Algorithms – 2020

NP-Hard (part 2)

Recap

- HW (HW7) over flows:
  - now posted.
  - due next week - oral grading on next Thurs & Fri.
  - sign-up with group on Canvas
- Reading:
  - Thursday this week
  - Switching to a different book (still on Perusall) for next week
  - Last reading: Sunday, Nov 15
- No class on Monday, Nov. 16 (rest of days as usual)
- Check for final exam conflict

Quantifying Hardness:

Fundamental question:

Are there "harder" problems?

How do we rank?

Polynomial Hierarchy

↳ runtimes:

- linear
- $n \log n$
- quadratic
- ⋮

} polynomials (in input)

?? → subexponential → Ex: factoring

→ exponential (backtracking chapter)

→ worse? yes.

Undecidability:

Some problems are impossible to solve!

The Halting Problem:

(Turing & Church-30's)

Given a program P and input I, does P halt or run forever if given I?

P: code: Python code

I

forever ← infinite loop

Output: True / False ← decision

(Utility should be obvious!)

Note: Can't just simulate P on I. Why?

If it goes forever, won't stop & let me answer

↳ I don't know when to output true.

Thm [Turing 1936]:

The halting problem is undecidable.

(That is, no such algorithm can exist.)

Proof: (by contradiction) suppose we have such a program h:

$$h(P, I) = \begin{cases} \text{True} & \text{if } P \text{ halts on } I \\ \text{False} & \text{otherwise (infinite loop)} \end{cases}$$

(P: program, I: input)

Need a contradiction now...

(see last notes)

So... what next?

Clearly, many things are solvable in polynomial time. :)

Some things are impossible. :(

But - what is in between?

↑ ?? what can we do?

Idea:

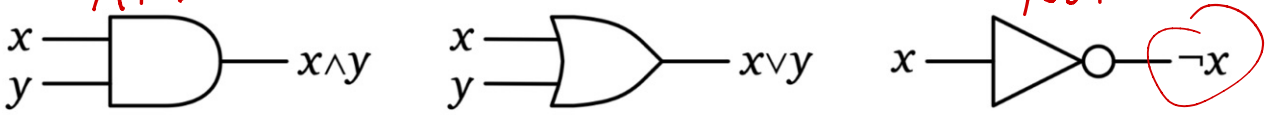
Set the idea of what are limits of (practical) computing.

- If NP-Hard, reasonable to use heuristics or approximations.

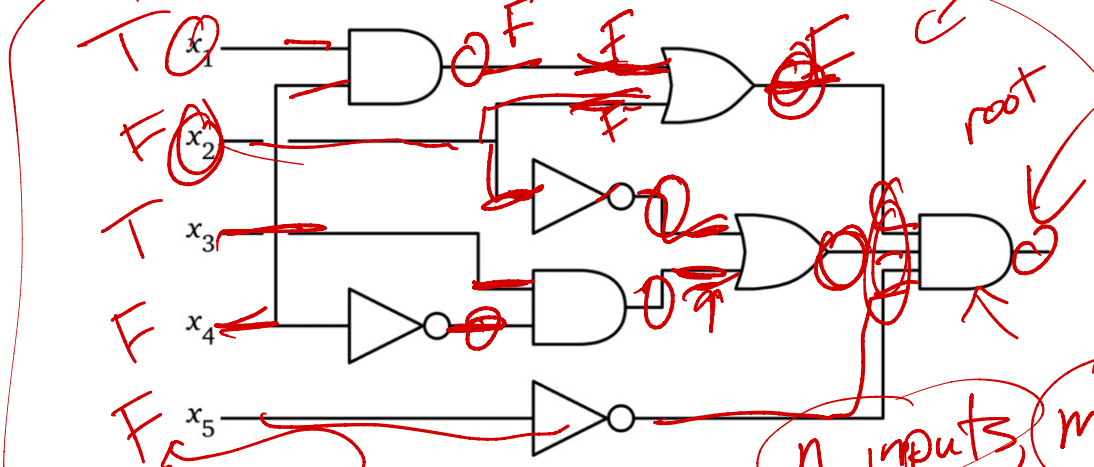
The first problem found: Boolean circuits

x	$\neg x$
0	1
1	0

AND OR NOT



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



A boolean circuit. inputs enter from the left, and the output leaves to the right.

leaves

tree

has $O(n+m)$?? is it T or F?

root

n inputs, m gates

Given input, check if T in $O(n+m)$ time

Given a set of inputs, can clearly calculate output in linear time (in # inputs + # gates).

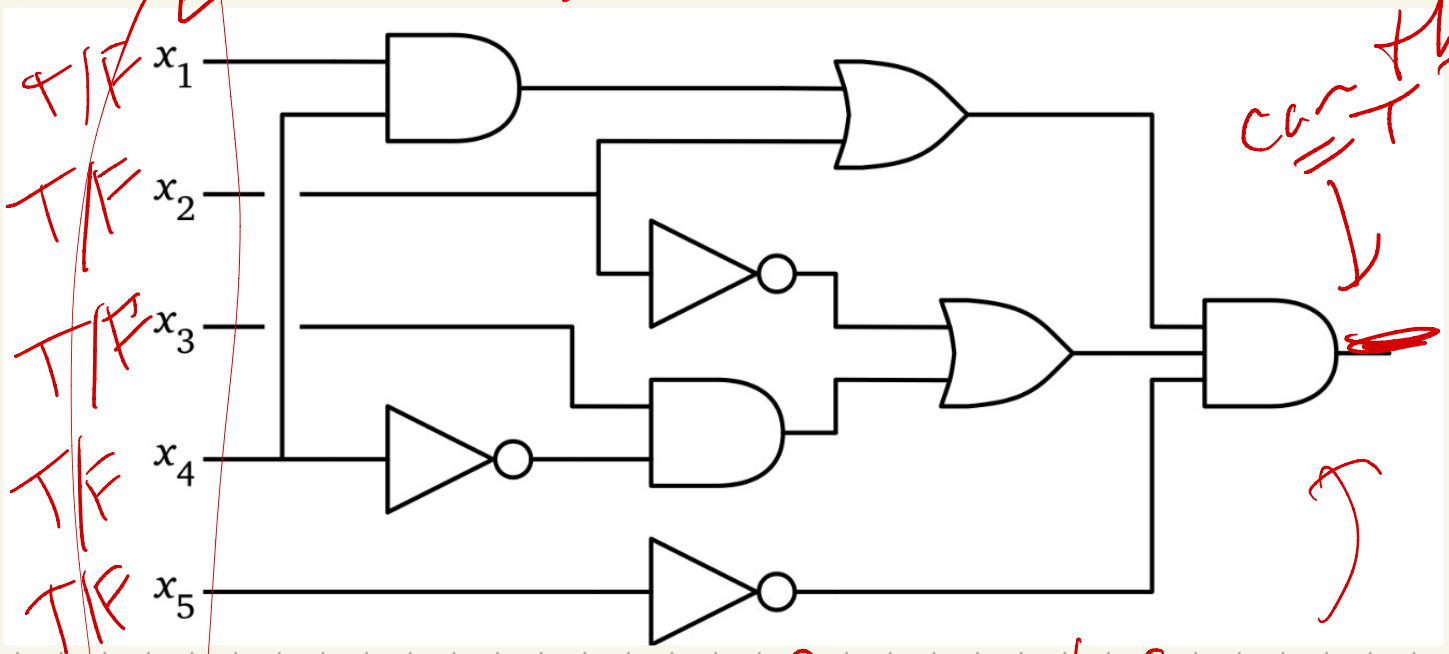
How? Move from inputs "up" in tree

x	y	output $x \wedge y$
0	0	0
1	0	0
0	1	0
1	1	1

OR

x	y	output
0	0	0
0	1	1
1	0	1
1	1	1

Q: Given such a boolean circuit, is there a set of inputs which result in TRUE output?



Known as CIRCUIT SATISFIABILITY (or CIRCUIT SAT)

"Clearly" check in exponential time:
2^n possible T/F inputs
for each, trace in $O(m+n)$ time

Best known algorithm:

try all possible 2^n inputs
if find true
output true
else False

Running time:

↳ $O(m+n)(2^n) = O(m2^n)$

↑ check each

Note:

Might be a polynomial algorithm; Hasn't been found yet.

P, NP, & co-NP

Consider only decision problems:
so Yes/No output

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

Ex: Is this list sorted?
Is value x in list?
Is flow in G of value ≥ 25?

NP: Set of problems such that, if the answer is yes & you hand me proof, I can verify/check in polynomial time.

Ex: CircuitSAT
Is list sorted?
all of P

Co-NP: Can verify a "No" answer.
sorted, is x in set, flow.

Ex: primality testing: Is x prime?

O(n) = n^1
n^2
n^c
n^constant

P ⊆ NP
think ⊆

P ⊆ co-NP

- $P \subseteq NP$, $P \subseteq co\text{-}NP$
- circuitSAT $\in NP$
  $\in co\text{-}NP$
  unknown if $cSAT \in P$?

- primality testing:
  → "obviously" in co-NP
  is it in NP or P?
  ↳ in NP, not obvious

→ give you $p \neq \emptyset$ which you can multiply in $O(n)$ +, =, x?

Q: NP = coNP?
(open)

Dfn: NP-Hard

X is NP-Hard

$\iff$

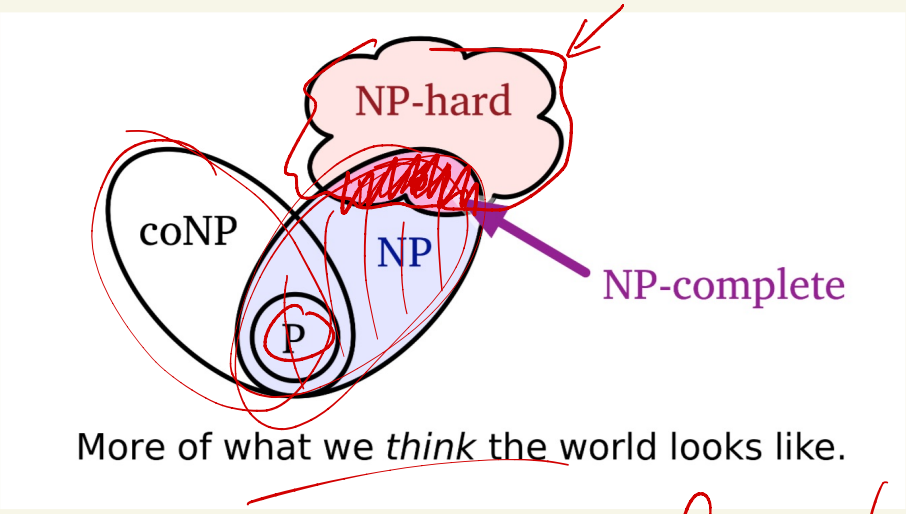
If X could be solved in polynomial time, then P=NP.

So if <u>any</u> NP-Hard problem could be solved in polynomial time, then all of NP could be.

Cook-Levine Thm:

Circuit SAT is NP-Hard.

unknown if P=NP?



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

(beyond scope of class)

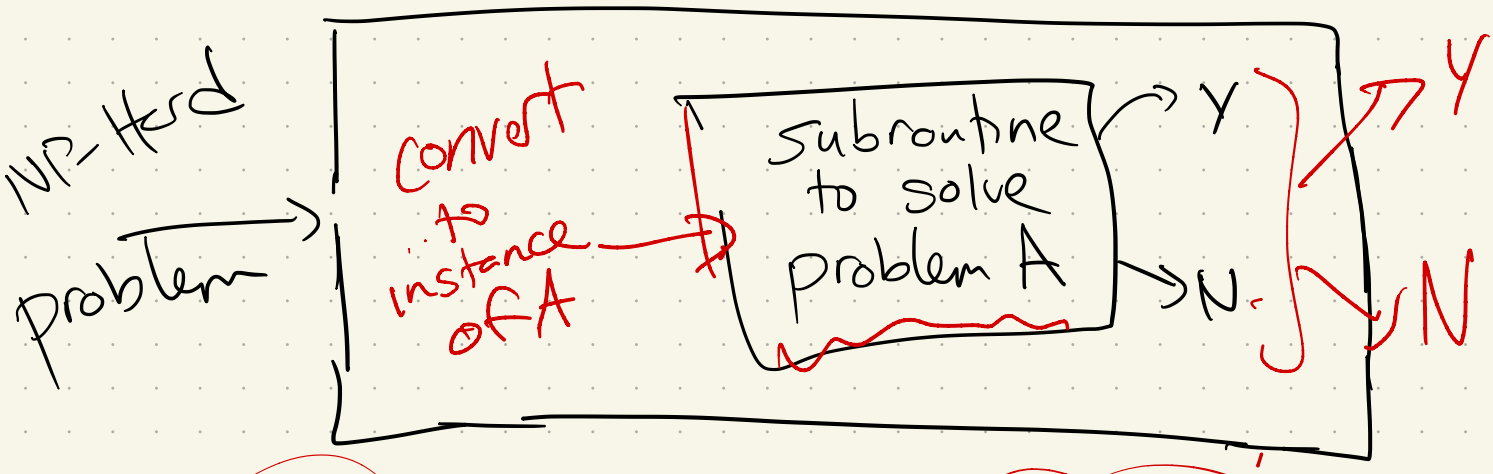
NP-Complete: :=

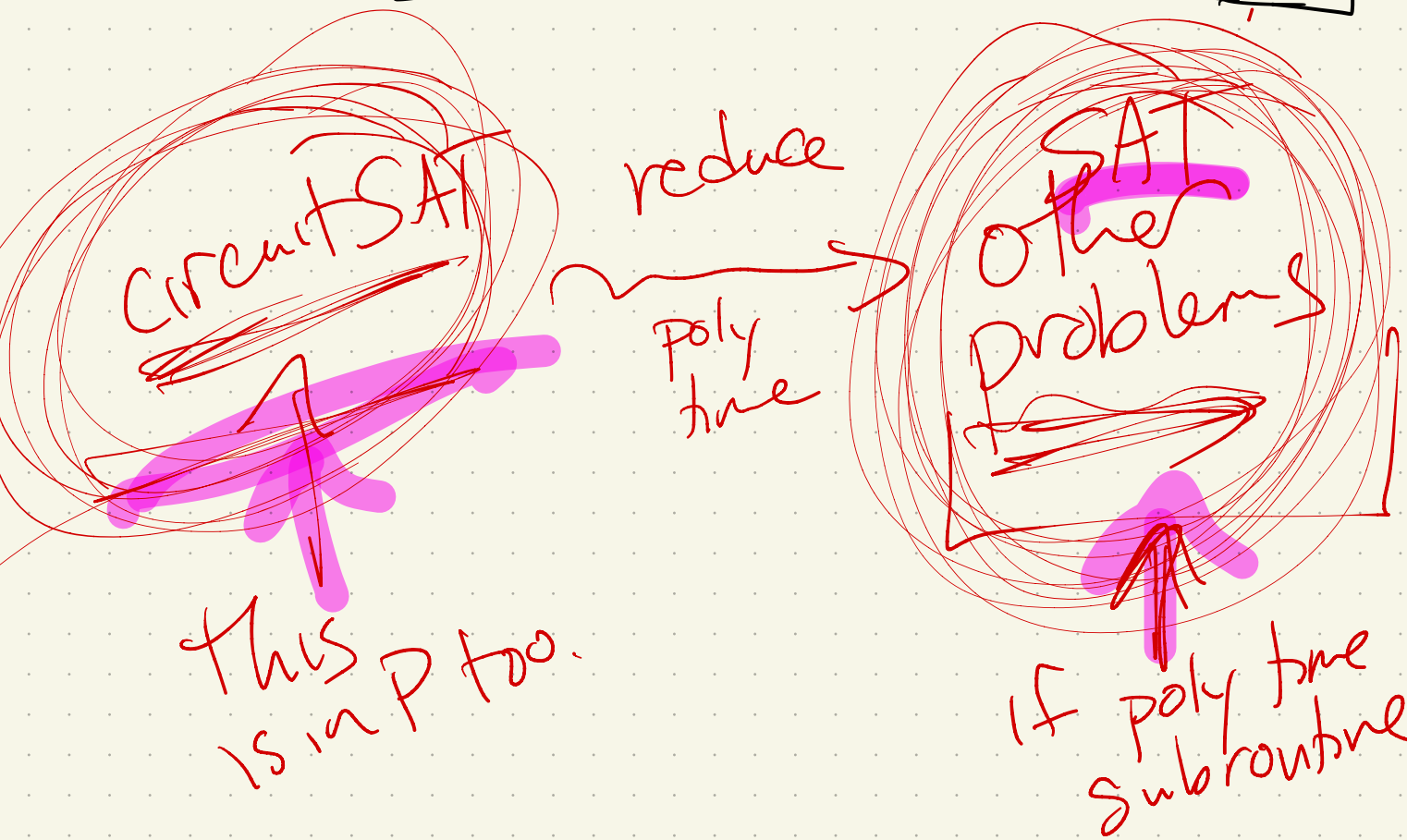
- in NP
- and is NP-Hard

If in P also, then NP=P

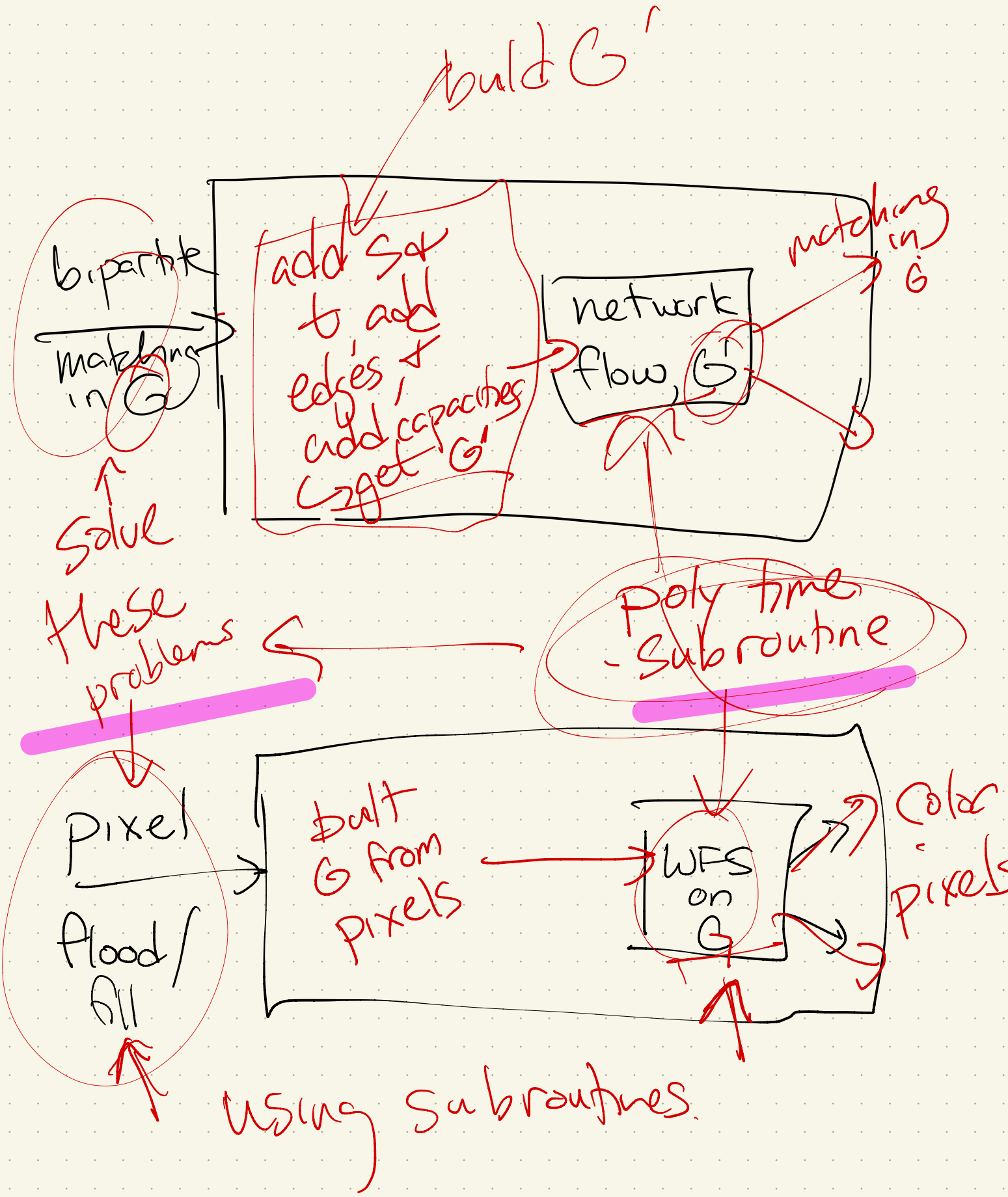
To prove NP-Hardness of A:

Reduce a known NP-Hard problem to A.





We've seen reductions!



This will feel odd, though:

To prove a new problem is hard, we'll show how we could solve a known hard problem using new problem as a subroutine.

Why?

Well, if a poly time algorithm existed, than you'd also be able to solve the hard problem!

(Therefore, "can't" be any such solution.)

Other NP-Hard Problems:

SAT: Given a boolean formula, is there a way to assign inputs so result is 1?

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

and, or, not, $\Rightarrow$, $=$ & $\neq$, ...

T T T D=T F T

n variables, here: $a, \ldots, d$ $x_1, \ldots, x_n$

m clauses: 5 = m

In NP!

Given n T/F inputs, scan the clauses & check each in O(1)

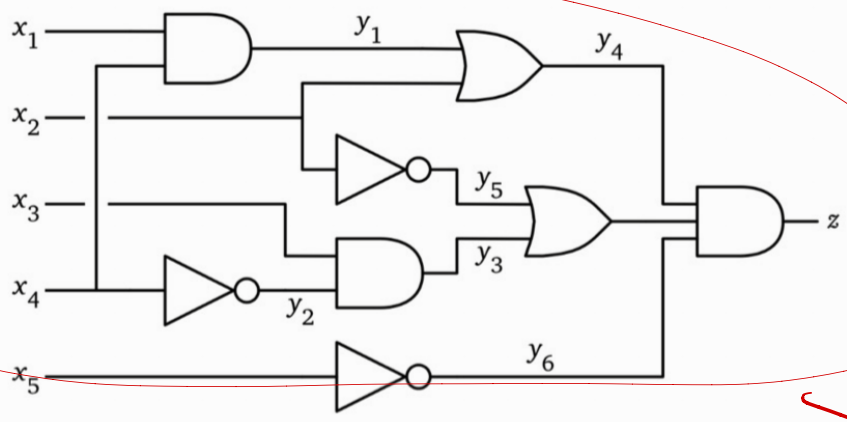
Poly!

Total: $O(n+m)$

nm if clauses are O(n) long

Thm: SAT is NP-Hard.

pf: Reduce CIRCUIT SAT (known NP-Hd) to SAT: (new one)



poly

$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge$$
$$(y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

SAT

A boolean circuit with gate variables added, and an equivalent boolean formula.