Reductions

Def: "Problem A reduces in polynomial time to Problem B"
if "you can use any efficient alg for B to efficiently solve A"
aka: "$A \leq_p B$" (A's difficulty $\leq$ B's difficulty)
(A is at most as hard as B)

Two-player Zero sum game
Input: Payoff matrix M Solution: Row player's optimal strategy

Thm: Zero Sum $\leq_p$ Linear Programming

Zero Sum input

$M = \begin{array}{|c|c|}\hline 3 & 1 \\ \hline 2 & 4 \\ \hline \end{array}$

Reduction alg →

LP input

$$\max \Sigma$$
$$\text{s.t. } \Sigma \leq 3p_1 + 2p_2$$
$$\Sigma \leq p_1 + 4p_2$$
$$p_1 + p_2 = 1$$
$$p_1, p_2 \geq 0$$

LP alg →

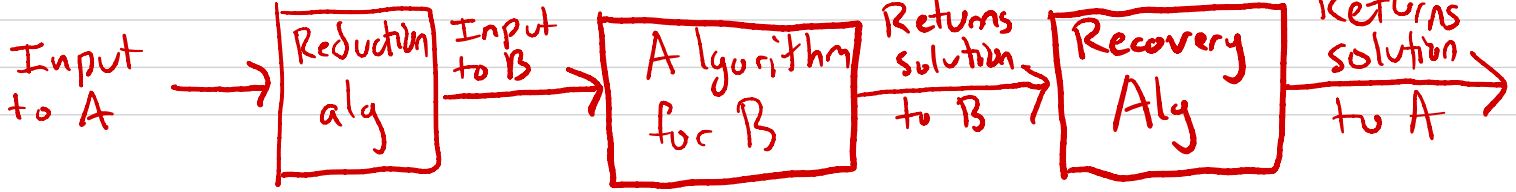
LP solution

$\Sigma = 2.5$
$p_1 = 1/2$
$p_2 = 1/2$

Recovery alg →

Zero Sum solution

$p_1 = 1/2$
$p_2 = 1/2$



Rudrata / Hamiltonian Cycle

Input: Graph $G = (V, E)$

Solution: tour of G

(= cycle visiting each vertex exactly once)

Min-TSP

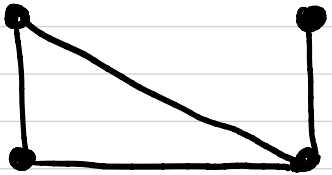
Input: cities $1, \ldots, n$

pairwise distances d_{ij}, $\forall i \neq j$

Solution: tour of minimum distance

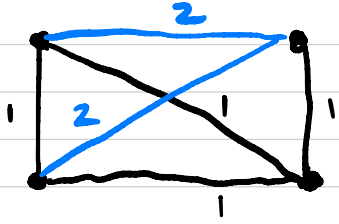
Thm: Rudrata Cycle $\leq_p$ Min-TSP (neither known to be in P!)

Rudrata instance G

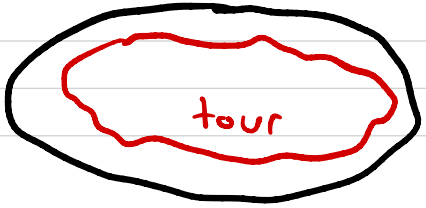


Reduction →

Min-TSP instance



Min-TSP solution



Recovery →

- Output tour if total weight $= n$
- Output "no tour" otherwise

Reduction notes

- Reduction & Recovery algs must be efficient (poly-time)
- $A \leq_p B$ and $B \leq_p C \Rightarrow A \leq_p C$
- Can prove $A \leq_p B$, even if A & B not known to be efficient
- "$A \leq_p B$" and "A reduces to B" most confusing thing in Algos

#1 Most Common Mistake:

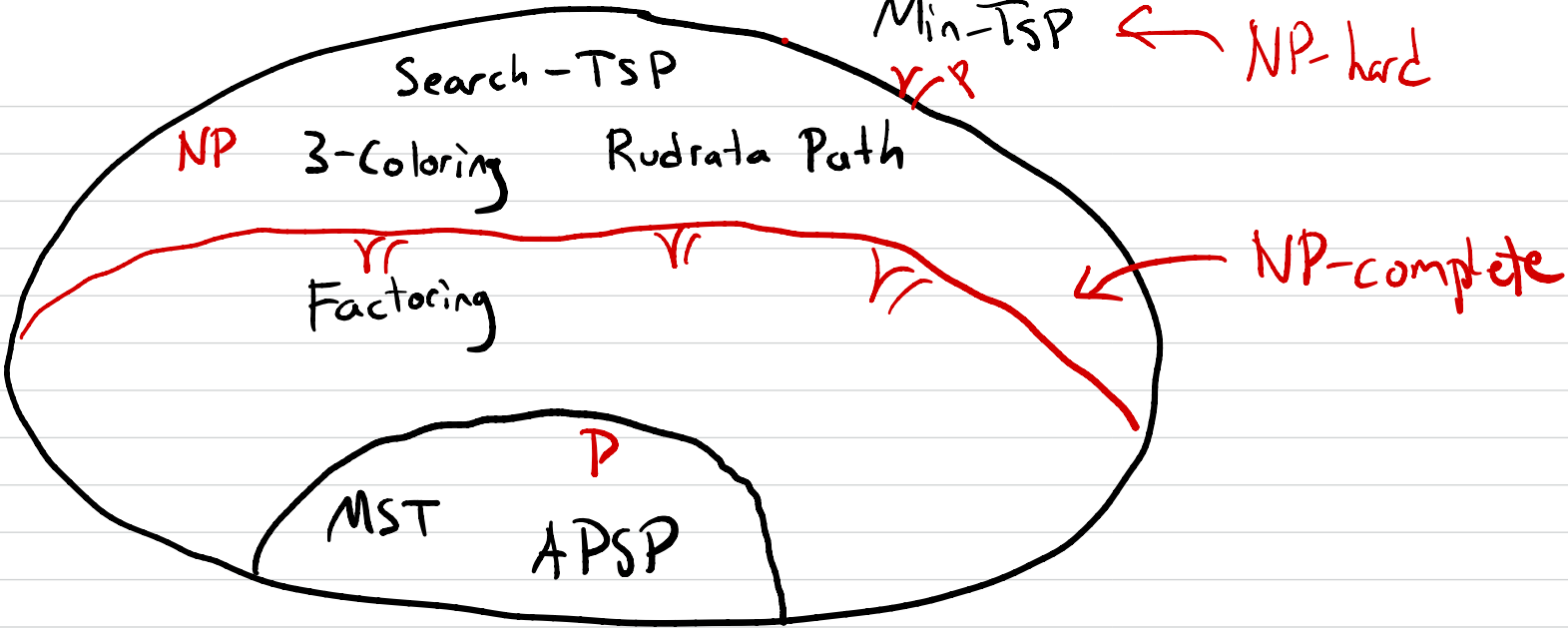
"Prove" $A \leq_p B$ by taking Instance of B $\xrightarrow{\text{reduction}}$ Instance of A ✗✗

Quiz: We have seen $A \leq_p B$ $B \leq_p A$

A = LP, B = Max Flow

A = Max Flow, B = Bipartite matching

✗ ✗



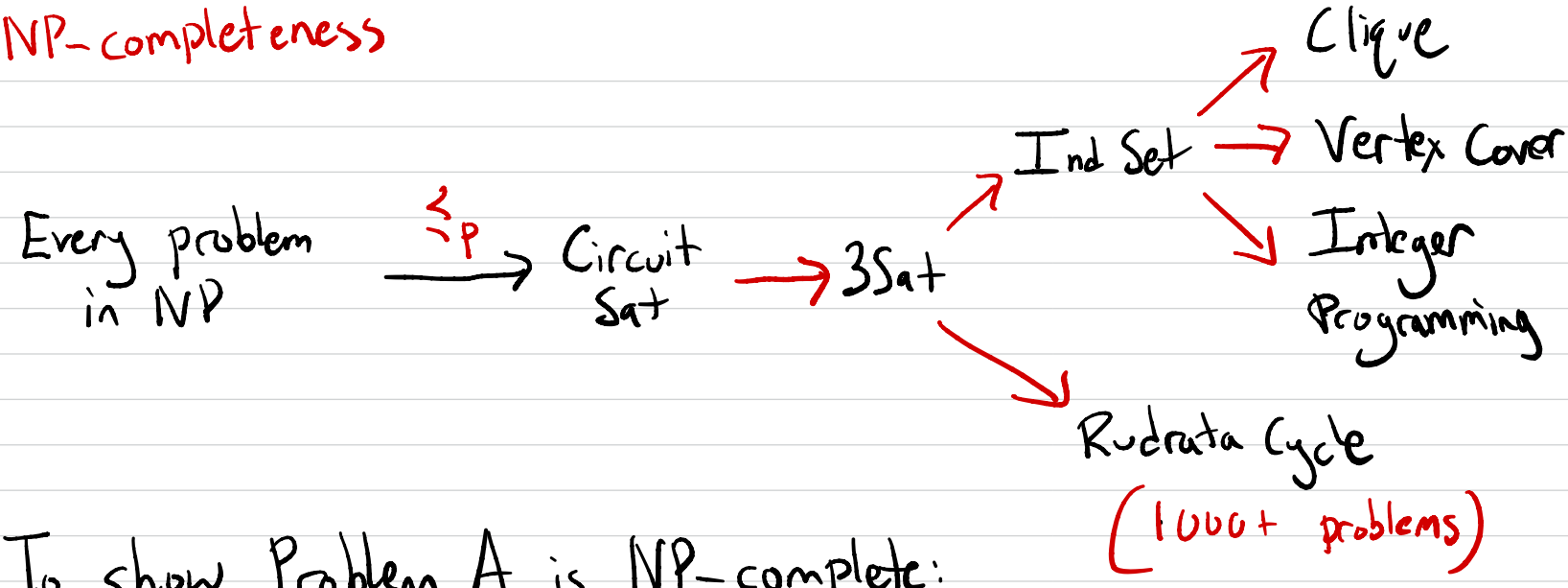
Def: A problem A is NP-hard if every problem $B \in NP$ reduces to A ($B \leq_P A$)

Def: A is NP-complete if $A \in NP$ & A is NP-hard.

Fact: If A & B are NP-complete, $A \leq_P B$ and $B \leq_P A$.

Fact: $\exists$ poly-time alg for NP-complete problem $\Rightarrow$ P=NP.

NP-completeness



To show Problem A is NP-complete:

1.) $A \in NP$ (show verification algorithm)

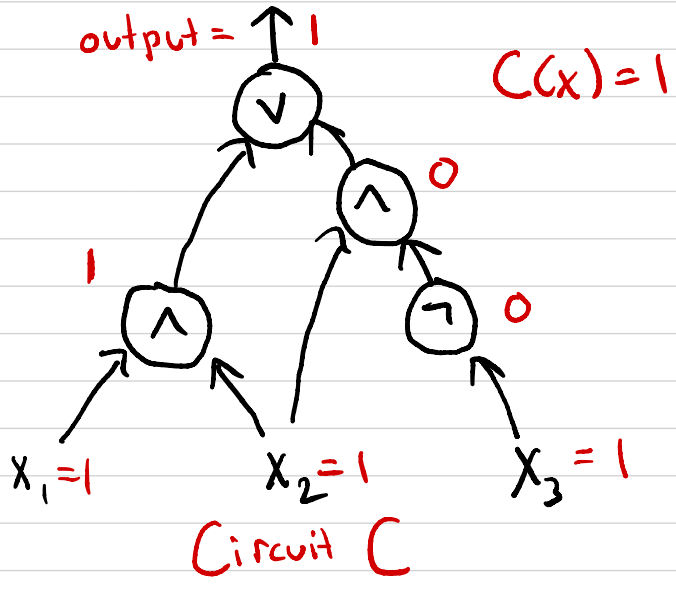
2.) Pick some (usually well-known) NP-complete problem B.
and show $B \leq_P A$. (Example: $3Sat \leq_P A$)

Circuit Sat

Def: A **Boolean circuit** is a directed acyclic graph (DAG)

- input nodes $x_1, \ldots, x_n$
- one output node
- gates marked

$\wedge$ AND, $\vee$ OR, $\neg$ NOT



Input: A circuit C

w/ n inputs and m gates

Solution: An assignment $x \in \{0,1\}^n$

s.t. $C(x) = 1$.

Cook-Levin Theorem: Circuit Sat is NP-complete

3Sat = THE NP-complete problem

Input: 1.) n Boolean variables $x_1, \ldots, x_n \in \{0,1\}$

2.) m $\leq$ 3-variable clauses $(x_1 \vee \overline{x_2} \vee x_8) \wedge (x_5 \vee \overline{x_6} \vee \overline{x_9}) \wedge (x_7 \vee x_{10}) \wedge \cdots$

Solution: An assignment $x_1, \ldots, x_n \to \{0,1\}$ satisfying all the clauses

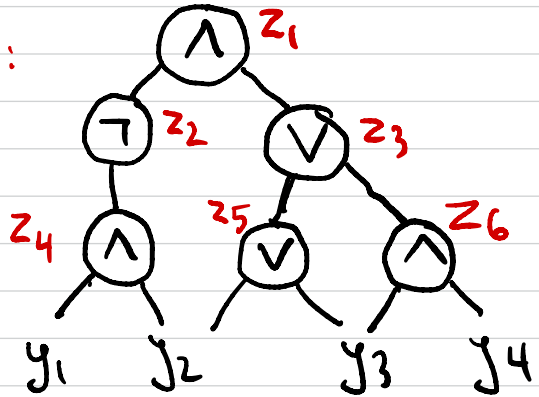
Thm: ~~3~~ kSat is NP-complete, for all $k \geq 3$ (see textbook)

Pf:
- 3Sat $\in$ NP.
- Circuit-Sat $\leq_p$ 3Sat. (next slide) □

Fun facts:
- 2Sat $\in$ P
- $\exists$ heuristic algs (e.g. DPLL) which do well in practice

Thm: Circuit Sat $\leq_p$ 3Sat

Circuit-Sat input:



n variables

m gates

Poly-time reduction:

$z_1 \wedge \boxed{(z_1 = z_2 \wedge z_3)}$

$\wedge (z_2 = \overline{z_4})$

$\wedge (z_3 = z_5 \vee z_6) \wedge \ldots$

3Sat clauses →

forbidden assignments $z_1\ z_2\ z_3$	3Sat Clauses
0 1 1	$(z_1 \vee \overline{z_2} \vee \overline{z_3})$
1 0 0	$\wedge (\overline{z_1} \vee z_2 \vee z_3)$
1 0 1	$\wedge (\overline{z_1} \vee z_2 \vee \overline{z_3})$
1 1 0	$\wedge (\overline{z_1} \vee \overline{z_2} \vee z_3)$

Poly-time recovery: $y_1, \ldots, y_n, z_1, \ldots, z_m \in \{0,1\}$ satisfying 3Sat inst. $\longrightarrow$ $y_1, \ldots, y_n$ satisfying Circuit Sat inst.

Integer Programming (IP)

Input: A linear program

Solution: An integer solution to LP

Thm: Ind-Set $\leq_p$ Integer Programming

Pf: Ind Set instance
- $G = (V, E)$
- $V = \{1, 2, \ldots, n\}$
- integer k

poly-time Reduction →

IP instance

$0 \leq x_i \leq 1$

$\sum_{i=1}^{n} x_i = k$

$\forall (i,j) \in E,\ x_i + x_j \leq 1$

Proof part 1: If $\exists$ ind set I of size k, $\exists$ solution to IP instance. (Just set $x_i = \begin{cases} 1 & \text{if } i \in I \\ 0 & \text{if } i \notin I \end{cases}$)

Proof part 2: If $(x_1, \ldots, x_n)$ is solution to IP, can "recover" and ind set of size k. (Just put $i \in I$ if $x_i = 1$)