Complexity Classes
P, NP, coNP
Reductions, NP-completeness

CS 4231, Fall 2017 Mihalis Yannakakis

Tractable vs. Intractable Problems

- Shortest simple path
- Fractional Knapsack problem
- Linear Programming
- Graph 2-Colorability
- Node cover in bipartite graphs
- Shortest tour that traverses all edges of a graph (Chinese Postman problem)

Polynomial Time

- Longest simple path
- 0-1 Knapsack problem
- Integer Linear Programming
- Graph 3-Colorability
- Node cover in general graphs
- Shortest tour that visits all nodes of a graph (Traveling Salesman problem)

Exponential Time as far as we know (but not proven)

Why Polynomial Time?

- P= problems solvable in polynomial time: $O(n^c)$ for some constant *c*, for example $O(n),\ O(n^2 \log n),\ O(n^4),\ O(n^{100})$
- A generous qualitative notion of tractability

- Advantages:
- Dramatic difference in rate of growth between polynomials and exponentials
- Not solvable in P means certainly intractable (in worst case)
- Polynomials closed under +, ×, *composition*: poly(poly(n)) is poly(n), so combining polynomial algorithms gives a polynomial algorithm
- *Robust* notion, invariant to reasonable changes in
  - Representation (encoding) of the input and output
  - Computational model

Encoding (Representation) of Objects

- Objects (graphs etc) can be represented in various ways, for example graphs via adjacency matrix or adjacency lists. Such reasonable representations (i) are polynomially related in size, and (ii) we can translate between them in polynomial time.

- Ultimately, everything → $\{0,1\}^*$

- Binary standard encoding: Assume all objects, inputs, outputs of problems are encoded in binary in some standard way, and that we can determine in polynomial time if a binary string is a valid encoding (and decode it).

- Size of input = number of bits

Encoding ctd.

- In particular, numbers are written in binary; logN bits to represent N: exponential difference between size (#bits) and the value of the number

- Using other constant bases, or other alphabets of constant size to encode input, does not change much the size.

- Sometimes, we want to study the complexity of algorithms/problems when numbers are written in *unary*: N is 11..111 (N times). Then we stress this explicitly.
- Algorithms that run in polynomial time under this unary encoding of numbers (and standard binary encoding for other objects) are called pseudopolynomial algorithms. - *Not* a polynomial time algorithm

Types of Computational Problems

Input

Instance of the problem

Output

s-t Reachability: Graph G,s,t $\longrightarrow \begin{cases} \text{Yes, if s can reach t} \\ \text{No, if s cannot reach t} \end{cases}$

A Decision Problem: Yes/No output , just one bit of output

- s-t Distance: Weighted graph G,w; s,t → distance(s,t)

  one correct output (a function)

- s-t shortest path: Weighted graph G,w; s,t → shortest s-t path

  possibly, multiple correct outputs

Optimization → Decision Problem

- It is common to define complexity classes like P and NP as classes of *decision problems* - problems with Yes/No (0/1) answer. Function and optimization problems studied in terms of a decision problem version.

- Optimization problem Π: Every instance I has a set of solutions, every solution has a value (profit or cost). The problem is to compute the maximum or minimum possible value and a solution that achieves it.

- Corresponding Decision problem for Π:

  Input: instance I of Π, bound b on value

  Output: Yes iff optimum value ≥b for maximization problem (resp. optimum value ≤b for minimization problem), else No

Examples of Decision versions

- Graph Coloring problem:

  Input: Undirected graph G, bound b
  Output: Yes if ∃ legal coloring with at most b colors, No otherwise

- (0-1) Knapsack:

  Input: values $v_1,\dots,v_n$, weights $w_1,\dots,w_n$ of the items, knapsack capacity B, bound b on total value
  Output: Yes if ∃ subset S of items s.t. $w(S) \le B$ and $v(S) \ge b$

- Node Cover:

  Input: Graph G, bound b
  Output: Yes, if ∃ node cover S s.t. $|S| \le b$, No otherwise

- Integer Linear Programming:

  Input: ILP with rational coefficients, bound b
  Output: Yes, if ∃ integer solution with value $\ge b$, No otherwise

  - Equivalent to Integer Linear Inequalities Solvability: Does a given set of linear inequalities have an integer solution?

Optimization vs. Decision Problem

- Optimization problem Π is at least as hard as the corresponding Decision problem: solving opt. Π gives immediately answer for decision

- In most cases, it is possible (but harder) to go also the other way using a subroutine for the Decision problem repeatedly (but only a polynomial number of times).
- Step 1. Find the optimal value
- Step 2. Find an optimal solution, one piece at a time

Optimization vs. Decision Problem

- Example- Node Cover: Input graph G=(N,E)
- Suppose we have a routine DecNC for the decision problem

1. Find the size c* of the minimum node cover: Call DecNC for inputs (G,b), b=0,1,...,n (or do binary search on b) to find the minimum b for which DecNC returns Yes.
2. Find a minimum node cover, one node at a time:

- If c*=0 (i.e. G has no edges) then return ∅, else
- 1. Find a node u for which DecNC with input (G-u, c*-1) returns Yes. (Try all nodes until we find one such node.)
- 2. Then call recursively the minimum node cover algorithm on the graph G-u and optimal value c*-1, and output the set that it returns together with node u

Functional problems → Decision problems

- Factoring – decision version

Input: Number N, number b (note: input size = logN+logb)

Output: Yes, if N has a nontrivial ($\neq$1,N) factor $\leq$ b, else No

- Can factor N with polynomially many calls to an algorithm for the decision problem: Use binary search to find the smallest nontrivial factor of N (using logN calls to the decision algorithm), divide N by the factor, and repeat.

- Any "functional" problem Π (only one correct output) can be reduced to a series of calls to a decision problem:

Input: Instance x of Π, index i

Output: Yes, if the i-th bit of the output of Π on input x is 1, No otherwise

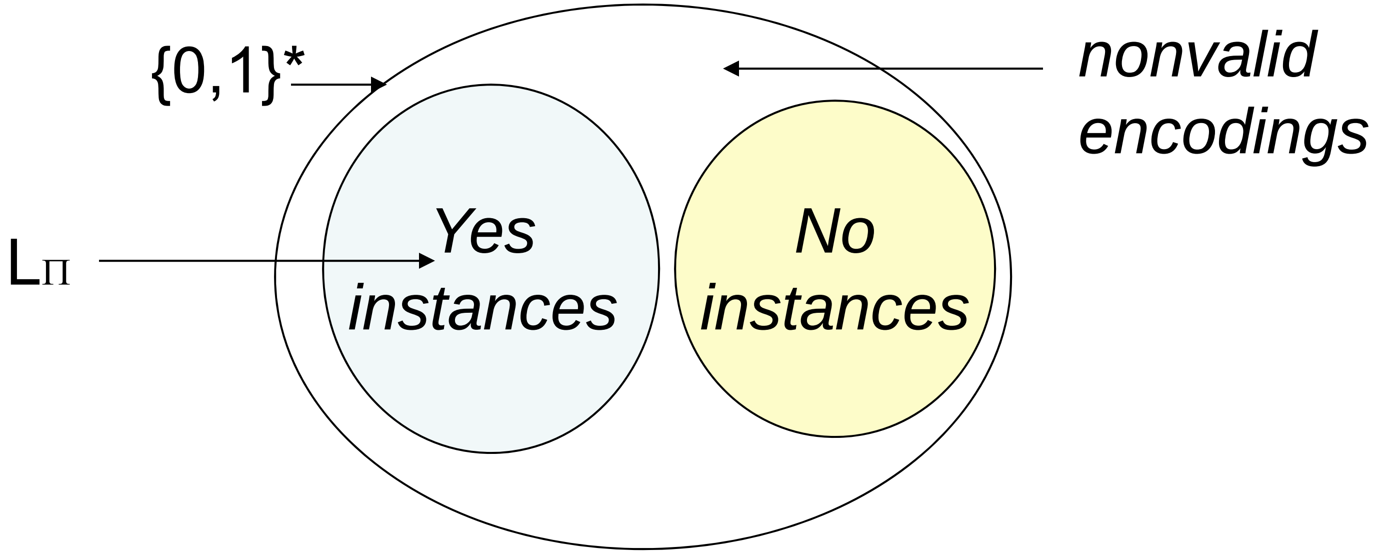
Decision Problems ↔ Languages over {0,1}

- Language over alphabet {0,1}=set of binary strings: $L \subseteq \{0,1\}^*$
- Every language L corresponds to a decision problem:

  Input: binary string $x \in \{0,1\}^*$

  Output: Yes if $x \in L$, No otherwise
- Every decision problem Π corresponds to a language

  L_Π = set of encodings of Yes instances of Π



$\{0,1\}^*$

nonvalid encodings

L_Π

Yes instances

No instances

Nonvalid encodings of instances lumped with the No instances in definition of L_Π

P as a class of Languages (Decision Problems)

- P = class of languages L (decision problems Π) that can be decided in polynomial time, i.e. there is a polynomial time algorithm which on input $x \in \{0,1\}^*$ returns Yes if $x \in L$ (resp. $x \in L_\Pi$, i.e., x is a Yes instance of Π), No otherwise.

- P is closed under union, intersection, complement, difference, i.e. if L1, L2 are in P then so are $L1 \cup L2$, $L1 \cap L2$, $\{0,1\}^* - L1$, $L1 - L2$

For a decision problem Π, binary strings that are not valid encodings of instances were lumped with the No instances; Not important, since we can determine in polynomial time whether a string is a valid encoding or not.

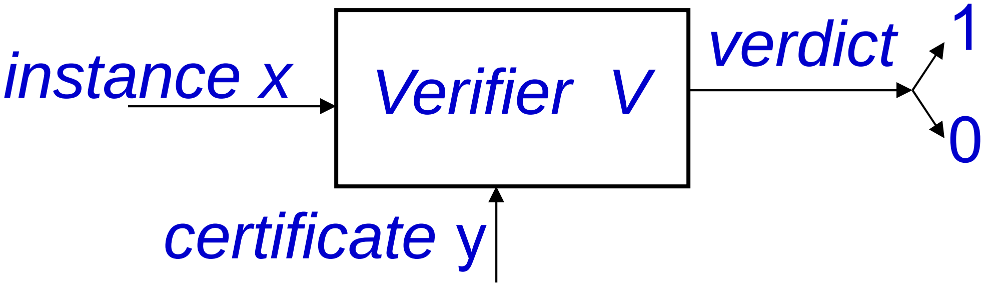
L_Π =\{Yes instances of Π\}$\in$P iff $\{0,1\}^* - L_\Pi \in$ P iff

\{No instances of Π\}$\in$P

Class NP

- Languages L (decision problems Π) such that we can verify efficiently that $x \in L$ (x is a Yes instance) if we are also given a short certificate (witness/proof/solution) y
- Formally, NP = class of languages L s.t. there is a constant c and a two-input polynomial time algorithm V ("the verifier") such that

$$L = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^*,\ |y| \le |x|^c,\ V(x,y) = 1\}$$



- $x \in L \ \Rightarrow\ \exists y \in \{0,1\}^*,\ |y| \le |x|^c \text{ s.t. } V(x,y) = 1$
- $x \notin L \ \Rightarrow\ \forall y \in \{0,1\}^*,\ |y| \le |x|^c.\ \ V(x,y) = 0$

Class NP

- The name NP stands for Nondeterministic Polynomial time:
- a nondeterministic algorithm *guesses* a certificate and then verifies it.

- Examples
- Node Cover: Instance x=(G,b).
  certificate y=node cover S s.t |S|≤b
- 0-1Knapsack problem: Instance x= ($v_1,\dots,v_n$, $w_1,\dots,w_n$,B, b)
  certificate y=set $S\subseteq\{1,\dots,n\}$ s.t. w(S)≤B, v(S)≥b
- (Formula) Satisfiability: Instance x=Boolean formula *f.*
  certificate y=truth assignment to the variables s.t. f(y)=true

More Examples

- Graph 3 Colorability: Instance: Graph G

  certificate y = assignment of a “color”$\in$\{1,2,3\} to each node s.t. adjacent nodes are assigned different colors

- Graph Hamiltonicity: Instance: Graph G

  certificate y = cycle that goes through every node exactly once (Hamiltonian cycle)

- Graph Isomorphism: Instance: Two graphs G, H

  certificate= isomorphism f from G to H, i.e. bijection on nodes s.t. $(u,v)\in E(G)$ iff $(f(u),f(v))\in E(H)$

- Number Compositeness: Instance: number N

  certificate = a factor of N

Examples ctd.

- Sometimes it is not entirely obvious that there is a “short” certificate (i.e. polynomially bounded), and a nontrivial proof is required. For example:
- Linear Programming
- Integer Linear Programming
- If there is a (integer) solution to a set of linear inequalities then there is one with a number of bits that is bounded by a polynomial in the input size (number of variables and inequalities in the input set and number of bits in the coefficients)

Big Open Question

$P = NP$?

Conjecture: $P \neq NP$

Class coNP

- Definition of NP is nonsymmetric with respect to Yes, No

$$\text{coNP} = \{\ L \mid \overline{L} = \{0,1\}^* - L \in \text{NP}\ \}$$

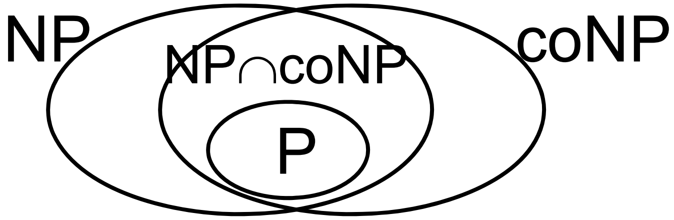
- A decision problem Π is in coNP if the complement of its Yes language L_Π is in NP $\Leftrightarrow$ its No language (set of No instances) is in NP

- Properties:

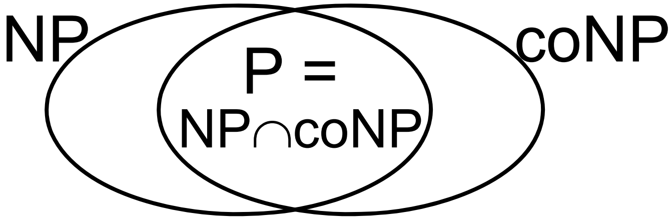
- P$\subseteq$NP, P$\subseteq$coNP , thus, P$\subseteq$NP$\cap$coNP
- NP is closed under union, intersection
- coNP is also closed under union, intersection
- NP (and coNP) closed under complement iff NP=coNP (conjectured not)

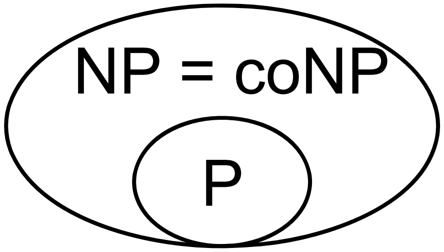
Relations between P, NP, coNP

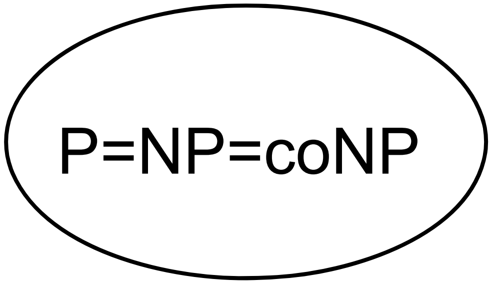
Conjectured relation:
all distinct



Other possibilities:







Fundamental Questions

- P = NP?

Is it always as easy to generate a proof as it is to check a proof that is given to us?

- NP = coNP?

Is it always possible to provide simple convincing evidence that something does not exist, as it is to show that something exists by exhibiting it?

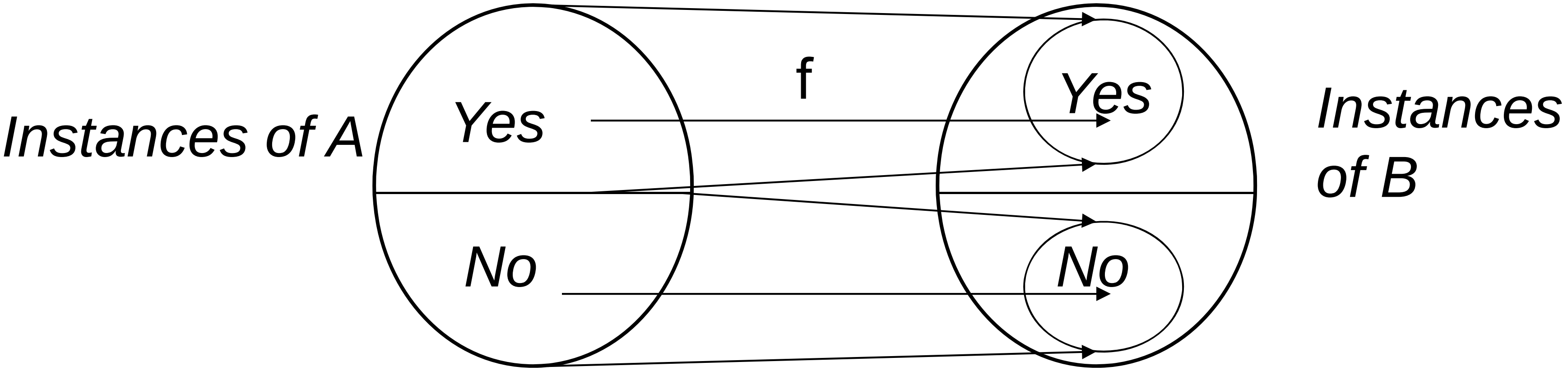
Reductions between Decision Problems

- Polynomial time Reduction from language L to L’ (notation $L \leq_P L'$): polynomial time computable function f: $\{0,1\}^* \rightarrow \{0,1\}^*$

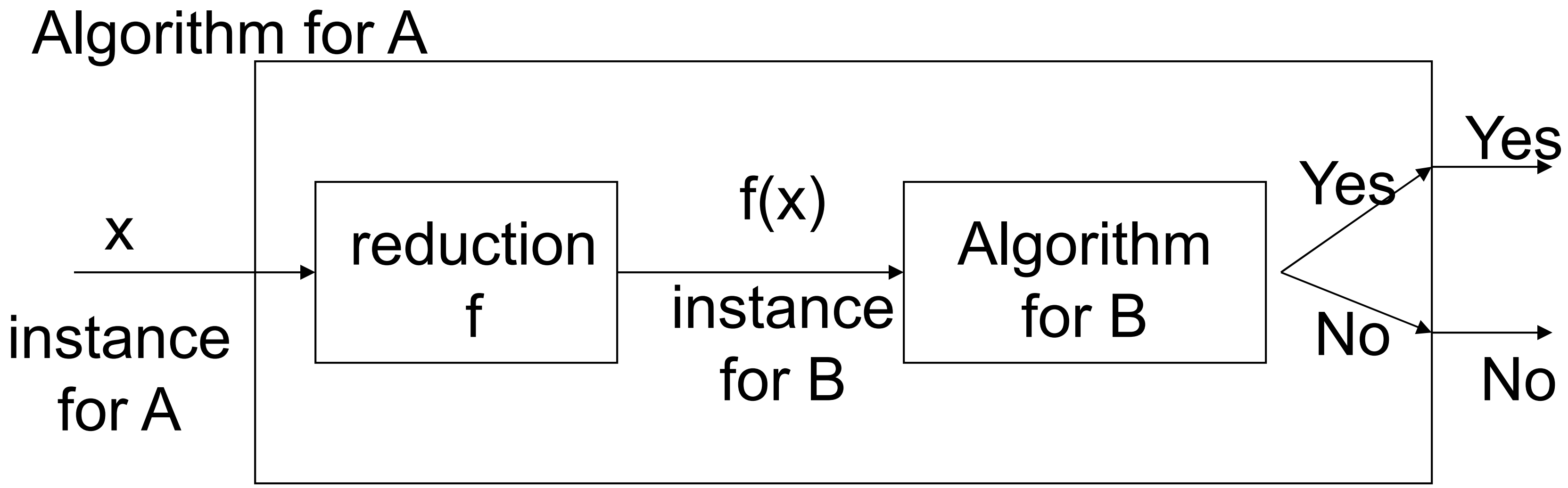
  s.t. $\forall x \in \{0,1\}^*$, $x \in L$ iff $f(x) \in L'$
- Polynomial time Reduction from decision problem A to B (notation $A \leq_P B$): reduction from L_A to L_B

  Polynomial time mapping f from instances of A to instances of B such that: Yes instances of A → Yes instances of B and No instances of A → No instances of B

  We can ignore invalid encodings (since they are efficiently recognizable and are irrelevant to the decision problem).



Reductions between Decision Problems



- If B$\in$P then also A$\in$P
- If A$\notin$P then also B $\notin$P
- If B$\in$NP then also A$\in$NP
- If A$\notin$NP then also B $\notin$ NP

Reductions compose: A $\leq_P$ B and B $\leq_P$ C $\Rightarrow$ A $\leq_P$ C

Complementation: A $\leq_P$ B $\Rightarrow$ $\bar{A} \leq_P \bar{B}$

NP-hardness and NP-completeness

- A language L (decision problem) is NP-hard if every language L' in NP reduces to it: $\forall L' \in NP$, $L' \leq_P L$
- A language L (decision problem) is NP-complete if
  1. L is in NP, and
  2. L is NP-hard

- coNP-hardness, coNP-completeness defined analogously

- If A is NP-hard and A $\leq_P$ B then B is also NP-hard

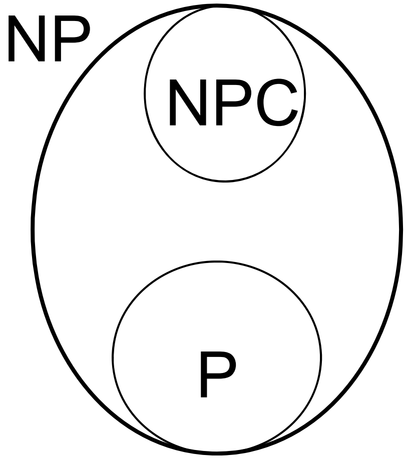
- If A is NP-hard then its complement is coNP-hard. Why?
- coNP-complete problems = complements of NP-complete problems. Why?

NP-complete problems and P vs NP

- If any NP-hard problem L is in P then P=NP

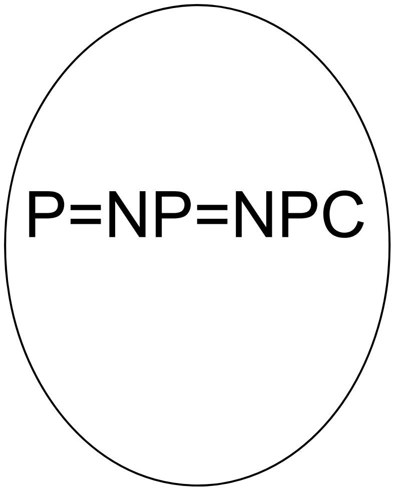
  Proof: For every $A \in NP$, $A \leq_P L$ and $L \in P$ imply $A \in P$

- Either all NP-complete problems are in P (and P=NP) or no NP-complete problem is in P (and $P \neq NP$)



Conjectured relation

or



A first NP-complete problem: Circuit Satisfiability

- Input: Boolean combinational circuit C (composed of AND,OR, NOT gates) with several inputs, one output
- Output: Yes, if there is an input assignment (assignment of 0, 1 to each input) for which C outputs 1, No otherwise

C = directed acyclic graph

indegree 0 nodes = inputs

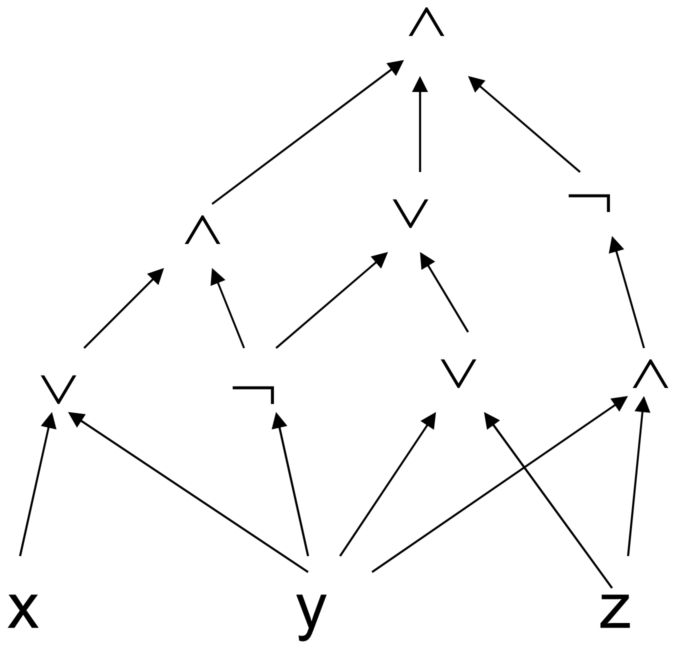
outdegree 0 node = output

internal nodes ↔ gates

NOT gate: 1 incoming edge – negates its input

AND, OR gates: 2 or more incoming edges – compute $\wedge$, $\vee$ of its inputs

Example

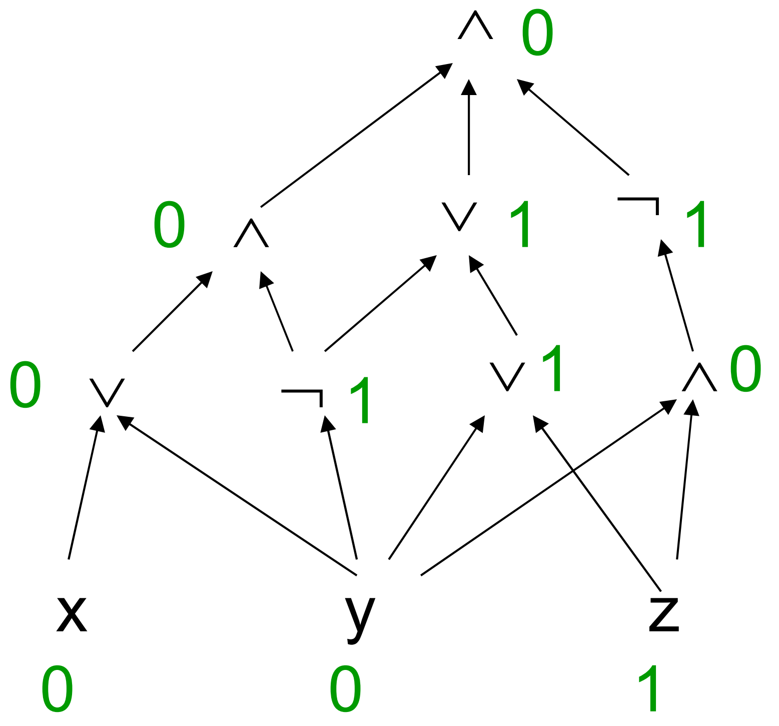


Output

Gates

Inputs

Example



Output

Gates

Inputs

C(001) = 0

C(101) = 1 ⇒ C is satisfiable

CIRCUIT-SAT is NP-complete

1. In NP: certificate = satisfying input assignment.

   Propagate values through C and verify output =1

2. NP-hard. To prove formally, must have a detailed formal model of computation and what is a polynomial algorithm.

   Usually formal model =some form of Turing machine or RAM (Random Access Machine). Different “reasonable” formal models can simulate each other with polynomial overhead, so classes P, NP invariant under which model used.

   Ingredients: program(= set of instructions), program counter, auxiliary state of machine, working space, input

   Configuration: global state of machine (incl. memory)

   Each step maps a configuration to a new configuration

   size of “relevant” space ≤ poly(# steps)

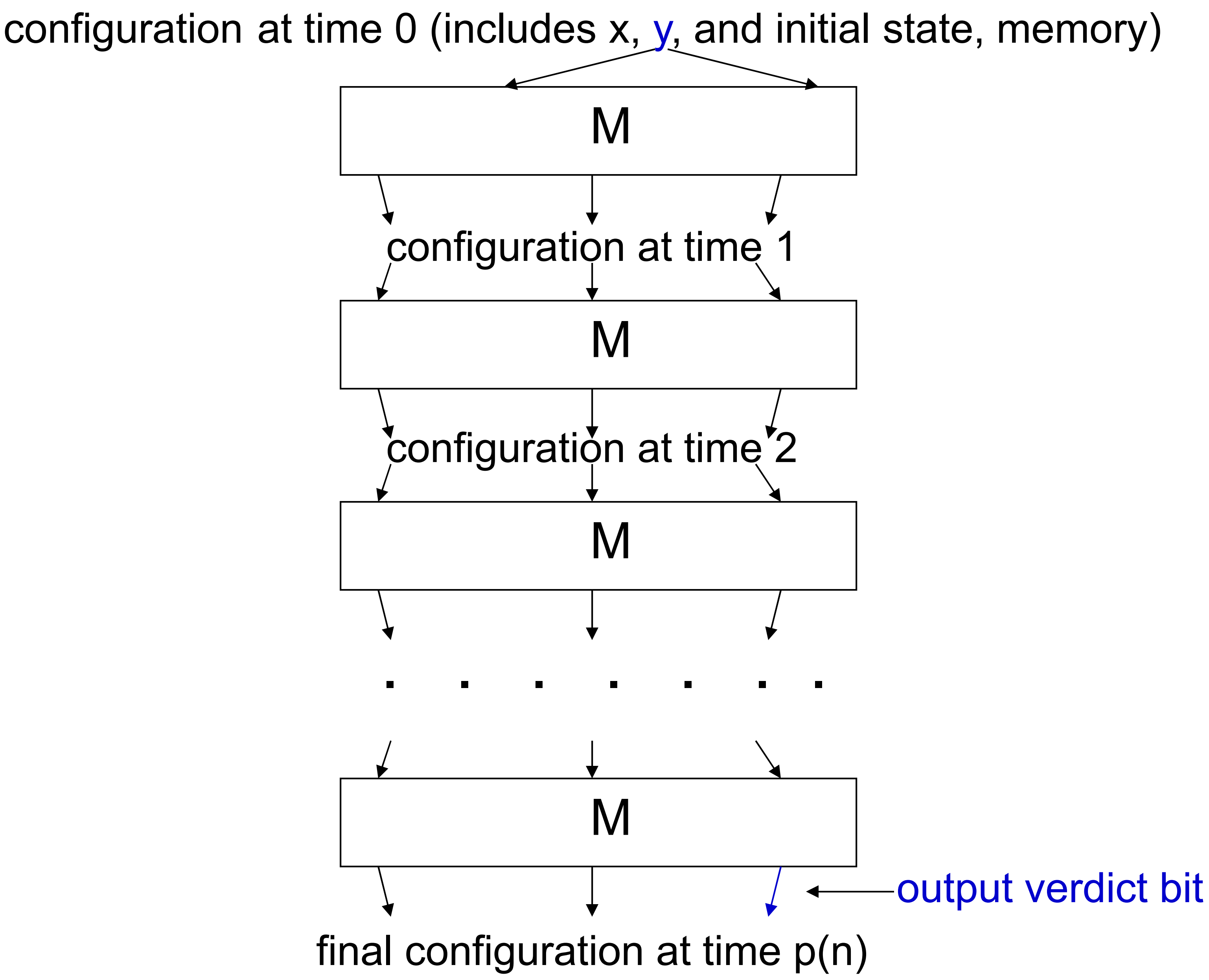
Reduction

- Take any language L in NP → polytime algorithm V(x,y) outputs Yes iff $x \in L$ and y is a certificate for it,

  p(|x|) bound on length of certificate and running time of V.
- Want poly-time algorithm which takes input string x and produces circuit C such that $x \in L$ iff C is satisfiable.

- Consider computation of V on input x, y (we know that $|y| \leq p(|x|)$ but not the actual bits of y); final verdict (1/0) written on special cell

- Configuration at time i is produced from configuration at time i-1 by a “small” (poly-size) combinational circuit M with inputs & outputs corresponding to bits of the configuration.



configuration at time 0 (includes x, y, and initial state, memory)

M

configuration at time 1

M

configuration at time 2

M

. . . .

M

output verdict bit

final configuration at time p(n)

Reduction ctd.

- Glue the copies of M (one for each time step) together by having the output gates of each copy feed the inputs to the next copy →
- one global combinational circuit for the whole computation

- Set the initial state, memory and the input bits corresponding to x to their known values and propagate the known values to simplify the circuit.
- In resulting circuit C, only free input bits are y
- output = gate that computes verdict bit in final configuration

$x \in L \Leftrightarrow \exists y.\ V(x,y)=1 \Leftrightarrow \exists y.\ C(y)=1$

NP-completeness Reductions

Proving a problem B is NP-complete

- Show the problem B is in NP
- Show B is NP-hard

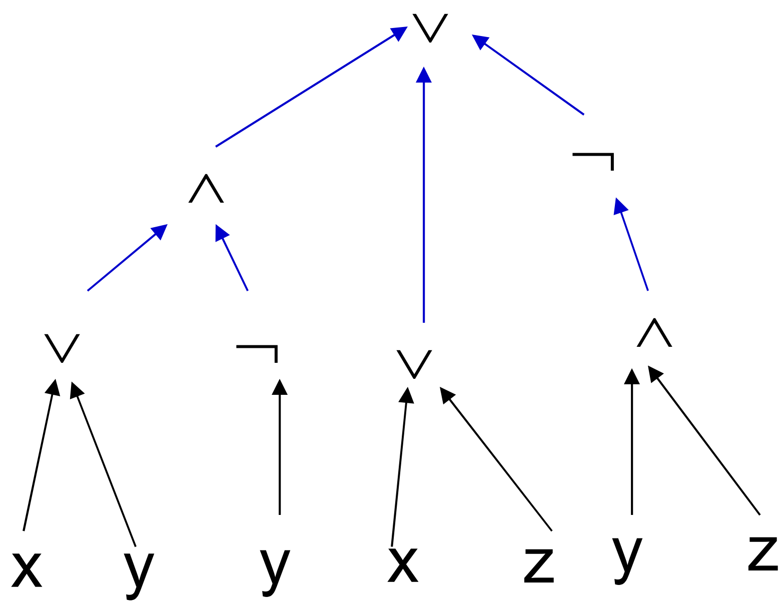
1. Pick a problem A that you know is NP-hard, preferably a problem close to B
2. Find an algorithm f that maps instances of A to instances of B
3. Prove that if x is a Yes instance of A then f(x) is a Yes instance of B
4. Prove that if x is a No instance of A then f(x) is a No instance of B; or equivalently show, if f(x) is a Yes instance of B then x is a Yes instance of A
5. Prove that f runs in polynomial time

* If B is not a decision problem, formulate the corresponding decision problem (NP is a class of decision problems)

Boolean Formula Satisfiability

- Special case of Circuit-SAT
- Boolean Formula with operators ¬, ∧, ∨ equivalent to circuit that is a tree, the parse tree of the formula, except for the inputs

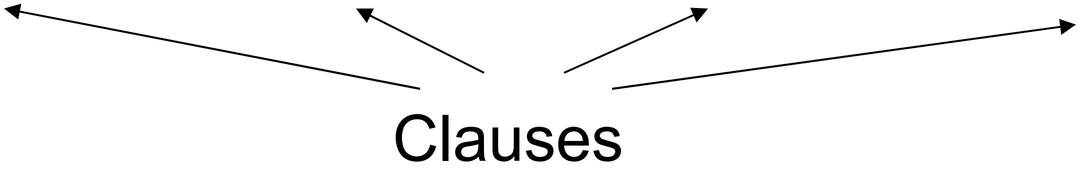
$((x \vee y) \wedge (\neg y)) \vee (x \vee z) \vee (\neg(y \wedge z))$



CNF Satisfiability

- Boolean formula is in Conjunctive Normal Form (CNF) if it is the AND of clauses where a clause is the OR of literals. A literal is a variable or its negation.

$$(x \vee y) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (y \vee z)$$



Clauses

CNF formula satisfiable $\Leftrightarrow \exists$ truth assignment to the variables that satisfies all the clauses

Disjunctive Normal Form (DNF): OR of ANDs of literals

Every Boolean function has an equivalent CNF (and DNF) formula, but the conversion from a circuit or formula to a CNF or DNF formula may incur an exponential blowup in size.

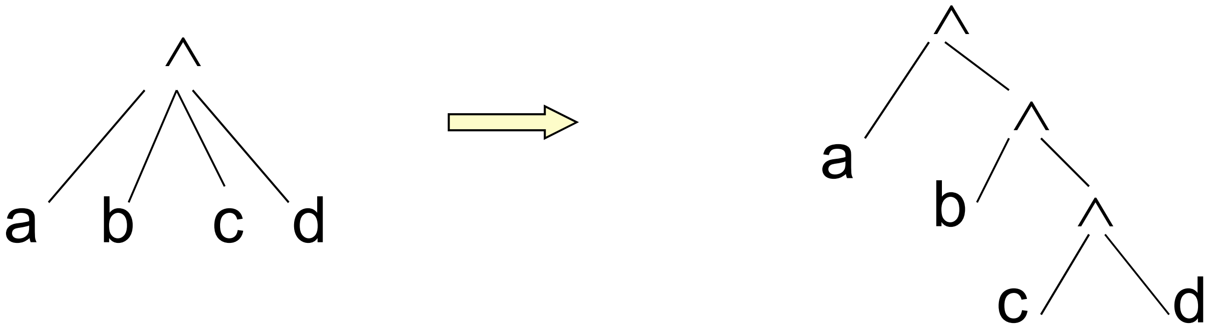
- Not a polynomial time reduction

3SAT is NP-complete

- 3SAT = Satisfiability of a CNF Boolean formula with 3 literals in each clause (a 3CNF formula).
- 3SAT in NP: “guess” a satisfying truth assignment (the certificate) and check that it satisfies all the clauses
- 3SAT is NP-hard: Reduction from CIRCUIT-SAT.

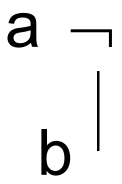
Given circuit C, will construct in polynomial time a 3CNF formula ϕ such that C is satisfiable iff ϕ is satisfiable.

1. Transform C to a circuit C’ where all the AND and OR gates have fan-in (in-degree) 2

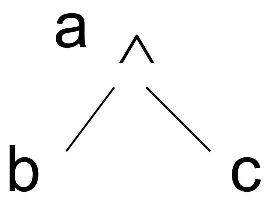


From CIRCUIT-SAT to 3SAT ctd.

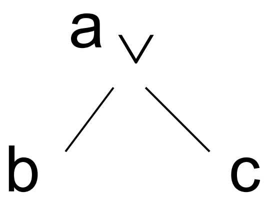
2. Introduce a variable for every input and gate of the circuit.
3. Include a set of clauses for each gate stating that the truth value of the corresponding gate variable is the NOT/AND/OR of the values of its input variables



$a= \neg b \Leftrightarrow$ conjunction of clauses $(a \vee b)$, $(\neg a \vee \neg b)$



$a=b \wedge c \Leftrightarrow$ clauses $(\neg a \vee b)$, $(\neg a \vee c)$, $(a \vee \neg b \vee \neg c)$



$a=b \vee c \Leftrightarrow$ clauses $(a \vee \neg b)$, $(a \vee \neg c)$, $(\neg a \vee b \vee c)$

From CIRCUIT-SAT to 3SAT ctd.

4. Let ϕ' = conjunction of all the gate clauses and an additional unit clause z, where z is the variable corresponding to the output gate.

ϕ' is a CNF formula with *at most* 3 literals per clause, and

ϕ' is satisfiable iff the circuit C' is satisfiable.

5. Transform ϕ' to a 3CNF formula ϕ with *exactly* 3 literals per clause:

- Replace each 2-literal clause $(r_1 \vee r_2)$ by two clauses $(r_1 \vee r_2 \vee p)$, $(r_1 \vee r_2 \vee \neg p)$, where p is a new variable (or any other variable of ϕ')
- Replace the 1-literal clause z by 4 clauses $(z \vee p \vee q)$, $(z \vee \neg p \vee q)$, $(z \vee p \vee \neg q)$, $(z \vee \neg p \vee \neg q)$

- The resulting 3CNF formula ϕ is satisfiable iff the circuit C is satisfiable.
- ϕ can be constructed from C in polynomial time

3SAT $\leq_P$ 0-1 Integer Linear Inequalities

- 3SAT formula ϕ with clauses $C_1,\dots,C_m$, variable $x_1,\dots,x_n$
- Set of linear inequalities with 0-1 variables $x_1,\dots,x_n$ and m constraints, one for each clause:

  For each clause $C_j = a \vee b \vee c$ where a,b,c are literals, we have a constraint $\alpha+\beta+\gamma \geq 1$ where if $a=x_i$ then $\alpha=x_i$ and if $a=\neg x_i$ then $\alpha=1-x_i$, and similarly with β, γ

  Example: $\phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4) \wedge (x_2 \vee x_3 \vee x_4)$

  $$x_1 + x_2 + (1 - x_3) \geq 1, \;\; (1 - x_1) + (1 - x_2) + (1 - x_4) \geq 1, \;\; x_2 + x_3 + x_4 \geq 1$$

- ϕ is satisfiable $\Leftrightarrow$ Constraints have a 0-1 solution

(Maximum) Independent Set Problem

- Independent set in a (undirected) graph G: subset I of nodes that are pairwise nonadjacent.

Example: edges = conflicts; Independent set = conflict-free

- (Maximum) Independent Set Problem

Input: Graph G

Output: An independent set of maximum cardinality

- Decision version: Input: graph G, number k
- Question: ∃ independent set of size ≥k?

- In NP: certificate= independent set I of size ≥k
- NP-hard: Reduction from 3SAT

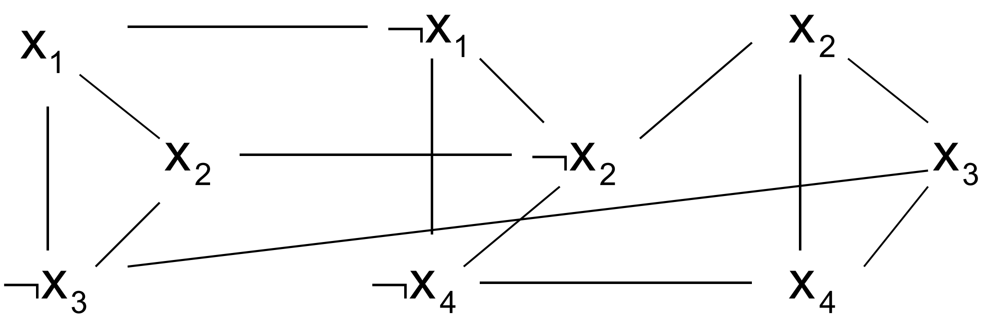
3SAT $\leq_P$ Max Independent Set

- 3SAT formula ϕ with clauses $C_1,\ldots,C_m$, variable $x_1,\ldots,x_n$
- Graph G, number k=m

  Nodes: One node for each literal of each clause

  Edges: Connect literals in same clause and complementary literals
- ϕ is satisfiable iff G has independent set with m nodes

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4) \wedge (x_2 \vee x_3 \vee x_4)$$



3SAT $\leq_P$ Max Independent Set Proof ctd.

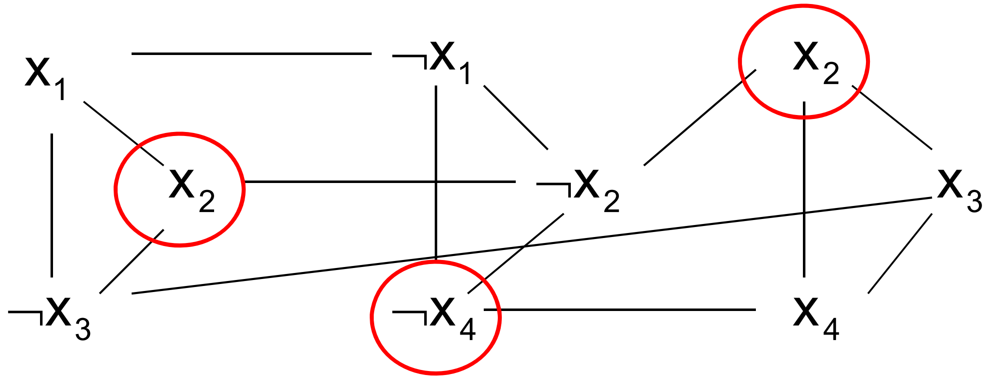
1. If ϕ is satisfiable then $\exists$ independent set with m nodes

Take a satisfying truth assignment, and let I include one true literal node from each clause

2. If $\exists$ independent set I with m nodes then ϕ is satisfiable

Set a variable x_i true (1) if a node for literal x_i is in I, false (0) if a node for literal $\neg x_i$ is in I, and arbitrarily otherwise

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4) \wedge (x_2 \vee x_3 \vee x_4)$$



Assignment: x_1 = 0/1, x_2 =1, x_3 = 0/1, x_4 = 0

(Maximum) Clique Problem

- Clique in a (undirected) graph G: subset C of nodes that are pairwise adjacent.
- (Maximum) Clique Problem

Input: Graph G

Output: A clique of maximum cardinality

- Decision version: Input: graph G, number k
- Question: ∃ clique of size ≥k?

- In NP: certificate= clique of size ≥k
- NP-hard: Reduction from Max Independent Set

Independent Set $\leq_P$ Clique

- Graph G, number k for Independent Set problem

→ Graph G' = complement of G, same number k for Clique

G' has same nodes as G, an edge is in G' iff it is not in G

- A set I of nodes is independent in G iff I is a clique in G'

- G has independent set of size k iff G' has clique of size k

Node (Vertex) Cover Problem

- Node Cover in a undirected graph G: subset C of nodes such that every edge has a node in C
- (Minimum) Node Cover Problem

Input: Graph G

Output: A node cover of minimum cardinality

- Decision version: Input: graph G, number k
- Question: ∃ node cover of size ≤k?

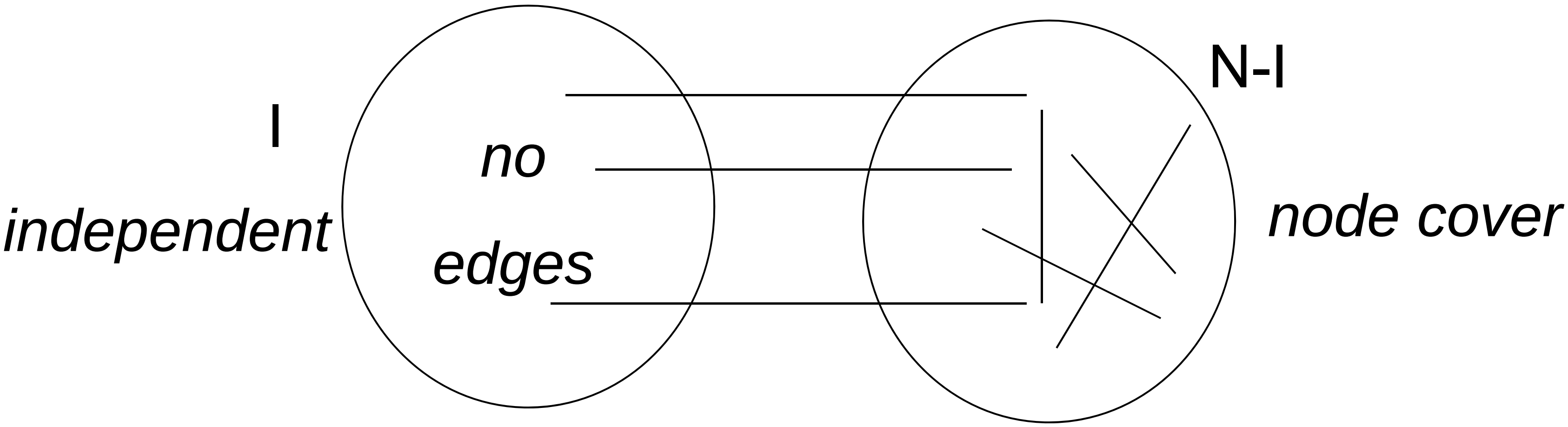
- In NP: certificate= node cover of size ≤ k
- NP-hard: Reduction from Max Independent Set

Independent Set $\leq_P$ Node Cover

- Graph G=(N,E), number k for Independent Set problem

→ Graph G' = same graph G, k' =n-k , where n =#nodes

- A set I of nodes is independent iff N-I is a node cover



•G has independent set of size ≥k iff G has node cover of size ≤n-k

Subset Sum

- Input: set S of (positive) integers, another integer t
- Question: ∃ subset T of S that sums to t?
- Note: numbers given in binary
- There is a pseudopolynomial algorithm (*HW*)

- In NP: certificate = subset T
- NP-hard: Reduction from Node Cover
- Given graph G=(N,E), bound k for Node Cover → instance of Subset Sum where S has one integer a_i for every node i of G, and one integer b_{ij} for every edge (i,j) of G.
- If G has e edges, then each integer has 2e+1 bits

$$\text{Target number } t = k \cdot 4^e + \sum_{i=0}^{e-1} 2 \cdot 4^i$$

Node Cover $\leq_P$ Subset Sum

2e+1 bits: leading bit + 2 bits per edge (edges in any order)

Leading bit 1 for node-numbers a_i, 0 for edge-numbers b_{ij}

edge bits:

a_i: 1 -- -- -- 00 01 -- -- 01 if i $\in$ edge, 00 otherwise

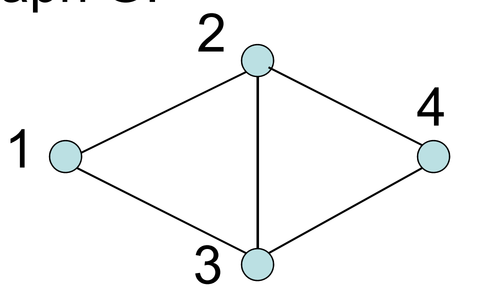
b_{ij}: 0 00 .. .. 00 01 …. All 0 except 01 at edge (i,j)

Bin(k) 10 10 10 10 10 Target t = k in binary followed by 10 for all edges

- For any subset T, when we add up the numbers in T, there is no carry from bits of one edge to the next or to the leading bit, because only three numbers have a 1 in the 2 bits for an edge

Example

Graph G:



k =3

edges

		(1,2)		(1,3)		(2,3)		(2,4)		(3,4)	
a_1:	1	0	1	0	1	0	0	0	0	0	0
a_2:	1	0	1	0	0	0	1	0	1	0	0
a_3:	1	0	0	0	1	0	1	0	0	0	1
a_4:	1	0	0	0	0	0	0	0	1	0	1
b_{12}:	0	0	1	0	0	0	0	0	0	0	0
b_{13}:	0	0	0	0	1	0	0	0	0	0	0
b_{23}:	0	0	0	0	0	0	1	0	0	0	0
b_{24}:	0	0	0	0	0	0	0	0	1	0	0
b_{34}:	0	0	0	0	0	0	0	0	0	0	1

Target number t: 1 1 1 0 1 0 1 0 1 0 1 0

Node Cover $\leq_P$ Subset Sum

- If $\exists$ node cover C with k nodes then $\exists$ subset T of S that sums to t

- T = $\{ a_i \mid i \in C \} \cup \{ b_{ij} \mid i \notin C \text{ or } j \notin C \}$ sums to t

- If $\exists$ subset T of S that sums to t then $\exists$ node cover C with k nodes

- Because there is no carry from bits of one edge to the next and t has 10 for all edges, T must contain for each edge (i,j) at least one of a_i, a_j

- Because of the k in the leading bits of t, T must contain exactly k numbers $a_i \Rightarrow C = \{ i \mid a_i \in T\}$ is a node cover of size k

0-1 Knapsack

Input: integers $v_1,\ldots,v_n$, $w_1,\ldots,w_n$ (values, weights of the items), W (knapsack capacity), bound b on total value

Question: $\exists$ subset $C \subseteq \{1,\ldots,n\}$ s.t. $w(C) \leq W$ and $v(C) \geq b$?

Reduction from Subset Sum

Instance $S=\{s_1,\ldots,s_n\}$, t of Subset Sum $\rightarrow$

instance of 0-1 Knapsack: n items, $v_i = w_i = s_i$,

knapsack capacity W=t, value bound b=t

- $\exists$ subset T of S that sums to t iff
  $\exists$ subset $C \subseteq \{1,\ldots,n\}$ s.t. $w(C) \leq W$ and $v(C) \geq b$

(Directed) Hamiltonicity

- Input: (Directed) graph H
- Question: ∃ Hamiltonian cycle (cycle that visits every vertex exactly once)?

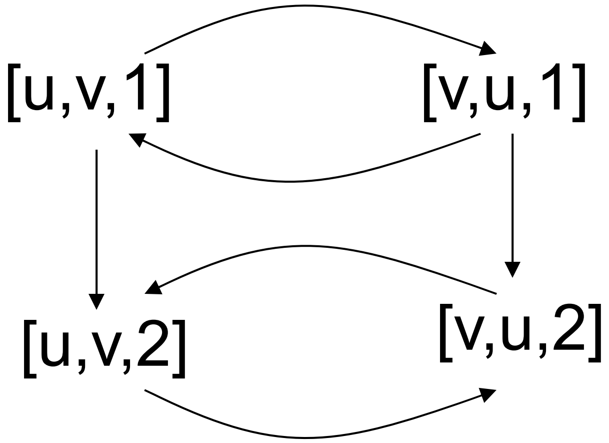
- In NP: certificate = Hamiltonian cycle
- NP-hard: Reduction from Node Cover

- Given (graph G, k)=instance of Node cover, construct another graph H such that G has a node cover of size k iff H has a Hamiltonian cycle

Node Cover $\leq_P$ Hamiltonicity

- Nodes of H: k “selector” node $s_1,\ldots s_k$ and

4 nodes for every edge (u,v) connected as in the following “gadget”:



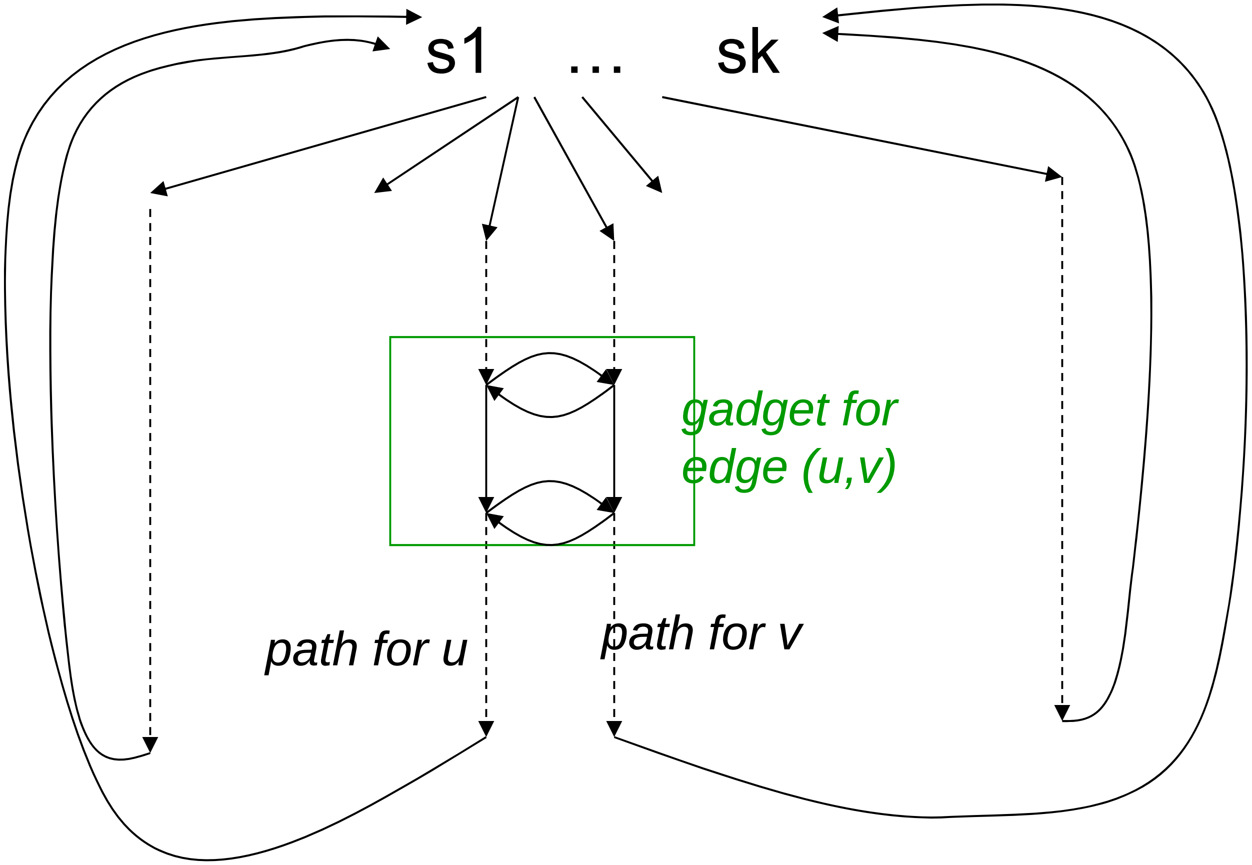
Outside edges only into [.,.,1], and out of [.,.,2]

Three ways to cover the nodes of the gadget:

1. start at [u,v,1], visit all nodes and leave at [u,v,2]
2. start at [v,u,1], visit all nodes and leave at [v,u,2]
3. [u,v,1] to [u,v,2] …. later on [v,u,1] to [v,u,2]

Node Cover $\leq_P$ Hamiltonicity ctd.

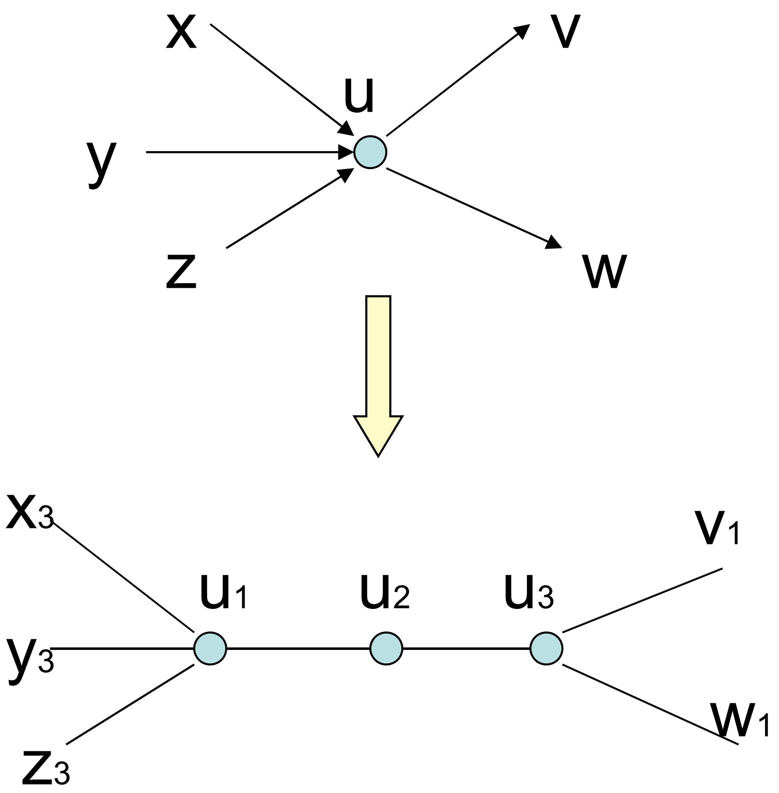
- Ignore isolated nodes of G (if it has any)
- For each node u of G, all [u,v,1],[u,v,2] nodes are connected in a path (in arbitrary order of u's edges), first node of path has edge from all selector nodes, last node has edge to all selector nodes



G has node cover with k nodes ⇔

H is Hamiltonian

Directed Hamiltonicity $\leq_P$ Undirected



Traveling Salesman Problem

- Input: Complete undirected graph G with weights d(i,j)>0 on its edges ("n cities and their pairwise distances")
- Output: Shortest (minimum total distance) "tour" (Hamilton cycle) that visits every city exactly once.
- Decision version: Given (G,d) and bound b, is there a "tour" with total distance ≤b ?

- In NP: certificate= tour
- NP-hard: Reduction from Undirected Hamiltonicity
- Given graph H=(N,E), let G have same nodes and define d(i,j)=1 if (i,j)∈E, else d(i,j)=2. Then H is Hamiltonian iff G has a tour with total distance ≤n