Lecture 3 Discussion

Instructor: Karteek Sreenivasaiah

23th April 2020

Summary

The main topics covered in the third NPTEL lecture:

- Many-one reductions
- Notions of NP hardness and completeness
- SAT and 3SAT

Reductions

How can we show that a problem A is easier than problem B?

Reductions

How can we show that a problem A is easier than problem B?

- A natural idea is to show:
  If we can solve problem B, then we can also solve problem A.
  (with maybe a small amount of extra resources used)

Reductions

How can we show that a problem A is easier than problem B?

- A natural idea is to show:
  If we can solve problem B, then we can also solve problem A.
  (with maybe a small amount of extra resources used)

- Alternate wording:
  If we can *easily modify* instances of problem A to look like instances of problem B.

- This idea is formalized to obtain the notion of *reductions*.
  A reduction is a function that takes an instance of problem A and makes it look like an instance of problem B.

Reductions

Fix the model of computation as Turing machines.

Many-one Reductions

Let $A, B \subseteq \{0,1\}^*$ be two languages. We say A reduces to B via a many-one reduction, denoted $A \leq_m B$, if and only if $\exists$ a function $f : \{0,1\}^* \to \{0,1\}^*$ such that:

- f is computable by a Turing machine
- $\forall x \in \{0,1\}^*$, we have $x \in A \iff f(x) \in B$

In other words: the *Yes* instances of A are mapped to *Yes* instances of B, and the *No* instances of A are mapped to *No* instances of B.

Reductions

Example 1: Let A, B be languages, and $B \in \mathrm{P}$.
If $A \leq_m B$ via a polynomial time computable reduction f, then we can conclude $A \in \mathrm{P}$.

Reductions

Example 1: Let A, B be languages, and $B \in \mathrm{P}$.
If $A \leq_m B$ via a polynomial time computable reduction f, then we can conclude $A \in \mathrm{P}$.

Proof:

Let $B \in \mathrm{P}$ via machine M with running time $O(n^k)$.
Construct a machine N to decide A as follows:

- Let x be an input. (we need to decide if $x \in A$)
- Compute $y = f(x)$
- Check if $y \in B$ by running M on y.
- Accept if M accepts. Reject if M rejects.

Analysis: Let $|x| = n$. If computing f takes time n^c, then $|y| \leq n^c$.
Machine M takes time $O(|y|^k)$. This equals $O(n^{ck})$.
Hence running time N is $O(n^c + n^{ck})$.

Reductions

Example:

Let A, B be languages, and $B \in \mathsf{DTIME}(n^5)$.

Suppose we want to show $A \in \mathsf{DTIME}(n^5)$ using a reduction to B.

Then the reduction itself had better not exceed time $O(n^5)$! Further, it had better not write strings that are too long.

Reductions

Example:
Let A, B be languages, and $B \in \mathsf{DTIME}(n^5)$.

Suppose we want to show $A \in \mathsf{DTIME}(n^5)$ using a reduction to B.

Then the reduction itself had better not exceed time $O(n^5)$! Further, it had better not write strings that are too long.

We can write:
If $A \leq_m B$ via a reduction f computable in time $O(n)$, then $A \in \mathsf{DTIME}(n^5)$.

Note: We restricted the length of the string output by the reduction by demanding that the reduction run in time $O(n)$.
(There are other ways to achieve this, but we do not go into it here)

Reductions

The complexity of reductions hint at how *similar* two languages are.

Example:
Let $A \leq_m B$ via a reduction f that uses time $\Theta(n^3)$,
and $A \leq_m C$ via a reduction g that uses time $O(n)$.

Reductions

The complexity of reductions hint at how *similar* two languages are.

Example:
Let $A \leq_m B$ via a reduction f that uses time $\Theta(n^3)$,
and $A \leq_m C$ via a reduction g that uses time $O(n)$.

It takes only $O(n)$ time to *convert* an instance of A to an instance of C. But it takes n^3 time to convert it to an instance of B.

So intuitively, A and C are very similar to each other as opposed to A and B.

However, there might be a more clever reduction from A to B that uses lesser time!

Hardness and Completeness

A problem L is *hard* for a class $\mathcal{C}$ means:
L is harder than all problems in $\mathcal{C}$.

Formally:
A language L is hard for $\mathcal{C}$ if and only if:

- $\forall A \in \mathcal{C}$, we have $A \leq_m C$ via a function f
- Complexity of f is *appropriately* bounded.

Hardness and Completeness

A problem L is *hard* for a class $\mathcal{C}$ means:
L is harder than all problems in $\mathcal{C}$.

Formally:
A language L is hard for $\mathcal{C}$ if and only if:

- $\forall A \in \mathcal{C}$, we have $A \leq_m C$ via a function f
- Complexity of f is *appropriately* bounded.

In the case of NP: The reduction f has to be computable in polynomial time.

Hardness and Completeness

A problem L is *complete* for a class $\mathcal{C}$ means:

- L is harder than all problems in $\mathcal{C}$ under suitable reductions.
- L is in the class C.

Intuitively: L completely captures the complexity of the class $\mathcal{C}$.

Hardness and Completeness

A problem L is *complete* for a class $\mathcal{C}$ means:

- L is harder than all problems in $\mathcal{C}$ under suitable reductions.
- L is in the class C.

Intuitively: L completely captures the complexity of the class $\mathcal{C}$.

Example: A language L is NP-complete if and only if:

- $\forall A \in \mathrm{NP}$, $A \leq_m L$ in polynomial time.
- $L \in \mathrm{NP}$

Hardness and Completeness

A problem L is *complete* for a class $\mathcal{C}$ means:

- L is harder than all problems in $\mathcal{C}$ under suitable reductions.
- L is in the class C.

Intuitively: L completely captures the complexity of the class $\mathcal{C}$.

Example: A language L is NP-complete if and only if:

- $\forall A \in \mathrm{NP}$, $A \leq_m L$ in polynomial time.
- $L \in \mathrm{NP}$

Exercise

Exercise 1: Let L be NP-complete under polynomial time reductions. Then we have: If $L \in \mathrm{P}$, then $\mathrm{P} = \mathrm{NP}$.

Exercise 2: Show that "$\mathrm{P} = \mathrm{NP}$" under exponential time reductions.

More precisely: Show a language in P that is NP-complete under exponential time reductions.

Thank you!