Hello Class

It’s me again!

Today

Example of an NP-Complete problem: Circuit-satisfiability

Proof that it is in NP (verifiable in polynomial time)

An somewhat informal appeal that it is also NP-Hard

Also today: Pictures! (Figures) So please speak up if you can’t see.

Brief Review

The class of NP, NP-Complete, NP-Hard, etc. are of no practical valuable if they are all empty.

To show they are not empty requires a single example.

NP-classification is important for a variety of reasons, the most obvious related to cyber-security.

Circuit Satisfiability: Circuit Design: Principles

A circuit is a **D**irect **A**cyclic **G**raph (DAG). In examples, direction is left to right.

It forms a function from $\{0,1\}^n \Rightarrow \{0,1\}^m$

Input edges to the graph take a single boolean input from the set of inputs

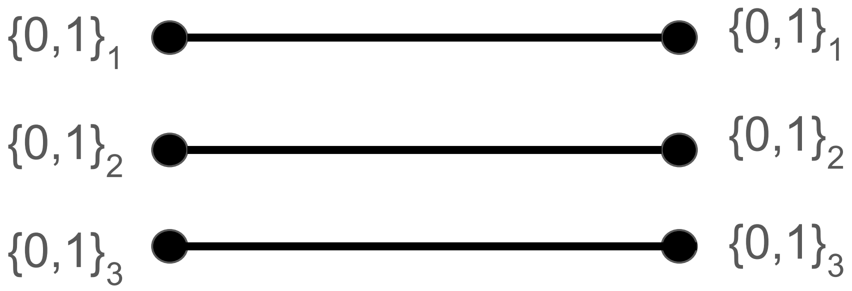
Each boolean in the input set corresponds to a single input edge

Output edges to the graph emit a single boolean output to the set of outputs

Each boolean in the output set corresponds to a single output edge

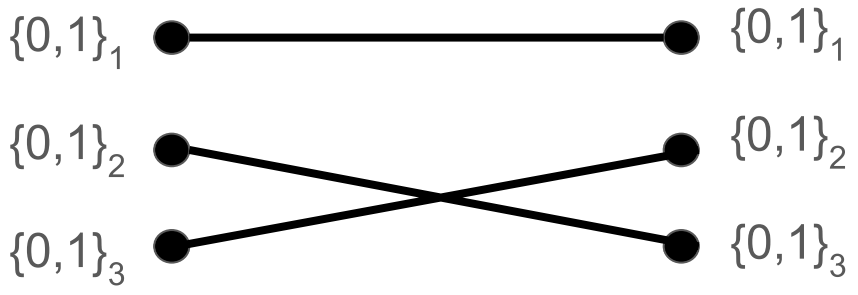
Circuit Satisfiability: Circuit Design: Examples

Take n = 3, m = 3.



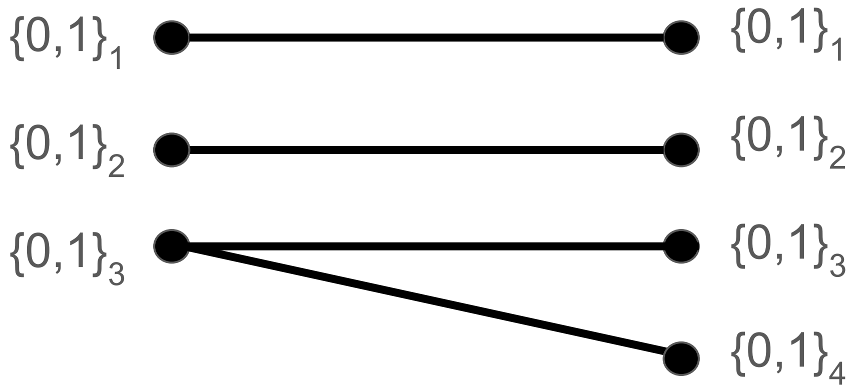
Circuit Satisfiability: Circuit Design: Examples

Take n = 3, m = 3.



Circuit Satisfiability: Circuit Design: Examples

Take n = 3, m = 4.



Circuit Satisfiability: Circuit Design: Logic Gates

To make things more exciting, we add the concept of logic gates.

Graph nodes can be logic gates, which are functions.

The NOT gate: $\{0,1\} \Rightarrow \{0,1\}$ *inverts* input: NOT(1) = 0, NOT(0) = 1

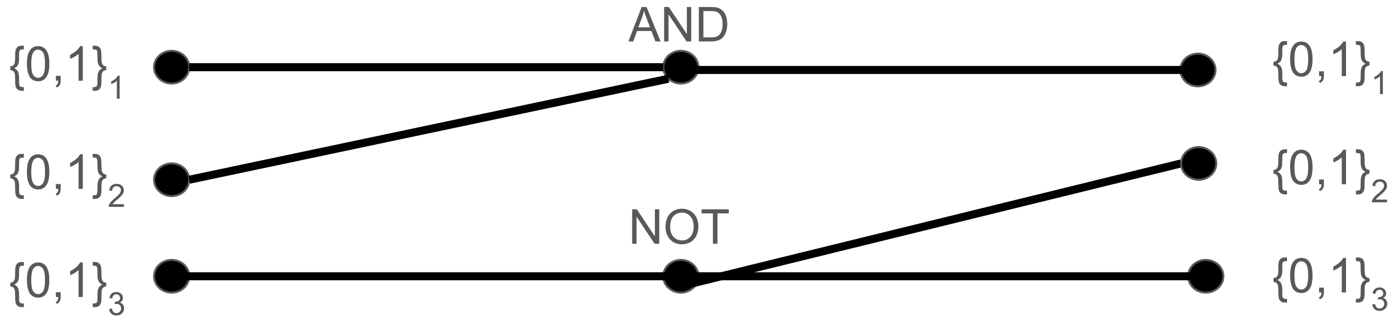
The AND gate: $\{0,1\}^n \Rightarrow \{0,1\}$ emits 0 unless all inputs are 1, then 1

The OR gate: $\{0,1\}^n \Rightarrow \{0,1\}$ emits 1 unless all inputs are 0, then 0

Nodes may always generate multiple identical outputs, ie “multiple wires”

Circuit Satisfiability: Circuit Design: Examples

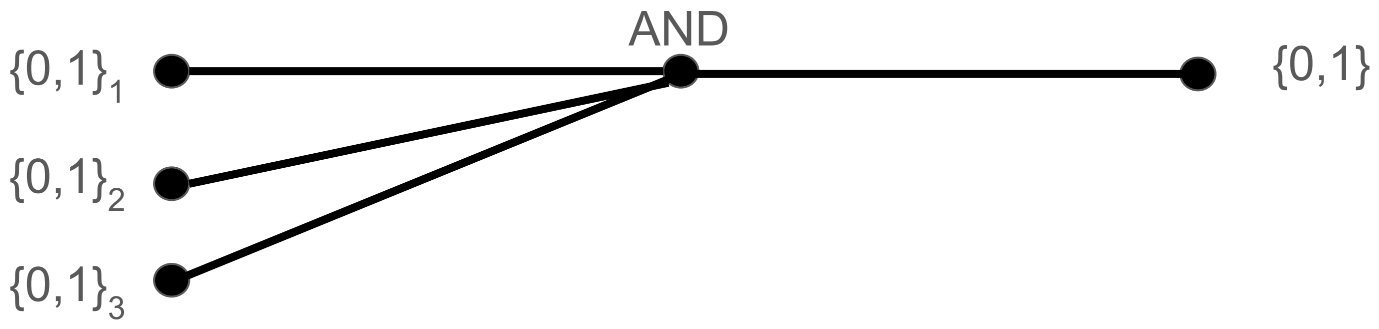
Take n = 3, m = 3.



Here, for example, c(1,1,0) = <1,1,1> or c(1,0,1) = <0,0,0>

Circuit Satisfiability: Circuit Design: Examples

Take n = 3, m = 1.



Here, for example, c(1,1,1) = 1 while another other inputs give c = 0.

Circuit Satisfiability: Satisfiability

Satisfiable means that something might be “true”

For a circuit to be satisfiable, we consider circuit with a single output.

If there is a set of inputs for which the output is 1, then it is satisfiable.

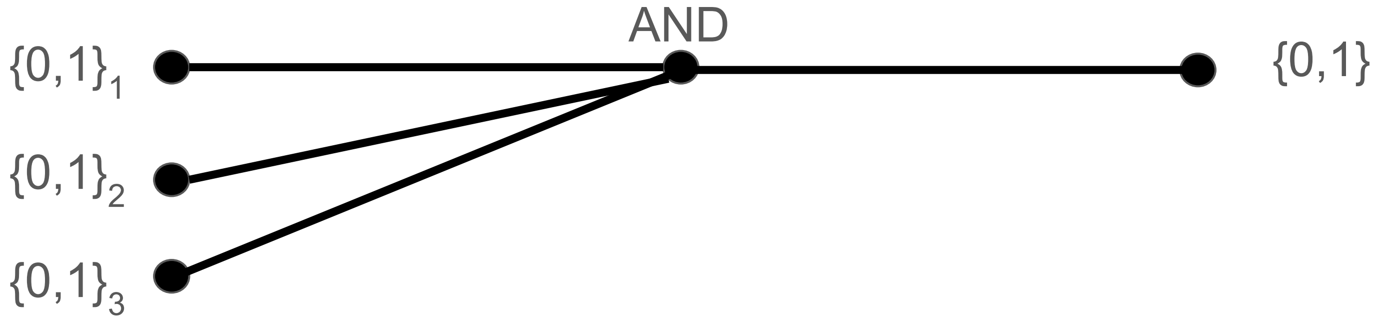
Formally: $\exists e \in \{0,1\}^n : c(e) = 1$

Otherwise it is *unsatisfiable*.

As an aside, if all inputs evaluate to true, then a circuit is *valid*.

Circuit Satisfiability: Satisfiable Example

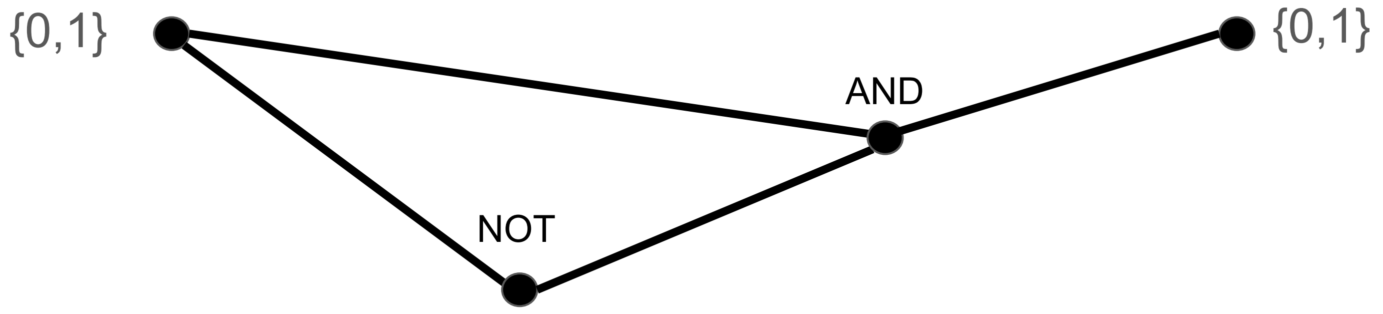
Take n = 3, m = 1.



This is satisfiable because c(1,1,1) = 1

Circuit Satisfiability: Unsatisfiable Example

Take n = 1, m = 1.



See that c(0) = 0 and c(1) = 0 as well. (Change AND to OR for a valid circuit)

Circuit Satisfiability: Closing Remarks

The *size* of a circuit is the number of gates.

The *size* of the input is the number of booleans.

We take as trivial the encoding of the circuit as a graph in memory.

Consider this as an exercise, taking care to preserve polynomial constraints

The text would like me to note that this problem is important because people get jobs making circuits and this problem has implications there. C.R.E.A.M.

CIRCUIT-SAT is in NP (Lemma 34.5)

Consider algorithm *A*. It’s aim is to verify a solution in polynomial time.

A takes as input a circuit *C* and a *certificate* and gives boolean output.

The certificate encodes outputs of a graph edges in the circuit. Crucially, its size is bounded by a polynomial of the the size of the circuit.

A considers each logic gate in *C* and, if all operations are correct the *C* output 1, then it returns true.   Otherwise, it returns false.

With a satisfying certificate, *A* completes in polynomial time and returns true.

CIRCUIT-SAT is NP-Complete Sketch

And I quote:

“The actual proof of this fact is full of technical intricacies, and so we shall settle for a sketch of the proof based on some understanding of the workings of computer hardware.”

CIRCUIT-SAT is NP-Hard Sketch: Computers

Perhaps unsurprisingly, computers can be made out of circuits. Moreover, circuits can compute algorithms on problems in NP.

How?

A *program* is stored in memory as a sequence of instructions. These instructions are combinations of logic gates. (Otherwise may be memory references).

Now, if we are ahead of schedule, we will build an addition circuit from logic.

Otherwise, we will go straight to the Lemma.

An Aside: Circuit Addition: Problem Statement

Consider addition between two-digit binary numbers. It can be encoded as:

$\{0,1\}^4 \Rightarrow \{0,1\}^3$ where the interpretation of input is two adjacent two-digit numbers

Output is the 3 digit sum.

Binary encoding is standard:

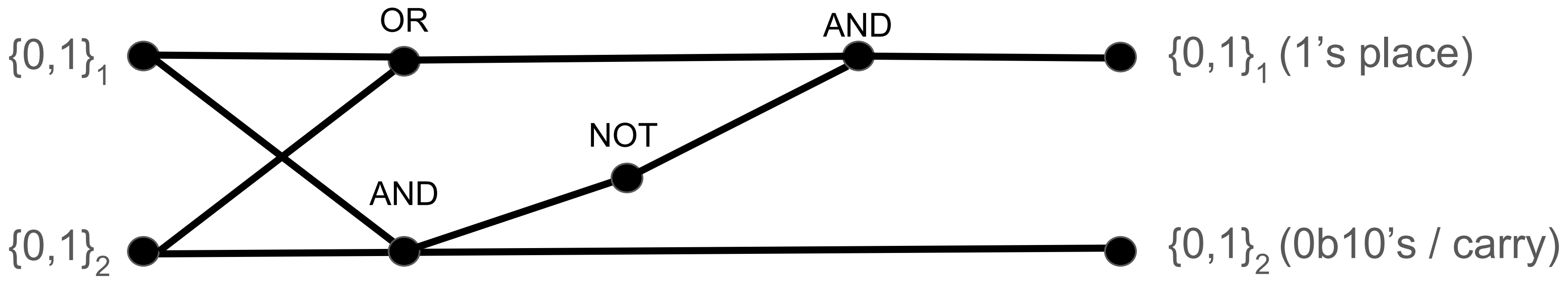
0b00 = 0, 0b01 = 1, 0b10 = 2, etc.

An Aside: Circuit Addition: Circuit Part 1

First we consider addition between single digit binary numbers with carrying.

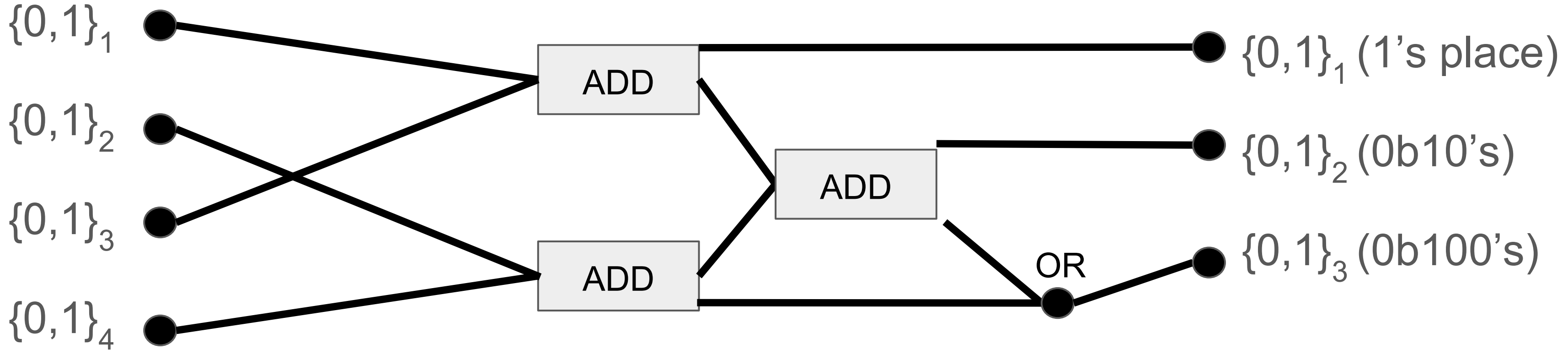
That is, $\{0,1\}^2 \Rightarrow \{0,1\}^2$ where input is two adjacent single-digit numbers

Output is the sum, with the second value holding the 0b10's place to carry



An Aside: Circuit Addition: Circuit Part *n*

From there, we can simply rearrange inputs to group 1's and 10's places and add



Fairly straightforward to extend to general addition and, frankly, everything else.

CIRCUIT-SAT is NP-Hard Sketch

Imagine encoding everything in a computer's memory that a program is allowed to use, everything in registers.

This is the computer's *state* which the text calls a *configuration*.

Imagine executing instructions as transitions between states.

This is done using boolean circuits!

GASP!

CIRCUIT-SAT is NP-Hard (Lemma 34.6)

Definitions:

L is a language in NP.

F is the polynomial time algorithm used for reduction

It maps strings *x* in *L* to circuit *C* = *f* (*x*)

It preserves the feature that *x* is in *L* iff *C* is satisfiable

M is a mapping circuit from one configuration to another

Also recall previously defined algorithm *A* which is polynomially complex

CIRCUIT-SAT is NP-Hard (Lemma 34.6)

Algorithm *F*:

For input *x*, create a circuit *C*. It will contain polynomially many instances of *M*.

If *x* is in *L* there will be a satisfying input to *C*. Call this *y*.

C takes an input the algorithm *A*, the input *x* and the input *y* and no other input.

The result? The computer computes *C*(*y*) = *A*(*x*,*y*) in polynomially many steps.

Why? *A* runs in polynomially many *M*'s which are, themselves, polynomial.

CIRCUIT-SAT is NP-Hard (Lemma 34.6)

Note then that if *C* is not satisfiable, then the algorithm does not return true because of the correctness of *A*.

Likewise, if the algorithm returns true, then *C* is necessarily satisfiable and so to then is *x*.

The algorithm fits in polynomial space because it runs in polynomial time and updates at most polynomial memory locations per unit time.

CIRCUIT-SAT is NP-Complete (Lemma 34.6)

This is, by definition, an immediate consequence of membership in NP and NP-Hard.

Any questions?