Notes, 10(a)

ECE 606

In the last lecture, we defined complexity classes, which are sets of decision problems.

Given a complexity class $\mathcal{C}$, when we say a problem $q \in \mathcal{C}$, that can legitimately be seen as a characterization of an upper-bound of the computational hardness of the problem q.

E.g., if we identify that $q \in \mathbf{P}$, then we know that we do not need an algorithm that is any more inefficient than polynomial-time.

Similarly, if $q \in \mathbf{NP}$, we need no worse than a polynomial-time non-deterministic algorithm for q.

In this lecture, we consider an approach to lower-bounding the computational hardness (difficulty) of a (decision) problem.

Towards this, we begin with a way to compare the computational hardness of one decision problem against another decision problem. We then extend the approach to comparing the computational hardness of a problem against an entire complexity class.

Reductions

We now consider comparing two *problems* from the standpoint of computational hardness. Then, we extend such a notion to the computational hardness of a problem compared to every problem in a complexity class.

Terminology: *hard* means difficult. For example, we may say, "problem B is computationally at least hard as problem A."
as

To start us off: suppose a problem can only be one of "hard" or "easy." Then, a logic-based notion of what it means for a problem B to be at least as hard as problem A is:

$$B \text{ is easy} \implies A \text{ is easy.}$$

The only possibilities that the above implication admits:

(1) A is easy and B is easy.
(2) A is hard and B is hard.
(3) A is easy and B is hard.

It does not admit:

(4) A is hard and B is easy.

Terminology and notation: if indeed we are able to establish that B is easy implies A is easy, then we say that there exists a *reduction* from A to B. Also, we adopt the symbol "$\leq$" and write $A \leq B$.

Note: we still need to more precisely clarify what we mean by "easy" and "hard." One way to do this is to more precisely specify what it means for A to reduce to B, i.e., $A \leq B$.

E.g., "easy" may mean "algorithm exists for the problem," and "hard" means "algorithm does not exist for the problem." Now, what we mean by $A \leq B$ is: if an algorithm exists for B, then one exists for A.

The Cook reduction

We write this as "$\leq_c$." Under this reduction, we equate "easy" with "$\in \mathbf{P}$." That is, we say, for decision problems A and B, that $A \leq_c B$ if: $B \in \mathbf{P} \implies A \in \mathbf{P}$.

Example: consider HAMPATH and HAMPATHSTARTEND:

HAMPATH: given as input a non-empty undirected graph G, is there a simple path in G of all its vertices?
HAMPATHSTARTEND: given as input a non-empty undirected graph G and two distinct vertices in it, a, b, is there a simple path $a \rightsquigarrow b$ in G of all its vertices?

Claim 1. HAMPATH $\leq_c$ HAMPATHSTARTEND.

To prove the claim, assume that H_{SE} is a polynomial-time algorithm for HAMPATHSTARTEND. Then, we can show that HAMPATH $\in \mathbf{P}$ by construction.

$H(G = \langle V, E \rangle)$

if $|V| = 1$ **then return** true
foreach $u \in V$ **do**
    foreach $v \in V \setminus \{u\}$ **do**
        if $H_{SE}(G, u, v) =$ true **then return** true
return false

Claim 2. HAMPATHSTARTEND $\leq_c$ HAMPATH.

Assume that H is a polynomial-time algorithm for HAMPATH. Then, we can show that HAMPATHSTARTEND $\in \mathbf{P}$ by proposing the following algorithm, H_{SE}, for it.

$H_{SE}(G = \langle V, E \rangle, a, b)$

if $a = b$ *or* $|V| = 1$ **then return** false
Let $V' \leftarrow V \cup \{x, y\}$, where $x, y \notin V$
Let $E' \leftarrow E \cup \{\langle x, a \rangle, \langle b, y \rangle\}$
return $H(\langle V', E' \rangle)$

LongSimplePath: given input (i) undirected $G = \langle V, E \rangle$, (ii) two distinct vertices $a, b \in V$, and, (iii) $k \in \mathbb{Z} \cap [1, |V| - 1]$, does there exist simple $a \rightsquigarrow b$ of $\geq k$ edges?

Claim 3. HamPathStartEnd $\leq_c$ LongSimplePath.

Proof. Invoke a polynomial-time algorithm for LongSimplePath with inputs $G, a, b, |V|-1$, and return whatever it returns. $\square$

Is it possible that $A \leq_c B$, but $B \not\leq_c A$?

Yes. Let $A =$ LongSimplePath. Let $B =$ Halt, i.e., the decision-problem: given as input an algorithm α and a string x, does running α with input x eventually halt (terminate)?

Another example: let ShortSimplePath be the problem: given inputs (i) undirected $G = \langle V, E \rangle$, (ii) two distinct vertices $a, b \in V$, and, (iii) $k \in \mathbb{Z} \cap [1, |V| - 1]$, does there exist simple $a \rightsquigarrow b$ of $\leq k$ edges?

Then: ShortSimplePath $\leq_c$ LongSimplePath.

But: LongSimplePath $\leq_c$ ShortSimplePath $\implies$ $\mathbf{P} = \mathbf{NP}$, which is highly unlikely to be true.

In other words, if we adopt the customary assumption $\mathbf{P} \neq \mathbf{NP}$, then LongSimplePath $\not\leq_c$ ShortSimplePath.

So shall we continue using $\leq_c$ as our way of comparing the computational hardness of decision problems?

Turns out that $\leq_c$ is not what is customarily used in this context. Because it has a highly undesirable property.

Claim 4. *Let p be a decision problem and q its complement. Then $p \leq_c q$.*

Definition: a set S is said to be *closed* under a binary operation $\boxplus$ if: given any two $e_1, e_2 \in S$, it is the case that $e_1 \boxplus e_2 \in S$.

e.g., $\mathbb{R}$ is closed under, $-$, i.e., subtraction, but $\mathbb{Z}^+$ is not.

Definition: we say that a complexity class $\mathcal{C}$ is closed under a reduction $\leq$ if: given any two decision problems p_1, p_2, it is the case that $p_1 \leq p_2$ and $p_2 \in \mathcal{C} \implies p_1 \in \mathcal{C}$.

And now, our punchline:

Claim 5. *If **NP** is closed under $\leq_c$, then **NP** $=$ **co-NP**.*

So, **NP** $\neq$ **co-NP** $\implies$ **NP** is not closed under $\leq_c$.