REDUCING ONE (DECISION) PROBLEM TO ANOTHER

input
for A → solve A → output of A

A

B

yes

no
REducing one (Decision) Problem to Another

input for A → Transform to input/problem B → solve B → output of B → solve A using B → output of A
REducing one (Decision) problem to another

input for A → Transform to input/problem B → solve B

output of B

output of A

solve A using B : $T_R + T_B$
Reducing one (decision) problem to another

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

\[ T_R + T_B \]

Solve A using B

is L.I.S. (A) \( \geq k \) ? → YES → NO

Output of B → YES → NO

Output of A → NO → NO
REDUCING ONE (DECISION) PROBLEM TO ANOTHER

input for A → Transform to input/problem B

solve A using B \( : T_R + T_B \)

\( T_R \) \( T_B \)

output of B \( \rightarrow \) yes \( \rightarrow \) yes

output of A \( \rightarrow \) no \( \rightarrow \) no

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

\( \rightarrow \) copy A → B \( \rightarrow \) sort B

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

\( \rightarrow \) yes \( \rightarrow \) yes

\( \rightarrow \) no \( \rightarrow \) no
LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

input for D

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

output of D

C

D

YES

NO
LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

\[ \text{solve } D : \text{known } \Omega(f(n)) \]
LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

input for D

Transform to input/problem C

$T_R = o(f(n))$

solve C

$T_C$

output

of C

YES

output

of D

NO

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

YES

NO
For D

\[ T_R = \gamma(\lambda_G) \]

Transform to problem C

Solve D: Known \( \gamma(\lambda_G) \) \( \Rightarrow T_C = \gamma(\lambda) \)

Solve C

Yes

Output of C

Yes

Output of D

A

Lower bound for one (decision) problem via another
Lower bound for one (decision) problem via another:

- Transform to input/problem C
  - $T_R = o(f(n))$
- Solve C
  - $T_C = \Omega(f(n))$
- Solve D: known $\Omega(f(n)) = T_R + T_C \Rightarrow T_C = \Omega(f(n))$

Quick transformation $T_R \Rightarrow C$ is at least as difficult as D

(fast solution for C ⇒ fast solution for D)
The convex hull problem (definition by picture)

**C.H. input:** list of (2D) points

\[
\begin{bmatrix}
P_0 & P_5 & P_3 & P_{12} & P_i & \ldots & \text{etc...}
\end{bmatrix}
\]

\[(x_{12}, y_{12})\]

**C.H. output:** a graph (cycle) containing only the "extreme" points

\[
\begin{bmatrix}
P_3 & P_2 & P_{13} & P_5 & P_{10} & P_6 & P_8
\end{bmatrix}
\]
Example

LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

Input

for sorting

Solve sorting: known 2(nlogn)

Sorted output


Example

Transform to 2D point set

\[ T_R = \Theta(n) = o(n\log n) \]

Solve sorting: known \( \Omega(n\log n) \)

\[ (x_i, x_i^2) \]

\[ \Theta(1) \uparrow \]

\[ x_i \]
LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

Example

input for sorting

Transform to 2D point set
\[ T_R = \Theta(n) = o(n\log n) \]

solve C.H.
\[ T_{C.H.} \]

Sorted output

solve sorting: known \( \Omega(n\log n) \)

\[ (x_i, x_i^2) \]
\[ \Theta(1) \uparrow \]
\[ x_i \]

\[ \Theta(n) \]

\( x_6 \) \( x_3 \) \( x_5 \) \( x_4 \) \( x_1 \) \( x_2 \)
Lower bound for one (decision) problem via another

Example

Transform to 2D point set

\[ T_R = \Theta(n) = o(n \log n) \]

Solve C.H.

\[ T_{C.H.} \]

\[ T_R' = \Theta(n) \]

Sorted output

Solve sorting: known \( \Omega(n \log n) \)

\( \{x_i, x_i^2\} \)

\( \Theta(1) \uparrow \)

\( x_i \)

\( \Theta(n) \)

\( \rightarrow \)

\[ \{P_1, P_2, P_3, P_4, P_5, P_6\} \]

\( \Theta(n) \)

\( x\)-coordinates
LOWER BOUND FOR ONE (Decision) PROBLEM VIA ANOTHER

Example

Transform to 2D point set

\[ T_R = \Theta(n) = o(n \log n) \]

Solve C.H.

\[ T_C.H. \]

\[ T_{R'} = \Theta(n) \]

Sorted output

Solve sorting: known \( \Omega(n \log n) = T_R + T_{R'} + T_{C.H.} \) \( \Rightarrow T_{C.H.} = \Omega(n \log n) \)

\[ (x_i, x_i^2) \]
\[ \Theta(n) \]
\[ \Theta(1) \]

\[ x_i \]

\[ x_6, x_5, x_4, x_1, x_2 \]

\[ P_1, P_2, P_3, P_4, P_5, P_6 \]

\( \Theta(n) \) \( x \)-coordinates
NPC REDUCTION FROM ONE (DECISION) PROBLEM TO ANOTHER
NPC REDUCTION
FROM ONE (DECISION) PROBLEM TO ANOTHER

\[ D \rightarrow C \]

D: known NPC

Input for D

Output of D

Yes

No
NPC REDUCTION FROM ONE (DECISION) PROBLEM TO ANOTHER

D: known NPC

input for D

Transform to input/problem C

solve C

output of C

output of D

YES

NO

YES

NO
NPC Reduction: From one (decision) problem to another

\[ T_R = \text{polynomial} \]

\[ T_C \]

\[ D: \text{known NPC} \]
NPC REDUCTION FROM ONE (DECISION) PROBLEM TO ANOTHER

\[ T_R = \text{polynomial} \]

\[ T_C \]

Input for D → Transform to input/problem C → Solve C → output of C

\[ \text{output of } C \]

Yes \rightarrow Yes

No \rightarrow No

output of D

\[ \text{output of } D \]

Yes \rightarrow Yes

No \rightarrow No

Solve D: known NPC = \[ T_R + T_C = \text{poly-time} + T_C \]
NPC REDUCTION FROM ONE (DECISION) PROBLEM TO ANOTHER

\[ T_R = \text{polynomial} \]

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

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

(or \( T_c \text{ is polynomial } \Rightarrow P=NP \))
NPC REDUCTION

FROM ONE (DECISION) PROBLEM TO ANOTHER

\[ D \rightarrow C \]

\[ \text{input for } D \rightarrow \text{Transform to input/problem } C \]

\[ T_R = \text{polynomial} \]

\[ \text{solve } C \rightarrow \text{output of } C \]

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

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

\& if C is in NP then C is NPC

(or \( T_C \) is polynomial \( \Rightarrow P=NP \))
NPC REDUCTION FROM ONE (DECISION) PROBLEM TO ANOTHER D C

LIKE ESTABLISHING AN ALMOST CERTAIN LOWER BOUND FOR C.

\[ T_{R} = \text{polynomial} \]

solve \( D \): known NPC \[ = T_{R} + T_{C} = \text{poly-time} + T_{C} \]

\[ \Rightarrow T_{C} \text{ is NP-hard} \]

& if C is in NP
then C is NPC)

(or \( T_{C} \) is polynomial \( \Rightarrow P=NP \))
$A \leq_p B$ : in polynomial time, $A$ can be reduced to $B$
\[ A \leq_p B \quad : \quad \text{in polynomial time, } A \text{ can be reduced to } B \]

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

Solving \( B \) in poly-time \( \implies \) solving \( A \) in poly-time
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{every problem in NP}\} \leq_p B \text{ then } B \text{ is NP-hard}

and if B is in NP, ... ?
A ≤_p B : in polynomial time, A can be reduced to B

⇒ B is at least as hard as A.
Solving B in poly-time ⇒ solving A in poly-time

If \{every problem in NP\} ≤_p B

then B is NP-hard
(and if B is in NP, then B is NPC)
A \leq_p B : \text{ in polynomial time, } A \text{ can be reduced to } B

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

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

\text{If } \{\text{every problem in NP}\} \leq_p B
\text{ or } \exists \text{ any NPC problem}\}
\text{ then } B \text{ is NP-hard}
\text{ (and if } B \text{ is in NP, then } B \text{ is NPC})
\( A \leq_p B \) : in polynomial time, \( A \) can be reduced to \( B \)
\( \implies B \) is at least as hard as \( A \).
Solving \( B \) in poly-time \( \implies \) solving \( A \) in poly-time

If \( \{ \text{every problem in NP} \} \leq_p B \) for some \( \text{NPC problem} \) then \( B \) is NP-hard
(and if \( B \) is in NP, then \( B \) is NPC)

if \( A \) \& \( B \) are NPC then \( A \leq_p B \) \& \( B \leq_p A \)
\[ \sim A \equiv B \]
Thousands of NPC Problems

X

Y

T

S

B

W

D

A
Thousands of NPC Problems

Reductions/transformations between them resemble a di-graph.

\[ \Rightarrow \]
THOUSANDS OF NPC PROBLEMS

Reductions/transformations between them resemble a di-graph.

# Strongly connected components? Must be 1 precisely.
THOUSANDS OF NPC PROBLEMS

Reductions/transformations between them resemble a di-graph.

THEOREM

\exists \text{ at least one problem } A \text{ s.t. } A \geq P \{ \text{everything} \}
\& \{ \text{everything} \} \geq P \ldots \geq P \ldots \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?

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

\[ x_i : \text{input} \]

\[
\begin{align*}
0 & \quad x_1 \quad \text{OR} \\
1 & \quad x_2 \\
0 & \quad x_3 \quad \text{NOT} \\
1 & \quad x_4 \quad \text{AND} \\
0 & \quad x_5 \\
\end{align*}
\]

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

$x_i$: input

```
  0  |  x_1
  1  |  x_2

       OR
       └── OR
           └── OR
               └── AND

        NOT
        ┌───
        │ 0
        │  x_3
        └───

    AND
    ┌───
    │ 0
    │  x_4
    └───

    AND
    ┌───
    │ 0
    │  x_5
    └───

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?

\[ \begin{align*}
X_i: \text{input} \\
0 & x_1 \\
0 & x_2 \\
0 & x_3 \\
0 & x_4 \\
0 & x_5 \\
\end{align*} \]
Circuit SAT (satisfiability)
The first NPC problem.
Given a circuit, can the output ever be 1?

\[
\begin{align*}
&x_1, x_2, x_3, x_4, x_5 \text{ : inputs} \\
&\left\{ \begin{array}{c}
0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0 \\
\end{array} \right. \\
\end{align*}
\]
Circuit SAT (satisfiability)
The first NPC problem.
Given a circuit, can the output ever be 1?
$1 \in \text{NP} \ldots$ intuitive
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 \text{circuit-SAT}
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
\[ \text{\{} \text{everything} \text{\}} \leq_p \text{circuit-SAT} \]
...and we don't know how to solve this in polynomial time. (not in P)
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
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

[Diagram of a circuit with labels and gates]
Boolean SAT: \((x_1 \lor x_2 \lor \overline{x_3}) \land ((x_1 \leftrightarrow x_5) \lor (x_4 \rightarrow \overline{x_3}))\) → ? → 1

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

\[ w \land (w \leftrightarrow (Y_4 \land Y_2 \land Y_3)) \]
Given a circuit-SAT instance, transform (quickly) into a Boolean SAT.

\[ w \land (w \leftrightarrow (Y_4 \land \neg Y_2 \land \neg Y_3)) \land (Y_4 \leftrightarrow (Y_1 \lor Y_2)) \]
Boolean SAT: \((x_1 \lor x_2 \lor \overline{x_3}) \land ((x_1 \leftrightarrow x_5) \lor (x_4 \rightarrow \overline{x_3})) \rightarrow 1\)

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

\[
\begin{align*}
W \land (W \leftrightarrow & (Y_4 \land Y_2 \land Y_3)) \\
& \land (Y_4 \leftrightarrow (Y_1 \lor Y_2)) \\
& \land (Y_1 \leftrightarrow (C_1 \lor C_2))
\end{align*}
\]
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:

\[
\begin{align*}
  w & \land (w \leftrightarrow (Y_4 \land Y_2 \land Y_3)) \\
  & \land (Y_4 \leftrightarrow (Y_1 \lor Y_2)) \\
  & \land (Y_1 \leftrightarrow (C_1 \lor C_2)) \\
  & \land (Y_2 \leftrightarrow \overline{C_3})
\end{align*}
\]
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.

\[
\begin{align*}
w \land (w \leftrightarrow (y_4 \land y_2 \land y_3)) \\
\land (y_4 \leftrightarrow (y_1 \lor y_2)) \\
\land (y_1 \leftrightarrow (c_1 \lor c_2)) \\
\land (y_2 \leftrightarrow \overline{c_3}) \\
\land (y_3 \leftrightarrow (c_4 \land c_5))
\end{align*}
\]
Given a circuit-SAT instance, transform (quickly) into a Boolean SAT.

\[ \begin{align*}
&\text{circuit-SAT} \leq_p \text{Boolean SAT} \\
&\text{(actually also prototypical)}
\end{align*} \]

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

\[
\rightarrow \ ? \rightarrow 1
\]
3-SAT

\[(x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_4 \lor x_5) \land (x_2 \lor \overline{x}_4 \lor x_{13}) \land (x_2 \lor \overline{x}_3 \lor x_3) \ldots\]

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

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

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

3-SAT → boolean SAT → circuit SAT

construct graph (quickly)

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

\{k \text{ clauses}\}

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

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

\[k \text{ clauses}\]

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

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

construct graph (quickly)

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

\(k\) clauses

construct graph (quickly)

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

Would need 1 vertex per triple (can't have 2 in any triple)

Answer maps back to 3-SAT
\[(a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor \overline{b} \lor \overline{d})\] 

\[k\] clauses

3-SAT \rightarrow boolean SAT \rightarrow circuit SAT

Construct graph (quickly)

\[\leq 1\] per triple

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

If yes, then 3-SAT is \(\checkmark\)

If no, then 3-SAT is \(\times\)
\[(a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor \overline{b} \lor \overline{d})\]\n
\[\text{k clauses}\]

\[\text{3-SAT} \to \text{boolean SAT} \to \text{Circuit SAT}\]

\[\text{independent set}\]

\[\text{construct graph (quickly)}\]

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

\[\text{If yes, then 3-SAT is } \checkmark\]
\[\text{If no, then 3-SAT is } \times\]
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.

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

Size $k$ independent set in $G$?

$G \xrightarrow{\text{transform}} \text{complement } (G) = G^c$
CLIQUE in a graph: subset of $V$ s.t. all pairs of vertices share edges.

Size $k$ independent set in $G$? $\iff$ Size $k$ clique in $G^c$?

$G$ \quad \text{complement} (G) = G^c
CLIQUE in a graph: subset of $V$ s.t. all pairs of vertices share edges.

Size $k$ independent set in $G$? $\iff$ Size $k$ clique in $G^c$?

$G$ \quad \text{complement (} G \text{) } = G^c$

Asking if a graph has a clique of size $k$: 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