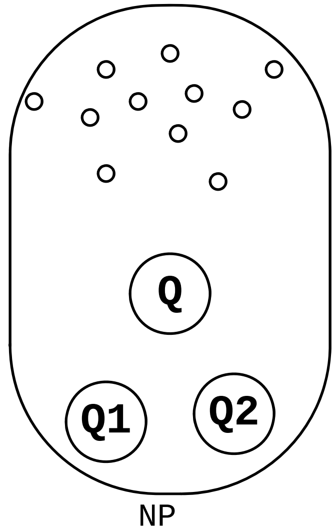
Exhibiting an NP-Complete Problem

Need to show one problem Q NP-complete via a direct proof.

That is, show how to reduce every other problem in NP to it (in)

Other problems can then be shown NP-hard by reducing Q to them



Initial Problem

There are different initial problems in the literature.

They are all based on encoding the computation of polynomial-time algorithms.

Rely on the "computational history" of a poly-time algorithm being polynomial in the size of the input.

Configurations of an Algorithm

Suppose A is a verification algorithm with time complexity bounded by $c \cdot n^k$

- On each step of the underlying machine model, it can write at most d bits
- So A uses at most $d \cdot c \cdot n^k$ bits of memory in any computation with an input of length
- If there are e bits of other state (registers, program counter), then one configuration of the machine needs

All Configurations

If we represent the configuration at each step in one computation, need at most $c \cdot n^k(e + d \cdot c \cdot n^k)$ =

There is a bit more. Have to have

- input x, |x| =
- certificate y, |y|
- instructions of A – size is

Next-Move Condition

Use some form of logic to describe the connection between one configuration and the next

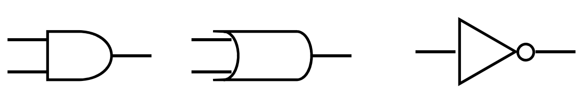
- Need one relating $config_i$ to $config_{i+1}$ for every i
- Things it must describe
  - Relationship of program-counter values
  - Changes in registers
  - Changes in memory
  - That everything else remains the same.

Circuit Satisifiability

Initial NP-complete problem in the book

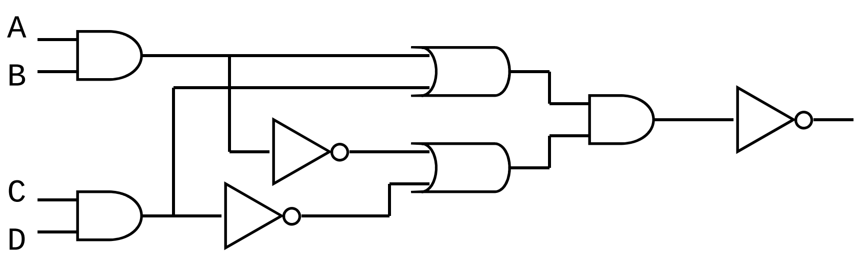
Does a Boolean circuit with one output (and no cycles) have an assignment to its inputs that makes its output true

Gates



Example

Is there a satisfying assignment for this circuit?



Circuit Satisfiability is in NP

Assume an instance is represented as a directed graph
- edges as wires
- nodes are labeled with the kind of gate
- number of inputs can grow linearly with representation: O(   ) to enumerate inputs

Problem is in NP

Certificate = assignment of 0 or 1

Need to check:
- Each gate behaves appropriately given its input and output
- output of whole circuit is

Circuit Sat is NP-Hard

Given any problem Q in NP

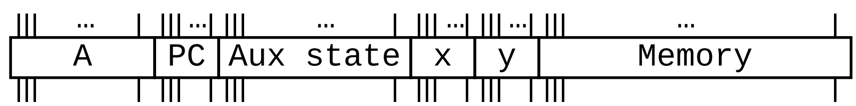
Need a polynomial-time function f that maps a string x to a circuit C, such that x is in Q if and only if

Must have a verification algorithm A(x, y) for Q with running time T(n) in

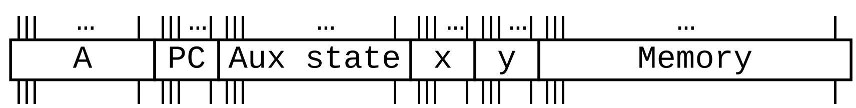
using certificates bounded by S(n) in

Configurations of Computation of A

A configuration is essentially wires



Initial configuration

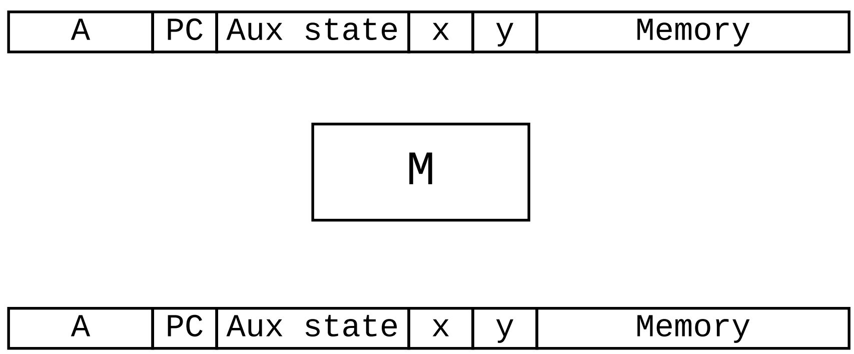


One Step of Computation of A

Function f builds T(n) such stages

One bit in final stage is designated as yes/no output

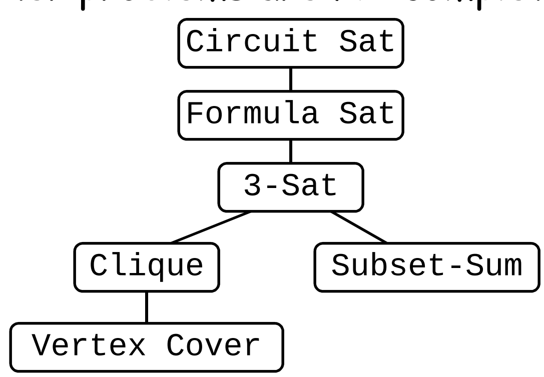
Config i



Config i+1

Reductions

Here is the sequence of problem reductions we will look at, to prove other problems are NP-complete.



Formula Satisfiability

Input: Boolean formula using:

$\wedge \quad \vee \quad \neg \quad \rightarrow \quad \leftrightarrow$

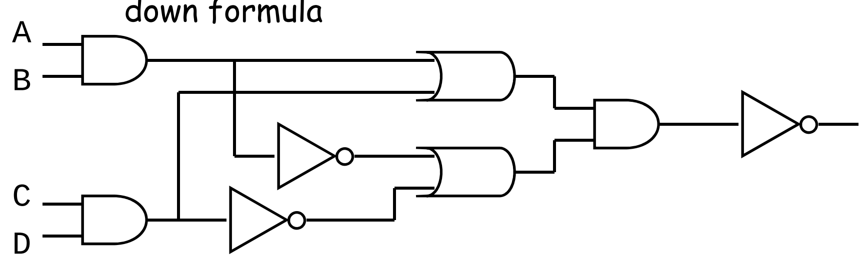
Plus parens and Boolean variables

Is there an assignment of true, false to the variables that makes the formulas true

$((x \wedge \neg y) \vee z) \leftrightarrow (w \rightarrow x)$

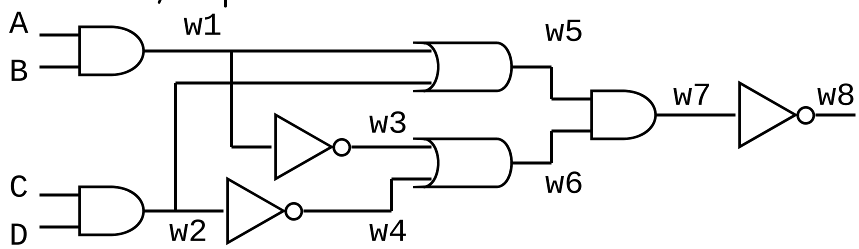
Reducing Circuit Sat to Formula Sat

Might seem like we can just work backwards from the output of the circuit, and write down formula



Instead, Variable per Wire, Sub-formula per Gate

Plus, output wire is true



$(w1 \leftrightarrow (A \wedge B))$	$(w2 \leftrightarrow (C \wedge D))$
$(w3 \leftrightarrow \neg w1)$	$(w4 \leftrightarrow \neg w2)$
$(w5 \leftrightarrow (w1 \vee w2))$	$(w6 \leftrightarrow (w3 \vee w4))$
$(w7 \leftrightarrow (w5 \wedge w6))$	$(w8 \leftrightarrow \neg w7)$

$w8$

Properties of Resulting Formula

1. Linear in the size of the circuit, hence
2. Can be constructed from the circuit in polynomial time
3. Is satisfiable if and only if the original

Polynomial-time reduction from circuit sat to formula sat

So formula sat is NP-Hard

Is formula sat in NP?

3-SAT

Conjunctive Normal Form (CNF):
AND of ORs of LITERALS
$(x \vee \neg y) \wedge (x \vee w \vee \neg z)$

3-CNF: Each clause has exactly 3 literals
$(x \vee \neg y \vee w) \wedge (x \vee w \vee \neg z)$

3-SAT Problem: Is a formula in 3-CNF satisfiable

Reduce Formula Sat to 3-SAT

Write formula as a tree, label each branch, proceed similarly to circuits

$((x \wedge \neg y) \vee z) \leftrightarrow (w \rightarrow x)$

$(b1 \leftrightarrow \neg y) \wedge$

$(b2 \leftrightarrow (x \wedge b1)) \wedge$

$(b3 \leftrightarrow (b2 \vee z)) \wedge$

Each Component Formula Has 3 Literals

Can convert each to 3-CNF

How? List cases that are false

$(b2 \leftrightarrow (x \wedge b1))$

b2	x	b1	
F	T	T	$(\neg b2 \wedge x \wedge b1) \vee$
T	F	T	$(b2 \wedge \neg x \wedge b1) \vee$
T	T	F	$(b2 \wedge x \wedge \neg b1) \vee$
T	F	F	$(b2 \wedge \neg x \wedge \neg b1)$

Then use DeMorgan's Laws

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

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

Result is in 3-CNF

$(b2 \vee \neg x \vee \neg b1) \wedge$
$(\neg b2 \vee x \vee \neg b1) \wedge$
$(\neg b2 \vee \neg x \vee b1) \wedge$
$(\neg b2 \vee x \vee b1)$

What about 2-literal clauses?

$(x \vee \neg z)$

Add a new variable p

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

What about 1-literal clauses?

Things to Check

- Translation can be done in polynomial time
- 3-SAT is in NP

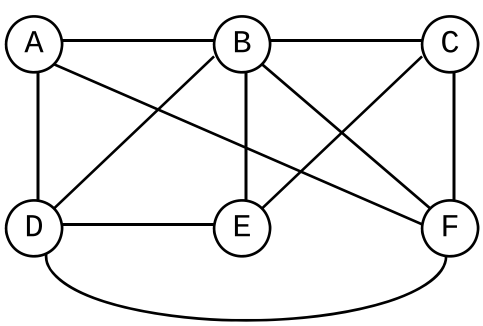
The 3-SAT problem is NP-complete

Branching Out from Logic

Graph problems

<u>Clique</u> of an undirected graph G = (N, E)

Set of nodes M such that (m1, m2) in E for every



Does this graph have a clique of 4 nodes?

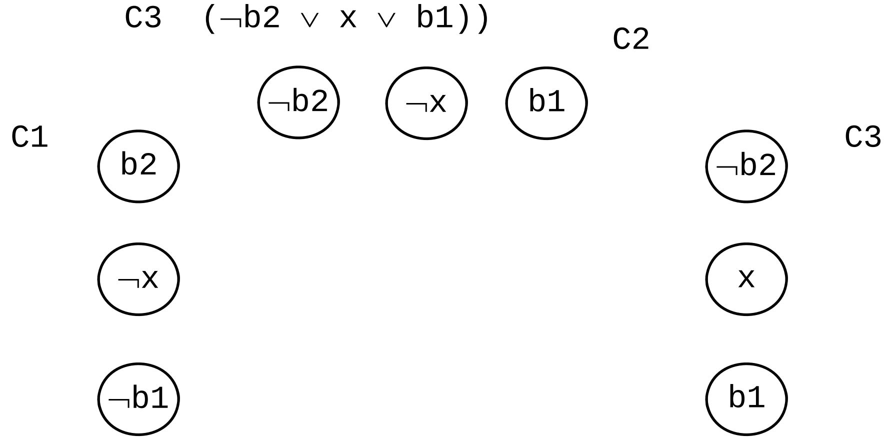
Does it have a clique of 5 nodes?

Solving 3-SAT with CLIQUE

- Each clause becomes 3 nodes, one for each literal
- For every pair of clauses, connect each pair of nodes that is compatible
  - Not compatible:
- If there are k clauses, want to know if there is a clique of size
  - Will be one exactly when all the clauses are satisfiable

Example

C1 $(b2 \vee \neg x \vee \neg b1)) \wedge$
C2 $(\neg b2 \vee \neg x \vee b1)) \wedge$
C3 $(\neg b2 \vee x \vee b1))$



Equivalence

Suppose formula is satisfiable
- Must make at least one literal true in each clause
- These literals must be compatible, so there are
- So there must be a

Suppose there is a clique of size k
- Must have exactly one node corresponding to each clause (why?)
- Tells us which literals to make true, hence which variables to make true or false
- Must be compatible literals

How Big is This Graph?

Number of nodes =

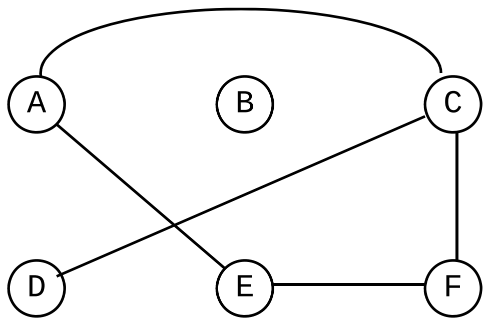
Number of edges

So we can reduce 3-SAT to CLIQUE in polynomial time

Is CLIQUE in NP?

Vertex Cover

Given an undirected graph G = (V, E) and integer j, is there a set of j nodes P such that for each edge {v, w} in E,



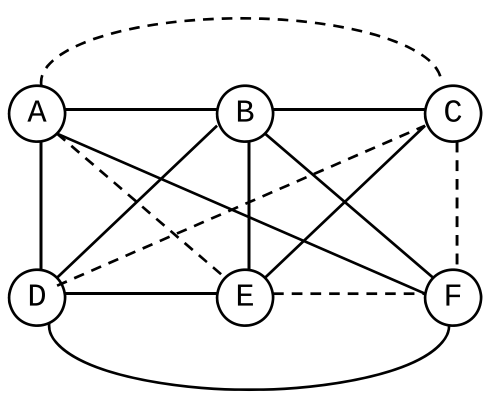
Does this graph have a vertex cover of 2 nodes?

Does it have a vertex cover of 1 node?

Complement Graph

This last graph is the complement of the example for clique

Has exactly the missing edges



Connection of Vertex Cover & Clique

Complement of a graph has a node cover of size j if and only if the original graph has a clique of size

If P is a vertex cover, then N – P is

Complement has no edge with both ends in here, hence original must have

Also, complement of a clique in the original graph is a vertex cover in

The Subset-Sum Problem

An NP-complete problem
- Won't cover the proof here
- You should know that it is NP-complete

Subset-Sum: Given
- Set S of integers
- A target value t

Is there a subset S' of S whose elements sum to t?

Examples

S = {3104, 11015, 22010, 117, 8506, 7011, 19103, 1010, 3912}

target t = 22324

Second example: same S

target t = 2324

Reduction from 3-SAT

Use decimal numbers, divide into positions

- Some for variables: Make sure
- Some for clauses: Make sure each has

Approximation Algorithms for NP-Complete Problems

If Q is NP-complete, then what can we do?

- Limit ourselves to small inputs
- Find special cases that can be solved in polynomial time
- Find poly-time algorithms that give good approximations

For example, for 3-SAT, is there a poly-time algorithm that is guaranteed to find an assignment (if it exists) that

Approximating Vertex Cover

If we could find the minimum size vertex cover, M, for a graph G, we could answer the question if G has a vertex cover of size k.

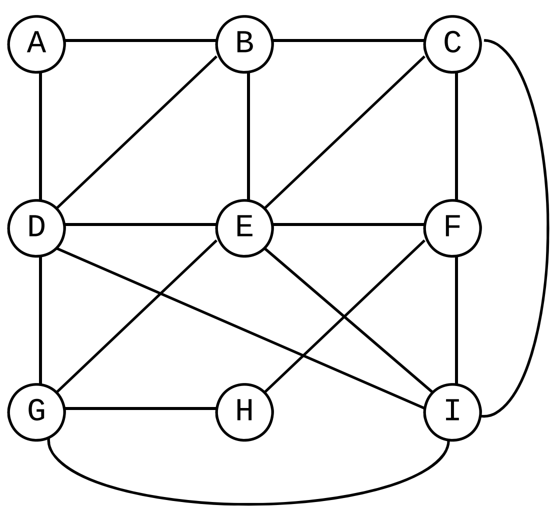
Will show a poly-time algorithm that finds a cover C of G such that |C|

Algorithm is Simple

Given G = (V,E)

1. Pick an edge {u, v} in E
2. Add u & v to
3. Remove all edges covered by C
4. Repeat until all edges covered

Example



Approximation Bound

Why is C within a factor of two of minimum M?

Let F be chosen edges. So C =

1. No two edges in F have a node in common. Suppose it has {u, v} and {u, w}. Assume {u, v} chosen first. Then
2. So, no cover can have a node x that covers
3. M must have at least one node for each edge in F, so $|M| \geq |F|$, $|C|$ =