CS314 - NP-Hardness

Note Title 10/28/2013

"Efficiency"

Fundamental question: Are there hard problems?

How hard – unsolvable? polynomial?

Undecidability

Some problems can not be solved:

Halting Problem: Given a program P and input I, does P halt given input I or does it run forever?

Output: true or false

Why useful?

Note: Our program can't just simulate $\langle P \rangle$ running on $\langle I \rangle$.

Why? if we simulate P on I & it runs forever, we don't actually output

Thm [Turing '36]: The halting problem is undecidable.

(That is, no such algorithm can exist.)

Proof: (by contradiction)

Suppose we have such a program h:

$$h(P, I) = \begin{cases} 1 & \text{if } P \text{ halts on } I \\ 0 & \text{otherwise} \end{cases}$$

Now define another program:

```
g(i):
    if h(i,i) = 0
        return 0
    else
        loop forever
```

try $g(g) \rightsquigarrow h(g,g)$

Now what does g(g) do?

calls h(g,g):

h(g,g) → if 1, then g halts on input g
then g(g) should loop forever

h(g,g) outputs 0
then it should halt
↳ this means it looped forever!

↳ h cannot exist.

So... what next?

Clearly, lots of things are do-able in "polynomial time.

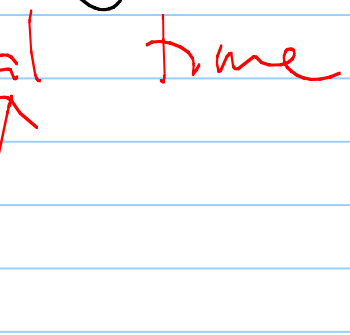
Some things are impossible.

But - is there anything in between?

Idea: exponential time

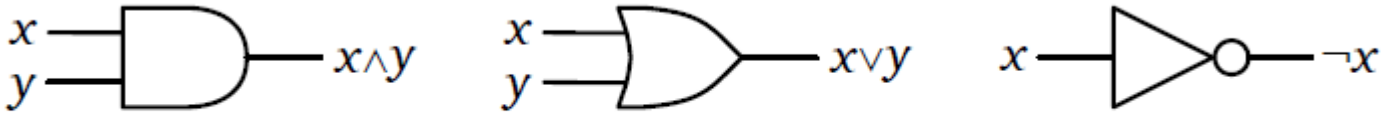
no n^c (poly)

$2^n \cdot n$



Candidate: Circuits

Boolean gates

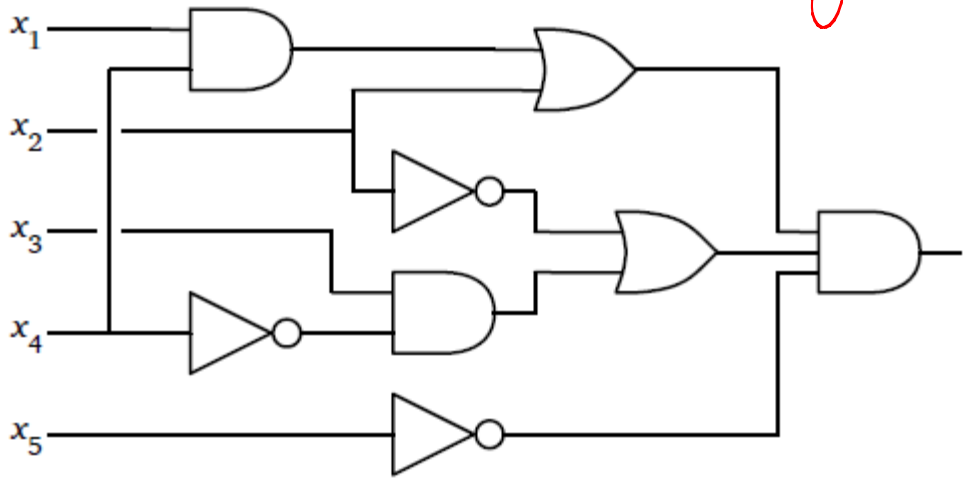


An AND gate, an OR gate, and a NOT gate.

- No loops
- Given inputs, can calculate output in linear time: basically evaluate in BFS order

Q: Given a boolean circuit, is there a set of inputs that evaluate to true?

Circuit satisfiability (Circuit SAT)



Best known algorithm:

try all 2^m possible inputs.

Running time: $2^m(n+m)$

current best known
↳ is no proof stating it couldn't be done faster.

P, NP, & co-NP

Consider decision problems: Yes or No.

P: set of decision problems that can be solved in polynomial time.

Ex: is this list sorted?

: is s connected to t in G?

NP: set of problems s.t., if the answer is yes, this can be checked in poly. time.

(So can verify a yes answer.)

Ex: Circuit SAT
given set of boolean inputs
can check that they give
yes answer.

---

co-NP: set of problems where
we can check a no answer
in poly time.

NP-Hard

Π is NP-hard $\iff$ If Π can be solved in polynomial time, then P=NP

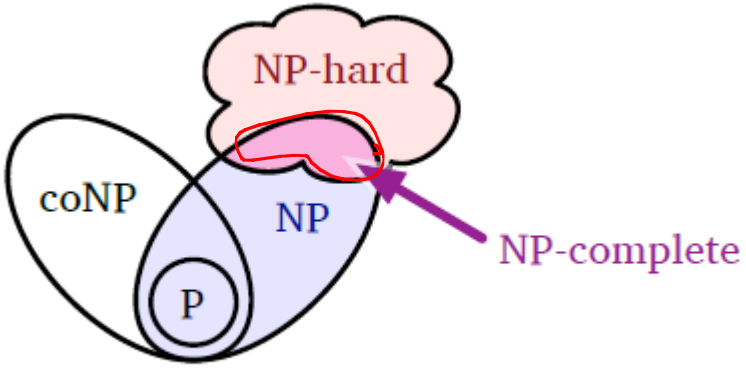
So if an NP-Hard problem can be solved in polynomial time, then any problem in NP can be solved in polynomial time.

(Paths story...)

P vs NP

A problem is NP-Complete if it is both:

- in NP
- NP-Hard

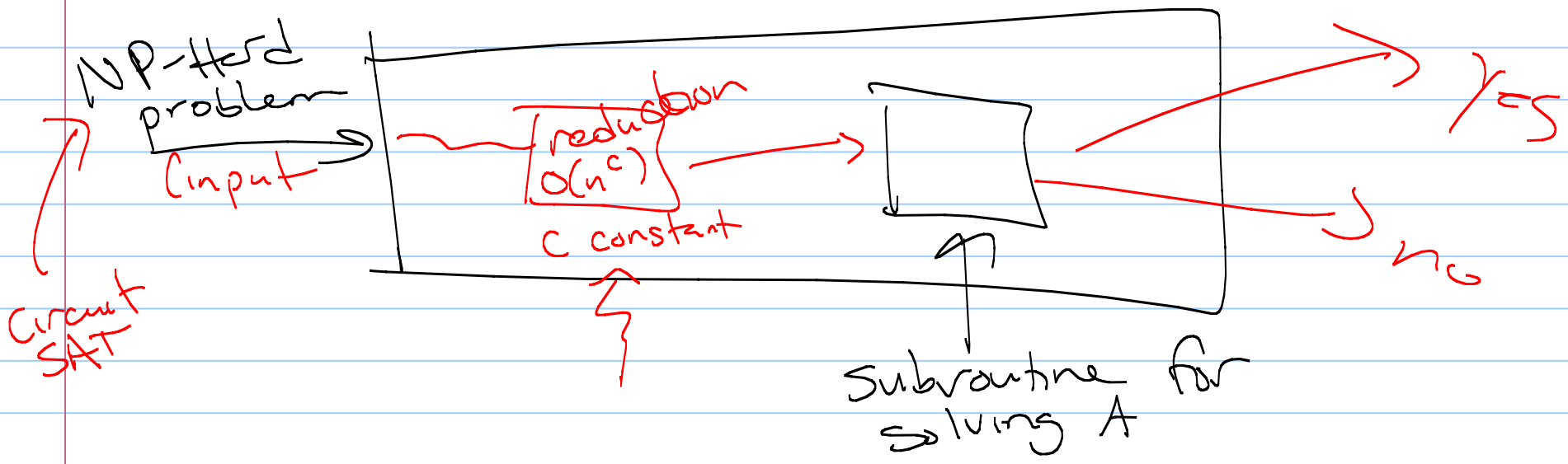


More of what we *think* the world looks like.

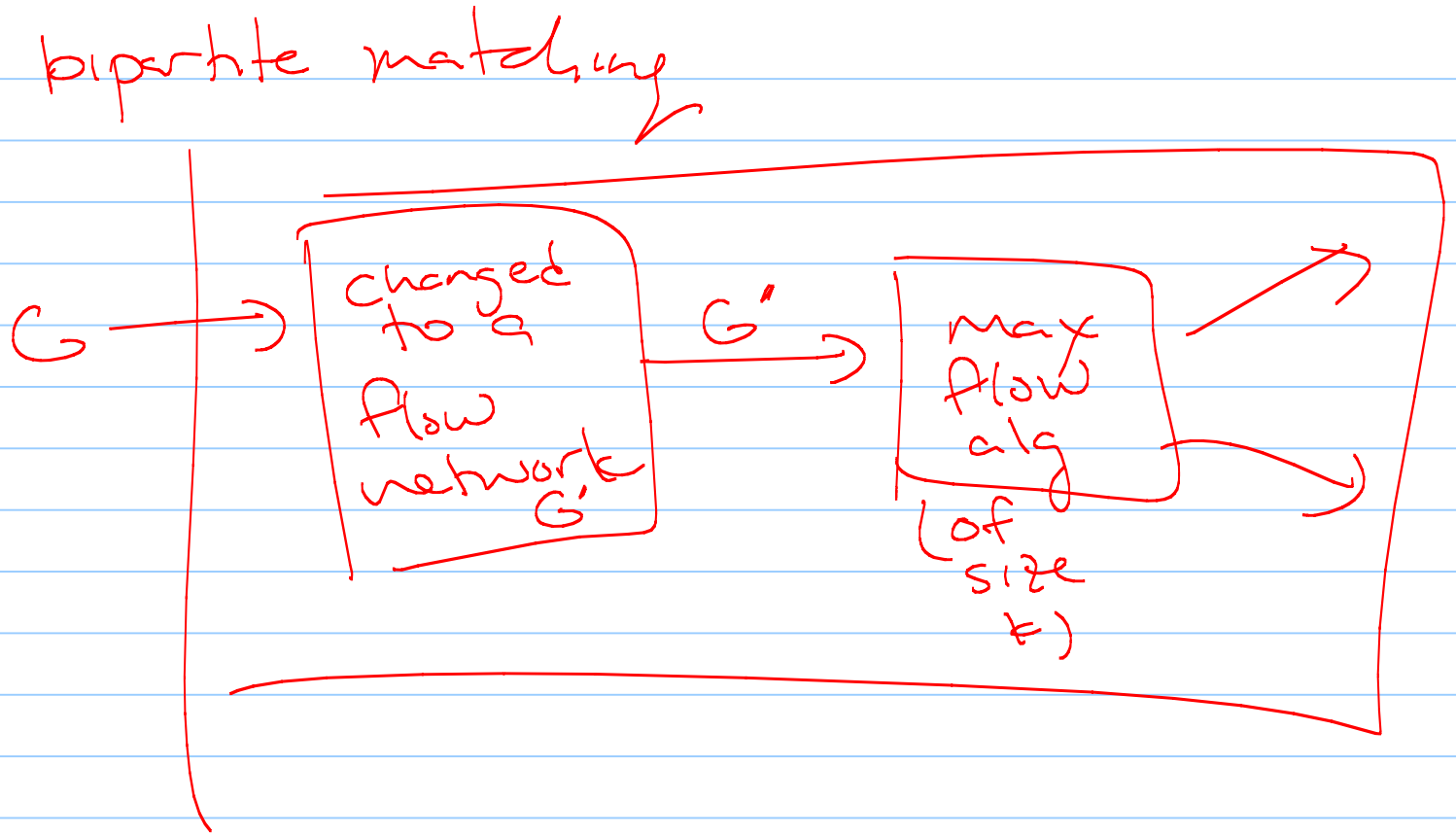
polynomial hierarchy

To prove NP-Hardness of A:

Reduce a known NP-Hard problem to A.



bipartite matching



So to prove your problem is hard, solve a different problem using your problem as a subroutine!

Cook's Thm: Circuit SAT is NP-Hard.

(just trust me)

Dfn: SAT takes a boolean formula & asks if it is possible to to assign booleans so the formula is true.

Ex: $(a \vee b \vee c \vee \bar{d}) \Leftrightarrow ((b \wedge \bar{c}) \vee \overline{(\bar{a} \Rightarrow d)} \vee (c \neq a \wedge b))$

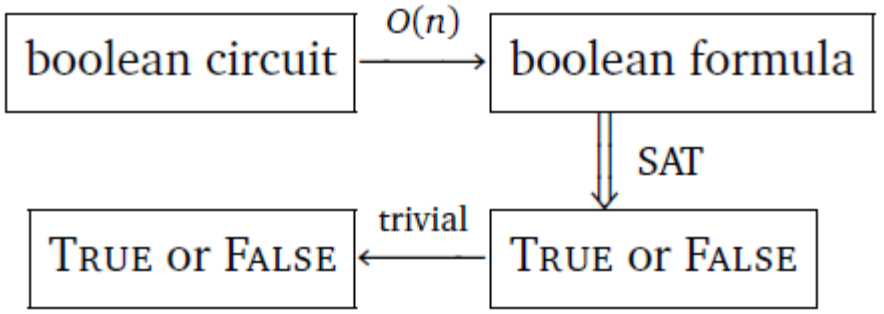
m variables
n clauses

in NP: given assignment a, b, c, d, can check if it evaluates to true in $O(m+n)$ time

Thm SAT is NP-hard.

Pf: Reduction:
Reduce circuit SAT to SAT

So our reduction looks like this:



CSR: