COMP3630/6360: Theory of Computation
Semester 1, 2022
The Australian National University

Time Complexity

This lecture covers Chapter 10 of HMU: Time Complexity

- NP-Hardness
- Polytime Reductions
- SAT is NP-hard

Additional Reading: Chapter 10 of HMU.

$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$

Question 1.1 ($\mathbf{P} = \mathbf{NP}$ problem)

Can we simulate a non-deterministic TM (NTM) in polynomial time on a (deterministic) TM?

Recall:

- **P**—problems that can be solved in polynomial time on a TM.
- **NP**—problems that can be solved in polynomial time on an NTM.

At this point, no one knows for sure, but "no" might be a good bet.

NP-complete problems

This is about decision problems (problems with yes/no answers). Equivalently, solving the membership problem $x \in L$.

Obviously $\mathbf{P} \subseteq \mathbf{NP}$.

Nobody knows for sure whether $\mathbf{NP} \subseteq \mathbf{P}$

Intuitively, **NP**-complete problems are the “hardest” problems in **NP**.

P Reducibility

Recall how we use mapping-reducibility to transfer (un)decidabilty from one problem to the next.

Definition 1.2

$f : \Sigma^* \longrightarrow \Sigma^*$ is a *polynomial time computable* (or **P** *computable*) function if some polynomial time TM M exists that halts with just $f(w)$ on its tape, when started on any input $w \in \Sigma^*$.

Definition 1.3

$A \subseteq \Sigma_1^*$ is *polynomial time mapping reducible* (or **P** *reducible*) to $B \subseteq \Sigma_2^*$, written $A \leq_{\mathbf{P}} B$, if a **P** computable function $f : \Sigma_1^* \longrightarrow \Sigma_2^*$ exists that is also a reduction (from A to B).

$\mathbf{P}$ Reducibility cont.

Theorem 1.4

If $A \leq_{\mathbf{P}} B$ and $B \in \mathbf{P}$ then $A \in \mathbf{P}$.

Proof.

To decide $w \in A$ first compute $f(w)$ (in $\mathbf{P}$) where f is the $\mathbf{P}$ reduction from A to B, and then run a $\mathbf{P}$ decider for B. This is still in $\mathbf{P}$ because $p_1(p_2(n))$ is a polynomial if $p_1(n)$ and $p_2(n)$ are. □

NP-Completeness

Definition 1.5

A language B is **NP**-*complete* if

1. $B \in \mathbf{NP}$
2. *every* $A \in \mathbf{NP}$ is **P** reducible to B.

Theorem 1.6

*If B is **NP**-complete and $B \in \mathbf{P}$ then $\mathbf{P} = \mathbf{NP}$.*

Theorem 1.7

*If B is **NP**-complete and $B \leq_{\mathbf{P}} C$ for $C \in \mathbf{NP}$, then C is **NP**-complete.*

Proof.

Polynomial time reductions compose. □

NP-Completeness

If there are any problems in $\mathbf{NP} \setminus \mathbf{P}$, the **NP**-complete problems are all there.

Every **NP**-complete problem can be translated in deterministic polynomial time to every other **NP**-complete problem.

So, if there is a **P** solution to one **NP**-complete problem, there is a **P** solution to every **NP** problem.

NP-Hardness by Reduction

Typical method: Reduce a known **NP**-hard problem P_1 to the new problem P_2 .

Basic Proof Strategy

NP-completeness is a good news/bad news situation.

- Good news: The problem is in **NP**!
- Bummer: The problem is **NP**-hard!

So, a typical **NP**-completeness proof consists of two parts:

1. Prove that the problem is in **NP** (i.e., it has **P** verifier).
2. Prove that the problem is at least as hard as other problems in **NP**.

A TM can simulate an ordinary computer in polynomial time, so it is sufficient to describe a polynomial-time checking algorithm that will run on any reasonable model of computation.

NP-Hardness

A problem is **NP**-*hard* if having a polynomial-time solution to it would give us a polynomial solution to every problem in **NP**.

Prove that the problem is **NP**-hard: The usual strategy is to find a polynomial-time reduction of a known **NP**-hard problem (say P_1) to the problem in question (say P_2).

The goal is to show that P_2 is at least as hard (in terms of polynomial vs. super-polynomial time) as P_1.

If P_1 can be translated to an equivalent problem P_2 in polynomial time, then a polynomial-time solution to P_2 would also give a polynomial-time solution to P_1: First reduce P_1 to P_2, then solve it.

NP-hardness cont.

Repeated warning: Make sure you are reducing the known problem to the unknown problem!

In practice, there are now thousands of known **NP**-complete problems. A good technique is to look for one similar to the one you are trying to prove **NP**-hard.

Boolean Formulae

Let $Prop = \{x, y, \ldots\}$ be a (finite) set of *Boolean variables* (or *propositions*).
A CFG for Boolean formulae over *Prop* is:

$$\phi \to p \mid \phi \wedge \phi \mid \neg\phi \mid (\phi)$$
$$p \to x \mid y \mid \ \ldots$$

We use abbreviations such as

$$\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2) \qquad \phi_1 \Rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$$
$$\text{FALSE} = (x \wedge \neg x) \qquad \text{TRUE} = \neg\text{FALSE}$$

Technically, we could handle countably infinite sets *Prop* if we had a naming scheme for variables, say, x_n for binary representations n of natural numbers.

Semantics of Boolean Formulae

A Boolean formula is either **TT** (for "true") or **FF** (for "false"), possibly depending on the interpretation of its propositions. Let $\mathbb{B} = \{\mathbf{FF}, \mathbf{TT}\}$.

Definition 2.1

An *interpretation* (of *Prop*) is a function $\pi : \mathit{Prop} \longrightarrow \mathbb{B}$.
For Boolean formulae ϕ we define π *satisfies* ϕ, written $\pi \models \phi$, inductively by:
Base: $\pi \models x$ iff $\pi(x) = \mathbf{TT}$.
Induction:

- $\pi \models \neg\phi$ iff $\pi \not\models \phi$.
- $\pi \models \phi_1 \wedge \phi_2$ iff both $\pi \models \phi_1$ and $\pi \models \phi_2$.
- $\pi \models (\phi)$ iff $\pi \models \phi$.

ϕ is *satisfiable* if there exists an interpretation π such that $\pi \models \phi$.

SAT—An **NP**-Complete Problem

$$SAT = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula } \}$$

Theorem 2.2

SAT is **NP***-complete.*

Proof of $SAT \in$ NP.

If $\pi \models \phi$ we use $\langle \pi \rangle$ as certicate. Had we chosen a countably infinite *Prop*, we'd restrict π to the propositions occurring in ϕ. $\square$

Proof of **NP**-Hardness of *SAT*

Let $A \in \mathbf{NP}$. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a deciding NTM with $L(M) = A$ and let p be a polynomial such that M takes at most $p(|w|)$ steps on any computation for any $w \in \Sigma^*$.

Construct a **P** reduction from A to *SAT*. On input w a Boolean formula ϕ_w that describes M's possible computations on w.

M accepts w iff ϕ_w is satisfiable. The satisfying interpretation resolves the nondeterminism in the computation tree to arrive at an accepting branch of the computation tree.
Remains to be done: define ϕ_w.

Proof of **NP**-Hardness of *SAT* cont.

Recall that M accepts w if an $n \leq p(|w|)$ and a sequence $(C_i)_{0<i\leq n}$ of IDs exist, where

1. $C_1 = q_0 w$,
2. each C_i can yield C_{i+1}, and
3. C_n is an accepting ID.

Let $C = Q \cup \Gamma \cup \{\#\}$. Each C_i can be represented as a $\#$-enclosed string over alphabet C no longer than $n + 3$.

ϕ_w

The Boolean formula ϕ_w shall represent *all* such sequences $(C_i)_{0<i\leq n}$ beginning with q_0w.

$$\phi_w = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

ϕ_{cell}

…describes an n^2 grid using propositions $Prop = \{\ x_{i,k,s} \mid i,k \in \{1,\ldots,n\} \wedge s \in C\ \}$.

$$\phi_{\text{cell}} = \bigwedge_{0<i,k\leq n} \left(\bigvee_{s\in C} x_{i,k,s} \wedge \bigwedge_{s\neq t\in C} (\neg x_{i,k,s} \vee \neg x_{i,k,t}) \right)$$

Row i in the grid corresponds to the ID C_i. Unused tape cells are blank.
Every grid cell contains exactly one symbol or a state.

ϕ_{start}

... specifies that the first row of the grid contains $q_0 w$ where $w = w_1 \ldots w_{|w|}$:

$$\phi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \bigwedge_{2<i\leq|w|+2} x_{1,i,w_{i-2}} \wedge \bigwedge_{|w|+2<i\leq n-1} x_{1,i,\sqcup} \wedge x_{1,n,\#}$$

ϕ_{move}

...ensures that C_i yields C_{i+1} by describing *legal* 2×3 windows of cells.

$$\phi_{\text{move}} = \bigwedge_{0<i,k<n} \bigvee_{\begin{array}{|c|c|c|}\hline a_1 & a_2 & a_3 \\ \hline a_4 & a_5 & a_6 \\ \hline \end{array} \text{ is legal}} \left(\begin{array}{lll} x_{i,k-1,a_1} & \wedge\ x_{i,k,a_2} & \wedge\ x_{i,k+1,a_3} \wedge \\ x_{i+1,k-1,a_4} & \wedge\ x_{i+1,k,a_5} & \wedge\ x_{i+1,k+1,a_6} \end{array} \right)$$

what is legal depends on the transition function δ.

ϕ_{accept}

...states that the accept state is reached:

$$\phi_{\text{accept}} = \bigvee_{0<i,k\leq n} x_{i,k,q_{\text{accept}}}$$

Finally we check that the size of ϕ_w is polynomial in $|w|$ and that ϕ_w is constructable in polynomial time.

—The End—