Computational Complexity

Big picture: • Turing Machines = Algorithms

• TM are limited (~~things~~ problems exist that are *not* computable

Among *problems* that *are* computable:

- Do some problems *require* more resources to solve?
- Are some problems *inherently* more difficult than others?

Big-O Notation: analyzes algorithms

Given an algo $\longrightarrow$ how "good" is it

Structural Complexity:

Studying problems, not algorithms

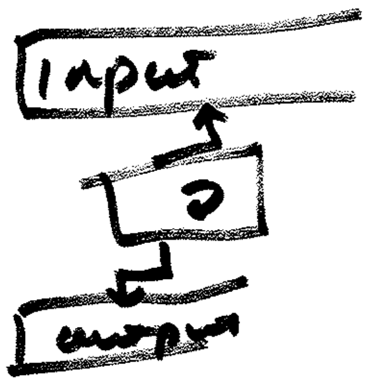
Resources wrt TM:

$$\begin{array}{lcl} & & \text{TM} \\ \text{① Input} & \longrightarrow & \text{Input: } x \in \Sigma^* \\ \text{② Input size} & \longrightarrow & \text{Input size: } |x| = n \end{array}$$

# of bits in x

③ Elem. Operation: State change

$0^n 1^n$



one state → another state

i.e. <u>time</u>

$T(n) = T(|x|) =$ number of state changes made by a TM on inputs of size n

= time taken by an algo/TM

$M(n) =$ number of unique memory cells (bits) used in the output tape.

let M be a TM such that

$$T(n) \in O(n^k), \text{ for all } n$$

then M is a ~~pol~~ <u>deterministic</u> polynomial time (ptime) Turing Machine

$$P = \{ L \mid \text{L is a language such that there exist a ptime deterministic TM that decides L} \}$$

- P is a class of <u>languages</u> (ie problems)
- Class of problems that have "efficient" solutions $\quad O(n^{100})$

Ex: Problems that are in P:

- is G acyclic
- searching, sorting
- Matrix inverse, solving a linear system.

?: Hamiltonian Path? $O(n!)$

Satisfiability $O(2^n)$

Nondeterminism

- "magic" computation
- Theoretical idea of computation
- Does not really exist
- Can always be "simulated" by deterministic computation but with an exponential blow up $O(2^n)$
- Not randomization, not quantum

2 stages to a non-deterministic algo:

1. Guess: a solution is guessed, called a certificate
2. Verification: verify the certificate:
   a) if valid: accept
   b) if invalid: reject

If any guess leads to an accept state, then the TM / Algo accepts

Non-deterministic Algo: Ham. Path

Input: a graph $G = (V, E)$

non. det. guess $\Big\{$ guess a path p, a permutation of V, $\pi(V) = v_1 v_2 \ldots v_n$

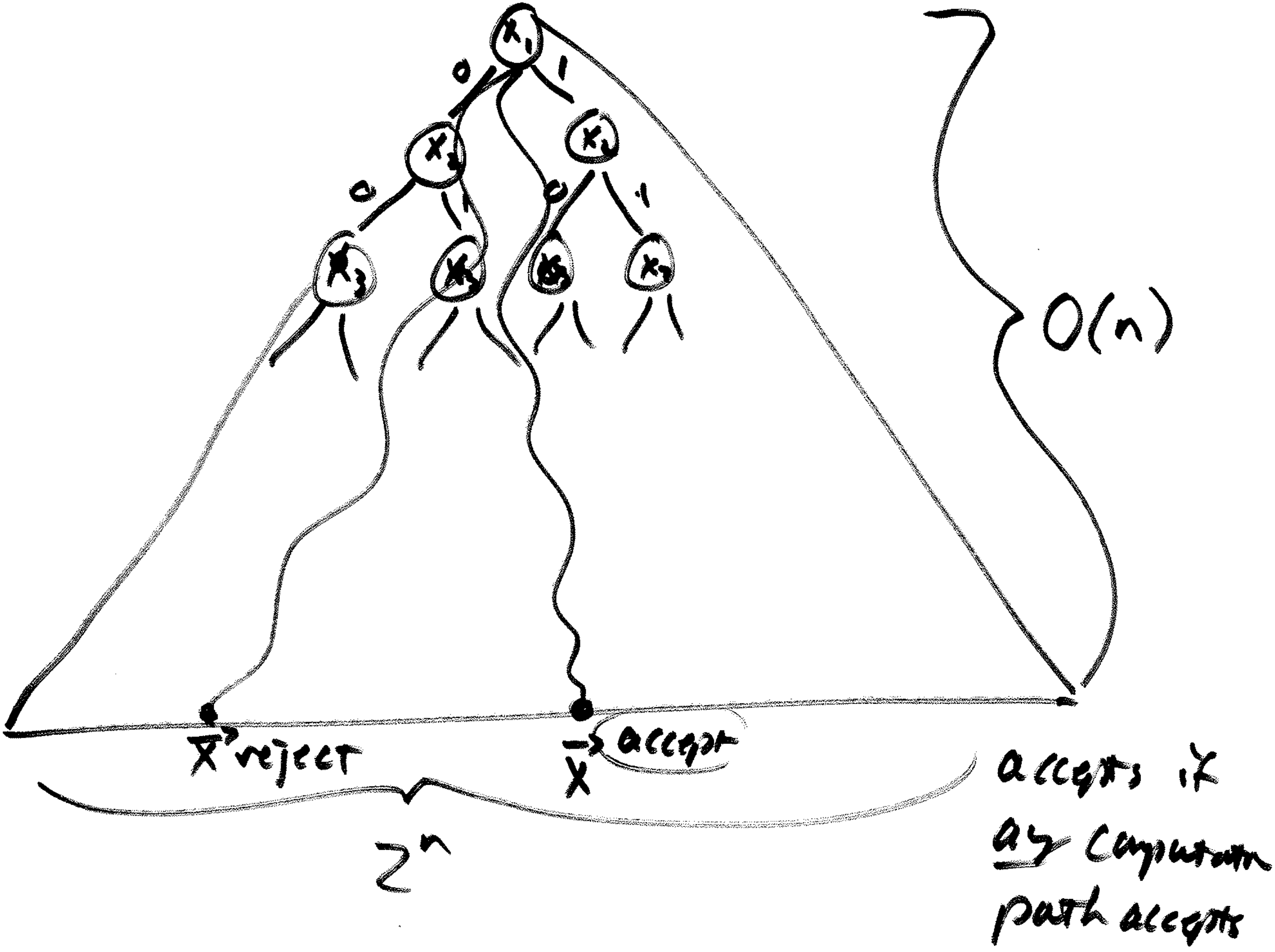
for $i = 1 \ldots n-1$

  if $((v_i, v_{i+1}) \notin E)$

    reject

(deterministic verification)

accept



x1
x2
x2
x3
0
1
O(n)
x reject
x accept
2^n
accepts if any computation path accepts

SAT:

$$\exists \underbrace{x_1\, x_2\, x_3 \ldots x_n}^{O(n)} \left[P(\vec{x}) \equiv 1 \right]$$

$\forall$ co-Nondeterminism

NP = Nondeterministic Polynomial time

$= \left\{ L \mid \begin{array}{l} L \text{ is a language such that there is} \\ \text{a nondeterministic polytime turing machine} \\ \text{that decides } L \end{array} \right\}$

Ham Path, SAT, have NP solutions

P vs. NP

- $P \subseteq NP$

Because: you can always <u>skip</u> the "magic" guessing phase and compute a solution (certificate) directly

$NP \subseteq P$ ie $P = NP$ or $P \neq NP$

nondeterminism is <u>not</u> inherently more powerful
you could simulate nondeterminism with $O(n^k)$ determinism

nondeterminism <u>IS</u> inherently more powerful

ie. some problems are inherently more difficult.

P vs NP

Reductions: • establish a relative complexity between problems

• a problem B is at least as difficult as a problem A

• $\underbrace{A \le_p B}$ "A reduces to B"

B is at least as hard / complex as A

not simplification reduction to the essentials

Not necessarily more difficult $\cancel{A <_p B}$

Turing Reduction :

bool isConnected(G, x, y) A   x , a , b , ... , y

int shortestPathDist(G, x, y)^B = $\begin{cases} \infty & \text{if no such path} \\ k & \end{cases}$

list<vertices> shortestPath(G, x, y)

---

bool isConnected(G, x, y) $\cancel{A \le_T B}$

int length = shortestPathDist(G, x, y)

if (length < ∞)

return true

else

return false

A

1. $A \leq_p B$

2. Suppose you have an algorithm for B

3. you can build an algo. for A:

   a) ~~use~~ run the solution for B

   b) use solution to answer A

isConnected $\leq_p$ Shortest Path Dist.

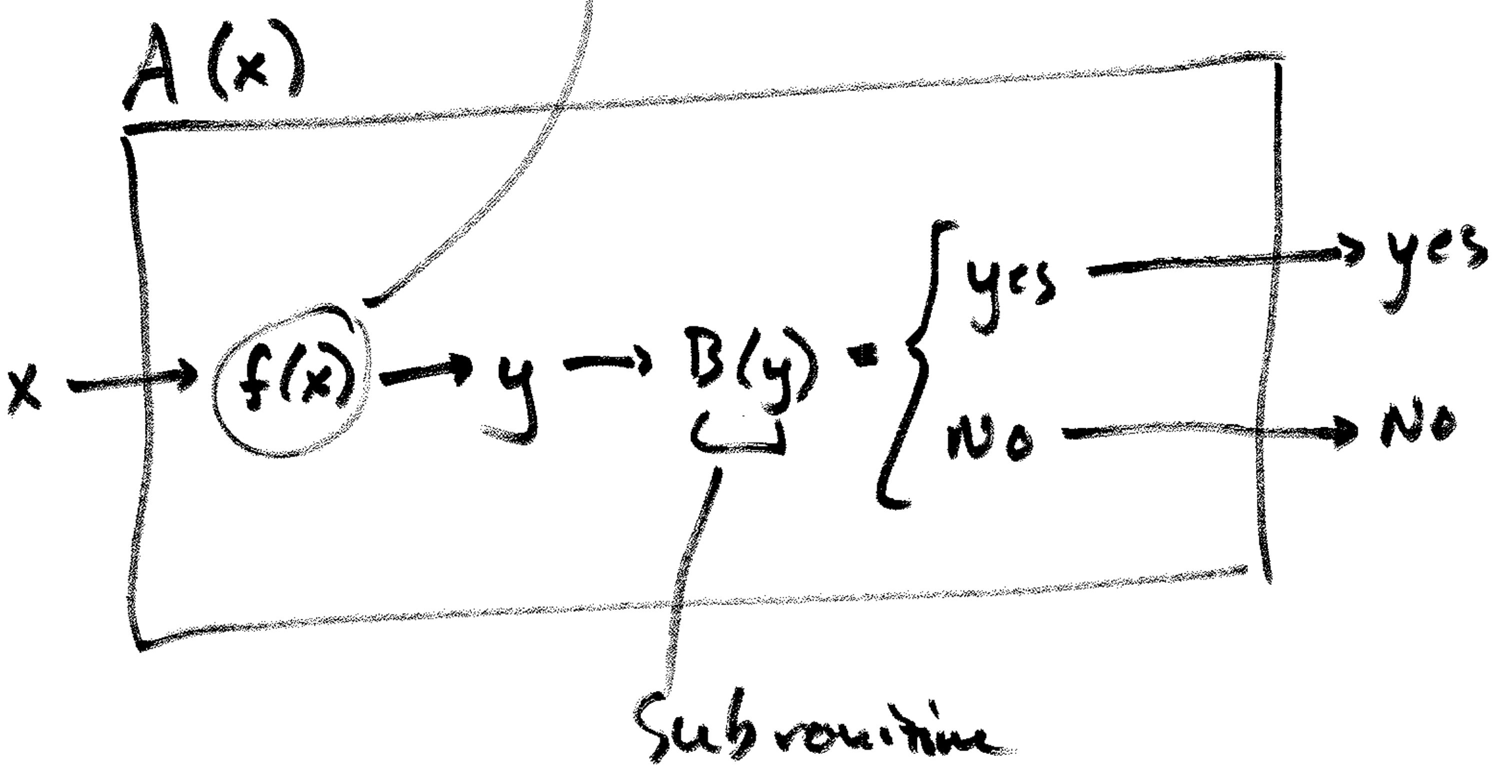
- isConnected is no harder than SPD
- SPD is at least as hard as isConne

Mapping Reductions:

Defn Let A, B be problems, $A \leq_p B$ A is poly-time reducible to B if there is a poly-time computable function f such that

- $f : \Sigma^* \longrightarrow \Sigma^*$
- if $x \in A \iff f(x) \in B$
    - yes instances of problem A map to yes instances of " B
    - no map to no
- f is computable by a poly-time deterministic T.M.

- Suppose you have an algorithm B for problem B
- suppose $A \le_p B$ via f
- Design Algo A' for A



P class

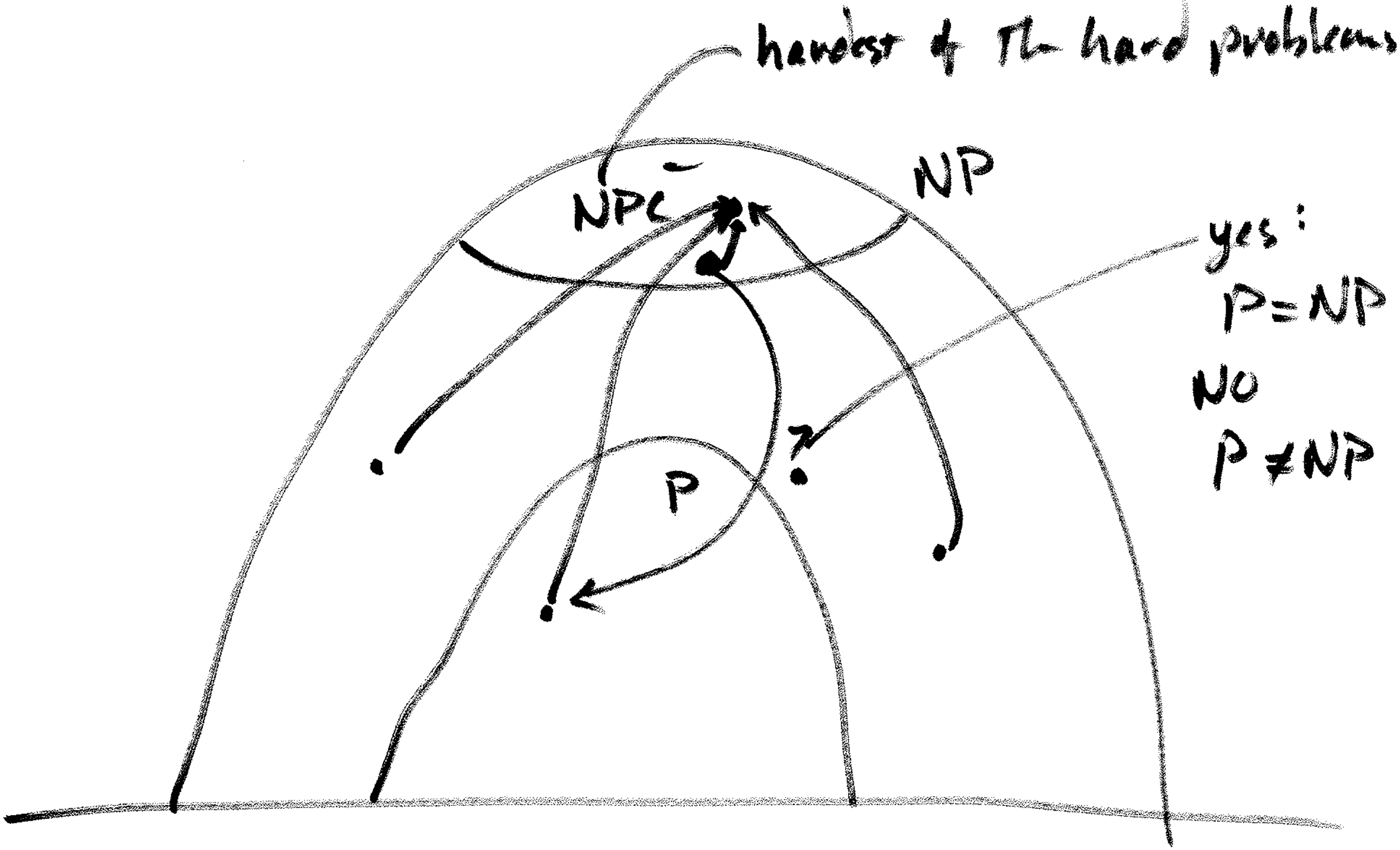
P problem

NPC = NP-Complete

= All problems B such that

- B is in NP and
- Every problem A in NP reduces to B

$$A \le_p B$$


hardest of the hard problems
NPC
NP
P
?
yes: P=NP
NO P≠NP

Goal: establish that certain difficult-looking problems are NP-C (they actually <u>are</u> difficult)

you need a starting point: the <u>first</u> NP-Complete problem.

$L_{NP} = \{ \langle M, x \rangle \mid$ M is a non-deterministic TM that accepts x in poly-time $\}$

"Canonically" complete

Cook-Levin: Satisfiability is NP-Complete

$$L_{NP} \leq_p SAT \leq_p 3\text{-CNF} \leq_p \text{Clique}$$

3-CNF: Conjunction Normal Form boolean predicate such that each "clause" has exactly 3 variables

$$\bigwedge_{i=1}^{m} C_i = C_1 \wedge C_2 \wedge C_3 \cdots \wedge C_m$$

clauses

- n variables: $X_1 \cdots X_n$
- each clause has ~~of~~ 3 variables
  a disjunction of

Ex:

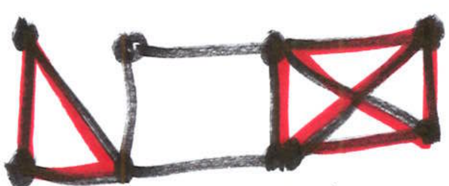
$$\underbrace{(x_1 \vee x_2 \vee \neg x_4)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3 \vee \neg x_5)}_{C_2} \wedge \underbrace{(x_2 \vee \neg x_3 \vee x_4)}_{C_3}$$

$n = 5$

$m = 3$

Clique : Given a graph $G = (V, E)$, an integer K, Does there exist $C \subseteq V$ such that C is a clique of size $|C| = k$

every vertex pair in C is connected



K_4

K_5

Brute Force

$\forall$ subsets of size k

$\binom{n}{k} = O(n^k)$

$K = n/2$

$O(n^{n/2})$

Show a problem B is NP-Complete

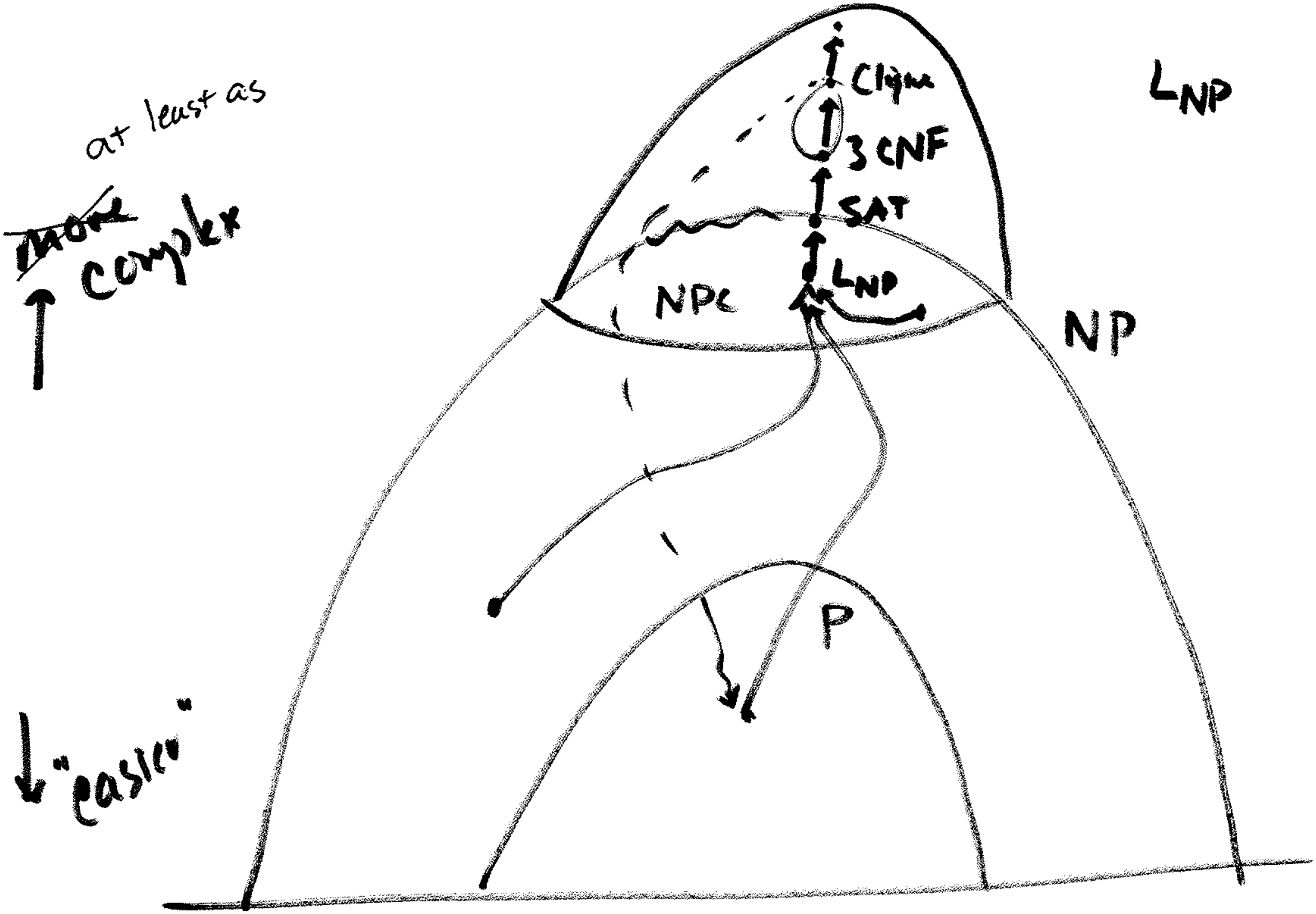
2. Choose a Known NP-C problem A to reduce from

1. Show B is in NP : design a guess + check algo for it

3. Define a mapping f: problem inputs in A → inputs in B
3-CNF ⟶ G

4. Show that f is poly-time computable

5. Prove that f preserves solution
yes ⟷ yes (2 proofs) $p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$
No ⟶ No



at least as
~~more~~ complex
"easier"
L_{NP}
Clique
3CNF
SAT
L_{NP}
NPc
NP
P

1. Show that your problem is in NP
2. Select a known NP-C problem A to reduce from (to your problem B)
3. Develop a mapping $A \leq_p B$
   map yes instances of A to
   yes instances of B
4. Prove your mapping preserves solutions
5. Show that your mapping is poly-time computable

$3CNF \le_p$ Clique

① Show Clique $\in NP$:

a) guess a subset $C \subseteq V$ of size $|C| = k$ $\quad O(k) = O(n)$

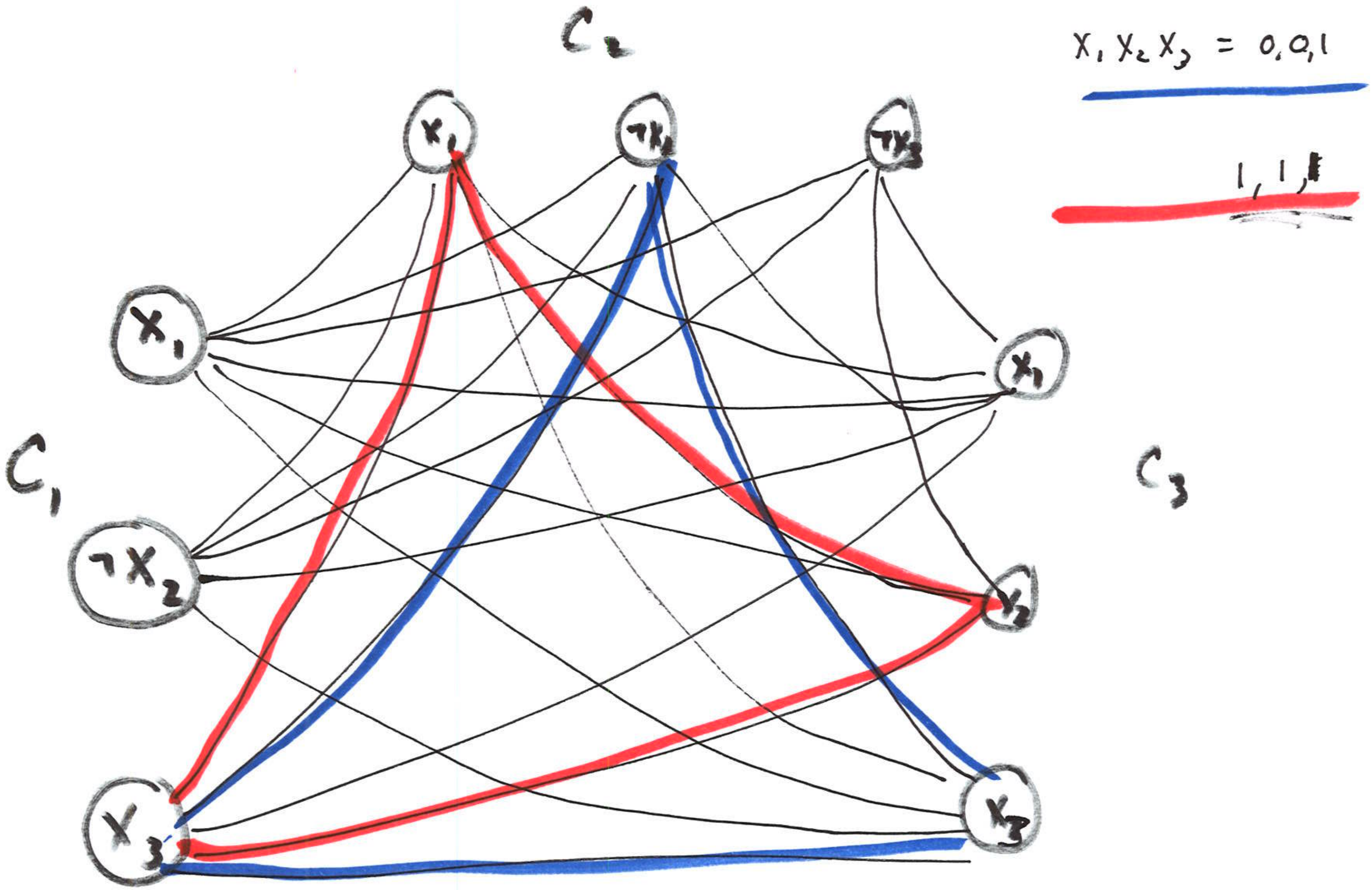
b) verify: given $C \subseteq V$, is it a clique?

for each pair $x, y \in C$
    if $((x, y) \notin E)$
        reject
accept

$\binom{k}{2} = O(n^2)$

accepts iff at least 1 subset C is a clique

$\therefore$ Clique $\in NP$



C_2
x_1
$\neg x_2$
$\neg x_3$
$x_1 x_2 x_3 = 0,0,1$
1, 1, 1
x_1
C_1
$\neg x_2$
x_3
x_1
C_3
x_2
x_3

② Select: 3CNF $\leq_p$ Clique

x_1	x_2	x_3	
0	0	0	×
0	0	1	☺

③ Develop a mapping:

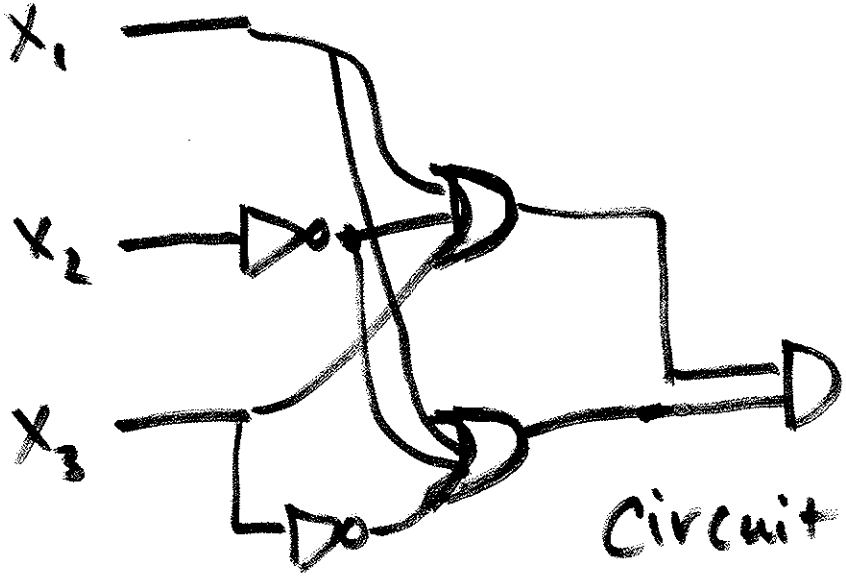
- map formulas $\longrightarrow$ graph

  "graph gadgets"

- formula:

$$\overbrace{(x_1 \vee \neg x_2 \vee x_3)}^{C_1} \wedge \overbrace{(x_1 \vee \neg x_2 \vee \neg x_3)}^{C_2} \wedge \overbrace{(x_1 \vee x_2 \vee x_3)}^{C_3}$$

- formula is satisfiable iff graph has a clique of size k



Circuit

Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ be a 3 CNF formula

clause with 3 variables/negation, n vars in general $x_1 \ldots x_n$

Create a graph:

- $3 \cdot k$ ~~variables~~ vertices, ~~one~~ for each variable in each clause

- ~~Edges~~ labels: $C_i = (x_1^i \vee x_2^i \vee x_3^i)$

$$\downarrow$$

$$v_1^i \quad v_2^i \quad v_3^i$$

- Edges: $(v_\alpha^i, v_\beta^j) \in E$ iff:

1. vertices are from different clauses ~~to~~ $i \neq j$

AND

2. the variables are consistent

v_α^i is not the negation of v_β^j

size: K

each pair is connected (thus forms a clique):

1) they are chosen from different clauses

2) they are not negations of each other

$X_i, \neg X_i$

$\therefore$ the vertices are connected.

(4) Prove the mapping preserves solutions

- Suppose ϕ is satisfiable
  at least 1 variable (or its negation) in each of the k clauses evaluates to true

- By construction: take the vertices corresponding to each true variable:

  what can you say about this set of vertices

(5) Mapping can be computed in p-time

for each clause C_i : $O(k)$

 for each var. x^i_j in C_i : 3

 L output a new vertex v^i_j

$\Big\}$ $O(k) = O(n)$

for each vertex pair v^i_α v^j_β : $\binom{3k}{2} = O(n^2)$

 apply the 2 conditions

 if $i \neq j$ and v^i_α is not the negation of v^j_β

 L add (v^i_α, v^j_β) to E

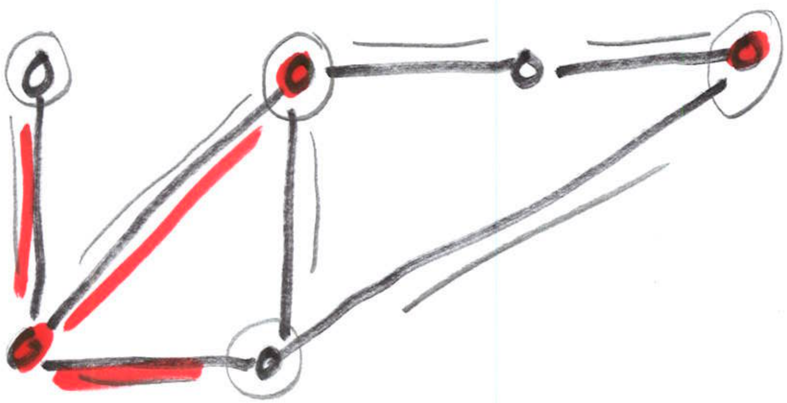
therefore

$$3CNF \le_p \text{Clique}$$

$$\Rightarrow \text{Clique is NP-C}$$

Show that vertex cover is NP-c

Given a graph $G=(V,E)$, find a subset $V' \subseteq V$ that "covers" every edge (by at least 1 endpoint)



$O(2^n)$

Show that Clique $\leq_p$ Vertex Cover (of size k)

- graph $\longrightarrow$ graph

① Show Vertex Cover is in NP:

a) guess a subset $V' \subseteq V$ of size k $O(k) = O(n)$

b) verify:

for each edge $e \in E$

$e = (x, y)$

if $(\neg(x \in V' \vee y \in V'))$ $\longrightarrow$ $x \notin V' \wedge y \notin V'$

reject

accept

($\Rightarrow$) if G has a clique of size k Then $\bar{G}$ has a vertex cover of size $n-k$

(P)

Let C be a clique of size k in G, $|C| = k$, let $e = (u, v)$ be an edge in $\bar{G}$, we need to show e is covered by some vertex in $V \setminus C$

G

$u \circ - - - - \circ v$

$$e = (u, v) \in \bar{G} \ (\bar{E}) \Rightarrow (u, v) \notin E$$

$$\Rightarrow u \notin C \vee v \notin C \quad \text{(or both)}$$

$$\Rightarrow u \in V \setminus C \ \cancel{\wedge} \vee \ v \in V \setminus C$$

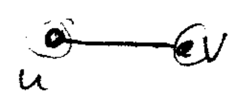
$$\Rightarrow (u, v) \text{ is covered by either } u \text{ or } v \text{ (or both)}$$

(⇐) let $C \subseteq V$ be a vertex cover of size $n-k$ in $\overline{G} = (V, \overline{E})$

$$\forall u,v \in V \Big[\overset{p}{(u,v) \in \overline{E}} \xrightarrow{} \overset{q}{u \in C \lor v \in C} \Big]$$

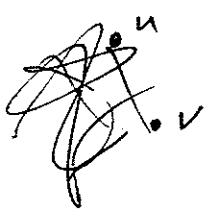
for evry pair of vertices (u,v), if u,v are connected in $\overline{G}$

then either u is in C or v is in C (or both)



$$\equiv \forall u,v \in V \Big[u \notin C \land v \notin C \Rightarrow \overset{(u,v) \in E}{\cancel{(u,v) \notin \overline{E}}} \Big]$$

$\overline{G}$



Then

G

$V \setminus C$ forms a clique of size $n-(n-k) = k$

$|V| = n$

$|C| = n-k$

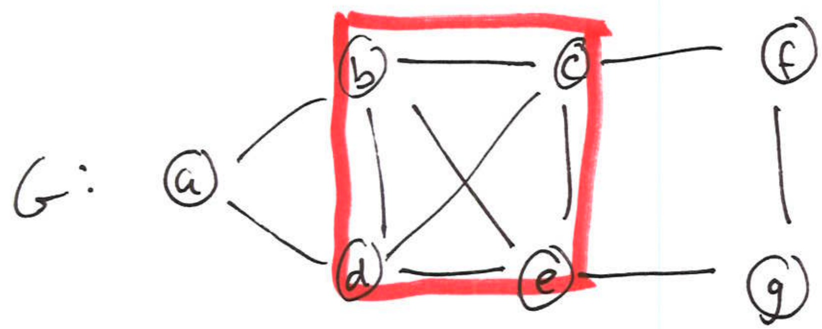
② Choose a known NP-C problem to reduce from

Clique $\leq_p$ Vertex Cover

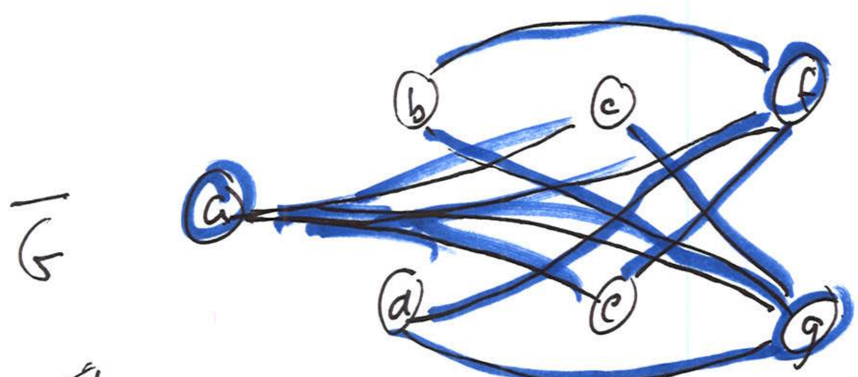
③ Come up with a mapping.

$n = 7$

$k = 4$ $\quad K$



$\Downarrow$



is there a vertex cover of size $\underline{\ 3\ }$ $\quad n - K$

mapping:

$\langle G, k \rangle \longrightarrow \langle \overline{G}, n-k \rangle$

claim:

G has a clique of size k $\iff$ $\overline{G}$ has a vertex cover of size $n-k$

(4) prove it ...

$\overline{G} \leftarrow (V, \emptyset)$

(5) for each pair of vertices $x, y \in V$:

if $(x, y) \notin E$

add (x, y) to $\overline{G}$

$\Big\} O(n^2)$

output $\langle \overline{G}, n-k \rangle$

$O(1)$