CAS CS 535: Complexity Theory

Lecturer: Mark Bun — Fall 2023

Lecture Notes 2: Time Complexity, P, NP, NP-Completeness

Reading.

- Arora-Barak § 1.6, 2.1-2.2, 2.6-2.7

Last time: Turing machines, decidability, undecidability

1 Time Complexity and $\mathbf{P}$

Recall:

- A TM M decides a language L if

$$x \in L \iff M(x) = 1.$$

- M runs in time $T(n)$ if it halts within $T(|x|)$ steps for every $x \in \{0,1\}^*$.

Definition 1. For a function $T(n)$, define the complexity class

$$\mathbf{DTIME}(T(n)) = \{L \subseteq \{0,1\}^* \mid L \text{ is decidable in time } O(T(n))\}.$$

Definition 2. The complexity class $\mathbf{P}$ is defined by

$$\mathbf{P} = \bigcup_{c=1}^{\infty} \mathbf{DTIME}(n^c) = \mathbf{DTIME}(\mathrm{poly}(n)).$$

The class $\mathbf{P}$ roughly captures problems which can be solved efficiently. Of course, there are reasonable criticisms as to whether this is actually the case. For instance:

- Is n^{100} time really "efficient"? Probably not, but "in practice" most polynomial runtimes for natural problems don't suffer from such high exponents. Plus, taking a fairly crude approach to defining efficiency lets us get away with working with the (relatively) simple TM model, and lets us design algorithms in a modular way.

- How about randomized or quantum computation? Later in the course, we'll look at the analogs $\mathbf{BPP}$ and $\mathbf{BQP}$ in these models, and many of the insights that we have from studying $\mathbf{P}$ will carry over. Current evidence actually suggests that $\mathbf{BPP} = \mathbf{P}$, while the jury's still out on $\mathbf{BQP}$, as well as on the question of whether general-purpose quantum computation is physically realizable.

Example 3.

$$\mathsf{CONNECTIVITY} = \{G \mid \text{ graph } G \text{ is connected } \} \in \mathbf{P}.$$

To solve this problem in poly-time, start at an arbitrary vertex and apply breadth-first search until its entire connected component is traversed, counting the number of vertices encountered. The graph is connected if and only if this is equal to the number of vertices in the entire graph.

Note that the exact running time of this algorithm depends on details such as how the input is presented (adjacency matrix? list?), and how data structures are implemented on a TM.

Example 4. It's important to note that $\mathbf{P}$ is a class of languages (decision problems), so to assert that some computational problem is in $\mathbf{P}$, one has to identify a decision version of it.

$$\mathsf{ADD} = \{\langle x, y, i\rangle \mid \text{ the } i\text{'th bit of } x + y \text{ is } 1\} \in \mathbf{P}.$$

The algorithm here is "gradeschool addition" which runs in time linear in the bit complexity of the input.

Of course, when discussing complexity, one doesn't have to stop at polynomial time. For instance, an important class of problems beyond $\mathbf{P}$ consists of those that can be solved in exponential time.

$$\mathbf{EXP} = \bigcup_{c=1}^{\infty} \mathbf{DTIME}(2^{n^c}).$$

2 Class NP

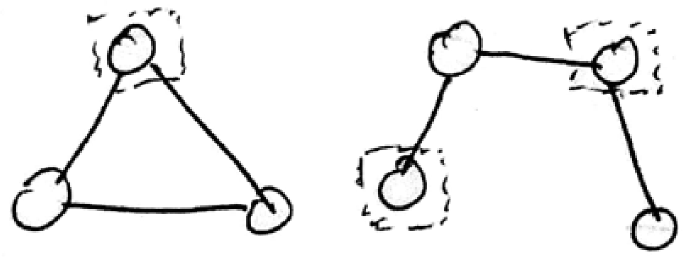
Whereas $\mathbf{P}$ is the class of languages for which membership can be decided efficiently, $\mathbf{NP}$ consists of languages for which membership can be verified efficiently.

Formally:

Definition 5. A language $L \in \mathbf{NP}$ if there exists a poly-time TM V ("verifier") and a polynomial p such that

$$x \in L \implies \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } V(x,u) = 1$$
$$x \notin L \implies \forall u \in \{0,1\}^{p(|x|)}, \quad V(x,u) = 0.$$

Example 6. An independent set in a graph is a subgraph containing no edges.



$$\mathsf{INDSET} = \{\langle G, k\rangle \mid G \text{ has an independent set of size } k\} \in \mathbf{NP}.$$

The certificate u consists of (the encoding of) k vertices in the graph. To verify the certificate, $M(\langle G, k\rangle, u)$ checks if u indeed encodes k vertices with no edges between any pair of them.

2.1 Relationships between P and deterministic time classes

$\mathbf{P} \subseteq \mathbf{NP}$. This is because a verifier can always ignore its certificate.

$\mathbf{NP} \subseteq \mathbf{EXP}$. Let M be a poly-time verifier for language $L \in \mathbf{NP}$. Then the following TM decides L in exponential time:

On input x:

- Run $M(x, u)$ for all $u \in \{0,1\}^{p(|x|)}$
- If any run accepts, output 1. Else, output 0.

The runtime of this algorithm is $2^{p(n)} \cdot \mathrm{poly}(n)$, hence $L \in \mathbf{EXP}$.

2.2 Nondeterministic TMs

An alternative (and in fact, the original) definition of $\mathbf{NP}$ is via nondeterministic TMs. There are several ways to define these, but here's the one from Arora-Barak.

An NTM N is a multi-tape TM, but it instead has two transition functions δ_0, δ_1. In each computation step, it has the option of either following the transition specified by δ_0 or by δ_1.

We say:

- $N(x) = 1$ if there exists a sequence of transitions leading the machine to output 1 on input x, and
- $N(x) = 0$ if for all sequences of transitions, the machine outputs 0 on input x.

Definition 7. For a function $T(n)$, define the complexity class

$$\mathbf{NTIME}(T(n)) = \{L \mid L \text{ is decided by an NTM in time } O(T(n))\}.$$

Example 8. Here's a nondeterminsitic TM deciding the language INDSET.

On input $\langle G, k\rangle$:

- "Nondeterministically guess" a sequence of k vertices. That is, use the two transition functions to write down the encoding of some sequence of k vertices.
- Check that all of the vertices written down are distinct and have no edges between them.

2.3 Equivalence between the two views

Theorem 9.

$$\mathbf{NP} = \bigcup_{c=1}^{\infty} \mathbf{NTIME}(n^c) = \mathbf{NTIME}(\mathrm{poly}(n)).$$

Proof. For the $\subseteq$ direction: Let $L \in \mathbf{NP}$ with poly-time verifier V. Define the following NTM:

On input x:

- Nondeterministically guess $u \in \{0,1\}^{p(|x|)}$
- Output $V(x, u)$.

For the $\supseteq$ direction: Let N be an NTM deciding L. We describe a verifier for L as follows. Its certificate consists of the sequence of nondeterministic choices (i.e., whether to take transition δ_0 or δ_1) that may be made by N. Then:

On input x, u:

- Simulate N on input x using the nondeterminstic choices encoded by u.
- Accept iff N accepts.

$\square$

2.4 Related classes

For a complexity class $\mathbf{C}$, define $\mathbf{coC} = \{L \mid \overline{L} \in \mathbf{C}\}$.

For example, the class $\mathbf{coNP} = \{L \mid \overline{L} \in \mathbf{NP}\}$, which is the set of languages for which <u>non</u>-membership is efficiently verifiable, i.e.,

$$x \notin L \implies \exists u \text{ s.t. } V(x,u) = 0$$
$$x \in L \implies \forall u \quad V(x,u) = 1.$$

Similarly with $\mathbf{EXP}$, we can define

$$\mathbf{NEXP} = \bigcup_{c=1}^{\infty} \mathbf{NTIME}(2^{n^c}).$$

3 P vs. NP, Reductions, and NP-Completeness

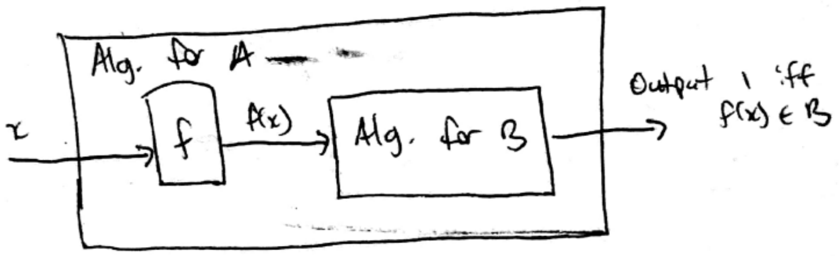
While we're still quite far from resolving central questions like $\mathbf{P} \overset{?}{=} \mathbf{NP}$ and $\mathbf{NP} \overset{?}{=} \mathbf{coNP}$, we have some tools for shedding light on the "structure" of these and other questions. The most important such tools are reductions between problems and the notion of completeness.

For languages A, B, the statement "A poly-time reduces to B" (written $A \leq_p B$) intuitively means:

- Any algorithm solving B is also useful for solving A
- Problem B is "at least as hard as" problem A.

Definition 10. Language A is poly-time reducible (a.k.a. Karp-reducible, many-to-one reducible) to B if there exists a poly-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that

$$x \in A \iff f(x) \in B.$$



Claim 11. *Poly-time reductions have the following properties.*

1. *Transitivity: If $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$.*
2. *If $A \leq_p B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.*
3. *If $A \leq_p B$ and $B \in \mathbf{NP}$, then $A \in \mathbf{NP}$.*

Definition 12. A language B is

- $\mathbf{NP}$-hard if $A \leq_p B$ for every language $A \in \mathbf{NP}$, and
- $\mathbf{NP}$-complete if B is $\mathbf{NP}$-hard <u>and</u> $B \in \mathbf{NP}$.

Consequences of $\mathbf{NP}$-hardness/completeness.

- If a language L is $\mathbf{NP}$-complete, then $L \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}$.
- If you believe $\mathbf{P} \neq \mathbf{NP}$, then you also believe that all $\mathbf{NP}$-hard (and hence, also all $\mathbf{NP}$-complete) problems are intractable.

Example of a reduction. A <u>vertex cover</u> of a graph G is a set of vertices S such that every edge in G is incident to some $v \in S$.

$$\mathsf{VERTEXCOVER} = \{\langle G, k\rangle \mid G \text{ has a vertex cover of size } k\}.$$

Claim 13.

$$\mathsf{INDSET} \leq_p \mathsf{VERTEXCOVER}.$$

The basic idea is that a set of vertices S is a vertex cover iff its complement $\overline{S}$ is an independent set. The following function computes the reduction f.

```
On input $\langle G, k\rangle$:
    Output $\langle G, |V| - k\rangle$.
```

Correctness of the reduction is just the statement that $\langle G, k\rangle \in \mathsf{INDSET} \iff \langle G, |V| - k\rangle \in \mathsf{VERTEXCOVER}$.

Next time: More on reductions, Cook-Levin Theorem