CS 6810: Theory of Computing **Spring 2026**

Lecture 2: January 22, 2026

Lecturer: Nicholas Spooner *Scribe: Thomas McFarland*

1 Introduction

Last lecture we covered Turing machines (**TM**), a theoretical model of computing which forms the foundational system of computational theory. Such a machine take an input x and (may) give an output in a number of steps, thus it is convention to use function notation, that is if a machine M outputs y when fed x, we say $M(x) = y$

Definition 1.1. *DTIME(T(n)) is the set of all languages $\mathcal{L} \subseteq \{0,1\}^*$ s.t. there exists a turing machine M that for all $x \in \{0,1\}^*$ runs in time[a] $\mathcal{O}(T(|x|))$ and $M(x) = 1 \iff x \in \mathcal{L}$*

[a]Time here referring to the number of steps the TM takes to reach a final state on a given input

The final condition in that definition says that M "decides" $\mathcal{L}$. This is equivalent to determining if x is in a language $\mathcal{L} \subseteq \{0,1\}^*$ or it's complement $\overline{\mathcal{L}}$. Thus, the above definition can be rephrased as the set of all languages that are decided in time $\mathcal{O}(T(n))$. Most of the problems in this course fall under decision problems, that is cases where a turing machine outputs either 1 or 0 deterministically for a given input x.

Definition 1.2.

$$\boldsymbol{P} = \bigcup DTIME(T(n))$$

for all polynomial $T(n)$

Generally when we speak of "efficient computation" in complexity theory, we are speaking of languages in **P**.

Conjecture 1.3 (Church-Turing Conjecture for **P**)**.** *Given that all theoretical models of efficient computation of functions over the natural numbers vary by only a polynomial factor of steps, the class* **P** *is the same regardless of the model used to define it.*

In essence the above means that, no matter how one tries to define complexity or complexity classes, the broad strokes of the resulting theories remain the same.

2 Beyond class P

If **P** is "efficient computation", **NP** is the class where the answer can be "efficiently verified" as correct.

Definition 2.1. *A language $\mathcal{L}$ is in* **NP** *if there is a polynomial $p(x)$ and V, a polynomial time TM[a] s.t. $\forall x \in \{0,1\}^*$, $x \in \mathcal{L} \iff \exists u \in \{0,1\}^{p(|x|)} \wedge V(x,u) = 1$*

[a] Called a verifier

We can say this colloquially as **NP** consists of all elements in the language have a "certificate" which can be verified in polynomial time.

Open problem 2.2 (**NP** vs. **Co-NP**)**.** *For P it is known that the complement of any language L in P is also in P. For NP this is not believed to be universally true.*

2.1 Another Definition of NP

While the above definition offers, the most intuitive picture of the class **NP**, the acronym itself comes from its alternate, historical definition via a Nondeterministic Turing Machine (**NDTM**).

Definition 2.3 (Nondeterministic Turing Machine)**.** *A Turing Machine with two transition functions (commonly called δ_0 and δ_1) is called a Nondeterministic Turing Machine.*

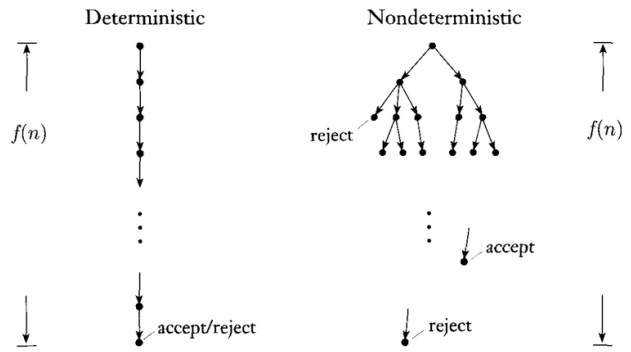

Figure 1: The computation tree for a deterministic vs. a nondeterministic turing machine

For a NDTM, at each step the turing machine can "use" either transition function. An output is accepted if any combination of steps from the two transition functions reaches an accepting state. That is, for NDTM M, $M(x) = 1$ if there exists a leaf in the tree of computations (see 1) choosing δ_0 or δ_1 in state q_{accept}[1]. The negation of this means that $M(x) = 0$ if and only if there is no path to an accept path in the computation tree. We time bound an NDTM via the max length of any path the machine can take on a given input.

We also define NTIME similarly to DTIME, such that a language $\mathcal{L}$ is in NTIME(T(n)) if there exists NDTM M running in time $\mathcal{O}(T(n))$ s.t. $\forall x M(x) = 1$ iff $x \in \mathcal{L}$.

Claim 2.4.

$$\boldsymbol{NP} = \bigcup NTIME(T(n))$$

for all polynomial $T(n)$

[1] Generally there may be multiple accepting states, but for simplicity we consider a NDTM with just one q_{accept}

We prove both directions of this, starting with a language $\mathcal{L}$ in NTIME(n^c). Since this language is in NTIME(n^c), we know the language can be decided in at most $\mathcal{O}(n^c)$ steps of a NDTM. We take as a certificate for a given $x \in \mathcal{L}$ the choices between δ_0 and δ_1 for a path that leads to accept. Then we run the NDTM deterministically as a verifier using that as a map. Since we are running it deterministically, and we know this path is of polynomial order length, we can instead run of a normal Turing Machine. Thus we have a polynomial time verifier, and have proven this language is in **NP**.

For the reverse direction, that is starting with a $\mathcal{L} \in$ **NP**, given the certificate length $p(|x|)$, we can enumerate every possible certificate with an NDTM then run the verifier. The runtime of this is $\mathcal{O}(p(|x|) + q(|x|))$, where the latter is the runtime bound of the verifier, which is a polynomial. Thus this language is in polynomial NTIME and the claim has been proven.

2.2 Some NP Problems

It turns out there are a lot of problems in **NP**, of varying level of individual complexity. A table of a few enlightening examples are listed below:

Problem Name	**Definition / Decision Question**	**Certificate**
INDSET	Given a graph G and integer k, does G contain an independent set of size k?	A set $I \subset V(G)$ with $\|I\| = k$
Traveling Salesman (TSP)	Given a graph with edge distances and a budget k, is there a path visiting each vertex with total distance $\leq k$?	The sequence of vertices (the path)
Graph Isomorphism	Given graphs G_1 and G_2, is there a bijection Π (that is, a relabelling) between their vertices such that $\Pi(G_1) = G_2$?	The permutation/bijection Π
Composite	Given an integer N, is it composite (i.e., not prime)?	Two integers $a, b > 1$ such that $N = a \times b$
Connectivity	Given a graph G and vertices s, t, is there a path from s to t?	The sequence of edges (or vertices) forming a path from s to t

All of these are in **NP**, however their complexity varies. Connectivity and composite[2] are in **P**. TSP is in **NP**, but more than that is **NP-Complete** (see section 3.1). Graph isomorphism is unproven and its proper location in this hierarchy is unknown, but based on some intermediate results it is probably not NP Complete[3].

3 Hierarchy of Computability

Definition 3.1.

$$\boldsymbol{EXP} = \bigcup DTIME(2^{n^c})$$

[2]While it is know that the composite decision problem, whether or not a number is composite, is in **P**, whether finding a and b is in **P** is still unknown.

[3]Interestingly this problem falls under a class of what is suspected to form **NP-Intermediate** problems, that is problems that are not in **P** nor are **NP-Complete**. It is a known result that **NP-Intermediate** problems exists if and only if **P** $\neq$ **NP**

for all $c \geq 1$

With these definitions in mind, we can make something of a intuitive claim.

Claim 3.2.

$$\mathbf{P} \subset \mathbf{NP} \subset \mathbf{EXP}$$

We first show that $\mathbf{NP} \subseteq \mathbf{EXP}$. Starting with language $\mathcal{L} \in \mathbf{NP}$ and an input x to decide on, enumerate all possible certificates (certificate length given by $p(|x|)$)

We check the verifier for each, showing that this algorithm decides $\mathcal{L}$. Enumeration and check is $2^{p(|x|)}(1 + q(|x|)) \in O(2^{n^c})$, where again $q(n)$ is the runtime bound of the verifier. Thus this algorithm decides $\mathcal{L}$ and is in **EXP**, proving the statement.

We also know all problems in P is in NP since the verifier can just be its turing machine and the certicate the input (i.e. $u = x$).

Every other part of that claim is unproven! We don't know $NP \neq EXP$ and don't know $P \neq NP$[4]

Open problem 3.3 (Complexity Hierarchy)**.** *Is there a hierarchy in the complexity classes* **P**, **NP**, *and* **EXP**, *or are two of the three classes equivalent.*

3.1 Reducibility and NP-Completeness

While we can't say $\mathbf{P} = \mathbf{NP}$ or $\mathbf{P} \neq \mathbf{NP}$ but we can define relationships between many problems in either, via reducibility.

Definition 3.4 (Polytime Reducibility)**.** *A language* $\mathcal{L}$ *is polytime reducible to language* $\mathcal{L}'$ *(Notation* $\mathcal{L} \leq_p \mathcal{L}'$*) if there exists a polytime computable function* $f : \{0,1\}^* \to \{0,1\}^*$ *(called a reduction) s.t.* $\forall x,\ x \in \mathcal{L} \iff f(x) \in \mathcal{L}'$

Intuitively, we can view this as a map from $\mathcal{L}$ to $\mathcal{L}'$ and from $\mathcal{L}^c$ to $\mathcal{L}'^c$

Claim 3.5.

$$\mathcal{L}' \in \mathbf{P} \wedge \mathcal{L} \leq_p \mathcal{L}' \implies \mathcal{L} \in \mathbf{P}$$

To prove, take input $x \in \mathcal{L}$, compute $f(x)$, then compute $M'(x)$ for turing machine which decides $\mathcal{L}'$. Then this is polytime (as its runtime is $\mathcal{O}(p_{M'}(|x|) + p_f(|x|))$). Given the definition of reducibility, this process is a turing machine which decides $\mathcal{L}$

This is very helpful for when it is hard to solve $\mathcal{L}$ but easy to find f.

Definition 3.6 (NP-Hardness)**.** *A language* $\mathcal{L}$ *is* **NP-Hard** *if for all* $\mathcal{L}'$ *in* **NP**, $\mathcal{L}' \leq_p \mathcal{L}$.

[4]The time hierarchy theorem does however show that $\mathbf{P} \subsetneq \mathbf{EXP}$

If $\mathcal{L}$ is **NP-Hard** and in **NP** we call it **NP-Complete**. Intuitively, we would say this definition defines a class of the "hardest problems in **NP**".

There are some immediate consequences and theorems from these definition, which will be useful.

1. If $\mathcal{L} \leq_p \mathcal{L}'$ and $\mathcal{L}' \leq_p \mathcal{L}''$ then $\mathcal{L} \leq_p \mathcal{L}''$ (transitivity)[5]
2. If $\mathcal{L}$ is **NP-Hard** and $\mathcal{L} \in \mathbf{P}$ then $\mathbf{P} = \mathbf{NP}$.
3. if $\mathcal{L}$ is **NP-Complete** then $\mathcal{L} \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}$

3.2 Does NP-C exist

Reasonably, one may ask whether there even exists **NP-Complete** problems. There are indeed many, but a good (if heavy handed) example to this problem is TMSAT.

Definition 3.7. *TMSAT is a decision problem which asks, given a tuple with a description of a turing machine α, an input string x, a certificate length n, and a time limit[a] t, is there a certificate u of length n s.t. M_α on (x, u) accepts within the given time limit.*

[a] Both the certificate length and time limit are represented by n 1s

Claim 3.8. *TMSAT is* **NP-Complete**

TMSAT is clearly in **NP**, the certificate is u. For the reduction for language $\mathcal{L} \in \mathbf{NP}$, let V be its verifier, $q(n)$ be its runtime bound, and $p(n)$ its certificate length bound. Our reduction maps x to $(V, x, 1^{p(|x|)}, 1^{q(|x|)+p(|x|)})$. Then, by definition of **NP**, TMSAT accepts x if and only if $x \in \mathcal{L}$. Thus, **NP-Complete** is not an empty class.

[5] This can be trivially proven by composing the two reductions together.