Limits of Computation

19 - How Hard is a Problem?

Bernhard Reus

The story so far

- We have seen **NP** contains problems that seem intractable.
- We don't know whether **P** = **NP** i.e. whether the seemingly intractable problems are actually tractable.
- Today we develop at least an "angle" to attack this problem.

We don't know

NP \ P

but we can look at the "hard" problems in there!

"Hard" Problems in (**N**)**P**

THIS TIME

- add complexity to problem reduction (of Lecture 9): *"if you can solve B then you can solve A as well"*. (*A* is not harder than *B*) *A≤**B***
- what are the "hardest" problems in (**N**)**P** (called *complete problems*)?
- why they are important?

see Lecture 9, Slides 18ff

Reduction


Emil Leon Post (1897-1954)

- to be able to show that problem *A* is no harder than problem *B, reduce the problem.*

decide membership in A in terms of B using f

- many-one reduction from *A* to *B (A≤B)* is given by a *total computable* function *f* on *(problem input)* data such that $\boxed{\mathtt{d} \in A \text{ if and only if } f(\mathtt{d}) \in B.}$
- *Many-one* as *f* maps (many values) to one value and "is in B?" can only be asked once at the end.

there are more general forms of reduction

Very Simple Example

- reduce the *Travelling Salesman Problem* where start = finish city to the *Travelling Saleman Problem* with different start and finish cities.
- *f([G,K,A]) = [G,K,A,A]*

encoding of graph

encoding of city start=finish

encoding of mileage limit

Reduction for Complexity Comparison

- For reduction between problems in a complexity class, the reduction must be "effective" ...
- ... i.e. must be computable (as before) ***and*** not carry out of the complexity class at hand.
- So for **NP** or **P** the reduction function *f* should be computable *in polynomial time*.
- we write $A \leq_p B$

R. Karp

"polynomial time reduction"
or Karp-reduction

Complete Problems

Definition (Complete Problem). For any complexity class $\mathscr{C}$ a problem H is called *complete* for the class $\mathscr{C}$ if problem H is itself in $\mathscr{C}$ and is "hardest" in $\mathscr{C}$ in the sense that for all other problems $A \in \mathscr{C}$ we have that $A \leq H$ (for an *appropriate* instantiation of reduction).

Important feature of **NP**-complete problems:
To show that **P** = **NP** it suffices to show that *any one* **NP**-complete problem *H* is actually in **P**.

First, we prove closure of **N**(**P**) under poly-time reduction:

Downward-closure (N)P

Theorem **(Downward closure of (N)P).** *If $A \leq_P B$ and B is in* ***NP*** *then A is also in* ***NP****. Similarly, If $A \leq_P B$ and B is in* ***P*** *then A is also in* ***P****.*

Proof. So assume $A \leq_P B$ and B is in **NP**.

<r>x computes $f(x)$. reduction function

```
p read xc {
  x := hd xc;
  c := hd tl xc;
  B
}
write y
```

verifier for B

```
q read xc {
  x := hd xc;
  c := hd tl xc;
  x := <r> x;        // x := f(x)
  B                  // run p on input f(x)
}
write y
```

verifier for A

Downward-closure NP

so we have now a program *q* such that

$$[\![q]\!]\,(x,c) = [\![p]\!]\,(f(x),c)$$

and for that it holds that

$$\begin{aligned}
[\![q]\!]\,(x,c) = \text{true for a certificate } c \quad & \text{iff} \quad [\![p]\!]\,(f(x),c) = \text{true for a certificate } c \\
& \text{iff} \quad f(x) \in B \\
& \text{iff} \quad x \in A
\end{aligned}$$

as required.

by reduction

Remains to check runtime

Assume program `r` runs in polynomial time $p_1(n)$
Assume program `p` runs in polynomial time $p_2(n)$

```
q read xc {
  x := hd xc;
  c := hd tl xc;
  x := <r> x;
  B
}
write y
```

body of p

$$
\begin{aligned}
time_q(d,c) &= 2+3+4+\text{“time for } \texttt{x := <r>x} \text{ ”} + p_2(|f(d)|) \\
&= 9+(1+p_1(|d|))+p_2(|f(d)|) \\
&\leq 10+p_1(|d|)+p_2(p_1(|d|) * |d|)
\end{aligned}
$$

$|f(d)| \leq p_1(|d|)*|d|$ why?

by inspecting how large a tree can be produced

so q runs in polynomial time

Why complete problems?

Theorem *If any **NP**-complete problem is already in **P** then **P** = **NP** (and the biggest open problem in theoretical computer science is solved).*

Proof: Let *H* be the **NP**-complete problem that is in **P**.
We only have to show that **NP** ⊆ **P** as the other direction holds anyway.
We'll give details on the next slide.

Proof

Let H be the **NP**-complete problem that, by assumption, is also in **P**.

show that $\mathbf{NP} \subseteq \mathbf{P}$

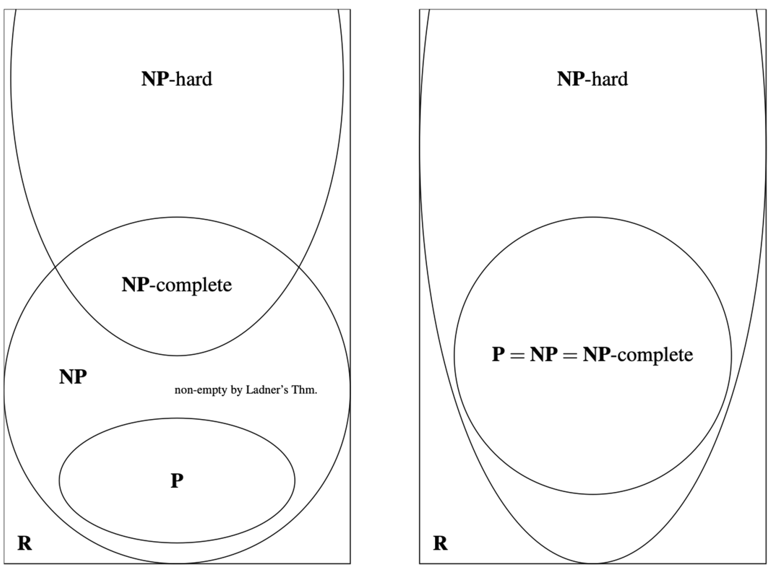
take any problem $A \in \mathbf{NP}$ and show that $A \in \mathbf{P}$ holds

$A \leq_P H$.

By the "Downward-closure of **P**" Thm.

$A \in \mathbf{P}$

NP-complete: 2 cases



$\mathbf{P} \neq \mathbf{NP}$ $\quad$ $\mathbf{P} = \mathbf{NP}$

END

Next time:
Example of $\mathbf{NP}$-complete problems