Intractibility and NP-Completeness II: Reductions

Lecture 17

Prof. Ronitt Rubinfeld
The Class NP

We might not know how to find a good input, but we can recognize a good one when we see it!
The Class NP

**NP**: Decision problems $D$ for which there exist polynomial time verification algorithm $V$

- if $D(x) = 1$ (x is a **yes-input**), then $\exists y, V(x,y) = True$
- if $D(x) = 0$ (x is a **no-input**), then $\forall y, V(x,y) = False$

**Notes**: $y$ must be “succinct” – poly size

$V$ is specific to $D$
The verifier –
Judge vs. detective

We don’t claim that V can *find* a certificate, just that V can check its correctness
(i.e., investigation can take months/years, trial takes days/weeks)
Is $P=NP$?

Exponential time solvable

Is this area empty?
A good reason to work on the problem

One of 7 Clay Millenium Prize problems
$1,000,000
NP-Complete Problems:

The “hardest” problems in NP
(picture assumes $P \neq NP$)

How to define NP-Complete?
Reductions: $B$ as hard as $A$

- $A \leq B$ if there is algorithm $R$ such that
  - $R$ maps inputs for $A$ into inputs for $B$ such that
    - “yes”- inputs for $A$ map to “yes”-inputs for $B$
    - “no”-inputs for $A$ map to “no”-inputs for $B$
  - $R$ is poly time
D is NP-Complete if

• $D \in \text{NP}$

• For all $A \in \text{NP}$, $A \leq D$ (i.e., $D$ is NP-hard)

A lot of problems!
How can we show anything to be NP-complete?

An important starting point:

Thm [Cook-Levin] 1971

c-SAT, SAT are NP-complete
SAT

Given Boolean formula $\Phi$

- $n$ boolean variables $x_1, \ldots, x_n$,
- ANDs, ORs, NOTs
- “and” of $m$ clauses (clause = “or” of literals)
- Is there a setting of the variables for which $\Phi$ evaluates to 1 (TRUE)?
Circuit-Satisfiability (cSAT)

Given a Boolean circuit $C$

- $n$ Boolean inputs, 1 output
- an acyclic diagraph whose vertices are AND, OR or NOT gates
- Is there a 0-1 setting of the inputs for which $C$ evaluates so that the output value is 1 (true)?

**Notation:** given an assignment $x_1, \ldots, x_n$

$C(x_1, \ldots, x_n)$ is the value of the output gate.
Cook-Levin Overview
(very high level “advertisement”)

• If \( L \in \text{NP} \), there is a poly time verifier \( V \) that checks certificates of “yes” instances

• For instance \( x \) (known) and for certificate \( y \) (variables with unknown settings), describe run of \( V \) on \( (x,y) \) via Boolean logic

  • Ensure validity of \( V \)’s run via even more Boolean logic

Fun for those that like to program in low level languages!
Cook-Levin Overview: the variables

• **Input and V’s program description** described by Boolean constants
  - E.g., $x_1=0,\ldots,x_n=1$

• **Certificate** described by Boolean variables
  - E.g., $y_1,\ldots,y_n$

• Each time step described by a “configuration” – use (many) Boolean variables to describe
  - contents of memory storage
  - Program counter, auxiliary machine storage
Cook-Levin Overview: the variables

Some examples of variables:

$Q_{i,q}$ denotes “At time $i$, V’s PC is $q$”

$S_{i,j,a}$ denotes “At time $i$, the $j$th location in memory is $a$”

There are a lot of these, but only polynomial number!
Cook-Levin Overview: the clauses

- Use Boolean formulas (or ckts) to check that
  - Each configuration description is valid
  - Start configuration encodes true input
  - Movement between configs legal according to V’s algorithm
  - End configuration is “PASS”
How can we show anything to be NP-complete?

An important starting point:

\textbf{Thm [Cook-Levin] 1971}

c-SAT, SAT are NP-complete

And a way to use it (from last lecture):

\textbf{Thm} If D NP-hard and D \leq C then C is NP-hard
An easier way to show NP-completeness

1. Show that $D \in \text{NP}$
2. For any $B$ that is NPC, show that $B \leq D$

Don’t forget to show that this is a legal reduction
What we’ve seen

- SAT, cSAT (Cook-Levin)
- 3SAT
- Clique
- Independent Set
- Vertex Cover

Right now
And more and more and more and more and more...

SAT, cSAT (Cook-Levin)

3SAT

Clique

Independent Set

Vertex Cover

3-colorability

Exact cover

Subset sum

And more and more and more and more and more and more...
Let’s end on an optimistic note
Two benefits of hard problems

• Source of randomness (Randomness and Complexity course 6.842)

• Cryptography!
  • A man that enjoys our sufferings