CAS CS 535: Complexity Theory

Lecturer: Mark Bun Fall 2023

Lecture Notes 3:

Reductions, NP-Completeness, Cook-Levin Theorem

Reading.

- Arora-Barak § 2.2-2.4

Last time: Time complexity, P, NP

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

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

$$= \text{ class of languages with poly-time verifiers}$$

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

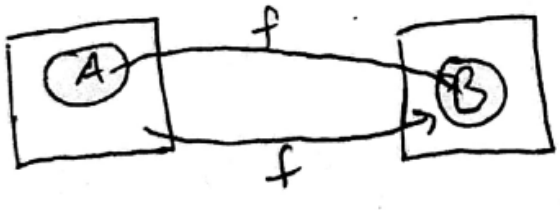
While we're still quite far from resolving central questions like $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ and $\mathbf{NP} \stackrel{?}{=} \mathbf{coNP}$, we have some tools for shedding light on the "structure" of these and other questions. The most important such tools are reductions between problems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

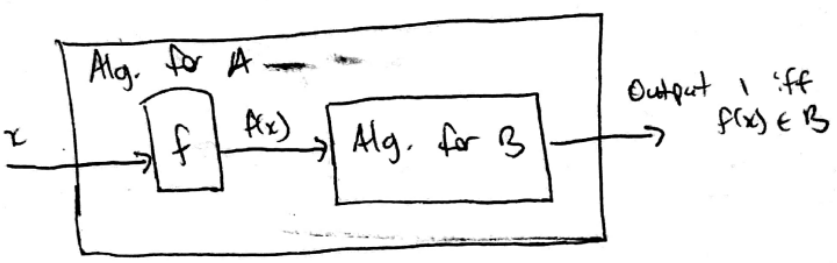
Definition 1. Language A is poly-time reducible (a.k.a. Karp-reducible, many-to-one reducible) to B if there exists a poly-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that

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



Claim 2. *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 $A \in \mathbf{NP}$.*



Example of a reduction. A vertex cover 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 3.

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

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 $\langle G, k\rangle \in \mathsf{INDSET} \iff \langle G, |V| - k\rangle \in \mathsf{VERTEXCOVER}$.

Definition 4. 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 and $B \in \mathbf{NP}$.

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.

2 An NP-Complete Problem

At first glance, **NP**-completeness seems like a shocking notion. But it turns out one can extract an example of an **NP**-complete problem more-or-less right out of the definitions.

Theorem 5. *The language*

$$\mathsf{TMSAT} = \{\langle \alpha, x, 1^m, 1^t \rangle \mid \exists u \in \{0,1\}^m \text{ s.t. } M_\alpha(x,u) \text{ outputs 1 within } t \text{ steps}\}$$

is **NP***-complete.*

Proof. There are two things to show: First that this language is in **NP**, and second that it's **NP**-hard.

TMSAT $\in$ NP. The certificate is the string $u \in \{0,1\}^m$. Thanks to the fact that m is presented in unary as part of the input, this certificate has length polynomial in the input length.

To verify the certificate, use the UTM to simulate $M_\alpha(x,u)$ for t steps and check if it accepts. This runs in poly-time because of the efficiency of the UTM simulation, combined with the fact that the time bound t is given in unary.

TMSAT is NP-hard. We show how to reduce an arbitrary language $L \in \mathbf{NP}$ to TMSAT. Let L have verifier $V(x,u)$, with certificate length $p(n)$ (i.e., $|u| = p(|x|)$) and runtime $q(n)$ as a function of the instance length $n = |x|$.

Our reduction computes the function

$$f(x) = \langle \lfloor V \rfloor, x, 1^{p(|x|)}, 1^{q(|x|)} \rangle.$$

This reduction works because

$$\begin{aligned} x \in L &\iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } V(x,u) \text{ accepts within } q(|x|) \text{ steps} \\ &\iff f(x) \in \mathsf{TMSAT}. \end{aligned}$$

□

3 Satisfiability

Ok, so an **NP**-complete problem exists. But it's quite unnatural (intrinsically making reference to Turing machines), and on its own doesn't help us understand the complexity of specific combinatorial problems we care about. To show that **NP**-completeness is actually a pervasive phenomenon across such problems, we need a more natural example: The Boolean Satisfiability problem.

Some notation/definitions to set up this problem:

Variables: $x_1, \ldots, x_n$

Literals: Variables and their negations, i.e., $x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}$

Formula: Sequence of literals connected by logical AND ($\wedge$), OR ($\vee$), with order determined by parentheses.

An important special case of Boolean formulas is those in CNF (Conjunctive Normal Form). A CNF consists of "clauses" connected by ANDs, where each clause is in turn an OR of literals. For example, let

$$\varphi(x) = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_{13}} \vee x_2) \wedge (x_1 \vee \cdots \vee x_{12}).$$

The width of a CNF is the maximum number of literals per clause (in this example, 12). The size is the number of clauses (in this example, 3).

The expressive power of CNF. Any Boolean function $f : \{0,1\}^k \to \{0,1\}$ can be written as a CNF, but the parameters aren't necessarily so good; the width may be as large as k and the size may be as large as 2^k.

A basic computational problem associated with Boolean formulas (and CNF in particular) is satisfiability: Does there exist an assignment to the variables x that causes the formula to evaluate to 1 (=TRUE)? Define the language

$$\mathsf{SAT} = \{\varphi \mid \varphi \text{ is a CNF formula s.t. } \exists x \quad \varphi(x) = 1\}.$$

For example, $\varphi(x, y, z) = (x \vee y) \wedge (\overline{x} \vee \overline{z}) \in \mathsf{SAT}$ because $x = 1, y = 1, z = 0$ forms a satisfying assignment.

4 Cook-Levin Theorem

Theorem 6 (Cook-Levin Theorem)**.** SAT *is* $\mathbf{NP}$*-complete.*

The rest of this section is devoted to sketching the proof, which I'll do differently from Arora-Barak. It's similar to the proof given in Sipser's *Theory of Computation*.

$\mathsf{SAT} \in \mathbf{NP}$. The certificate is the satisfying assignment. To verify it, check that assigning all variables in this way leads to all clauses being satisfied.

SAT is $\mathbf{NP}$-hard. Let $\mathbf{L} \in \mathbf{NP}$ with poly-time verifier $V(x, u)$. With polynomial runtime overhead, we can transform V into one satisfying the following assumptions:

- Single read/write tape
- Tape alphabet = $\{0, 1, \square\}$, never writes blank symbol
- Accepts by entering state q_{accept} (and staying there)
- Runtime is $T(n) = \text{poly}(n)$ where $n = |x|$
- Certificate is padded with $T(n)$ trailing 0's.

The goal is to exhibit a poly-time computable mapping $f : x \mapsto \varphi_x$ such that

$$\exists u \text{ s.t. } M(x, u) = 1 \iff x \in L \iff f(x) \in \mathsf{SAT} \iff \varphi_x \text{ is satisfiable.}$$

The key object that will help us think about and construct this mapping is the computation tableau, or transcript of V's execution when run on input $\langle x, u \rangle$. As shorthand, let $T = T(|x|)$. The tableau might look something like:

time step	state	head	tape content
$t = 1$	q_0	1	$x_1 x_2 \dots x_n u_1 \dots u_m 0 \dots 0$
2	q_5	2	$0 x_2 \dots x_n u_1 \dots u_m 0 \dots 0$
$\vdots$			
T	q_{accept}	42	$0100 \dots 011$

Observe that:

$$\begin{aligned} M(x,u) = 1 &\iff \text{tableau captures an accepting computation history} \\ &\iff (1)\ \text{First row is a valid starting configuration AND} \\ &\quad\ (2)\ \text{Final row is in the accept state AND} \\ &\quad\ (3)\ \forall t = 1, \dots, T-1,\ \text{Row } t+1 \text{ follows from row } t \text{ via } V\text{'s transition function.} \end{aligned}$$

We're going to build a formula $\varphi_x(s, h, c)$ such that $\varphi_x(s, h, c) = 1$ if and only if this system of checks passes.

The Boolean variables in this formula are as follows:

- s_q^t for every $t = 1, \dots, T$ and $q \in Q$. This variable should be set to 1 iff V is in state q at time t.
- h_ℓ^t for every $t = 1, \dots, T$ and $\ell = 1, \dots, T$. This variable should be set to 1 iff V's head is at location ℓ at time T.
- c_ℓ^t for every $t = 1, \dots, T$ and $\ell = 1, \dots, T$. This represents the content of cell ℓ at time t.

Here's how the components of our formula will capture the various tableau checks we need:

1. The first row is a valid starting configuration.

$$\underbrace{s^1_{q_{\text{start}}}}_{M \text{ starts in start state}} \wedge \underbrace{h_1^1}_{\text{head to the left}} \wedge \underbrace{\bigwedge_{i=1}^{n} (c_i^1 = x_i)}_{\text{input on tape}}$$

2. The final row is accepting.

$$s^T_{q_{\text{accept}}} \quad (V \text{ halts in the accept state})$$

3. For every t, one step of computation takes row t to row $t+1$

$$\bigwedge_{t=1}^{T-1} \bigwedge_{q \in Q} \bigwedge_{\ell=1}^{T} \bigwedge_{b=0,1} (V \text{ in state } q, \text{ head at } \ell \text{ at time } t \text{ reading bit } b \implies \text{state, head, tape at time } t{+}1 \text{ correct}).$$

The precise way to capture the condition on the right as a Boolean expression depends on V's transition function. To illustrate, let's say for example that if V is in some state q and reads the symbol 1, then it transitions to state q', writes the symbol 0, and moves the head one cell to the right. Then we can capture this condition with the expression

$$(s_q^t \wedge h_\ell^t \wedge (c_\ell^t = 1)) \implies s_{q'}^{t+1} \wedge h_{\ell+1}^{t+1} \wedge (c_\ell^{t+1} = 0) \wedge \left(\bigwedge_{\ell' \neq \ell} (c_{\ell'}^{t+1} = c_{\ell'}^t) \right).$$

To turn this expression into a CNF, we use the fact that

$$p \implies (q_1 \wedge q_2 \cdots \wedge q_k) \equiv (p \implies q_1) \wedge \cdots \wedge (p \implies q_k),$$

and that implication $p \implies q_i$ is a function of a constant number of bits, and hence can be written as a constant-size CNF.

4. We also need a bit of logic to enforce that our variables s_q^t and h_ℓ^t can actually mean what they're supposed to mean. Namely, the machine can only be in one state with its head in only one place at any point in time. We can capture the first condition using the formula

$$\bigwedge_{t=1}^{T} \bigwedge_{q \neq q'} \overline{s_q^t} \vee \overline{s_{q'}^t},$$

and similarly for the second condition.

The conjunction of the CNFs capturing all of these conditions is our desired formula φ_x.

5 New NP-Complete Problems from Old

A large part of what makes the Cook-Levin Theorem so powerful is that the **NP**-completeness of SAT can used to prove the **NP**-completeness of many other natural problems. For instance:

$$\begin{aligned} &\leq_p \text{dHAMPATH} \\ \text{SAT} \leq_p \text{3SAT} &\leq_p \text{INDSET} \leq_p \text{VERTEXCOVER} \\ &\leq_p 0/1 - \text{IP}. \end{aligned}$$