def IsSortingAlgo(a.java):
...
returns yes/No

def VerifyIsSortingAlgo(a.java, proof):
try a.java (T.txt.
"bad testcase T.txt
return yes/no.

CSE525 Lec19
NP-hardness

def VerifyIsNonsortedAlgo(a.java, T.txt):
out = a.java(T.txt)
if out is nonsorted, returnyes.

...

Debajyoti Bera (M21)

~~https://sites.google.com/a/iiitd.ac.in/cse525-m19~~

$a \le b$

if $b \le 10$, $a \le 10$

if $a \ge 19$, $b \ge 19$

Can this happen: 3SUM can't be solved in $O(n^{1.9})$ ← contradicts
~~COLL can be solved in $O(n^{1.5})$~~ ⇒ complex. of Algo3SUM $= O(n^{1.5})$
not possible

3SUM <= COLL reduction

if COLL is easy, 3SUM is easy
if 3SUM is hard, COLL is hard

3SUM <= COLL using a O(n) reduction. A is a Yes-inst. of 3SUM iff P = Reduce(A) is a Yes inst. of COLL.

- If COLL can be solved in time $O(n^{1.9})$ then 3SUM can be solved in ???
- ✗ If COLL can't be solved in time $O(n^{1.9})$ then … ??? $n = |A|$
- ✗ If 3SUM can be solved in time $O(n^{1.9})$ then ???
- ? If 3SUM can't be solved in time $O(n^{1.9})$ then ??? COLL can't be solved in $O(n^{1.9})$ time.

↳ AlgoColl $O(n^{1.9})$ time

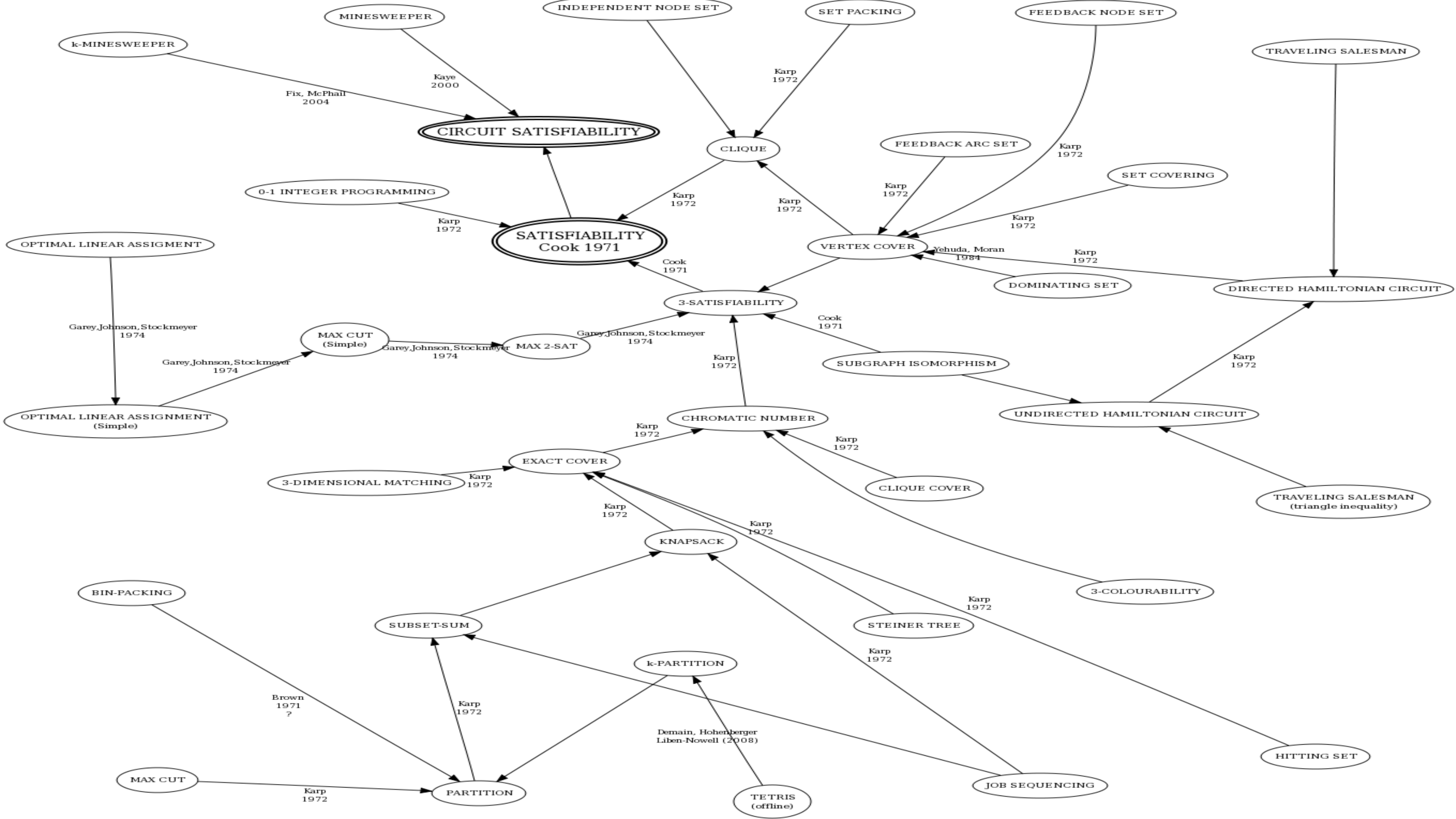
def Algo3SUM(A):
P = Reduce(A) ← $O(n)$
return AlgoColl(P) ← $* \; O(|P|^{1.9}) = O(n^{1.9})$

Total: $O(n^{1.9})$

Reality: We do not really know …

- Whether 3SUM can or cannot be solved in time $O(n^{1.9})$
- Whether COLL can or cannot be solved in time $O(n^{1.9})$

Suppose some bright student finds a super-fast algorithm for COLL. Then what ?
Suppose some bright student finds a $O(n^{1.9})$ lower bound for 3SUM. Then what ?
Suppose some bright student finds that COLL <= 3SUM. Then what ?



MINESWEEPER
INDEPENDENT NODE SET
SET PACKING
FEEDBACK NODE SET
k-MINESWEEPER
TRAVELING SALESMAN
Kaye
2000
Karp
1972
Fix, McPhail
2004
CIRCUIT SATISFIABILITY
CLIQUE
FEEDBACK ARC SET
Karp
1972
SET COVERING
0-1 INTEGER PROGRAMMING
Karp
1972
Karp
1972
Karp
1972
Karp
1972
Karp
1972
SATISFIABILITY
Cook 1971
OPTIMAL LINEAR ASSIGMENT
VERTEX COVER
Yehuda, Moran
1984
Karp
1972
Cook
1971
DOMINATING SET
DIRECTED HAMILTONIAN CIRCUIT
3-SATISFIABILITY
Cook
1971
Garey,Johnson,Stockmeyer
1974
MAX CUT
(Simple)
MAX 2-SAT
Garey,Johnson,Stockmeyer
1974
Garey,Johnson,Stockmeyer
1974
Karp
1972
SUBGRAPH ISOMORPHISM
Karp
1972
Garey,Johnson,Stockmeyer
1974
OPTIMAL LINEAR ASSIGNMENT
(Simple)
CHROMATIC NUMBER
UNDIRECTED HAMILTONIAN CIRCUIT
Karp
1972
Karp
1972
EXACT COVER
3-DIMENSIONAL MATCHING
Karp
1972
CLIQUE COVER
TRAVELING SALESMAN
(triangle inequality)
Karp
1972
Karp
1972
KNAPSACK
BIN-PACKING
3-COLOURABILITY
Karp
1972
SUBSET-SUM
STEINER TREE
Karp
1972
k-PARTITION
Brown
1971
?
Karp
1972
Demain, Hohenberger
Liben-Nowell (2008)
HITTING SET
MAX CUT
JOB SEQUENCING
Karp
1972
PARTITION
TETRIS
(offline)

3COL <= 4COL reduction

$poly(V_G, E_G) = (V_G+E_G)^{200}$

$poly(V_G, E_G) \Leftarrow V_{G'} = poly(V_G) \Leftarrow (V_{G'}+E_{G'})^{200}$

$\to V_G+1$

$E_G+V_G \leftarrow E_{G'} = poly(E_G)$

$(V+E)$

```
def Algo3COL(G):
    G' = Reduce(G)
    return Algo4COL(G')
```

3COL <= 4COL using a O(V+E) reduction.

→ If 4COL can be solved in time $O((V+E)^{200})$ then 3COL can be solved in ??? $O((V+E)^{200})$

X If 4COL can't be solved in time $O((V+E)^{200})$ then ... ???

X If 3COL can be solved in time $O((V+E)^{200})$ then ???

→ If 3COL can't be solved in time $O((V+E)^{200})$ then ??? → 4COL can't be solved in $O((V+E)^{200})$.

Reality: We do not really know ...

- Whether 3COL can or cannot be solved in time $O((V+E)^k)$ for any k
- Whether 4COL can or cannot be solved in time $O((V+E)^k)$ for any k

⇒ If 4COL can be solved in polytime, then 3COL can be solved in polytime.

⇒ If 3COL can't be solved in polytime, then 4COL can't be solved polytime.

$4COL \leq_p 3COL$: ⇒ If $3COL \in P$, then $4COL \in P$

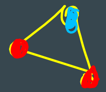
⇒ If $4COL \notin P$, then $3COL \notin P$

P vs NP problem

Answer Yes/No & proof if answer is yes.
If actual answer is Yes, proof can help me verify the "yes" fact.
If actual " " No, " should not convince me.

defn

PROB is NP-hard ⇔ If PROB can be solved in poly time, then every NP problem can be solved in poly time.



P : problems for which we know a polytime algorithm

NP : problems for which we know a fast (polytime) process algo. for yes-answer verification

→ proof: 2coloring

Given a graph G, can it be 2-coloured ? If yes, give me a proof that be verified fast.
Given a graph G, can it be 3-coloured ? If yes, give me a proof that be verified quickly.
Given an array A, is A sorted ? If yes, give me a proof that be verified quickly.
Given a graph G, does G have a clique with atleast k vertices? → X
Given ~~an array A and~~ an algorithm Algo, does Algo( ) return a sorted version of its input? If yes ...
Given a partially played chessboard, can white win from here (no matter what black plays)? If yes ... X
~~Given a 2-colourable graph G, can any 2-coloring of G be extended to a 3-colouring of G? If yes ...~~

* Given a digraph G, can G be topologically sorted ? If yes, give a proof that can be polytime verified.

Here is how you construct and explain the verification algorithms for NP. Take the example of 2COLOR.

```
def Verify2COLOR(instance G, proof C): // G is a graph with n vertices and C[1...n] lists the colour of each vertex
    If C uses more than 2 colours, return false
    For every edge e=(u,v) in G:
        If C[u] = C[v]: return false
    return true
```

Correctness claim:
(a) If G is 2 colorable, then there exists a proof C such that Verify2COLOR(G,C) returns true,
(b) If G is not 2 colorable, then for any G and any C, Verify2COLOR returns false.

Proof of (a): Let G be 2 colorable. So, define C as the list of the colours of its vertices. Since C is a valid colouring and uses only 2 colours, Verify2COLOR(G,C) will return true.

Proof of (b): We will prove the contrapositive of the claim. Suppose Verify2COLOR(G,C) returns true for some G and C. Then C must be using 2 colours and every edge of the graph must be assigned different colours. Hence, G must be 2 colorable.

Reading Assignment: NP-completeness chapter

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

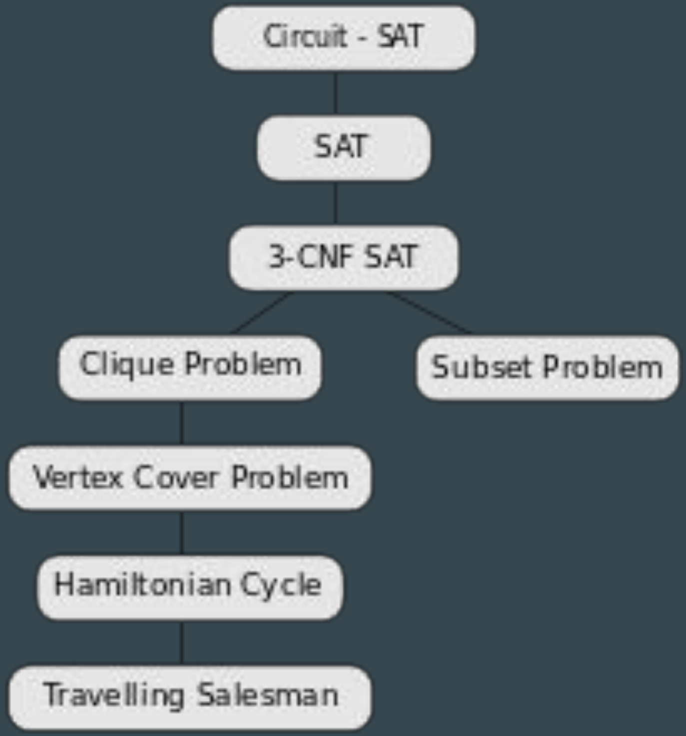
To prove that problem *A* is NP-hard, reduce a known NP-hard problem to *A*.

The cycle can be broken using a formal definition of NP-hardness.
For practical purpose, there are thousands of known NP-hard problems that can be used.
If no suitable problem is found, try a reduction from **3SAT** to π !
Last lecture: reductions from 3COLOR to 4COLOR and 2SAT to 3SAT.

Q: 3COLOR is known to be NP-hard. What can you say about 4COLOR from this statement?
Q: 2SAT is known to be polynomial-time. What can you say about 3SAT from this statement?
Q: 3SAT is known to be NP-hard. What can you say about 2SAT from this statement?

NP-complete = NP + NP-hard

Summary of NP-completeness



X and Y are NP-complete

- X is NP and X is NP-hard
    - All problems in NP have a reduction to X
- Y is NP and Y is NP-hard
    - All problems in NP have a reduction to Y
- $X \leq Y$ (by definition a reduction exists)
- $Y \leq X$ (by definition a reduction exists)
- If X can be solved in poly-time, then same for Y
- If Y can be solved in poly-time, then same for X
- If X can’t be solved in poly-time, same for Y
- If Y can’t be solved in poly-time, same for X