

Problems

An *abstract problem* is a relation from *instances* to *solutions*.

E.g., integer addition: instance $2 + 2$, solution 4.

E.g., equations: instance $x^2 = 9$, solutions 3 and -3 .

A *decision problem* has solutions = {yes,no}.

Any abstract problem has a related decision problem.

For integer addition, use instances like $2 + 2 \geq 3$.

An *encoding* maps abstract objects to (binary) strings.

E.g., a graph can be mapped to adjacency list or matrix.

Encodings map abstract problems to *concrete problems*.

The P Complexity Class

A problem Q is *polynomial-time solvable* if there exists an algorithm A and a constant k such that A solves all instances in $O(n^k)$, where n is the length of the encoded instance.

The *language* of a decision problem is the set of instances with yes solutions.

The polynomial complexity class P is defined:

$$P = \{L : L \text{ is a language accepted by a polynomial-time algorithm}\}$$

Reducibility

A problem Q can be reduced to a problem Q' if each instance I of Q can be transformed to an instance I' of Q' such that the solution to I' can be transformed back to a solution for I .

Finding the median reduces to sorting.

Difference constraints reduces to shortest-paths.

A language L_1 is *poly-time reducible* to L_2 ($L_1 \leq_P L_2$) if a poly-time alg. A satisfies $x \in L_1$ iff $A(x) \in L_2$.

If $L_1 \leq_P L_2$ and $L_2 \in P$, then $L_1 \in P$.

If $L_1 \leq_P L_2$ and $L_1 \notin P$, then $L_2 \notin P$.

Proof:

Let A_r be a $O(n^k)$ reduction alg. for $L_1 \leq_P L_2$.

Let A_2 be a $O(n^l)$ algorithm for L_2 .

We can define an algorithm A_1 for L_1 by

$$A_1(x) = A_2(A_r(x))$$

A_r is $O(n^k)$ implies the size of $A_r(x)$ is $O(|x|^k)$.

A_2 is $O(n^l)$ implies A_2 is $O(|x|^{kl})$ on an input of size $O(|x|^k)$.

It follows that $A_1 \in P$ because A_1 is $O(n^{kl})$.

Reducibility Examples

$$\text{DOUBLE} = \{ \langle x, y \rangle : 2x \leq y \}$$

$$\text{ADD} = \{ \langle a, b, c \rangle : a + b \leq c \}$$

$$\text{DOUBLE} \leq_P \text{ADD} \text{ by } f(x, y) = \langle x, x, y \rangle$$

$$\text{LCM} = \{ \langle m, x, y \rangle : m \text{ is the least common multiple of } x \text{ and } y \}$$

$$\text{GCD} = \{ \langle d, x, y \rangle : d \text{ is the greatest common divisor of } x \text{ and } y \}$$

$$\text{LCM} \leq_P \text{GCD} \text{ by } f(m, x, y) = \langle xy/m, x, y \rangle$$

$$\text{SIMPLE} = \{ \langle G, u, v, l \rangle : \text{there is a simple path from } u \text{ to } v \text{ with length } l \}$$

$$\text{HAMPATH} = \{ \langle G, u, v \rangle : \text{there is a simple path from } u \text{ to } v \text{ including all vertices} \}$$

$$\text{HAMPATH} \leq_P \text{SIMPLE} \text{ by}$$

$$f(G, u, v) = \langle G, u, v, n - 1 \rangle$$

where n is the number of vertices

The NP Complexity Class

Let A be an alg. with 2 inputs and 0-1 outputs.
 Let $L_A = \{x : \text{there is a } y \text{ with } A(x, y) = 1\}$.

A is a *verification algorithm*. y is a *certificate*.

E.g., if x is a Boolean formula, y is a truth assignment, and $A(x, y) = 1$ if y makes x true, then L_A is the set of satisfiable Boolean formulas.

$\text{NP} = \{L_A : A(x, y) \in \text{P and } y \text{ is poly-size}(x)\}$

More examples:

Does a weighted graph have negative cycles?

Do two numbers have a divisor greater than 1?

Is a number composite?

Can a logic circuit output a 1?

Can a set of numbers be split into equal sums?

Are there integers $p, q > 0$ s.t. $ap^2 + bq = c$?

It is believed that $\text{P} \neq \text{NP}$, but not proven.

NP-Completeness and Reducibility

L is *NP-hard* if $L' \leq_P L$ for every $L' \in \text{NP}$.

L is *NP-complete* if $L \in \text{NP}$ and L is NP-hard.

If L is NP-complete and $L \in \text{P}$, then $\text{NP} = \text{P}$.

No known L is both NP-complete and P.

CIRCUIT-SAT = set of satisfiable logic circuits

CIRCUIT-SAT is NP-complete because it is in NP and all NP problems \leq_P CIRCUIT-SAT.

Sketch of reduction to CIRCUIT-SAT.

Simulate a computer cycle by a logic circuit plus Boolean inputs/outputs.

Simulate a sequence of cycles by a sequence of circuits.

A poly-time verification algorithm and a poly-size certificate implies a poly-number of cycles, which implies a poly-size circuit.

NP-Completeness Proofs

To show that L is NP-complete

1. Show $L \in \text{NP}$.
2. Show $L' \leq_P L$ for NP-complete language L' .

SAT = set of satisfiable Boolean formulas

1. Truth assignments can be verified.
2. CIRCUIT-SAT \leq_P SAT as follows.
 Map each gate's output to a variable.
 Map each gate to a formula.
 AND all the formulas.

3-CNF-SAT = set of satisfiable conjunctions of clauses, where each clause is a disjunction of 3 literals.

1. Truth assignments can be verified.
2. SAT \leq_P 3-CNF-SAT as follows.
 Map each operator output to a variable.
 Create a dummy input for each negation.
 Each operator has 2 inputs and 1 output.
 Use truth tables to map each op. to clauses.
 AND all the clauses.

CLIQUE

A clique of an undirected graph is a subset of vertices with edges between all pairs.

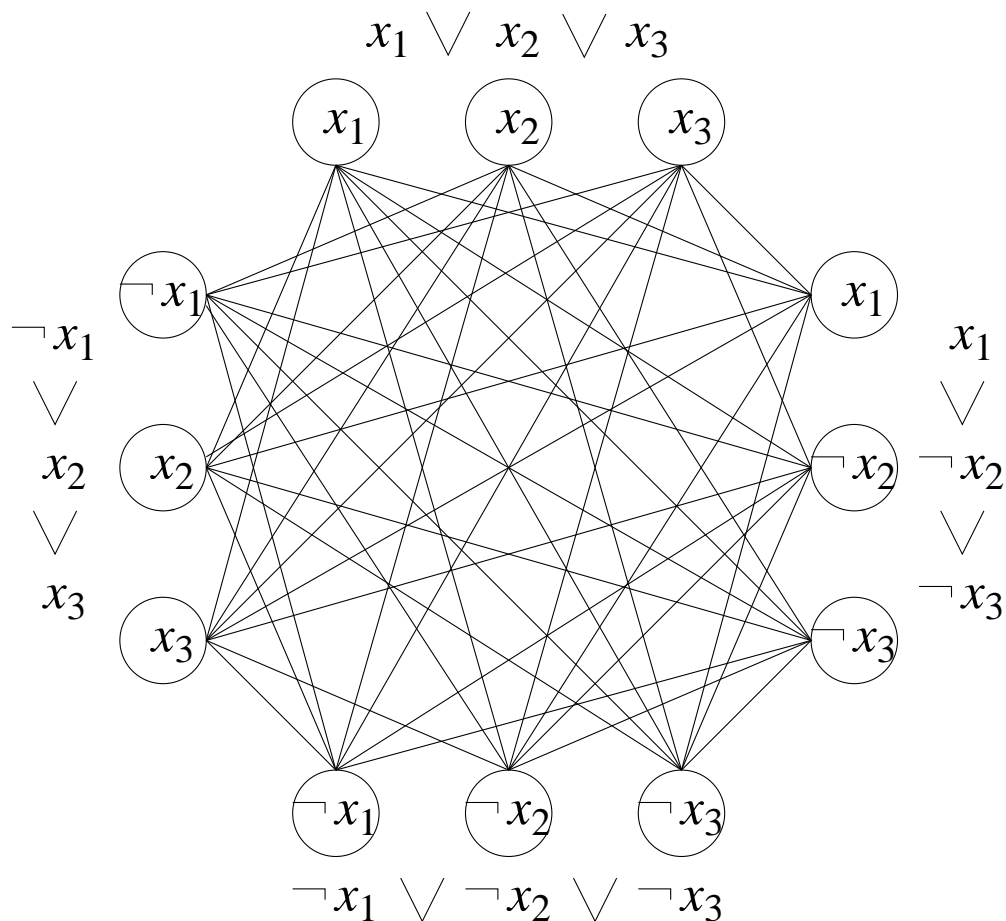
$\text{CLIQUE} = \{ \langle G, k \rangle : G \text{ has a clique of size } k \}$

1. It is polynomial to verify a clique.
2. $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$ as follows.

Map each of m clauses to three vertices.

Edges for consistent vertices of diff. clauses.

Satisfying assignment iff clique of size m .



VERTEX COVER

A vertex cover of an undirected graph is a subset of vertices that covers all the edges.

VERTEX-COVER =

$$\{\langle G, k \rangle : G \text{ has a vertex cover of size } k\}$$

1. It is polynomial to verify a vertex cover.
2. CLIQUE \leq_P VERTEX-COVER.

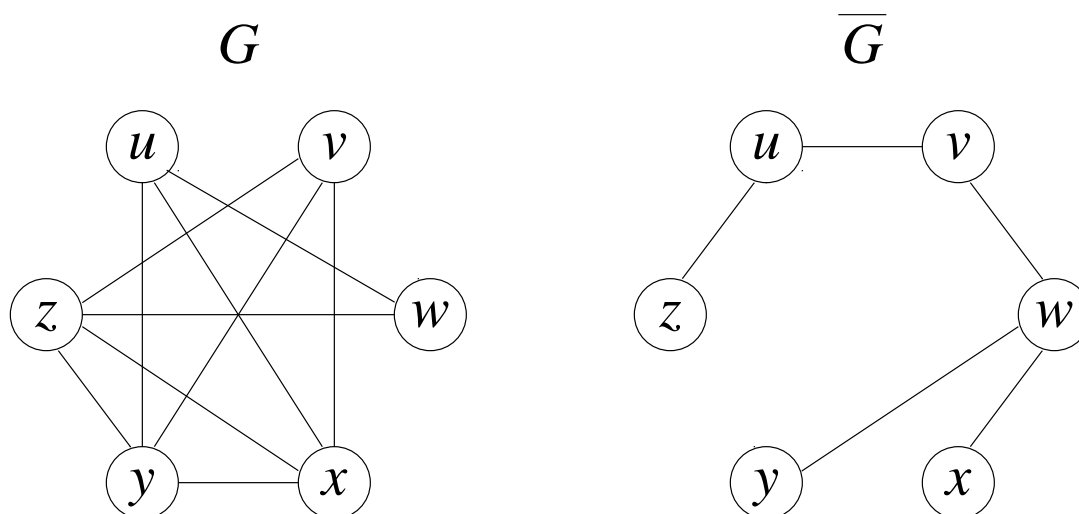
Map clique graph G to its complement \overline{G} .

\overline{V} is a clique of \overline{G} iff V is a vertex cover of G .

Proof of reduction:

If V is a clique of G , then \overline{G} has no edge in $V \times V$, so \overline{V} is a vertex cover of \overline{G} .

If V is not a clique of G , then \overline{G} has an edge in $V \times V$, so \overline{V} is not a vertex cover \overline{G} .



SUBSET SUM

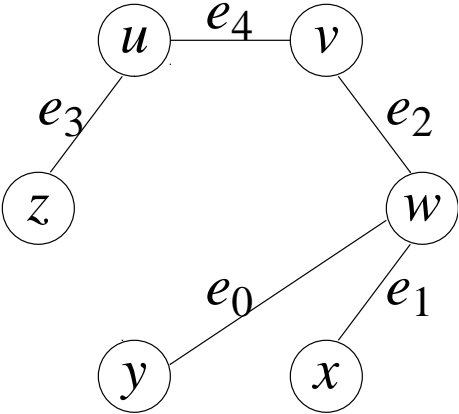
SUBSET-SUM =

$\{\langle S, t \rangle : \text{A subset } S' \subseteq S \text{ satisfies } t = \sum_{s \in S'} s\}$

1. It is polynomial to verify a subset sum.
2. VERTEX-COVER \leq_P SUBSET-SUM.

Map vertices to S using incidence matrix.

Must cover each edge to reach sum.

G	base 4					decimal				
	e_4	e_3	e_2	e_1	e_0					
	u	$=$	1	1	1	0	0	0	$=$	1344
	v	$=$	1	1	0	1	0	0	$=$	1296
	w	$=$	1	0	0	1	1	1	$=$	1045
	x	$=$	1	0	0	0	1	0	$=$	1028
	y	$=$	1	0	0	0	0	1	$=$	1025
	z	$=$	1	0	1	0	0	0	$=$	1088
	4^0	$=$	0	0	0	0	0	1	$=$	1
	4^1	$=$	0	0	0	0	1	0	$=$	4
	4^2	$=$	0	0	0	1	0	0	$=$	16
	4^3	$=$	0	0	1	0	0	0	$=$	64
	4^4	$=$	0	1	0	0	0	0	$=$	256
	t	$=$	2	2	2	2	2	2	$=$	2730