

Algorithms - Spring '25

- End of flows
- Complexity



Recap

Max-flow/Min Cut Algorithms:

Residual graph based

• FF? $O((V+E)F^*)$

• Edmonds-Karp: $O(E^2 \log E \log f^*)$

• BFS based: $O(VE^2)$

Technique	Direct	With dynamic trees	Source(s)
Blocking flow	$O(V^2E)$	$O(VE \log V)$	[Dinitz; Karzanov; Even and Itai; Sleator and Tarjan]
Network simplex	$O(V^2E)$	$O(VE \log V)$	[Dantzig; Goldfarb and Hao; Goldberg, Grigoriadis, and Tarjan]
Push-relabel (generic)	$O(V^2E)$	—	[Goldberg and Tarjan]
Push-relabel (FIFO)	$O(V^3)$	$O(VE \log(V^2/E))$	[Goldberg and Tarjan]
Push-relabel (highest label)	$O(V^2\sqrt{E})$	—	[Cheriyan and Maheshwari; Tunçel]
Push-relabel-add games	—	$O(VE \log_{E/(V \log V)} V)$	[Cheriyan and Hagerup; King, Rao, and Tarjan]
Pseudoflow	$O(V^2E)$	$O(VE \log V)$	[Hochbaum]
Pseudoflow (highest label)	$O(V^3)$	$O(VE \log(V^2/E))$	[Hochbaum and Orlin]
Incremental BFS	$O(V^2E)$	$O(VE \log(V^2/E))$	[Goldberg, Held, Kaplan, Tarjan, and Werneck]
Compact networks	—	$O(VE)$	[Orlin]

Figure 10.10. Several purely combinatorial maximum-flow algorithms and their running times.

FOR HW

Just use
 $O(VE)$ if
 unsure

Another: Exam scheduling

Input: n classes, r classrooms
 t time slots, P proctors

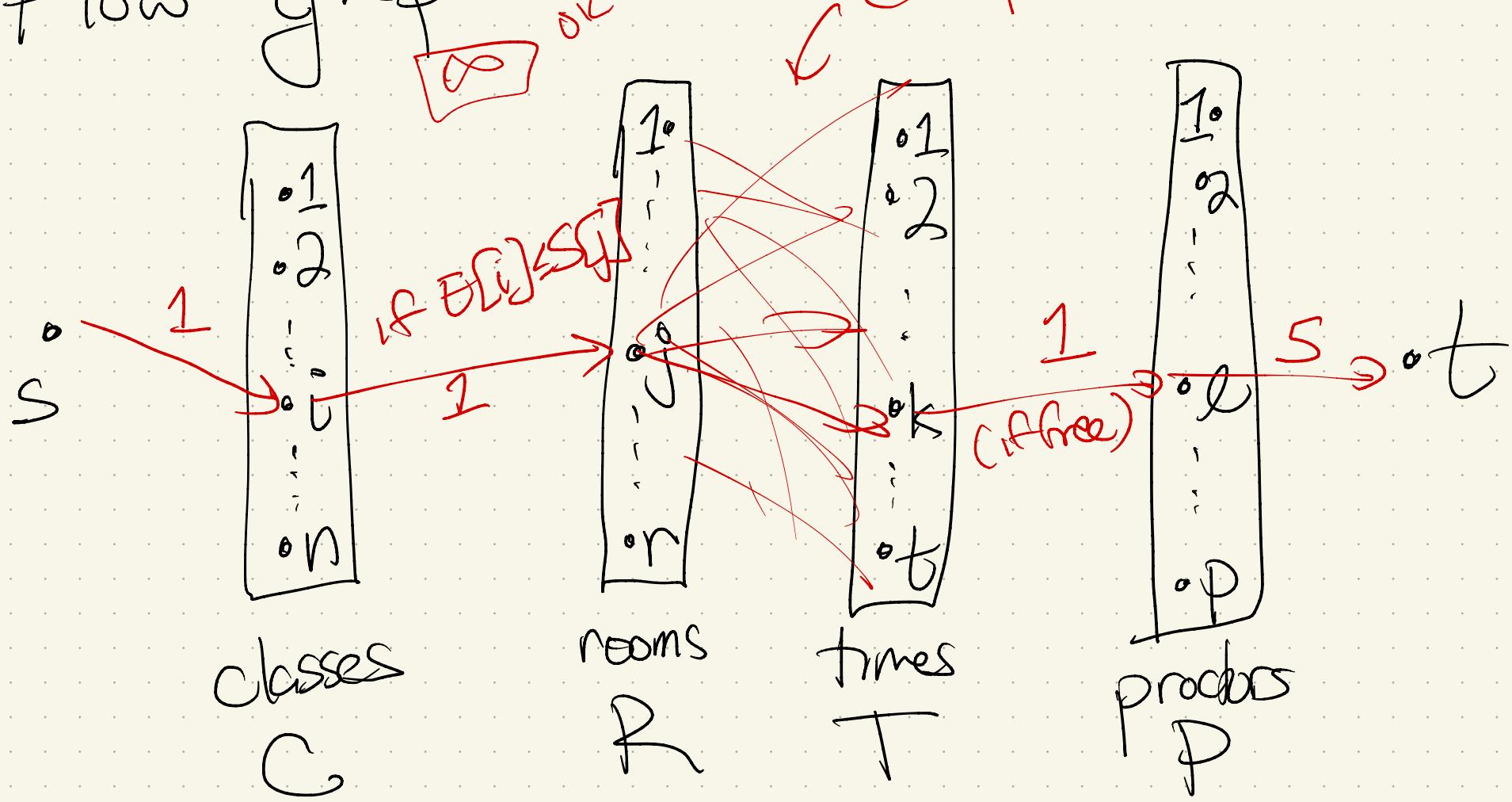
$E[1..n]$: # of students in each class

$S[1..r]$: capacity of each classroom

$A[1..t, 1..P]$: $A[k, l]$ is true
if l th proctor is free at time slot k

& each proctor gets ≤ 5 classes.

Flow graph:



Edges: $H \in C, S \rightarrow V$ with $cap = 1$,
 so flow paths "assign" 1 class to
 valid room, time & proctor

Then $C \rightarrow R$ edges:

If $E[i] \leq S[j]$: class will fit
in room.

So add edge $i \rightarrow j$ [for $i \in C, j \in R$].
capacity = ~~X 1~~

Then $R \rightarrow T$ edges:

add all edges $j \rightarrow k$ with
capacity = 1, since each room
is open to start at every time

Next: $T \rightarrow P$ edges

If $A[k, l]$ is true, then proctor
 l is open at time k .

→ add edge of capacity = 1
(so can't be assigned 2)

Finally: $P \rightarrow t$

Add all $l \rightarrow t$ edges, for $l \in P$

Capacity = ~~5~~ 5

Then: find max flow.

If $F = n$, done! ~~If $< n$: problem~~

Find flow paths:

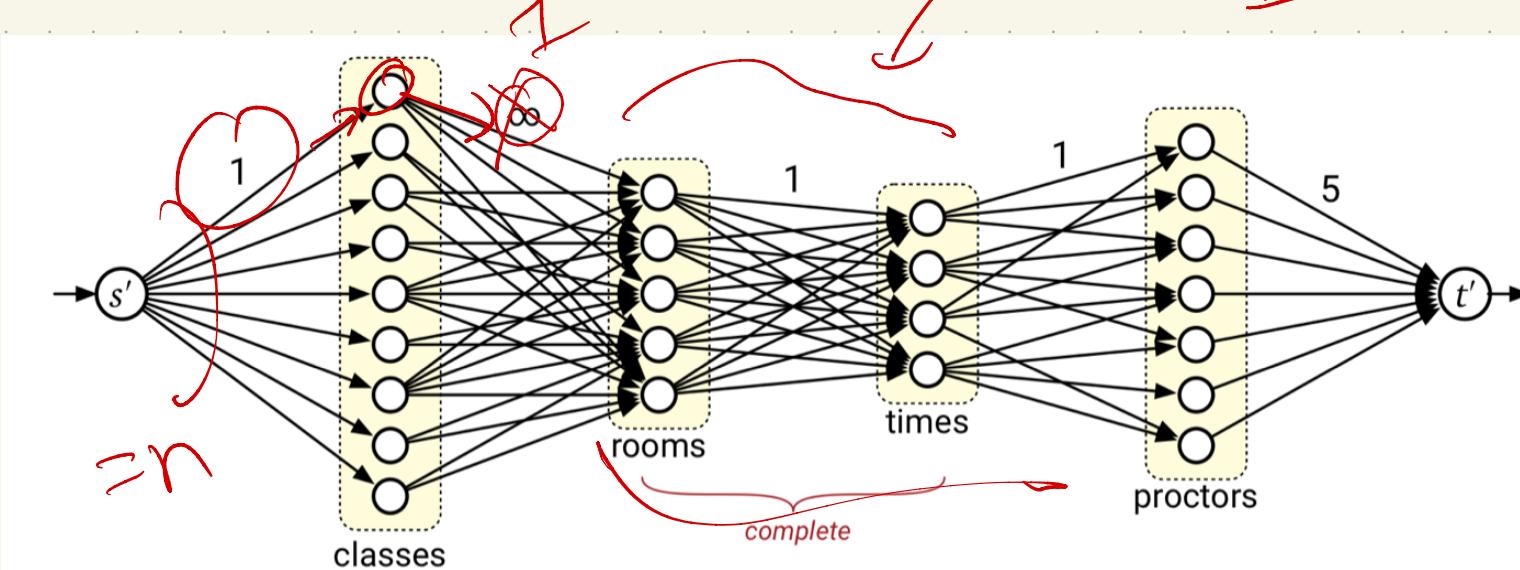


Figure 11.5. A flow network for the exam scheduling problem.

Must go $s \rightarrow i \rightarrow j \rightarrow k \rightarrow l \rightarrow t$.

So: If I flow of value n ,
can find assignment of exams.

Other way:

If can assign rooms, classes,
times, & proctors, can also
use each assignment to build
a flow path of value 1 in
G. So, assignment \Rightarrow flow.

Runtime:

$$V =$$

$$E =$$

$$+ O(nk) = O(VE) =$$

Quantifying Hardness:

Fundamental question:

Are there "harder" problems? (Yes)
Why should we care?

How do we rank?

The bad news! Undecidability

Some problems are impossible to solve!

The Halting Problem:

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

Output: True / False

(Utility should be obvious!)

Note: Can't just simulate P on I .

Why?

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 .

$$H(P, I) = \begin{cases} \text{True} & \text{if } P(I) \text{ halts} \\ \text{False} & \text{if } P(I) \text{ loops forever} \end{cases}$$

Need to find a contradiction now...

Define a program G, which uses H as a subroutine:

G(x): if $H(x, x) = \text{false}$
return false
else
loop forever

So: if $X(x)$ halts, loop forever
if $X(x)$ loops halt + return false

Now : What does $G(G)$ do?

If $H(G, G) = \text{false}$, then halts
↳ but if $H(G, G)$ is false,
means $G(G)$ has infinite loop!

If $H(G, G) = \text{true}$, then loops
forever

↳ but if $H(G, G)$ is true,
means $G(G)$ halts.

Logical contradiction

↳ no such function exists.

So... what next?

Clearly, many things are solvable in polynomial time.

Some things are impossible

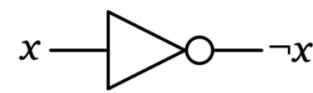
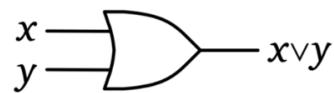
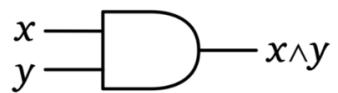
But - what is in between?

Idea:

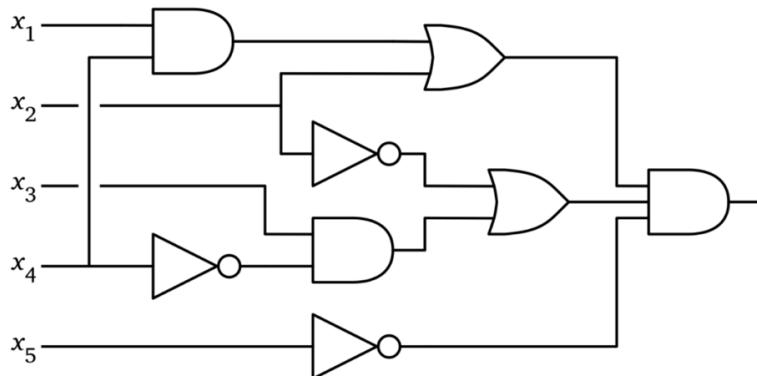
Try to formalize a notion of "hardness", to better understand what computer can do.

The first problem found;

Boolean circuits

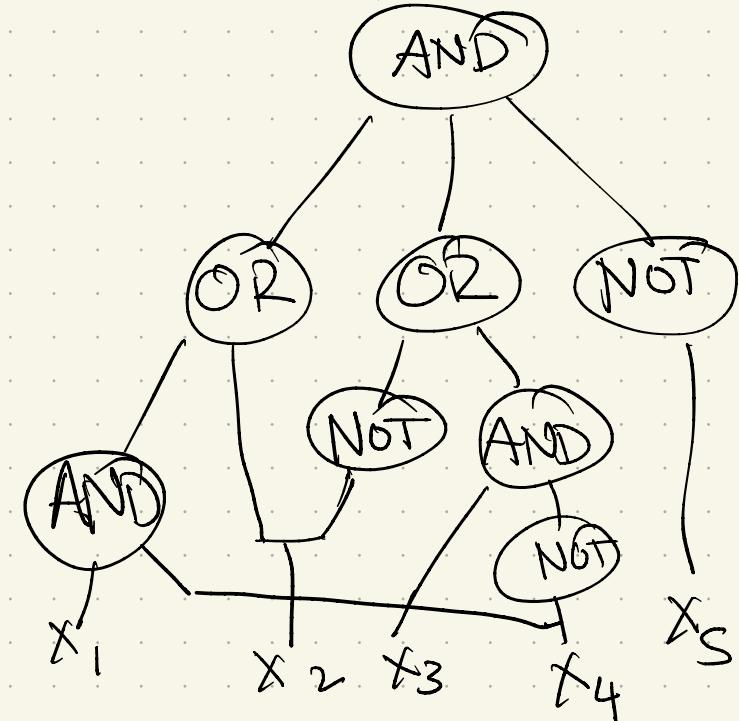


An AND gate, an OR gate, and a Not gate.



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

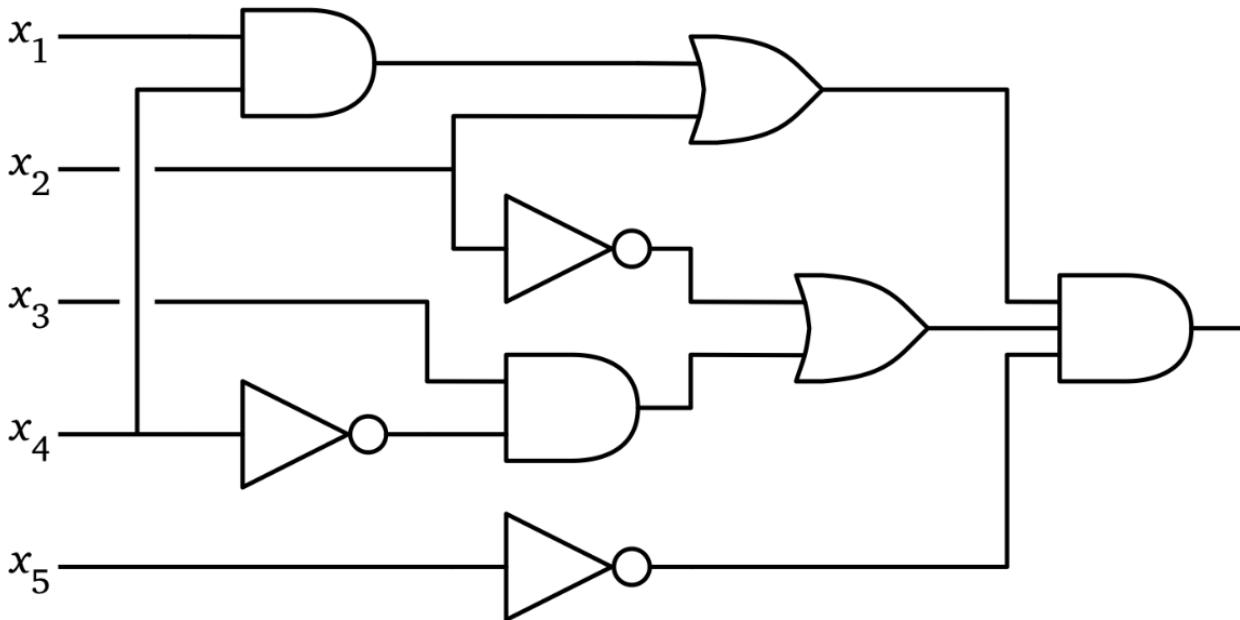
"Flipped view":



Given a set of inputs can clearly calculate
out put in linear time ($\text{in } \# \text{ inputs } + \# \text{ gates}$).

How?

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



Known as CIRCUIT SATISFIABILITY
(or CIRCUIT SAT)

Best known algorithm:

Try all possible inputs

If one works, return true
else return false

Runtime:

Note:

P, NP, + co-NP

Consider only decision
problems: so Yes/No output

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

Examples:

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

Examples:

Co-NP: Set of problems where we can verify a "no".

Examples:

Dfn: NP-Hard

X is NP-Hard



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

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

Note: Not at all obvious any such problem exists!

Cook-Levine Thm:

Circuit SAT is NP-Hard.

Proof (sketch):

Suppose I have an algorithm
to solve CIRCUIT-SAT. in poly time.

Take any problem in NP, A.

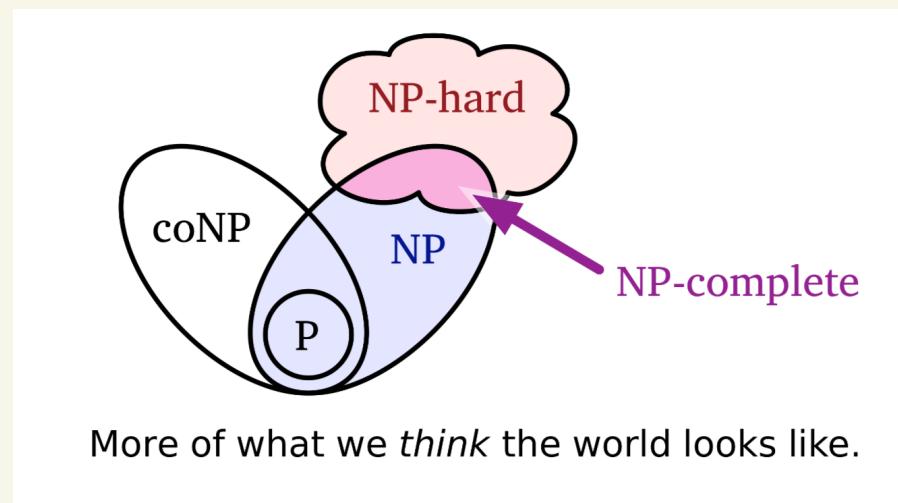
Reduce A to CIRCUIT-SAT.

in polynomial time: build circuit.

Therefore, I have a poly time alg
for A.



So, there is at least one problem that is NP-Hard, & in NP, but which we don't think is in P:

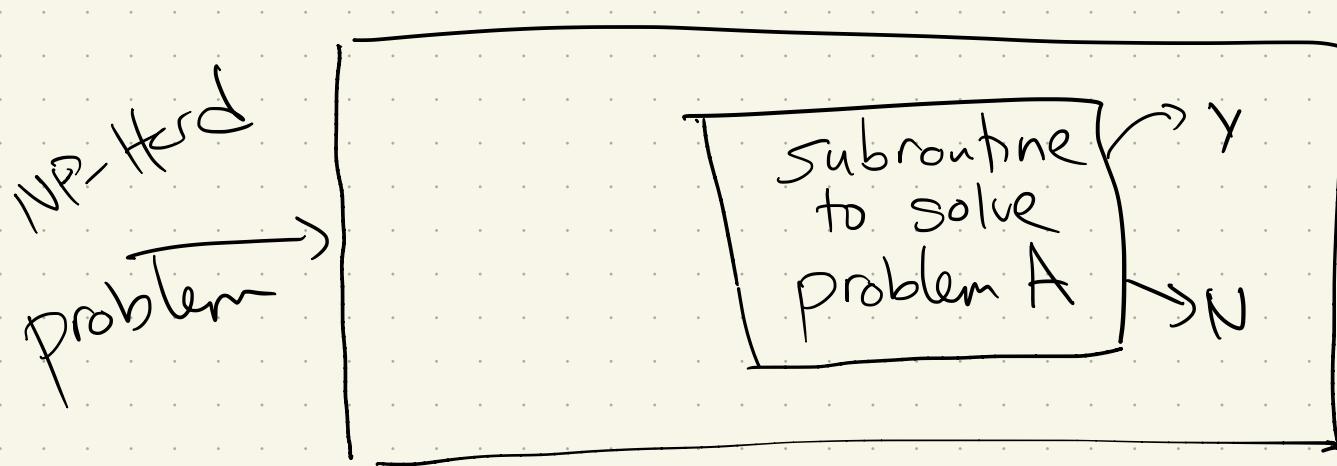


NP-Complete:

To prove NP-Hardness of A:

Reduce a known NP-Hard problem to A.

(Alternative is to show any problem in NP can be turned into A, like Cook.)



We've seen reductions!

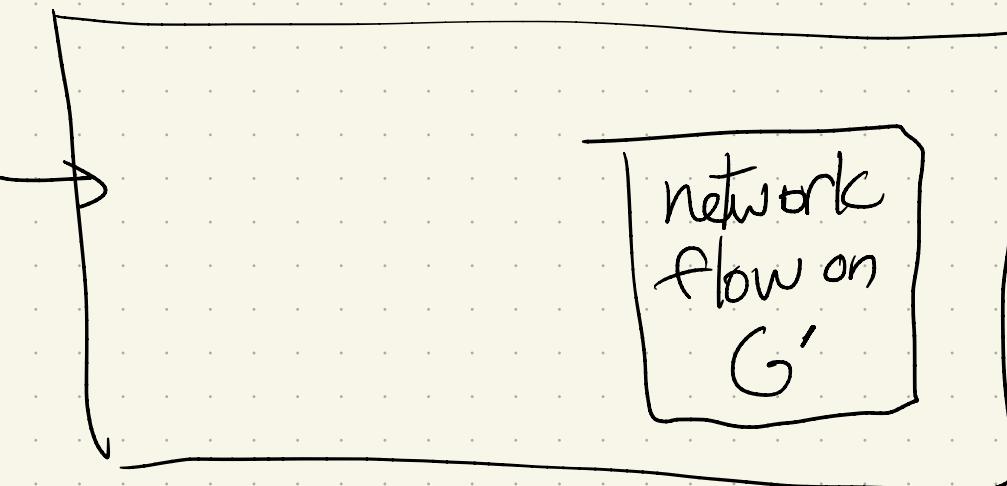
But used them to solve problems!

flood
fill
in pixels



WFS_{in}
G

bipartite
matching
in G



network
flow on
G'

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? Just like halting problem!

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

(Therefore, "can't" be any such alg)

Other NP-Hard Problems:

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

Ex:

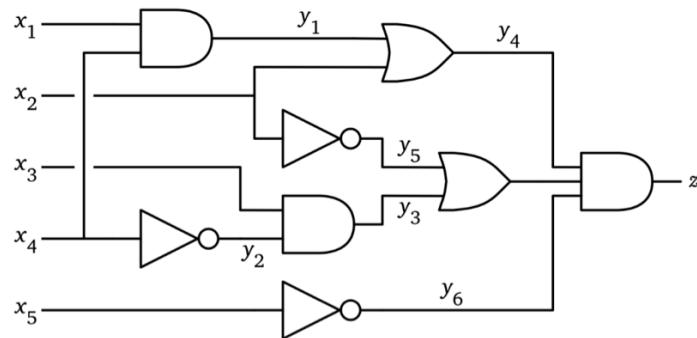
$$(a \vee b \vee c \vee \bar{d}) \Leftrightarrow ((b \wedge \bar{c}) \vee \overline{\bar{a} \Rightarrow d}) \vee (c \neq a \wedge b),$$

n variables, m clauses

First: in NP?

Thm: SAT is NP-Hard.

Pf: Reduce CIRCUIT SAT to SAT:



Input: CIRCUIT

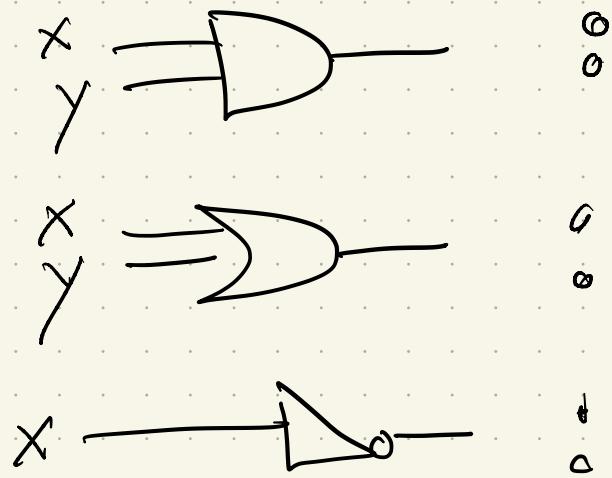
$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge \\ (y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

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

Convert in poly time to clauses:

More carefully:

1) For any gate, can transform:



2) "And" these together, & want final output true:

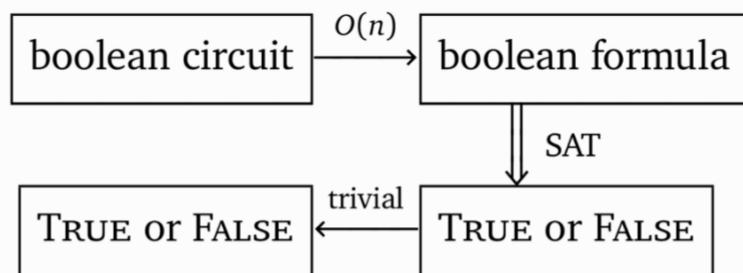
Is this poly-size?

Given n inputs + m gates:

Variables:

Clauses:

So our reduction:



$$T_{\text{CSAT}}(n) \leq O(n) + T_{\text{SAT}}(O(n)) \implies T_{\text{SAT}}(n) \geq T_{\text{CSAT}}(\Omega(n)) - O(n)$$

3SAT: 3CNF formulas!

Thm: 3SAT is NP-Hard

Pf: Reduce circuitSAT to 3SAT:

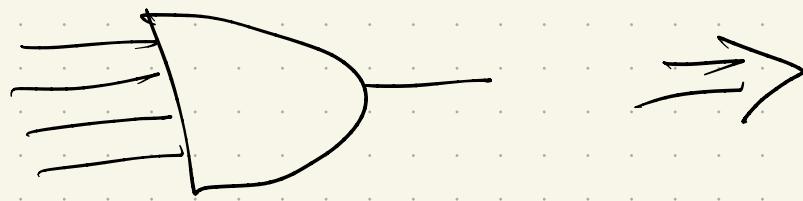
Need to show any circuit can be transformed
to 3CNF form

(so last reduction fails)

Instead 

Given a circuit!

- ① Rewrite so each gate has ≤ 2 inputs:



- ② Write formula, like SAT. Only 3 types!

$$y = a \vee b$$

$$y = a \wedge b$$

$$y = \overline{a}$$

③ Now, change to CNF:
go back to truth tables

$$a = b \wedge c \rightarrow (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee c)$$

$$a = b \vee c \rightarrow (\bar{a} \vee b \vee c) \wedge (a \vee \bar{b}) \wedge (a \vee \bar{c})$$

$$a = \bar{b} \rightarrow (a \vee b) \wedge (\bar{a} \vee \bar{b})$$

④ Now, need 3 per clause!

$$a \rightarrow (a \vee x \vee y) \wedge (a \vee \bar{x} \vee y) \wedge (a \vee x \vee \bar{y}) \wedge (a \vee \bar{x} \vee \bar{y})$$

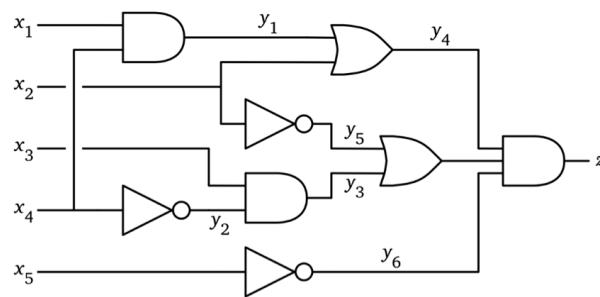
$$a \vee b \rightarrow (a \vee b \vee x) \wedge (a \vee b \vee \bar{x})$$

Note : Bigger!

How much

bigger?

(need polynomial)



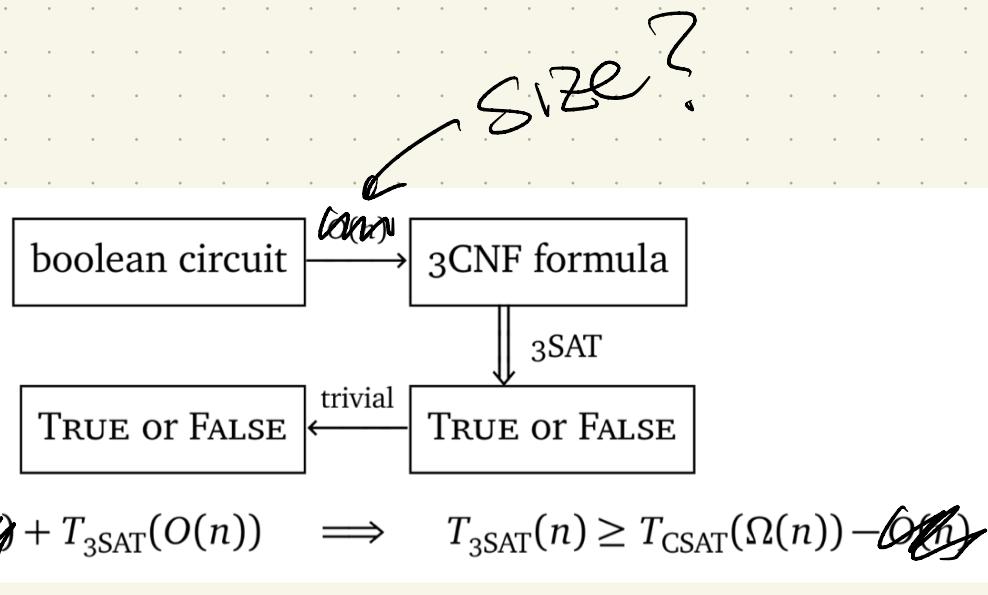
$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge \\ (y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

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



$$(y_1 \vee \overline{x_1} \vee \overline{x_4}) \wedge (\overline{y_1} \vee x_1 \vee z_1) \wedge (\overline{y_1} \vee x_1 \vee \overline{z_1}) \wedge (\overline{y_1} \vee x_4 \vee z_2) \wedge (\overline{y_1} \vee x_4 \vee \overline{z_2}) \\ \wedge (y_2 \vee x_4 \vee z_3) \wedge (y_2 \vee x_4 \vee \overline{z_3}) \wedge (\overline{y_2} \vee \overline{x_4} \vee z_4) \wedge (\overline{y_2} \vee \overline{x_4} \vee \overline{z_4}) \\ \wedge (y_3 \vee \overline{x_3} \vee \overline{y_2}) \wedge (\overline{y_3} \vee x_3 \vee z_5) \wedge (\overline{y_3} \vee x_3 \vee \overline{z_5}) \wedge (\overline{y_3} \vee y_2 \vee z_6) \wedge (\overline{y_3} \vee y_2 \vee \overline{z_6}) \\ \wedge (\overline{y_4} \vee y_1 \vee x_2) \wedge (y_4 \vee \overline{x_2} \vee z_7) \wedge (y_4 \vee \overline{x_2} \vee \overline{z_7}) \wedge (y_4 \vee \overline{y_1} \vee z_8) \wedge (y_4 \vee \overline{y_1} \vee \overline{z_8}) \\ \wedge (y_5 \vee x_2 \vee z_9) \wedge (y_5 \vee x_2 \vee \overline{z_9}) \wedge (\overline{y_5} \vee \overline{x_2} \vee z_{10}) \wedge (\overline{y_5} \vee \overline{x_2} \vee \overline{z_{10}}) \\ \wedge (y_6 \vee x_5 \vee z_{11}) \wedge (y_6 \vee x_5 \vee \overline{z_{11}}) \wedge (\overline{y_6} \vee \overline{x_5} \vee z_{12}) \wedge (\overline{y_6} \vee \overline{x_5} \vee \overline{z_{12}}) \\ \wedge (\overline{y_7} \vee y_3 \vee y_5) \wedge (y_7 \vee \overline{y_3} \vee z_{13}) \wedge (y_7 \vee \overline{y_3} \vee \overline{z_{13}}) \wedge (y_7 \vee \overline{y_5} \vee z_{14}) \wedge (y_7 \vee \overline{y_5} \vee \overline{z_{14}}) \\ \wedge (y_8 \vee \overline{y_4} \vee \overline{y_7}) \wedge (\overline{y_8} \vee y_4 \vee z_{15}) \wedge (\overline{y_8} \vee y_4 \vee \overline{z_{15}}) \wedge (\overline{y_8} \vee y_7 \vee z_{16}) \wedge (\overline{y_8} \vee y_7 \vee \overline{z_{16}}) \\ \wedge (y_9 \vee \overline{y_8} \vee \overline{y_6}) \wedge (\overline{y_9} \vee y_8 \vee z_{17}) \wedge (\overline{y_9} \vee y_8 \vee \overline{z_{17}}) \wedge (\overline{y_9} \vee y_6 \vee z_{18}) \wedge (\overline{y_9} \vee y_6 \vee \overline{z_{18}}) \\ \wedge (y_9 \vee z_{19} \vee z_{20}) \wedge (y_9 \vee \overline{z_{19}} \vee z_{20}) \wedge (y_9 \vee z_{19} \vee \overline{z_{20}}) \wedge (y_9 \vee \overline{z_{19}} \vee \overline{z_{20}})$$

So:



$O(\quad)$ \rightarrow

So: If could solve 3CNF, could
solve CIRCUITSAT in poly time.

Next time:

Can we do this with any
useful problems?

(Logic is all well + good...)

Maybe \rightarrow graphs?