q divides p. If yes, then *accept* else *reject*.

Proof of correctness

p is a composite number if and only if there exists a number q strictly between 1 and p such that q divides p.

2. Let G_1 and G_2 be two graphs. For a graph G let $V(G)$ and $E(G)$ denote the vertex set and edge set of G respectively. We say G_1 is *isomorphic* to G_2 (denoted as $G_1 \cong G_2$) if there is a bijective function

$$f : V(G_1) \longrightarrow V(G_2),$$

such that for all pair of vertices $u, v \in V(G_1)$,

$$(u, v) \in E(G_1) \iff (f(u), f(v)) \in E(G_2).$$

Then,

$$\mathsf{GI} = \{\langle G_1, G_2 \rangle \mid G_1 \cong G_2\}.$$

Claim 4. $\mathsf{GI} \in \mathsf{NP}$.

Certificate

A function $f : V(G_1) \longrightarrow V(G_2)$.

Verifier's Algorithm

Input: $\langle G_1, G_2, f \rangle$

(a) Check that f is indeed a bijective function from $V(G_1)$ to $V(G_2)$. If not, then *reject*.

(b) $\forall u, v \in V(G_1)$ check whether the following statement is true

$$(u, v) \in E(G_1) \iff (f(u), f(v)) \in E(G_2).$$

(c) If the check succeeds for all pairs then *accept* and if it fails for even one pair then *reject*.

3.

$$\mathsf{Clique} = \{\langle G, k \rangle \mid G \text{ has a complete subgraph of size at least } k\}.$$

Claim 5. $\mathsf{Clique} \in \mathsf{NP}$.

Certificate

$U \subseteq V(G)$.

Verifier's Algorithm

Input: $\langle G, k, U \rangle$

(a) Verify that U is indeed a subset of $V(G)$. If not, then *reject*.

(b) Verify that $|U| = k$. If not, then *reject*.

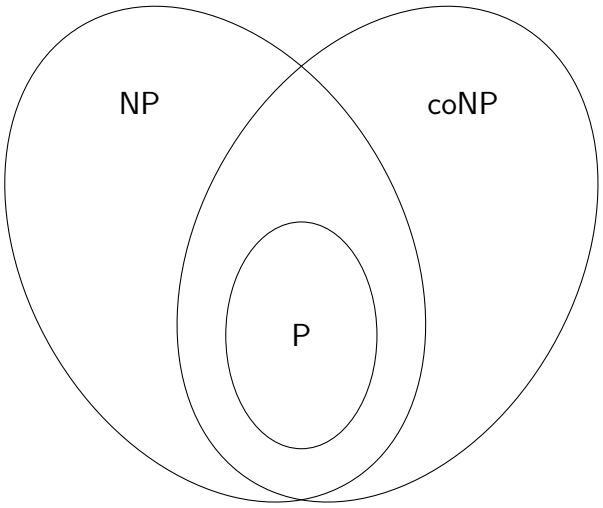
(c) $\forall u, v \in U$ check whether $(u, v) \in E(G)$. If the check succeeds for all pairs then *accept* else *reject*.

3.4 The Class coNP

- It is the class of problems whose complement is in NP.
- Formally,

$$\mathsf{coNP} = \{L \mid \overline{L} \in \mathsf{NP}\}.$$

- The following diagram illustrates the currently known relation between P, NP and coNP.



- It is an open question whether NP = coNP or not.
- Note that P is a deterministic class and hence is closed under complementation.

Exercise 1. Show that if NP $\subseteq$ coNP then NP = coNP.