

Lecture Notes 4:**Cook-Levin Theorem, Decision vs. Search, Time Hierarchy****Reading.**

- Arora-Barak § 2.2, 2.4-2.5

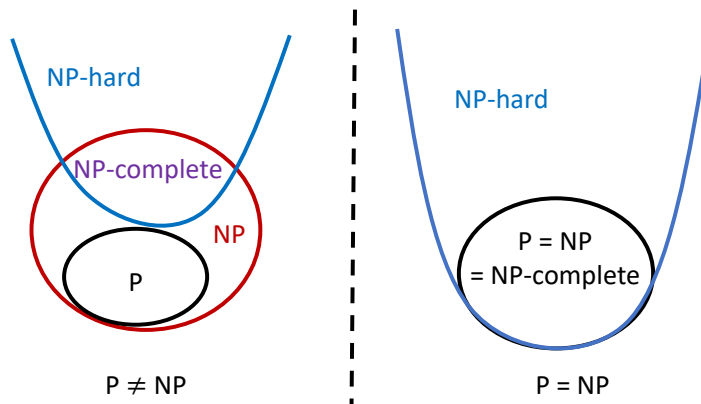
Last time: Reductions, NP-completeness, Cook-Levin Theorem

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 $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ such that

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

Definition 2. A language B is

- **NP-hard** if $A \leq_p B$ for every language $A \in \text{NP}$, and
- **NP-complete** if B is **NP-hard** and $B \in \text{NP}$.

**1 Satisfiability**

Last time, we sketched an argument that the following language is **NP-complete**:

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

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: $x_1, \dots, x_n$

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

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

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

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 $f : \{0, 1\}^k \rightarrow \{0, 1\}$ 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

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

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

2 Cook-Levin Theorem

Theorem 3 (Cook-Levin Theorem). *SAT is NP-complete.*

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*.

SAT $\in$ 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 NP-hard. Let $L \in \text{NP}$ 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

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

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:

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

We're going to build a formula $\varphi_x(s, h, c)$ such that $\varphi_x(s, h, c) = 1$ if and only if all of these conditions on the tableau hold.

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.

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

2. The final row is accepting.

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

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 .

3 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 be used to prove the **NP**-completeness of many other natural problems. For instance:

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

4 An NP-Completeness Example

In the *hitting set* HS problem, you are given a “ground set” U , natural number k , and sets $S_1, \dots, S_\ell \subseteq U$. Your goal is to determine whether there exists a set $T \subseteq U$ such that $|T| \leq k$ and $T \cap S_i \neq \emptyset$ for every $i \in [\ell]$. We can describe this by the language

$$\text{HS} = \{ \langle U, S_1, \dots, S_\ell, k \rangle \mid \exists T \subseteq U \text{ s.t. } |T| \leq k, T \cap S_i \neq \emptyset \forall i \in [\ell] \}.$$

This is in **NP**: The certificate is the description of the set T . To verify it, just check that $|T| \leq k$ and that T indeed intersects every set S_i .

To show that it's NP-hard, we can reduce from 3SAT. Let $\varphi = (\ell_1^1 \vee \ell_2^1 \vee \ell_3^1) \wedge \dots \wedge (\ell_1^m \vee \ell_2^m \vee \ell_3^m)$ be a 3SAT instance on variables $x_1, \dots, x_n$. Our reduction computes the following HS instance:

$$\begin{aligned} U &= \{x_1, \overline{x_1}, \dots, x_n, \overline{x_n}\} \\ S_1 &= \{\ell_1^1, \ell_2^1, \ell_3^1\} \\ &\dots \\ S_m &= \{\ell_1^m, \ell_2^m, \ell_3^m\} \\ S_{m+1} &= \{x_1, \overline{x_1}\} \\ &\dots \\ S_{m+n} &= \{x_n, \overline{x_n}\} \\ k &= n. \end{aligned}$$

This reduction takes time polynomial in the instance size. To show that it is correct, we need to show that φ is satisfiable $\iff \langle U, S_1, \dots, S_{m+n}, k \rangle \in \text{HS}$.

$\Rightarrow$: Let $(b_1, \dots, b_n)$ be a satisfying assignment to φ . We construct a hitting set T as follows. For each $i = 1, \dots, n$, if $b_i = 1$, place x_i in T . If $b_i = 0$, place $\overline{x_i}$ in T . This gives a hitting set of size $n = k$.

$\Leftarrow$: Let T be a hitting set of size $k = n$. Since T must intersect $S_{m+1}, \dots, S_{m+n}$, it must contain exactly one element of each pair. Construct the satisfying assignment $(b_1, \dots, b_n)$ by taking $b_i = 1 \iff x_i \in T$.

5 Decision vs. Search

For much of this class, we focus on decision problems (languages), i.e., those with yes/no answers. Why do we do this when computation encompasses so many other types of problems?

1. This is the simplest setting for studying complexity, where we already can't answer most of the interesting questions we want to ask.
2. Solving decision problems actually lets us solve other types of computational problems!

Today, we'll address the second point by describing about a generic "search-to-decision" reduction for NP. But to set up this general reduction, let's start by looking at a concrete problem.

Recall the decision problem

$$\text{SAT} = \{\varphi \mid \varphi \text{ has a satisfying assignment}\}.$$

This corresponds to a natural search problem:

SAT – Search: Given a CNF φ , output a satisfying assignment to φ if one exists, and $\perp$ otherwise.

Theorem 4. *If $\text{SAT} \in \mathbf{P}$, then there exists a poly-time algorithm solving SAT – Search.*

Proof. Let A be a poly-time algorithm deciding SAT. The idea is to repeatedly apply A to identify a satisfying assignment bit-by-bit.

First some notation: For a formula φ and $b \in \{0, 1\}$, let $\varphi|_{x_1=b}$ be the formula φ with variable x_1 replaced by the constant b . (And similarly if we want to fix more variables.)

Observe that:

$$\varphi \in \text{SAT} \iff \varphi|_{x_1=0} \in \text{SAT} \quad \text{or} \quad \varphi|_{x_1=1} \in \text{SAT}.$$

The following procedure uses A to construct a satisfying assignment $b_1, \dots, b_n$ to a formula $\varphi(x_1, \dots, x_n)$.

Alg Find(φ):

If $A(\varphi) = 0$: Output $\perp$.

For $i = 1, \dots, n$:

 If $A(\varphi|_{x_1=b_1, \dots, x_{i-1}=b_{i-1}, x_i=0}) = 1$: // i.e., $\varphi|_{x_1=b_1, \dots, x_i=0}(x_{i+1}, \dots, x_n) \in \text{SAT}$

 Set $b_i = 0$

 Else:

 Set $b_i = 1$

Output $(b_1, \dots, b_n)$

This algorithm uses $n + 1$ invocations of the poly-time algorithm A , so it runs in poly-time. $\square$

This result is just an example of a more general phenomenon. To state this, we need the following definition of an **NP** search problem.

Definition 5. Let $f : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}$ be a poly-time computable function (“verification function”), and let $p : \mathbb{N} \rightarrow \mathbb{N}$ be a polynomial (“certificate length”). The **NP** search problem associated to f and p is as follows.

Given $x \in \{0, 1\}^*$, find a $u \in \{0, 1\}^{p(|x|)}$ such that $f(x, u) = 1$ (if one exists).

For example, we can capture the SAT – Search problem by taking $f(\varphi, u) = 1 \iff u$ satisfies the CNF φ .

As another example, we can take $f(\langle G, k \rangle, u) = 1 \iff u$ is an independent set for G of size k .

Theorem 6. If $\mathbf{P} = \mathbf{NP}$, then every **NP** search problem can be solved in poly time.

Proof. Consider an **NP** search problem defined by verification function f and certificate length p . The idea will, again, be to construct a certificate bit-by-bit using our ability to solve a decision problem associated to f in poly-time.

One choice of decision problem that works here is

$$L_f = \{ \langle x, b \rangle \mid \exists z \in \{0, 1\}^{p(|x|)-|b|} \text{ s.t. } f(x, b \circ z) = 1 \}.$$

That is, L_f consists of pairs (x, b) such that there exists a way to extend the prefix b to a certificate $u = b \circ z$ for which $f(x, u) = 1$.

The language $L_f \in \mathbf{NP}$ because f is poly-time computable. Therefore, if $\mathbf{P} = \mathbf{NP}$, there is a poly-time algorithm A deciding L_f . This gives us the following search-to-decision reduction:

Alg Find(x):

If $A(x) = 0$: Output $\perp$.

For $i = 1, \dots, p(|x|)$:

 If $A(x, b_1, \dots, b_{i-1}, 0) = 1$: // i.e., $\langle x, b_1, \dots, b_{i-1}, 0 \rangle \in L_f$

 Set $b_i = 0$

 Else:

 Set $b_i = 1$

Output $(b_1, \dots, b_{p(|x|)})$. $\square$

Next time: Hierarchy theorems, Ladner’s theorem