REDUCING ONE (DECISION) PROBLEM TO ANOTHER

Input for A → solve A

Output of A → yes

Output of A → no
Reducing one (decision) problem to another

Input for A → Transform to input/problem B → Solve B

Output of B → Yes → Output of A → Yes

Output of B → No → Output of A → No

Solve A using B
REducing one (decision) problem to another

Input for A → Transform to input/problem B → Solve B

Solve A using B: $T_R + T_B$

Output of B → YES → YES
Output of A → NO → NO
Reducing one (decision) problem to another

Input for A → Transform to input/problem B → Solve B

\[ T_R \text{ and } T_B \]

Output of B → Yes/No → Output of A

Solve A using B: \[ T_R + T_B \]

Is L.I.S.(A) \[ \geq k \]?

Yes/No
REDUCING ONE (DECISION) PROBLEM TO ANOTHER

input for A → Transform to input/problem B

solve A using B : $T_R + T_B$

solve B

output of B

output of A

is L.I.S.($A$) \( \geq k \) ?

copy A → B

sort B

is L.C.S.($A, B$) \( \geq k \) ?

\( \text{YES} \)

\( \text{NO} \)

\( \text{YES} \)

\( \text{NO} \)
LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

input for $D$

solve $D$: known $\Omega(f(n))$

output of $D$

$C$ \rightarrow \text{YES}

$D$ \rightarrow \text{NO}
LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

input for D

Transform to input/problem C

solve C

output of C

YES

output of D

NO

YES

NO

solve D: known \( \Omega(f(n)) \)
LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

input for D

Transform to input/problem C

$T_R = o(f(n))$

solve C

output of C

output of D

NO

YES

solve D: known $\Omega(f(n)) = T_R + T_C \Rightarrow T_C = \Omega(f(n))$

C

D
Lower bound for one (decision) problem via another

This is just one way of solving D

solve D: known $\Omega(f(n)) = T_R + T_C \Rightarrow T_C = \Omega(f(n))$

- quick transformation $\Rightarrow$ C is at least as difficult as D
  (fast solution for C $\Rightarrow$ fast solution for D)

e.g. D= sorting, C= convex hull.

D might have another (easier) solution too, w/o using C
NPC REDUCTION FOR ONE (DECISION) PROBLEM VIA ANOTHER C D
NPC REDUCTION FOR ONE (DECISION) PROBLEM VIA ANOTHER

\[ \text{input for D} \rightarrow \text{D: known NPC} \rightarrow \text{output of D} \rightarrow \begin{cases} \text{YES} & \text{if D is \textit{YES}} \\ \text{NO} & \text{if D is \textit{NO}} \end{cases} \]
NPC REDUCTION FOR ONE (DECISION) PROBLEM VIA ANOTHER

D: known NPC
NPC REDUCTION FOR ONE (DECISION) PROBLEM VIA ANOTHER C → D

input for D

Transform to input/problem C

$T_R = \text{polynomial}$

solve C

$T_C$

output of C → YES

output of D → YES

D: known NPC
NPC REDUCTION FOR ONE (DECISION) PROBLEM VIA ANOTHER

input for D

Transform to input/problem C

$T_R = \text{polynomial}$

solve C

output of C

$T_c$

output of D

output

YES

NO

solve D: known NPC = $T_R + T_c = \text{poly-time} + T_c$

YES

NO
NPC REDUCTION FOR ONE (DECISION) PROBLEM VIA ANOTHER

\[ \text{solve } D: \text{ known NPC} = T_R + T_C = \text{poly-time} + T_C \]

\[ \Rightarrow T_C \text{ is NP-hard} \]

(or \( T_C \text{ is polynomial } \Rightarrow P = NP \))
NPC REDUCTION for one (decision) problem via another

\[ T_R = \text{polynomial} \]

\[ T_c \]

\[ T_c = T_R + T_c = \text{poly-time} + T_c \]

\[ \Rightarrow T_c \text{ is NP-hard} \]

& if C is in NP then C is NPC

(or $T_c$ is polynomial $\Rightarrow P=NP$)
NPC REDUCTION for one (decision) problem via another C (from D)

~like establishing an almost certain lower bound for C.

\[
\text{input for } D 
\xrightarrow{\text{Transform to input/problem C}} \text{C} \xrightarrow{\text{solution}} \text{output of } C \xrightarrow{\text{yes/no}} \text{yes/no}
\]

\[
T_R = \text{polynomial} \quad T_C
\]

\[
\text{solve D: known NPC } = T_R + T_C = \text{poly-time} + T_C
\Rightarrow T_C \text{ is NP-hard}
\]

\[
\text{if C is in NP then C is NPC}
\]

(or \( T_C \) is polynomial \( \Rightarrow P=NP \) (plus you rule and/or destroy the world))
$A \leq_p B \quad : \quad \text{in polynomial time, } A \text{ can be reduced to } B$
\[ A \leq_p B \text{ : in polynomial time, } A \text{ can be reduced to } B \]

\[ \Rightarrow B \text{ is at least as hard as } A. \]

Solving \( B \) in poly-time \( \Rightarrow \) solving \( A \) in poly-time
A \leq_p B : \text{ in polynomial time, A can be reduced to B}

\implies B \text{ is at least as hard as A.}

Solving B in poly-time \implies \text{solving A in poly-time}

If \{\text{any problem in NPC}\} \leq_p B \text{ then B is NPC.}

(assuming B is in NP, otherwise it is just NP-hard)
A \leq_p B : \text{ in polynomial time, } A \text{ can be reduced to } B

\implies B \text{ is at least as hard as } A.

Solving \( B \) in poly-time \( \implies \) solving \( A \) in poly-time

If \{ \text{any problem in } \text{NPC} \} \leq_p B \text{ then } B \text{ is NPC.}

(assuming \( B \) is in NP, otherwise it is just NP-hard)

Notice if \( A \& B \) are NPC then \( A \leq_p B \& B \leq_p A \).

\sim A \approx_p B
Thousands of NPC Problems
THOUSANDS OF NPC PROBLEMS

Reductions/transformations between them resemble a di-graph.
Thousands of NPC problems

Reductions/transformations between them resemble a di-graph.

\( \text{Is a DAG? NO.} \)
THOUSANDS OF NPC PROBLEMS

Reductions/transformations between them resemble a di-graph.

\[ \subseteq \text{ a DAG? No.} \]

strongly connected components?
Thousands of NPC Problems

Reductions/transformations between them resemble a di-graph.

\[ \geq \text{a DAG? No.} \]

\#strongly connected components?
Must be 1, precisely.
Thousands of NPC Problems

Reductions/transformations between them resemble a di-graph.

\( \subseteq \) a DAG? No.

# Strongly connected components?
Must be 1 precisely.

Theorem:
\( \exists \) at least one problem A s.t. A \( \geq_p \) everything?
THOUSANDS OF NPC PROBLEMS

Reductions/transformations between them resemble a di-graph.

\( \subseteq \) a DAG? No.

#strongly connected components? Must be 1. precisely.

**FACT**

\( \exists \) at least one problem \( A \) s.t.

\( A \geq_p \{ \text{everything} \} \)

& \( \{ \text{everything} \} \geq_p \cdots \geq_p \cdots \geq_p A \)
Circuit SAT (satisfiability)
The first NPC problem.
Circuit SAT (satisfiability)
The first NPC problem.
Given a circuit, can the output ever be 1?

binary
\( x_i \): input

\[
\begin{align*}
x_1 & \quad \text{OR} \\
x_2 & \quad \text{OR} \\
x_3 & \quad \text{NOT} \\
x_4, x_5 & \quad \text{AND} \\
\end{align*}
\]
Circuit SAT (satisfiability)
The first NPC problem.
Given a circuit, can the output ever be $1$?
Circuit SAT (satisfiability)
The first NPC problem.
Given a circuit, can the output ever be 1?

\[ x_i : \text{input} \]

\[
\begin{align*}
0 & \quad x_1 \\
1 & \quad x_2
\end{align*}
\]

\[
\begin{align*}
0 & \quad x_3 \\
1 & \quad x_4 \\
0 & \quad x_5
\end{align*}
\]

\[
\begin{align*}
\text{OR} & \quad 1 \\
\text{OR} & \quad 1 \\
\text{AND} & \quad \text{output}
\end{align*}
\]
Circuit SAT (satisfiability)
The first NPC problem.
Given a circuit, can the output ever be 1?

\[ x_i: \text{input} \]

0 \quad x_1 \quad \text{OR} \quad 1

0 \quad x_2 \quad \text{OR}

0 \quad x_3 \quad \text{NOT}

1 \quad x_4 \quad \text{AND}

0 \quad x_5 \quad \text{AND}

\text{(not yet, try again)}

\text{output}
Circuit SAT (satisfiability)
The first NPC problem.
Given a circuit, can the output ever be 1?
Circuit SAT (satisfiability)
The first NPC problem.
Given a circuit, can the output ever be 1?

- **Xi**: input
  - 0: x₁
  - 0: x₂
  - 0: x₃
  - 1: x₄
  - 1: x₅

- Output: 1 (yes)
Circuit SAT (satisfiability) 
The first NPC problem. 
Given a circuit, can the output ever be 1? 
1 - in NP ... intuitive

\[ x_i : \text{input} \]
Circuit SAT (satisfiability)
The first NPC problem.

Given a circuit, can the output ever be 1?

1 - in NP ... intuitive
2 - every problem can be described as a circuit
\{everything\} \leq_p circuit-SAT

\[ X_i : \text{input} \]

\[ x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \]

\[ \text{AND} \quad \text{AND} \]

\[ \text{OR} \quad \text{OR} \quad \text{NOT} \]

\[ \text{output} \]
Circuit SAT (satisfiability)
The first NPC problem.
Given a circuit, can the output ever be 1?
1 - in NP ... intuitive
2 - every problem can be described as a circuit
\{everything\} \leq_P circuit-SAT
... and we don't know how to solve this in polynomial time. (not in P)
Boolean SAT \[ (x_1 \lor x_2 \lor \overline{x_3}) \land ((x_1 \leftrightarrow x_5) \lor (x_4 \rightarrow \overline{x_3})) \rightarrow ? \rightarrow 1 \]
boolean SAT  but here's a reduction  circuit SAT

Boolean SAT  \((x_1 \lor x_2 \lor \overline{x_3}) \land ((x_1 \leftrightarrow x_5) \lor (x_4 \rightarrow \overline{x_3}))\)  \(\rightarrow?\rightarrow1\)

Given a circuit-SAT instance

```
c_1
---
c_2
---
 OR
---
 OR
---
 OR
---
 NOT
---
 c_3
---
 AND
---
 AND
---
c_4
---
c_5
---
```
Boolean SAT \( (x_1 \lor x_2 \lor \overline{x}_3) \land ((x_1 \leftrightarrow x_5) \lor (x_4 \rightarrow \overline{x}_3)) \rightarrow ? \rightarrow 1 \)

Given a circuit-SAT instance transform (quickly) into a Boolean SAT.

\[
\begin{align*}
C_1 \quad &\text{OR} \\
C_2 \quad &\text{OR} \\
C_3 \quad &\text{NOT} \\
C_4 \quad &\text{AND} \\
C_5 \quad &\text{AND}
\end{align*}
\]

\[
\begin{align*}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
w
\end{align*}
\]

\[
w \land (w \leftrightarrow (y_4 \land y_2 \land y_3))
\]
Boolean SAT \((x_1 \lor x_2 \lor \overline{x_3}) \land ((x_1 \leftrightarrow x_5) \lor (x_4 \rightarrow \overline{x_3})) \quad \rightarrow ? \rightarrow 1\)

Given a circuit-SAT instance, transform (quickly) into a Boolean SAT. 

\[
\begin{align*}
w & \land (w \leftrightarrow (y_4 \land y_2 \land y_3)) \\
& \land (y_4 \leftrightarrow (y_1 \lor y_2))
\end{align*}
\]
Boolean SAT \[(x_1 \lor x_2 \lor \bar{x}_3) \land ((x_1 \leftrightarrow x_5) \lor (x_4 \rightarrow \bar{x}_3)) \rightarrow ? \rightarrow 1\]

Given a circuit-SAT instance, transform (quickly) into a Boolean SAT.

\[w \land (w \leftrightarrow (y_4 \land y_2 \land y_3))\]
\[\land (y_4 \leftrightarrow (y_1 \lor y_2))\]
\[\land (y_1 \leftrightarrow (c_1 \lor c_2))\]
Boolean SAT: \((x_1 \lor x_2 \lor \overline{x_3}) \land ((x_1 \leftrightarrow x_5) \lor (x_4 \rightarrow \overline{x_3})) \quad \rightarrow ? \rightarrow 1\)

Given a circuit-SAT instance, transform (quickly) into a Boolean SAT:

\[
\begin{align*}
\neg w & \land (w \leftrightarrow (y_4 \land y_2 \land y_3)) \\
& \land (y_4 \leftrightarrow (y_1 \lor y_2)) \\
& \land (y_1 \leftrightarrow (c_1 \lor c_2)) \\
& \land (y_2 \leftrightarrow \overline{c_3})
\end{align*}
\]
Boolean SAT \( (x_1 \lor x_2 \lor \overline{x}_3) \land ((x_1 \leftrightarrow x_5) \lor (x_4 \rightarrow \overline{x}_3)) \rightarrow ? \rightarrow 1 \)

Given a circuit-SAT instance, transform (quickly) into a Boolean SAT.

\[
\begin{align*}
\text{w} \land (\text{w} \leftrightarrow (Y_4 \land Y_2 \land Y_3)) \\
\land (Y_4 \leftrightarrow (Y_1 \lor Y_2)) \\
\land (Y_1 \leftrightarrow (C_1 \lor C_2)) \\
\land (Y_2 \leftrightarrow \overline{C}_3) \\
\land (Y_3 \leftrightarrow (C_4 \land C_5))
\end{align*}
\]
boolean SAT \rightarrow \text{Circuit SAT}

\text{circuit-SAT} \leq_p \text{Boolean SAT}

\text{Boolean SAT} \quad (x_1 \lor x_2 \lor \overline{x_3}) \land ((x_1 \leftrightarrow x_5) \lor (x_4 \rightarrow \overline{x_3})) \quad \rightarrow ? \rightarrow 1

\text{Given a circuit-SAT instance transform (quickly) into a Boolean SAT.}

\text{by solving the Boolean SAT you get an answer for circuit SAT}

\begin{align*}
&\text{w} \land (\text{w} \leftrightarrow (Y_4 \land Y_2 \land Y_3)) \\
&\land (Y_4 \leftrightarrow (Y_1 \lor Y_2)) \\
&\land (Y_1 \leftrightarrow (C_1 \lor C_2)) \\
&\land (Y_2 \leftrightarrow \overline{C_3}) \\
&\land (Y_3 \leftrightarrow (C_4 \land C_5))
\end{align*}
3-SAT

\[(x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_4 \lor x_5) \land (x_2 \lor \overline{x_4} \lor x_{13}) \land (x_2 \lor \overline{x_3} \lor x_3) \land \ldots\]

\[n \text{ clauses each with 3 literals}\]
\[(a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor \overline{b} \lor \overline{d})\] 
\[\text{\{ k clauses \}}\]

\[\text{construct graph (quickly)}\]
\[(a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor \overline{b} \lor \overline{d})\]

\[k\ \text{clauses}\]

Construct graph (quickly)

for every clause-triple \((x \lor y \lor z)\)

make a triple of vertices, linked together
\[(a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor \overline{b} \lor \overline{d})\]

\[k \text{ clauses}\]

\[3\text{-SAT} \rightarrow \text{boolean SAT} \rightarrow \text{circuit SAT}\]

\[\text{construct graph (quickly)}\]
(a ∨ b ∨ c) ∧ (b ∨ c ∨ d) ∧ (a ∨ c ∨ d) ∧ (a ∨ b ∨ d)

k clauses

3-SAT → boolean SAT → circuit SAT

construct graph (quickly)
\[(a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor \overline{b} \lor \overline{d})\]  

\(k\) clauses

3-SAT → boolean SAT → circuit SAT

Construct graph (quickly)

Link every vertex to any other vertex that has the same variable but negated
\((a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor \overline{b} \lor \overline{d})\) \(k\) clauses

3-SAT → boolean SAT → Circuit SAT

Ask: does this graph have \(k\) independent vertices?
\[(a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor \overline{b} \lor \overline{d}) \}\]

\[k \text{ clauses}\]

3-SAT \rightarrow \text{boolean SAT} \rightarrow \text{circuit SAT}

```
\text{Ask: does this graph have } k \text{ independent vertices?}
```

To find \( k \) independent vertices, you must use each triple, because each triple can only have 1 independent vertex.
\[(a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor b \lor \overline{d})\] 

\[k \text{ clauses}\]

Construct graph (quickly)

3-SAT $\rightarrow$ boolean SAT $\rightarrow$ circuit SAT

Ask: does this graph have \( k \) independent vertices?

To find \( k \) independent, you must use each triple, because each triple can only have 1 independent vertex.

As soon as you select a vertex in a triple, that blocks you from using its negation anywhere else.
$(a \lor b \lor c) \\
\land (b \lor \overline{c} \lor \overline{d}) \\
\land (\overline{a} \lor c \lor d) \\
\land (a \lor \overline{b} \lor \overline{d})$

$k$ clauses

3-SAT \xrightarrow{\text{boolean SAT}} \text{Circuit SAT}

If you can't find $k$ independent vertices, then some triple had no selectable vertex... i.e. all 3 were blocked by selections in other triples. That corresponds to an unsatisfiable clause in 3-SAT.

Ask: does this graph have $k$ independent vertices?

If yes, then 3-SAT is $\checkmark$
If no, then 3-SAT is $\times$
\((a \lor b \lor c)\) and \((b \lor \overline{c} \lor \overline{d})\) and \((\overline{a} \lor c \lor d)\) and \((a \lor \overline{b} \lor \overline{d})\) \(k\) clauses

\[ \begin{align*}
\text{3-SAT} & \quad \text{boolean SAT} \\
\text{independent set} & \quad \text{Circuit SAT}
\end{align*} \]

Construct graph (quickly)

Ask: does this graph have \(k\) independent vertices?

If yes, then 3-SAT is \(\checkmark\)
If no, then 3-SAT is \(\times\)
CLIQUE in a graph: subset of $V$ s.t. all pairs of vertices share edges.
CLIQUE in a graph: subset of $V$ s.t. all pairs of vertices share edges.

$G$  complement ($G$)
CLIQUE in a graph: subset of $V$ s.t. all pairs of vertices share edges.

independent set size $k$ \iff clique size $k$
CLIQUE in a graph: subset of $V$ s.t. all pairs of vertices share edges.

independent set size $k \iff$ clique size $k$

$G \xrightarrow{\text{transform}} \text{complement}(G)$

Asking if a graph has a clique of size $k$ is NPC
OTHER HARD PROBLEMS

- knapsack: given item types, w/ size & value, fill a bag w/ max value (multiples ok)
- subset sum: does a subset of given integers sum to t?
- partition: split set of integers in 2 groups with equal sums
- planar SAT: SAT without wire crossings
- set cover: given some sets, select min # sets s.t. all elements are present
- hitting set: given sets, select min # elements s.t. all sets are represented
- longest path: visiting each vertex once.
- Steiner tree: ~MST on selected subset of a graph
- traveling salesman: min-cost simple cycle on weighted complete graph
OTHER HARD PROBLEMS

... & some are even harder

Tetris  Minesweeper  Lemmings

Mario Bros  Pac-man

Prince of Persia  Portal  Doom

etc