15-251 Spring 2015

Lecture 19 - NP and NP-completeness II

Last Time:

- How do you identify intractable problems? e.g. SAT, TSP, ...
- Can't prove they are intractable. Can we gather some sort of evidence?
- Poly-time reductions. $A \leq_T^P B$
- If we can show $L \leq_T^P A$ for <u>many</u> L, that can be good evidence that $A \notin P$.
- Definitions of $\mathcal{C}$-hard, $\mathcal{C}$-complete.
- What is a good choice for $\mathcal{C}$, if we want to show SAT is $\mathcal{C}$-hard?
- The complexity class NP. NP-hardness, NP-completeness.
- Cook-Levin Theorem: SAT is NP-complete.
- <u>Many</u> natural problems are NP-complete
- Is this good evidence that a problem is intractable?
- P vs NP question.

Today:

- Proof sketch of Cook-Levin Theorem
- Showing other natural problems are NP-complete.

  Circuit-SAT, 3SAT, CLIQUE, VERTEX-COVER

We'll actually first show that Circuit-SAT is NP-complete.
Then we'll show Circuit-SAT $\leq_T^P$ SAT and conclude SAT is NP-complete.

$\boxed{\text{Circuit-SAT}}$

<u>Input</u>: A Boolean circuit with AND, OR, and NOT gates.

<u>Output</u>: Yes, if there is an assignment to the input variables that makes the circuit output 1.
No, otherwise.

From our discussion about circuit complexity, recall the following two theorems.

<u>Thm 1</u>: Every function $f: \{0,1\}^n \to \{0,1\}$ can be computed by a Boolean circuit of size $O(2^n)$.

<u>Remark</u>: ① If we wanted to compute a function $f: \{0,1\}^k \to \{0,1\}^r$, we can do it with r circuits – one circuit for each output bit.

② If k and r are constants, then the size of the circuit is constant
So transforming constant length information into another constant length information can be done using constant size circuits.

<u>Thm 2</u>: If a language L can be decided in $O(T(n))$ time, then it can be computed by a circuit-family of size $O(T(n)^2)$.

We didn't prove this before. Let's sketch the proof now.

Proof Sketch:

Let L be decided by TM M (we'll assume M's tape is infinite in one direction – to the right)

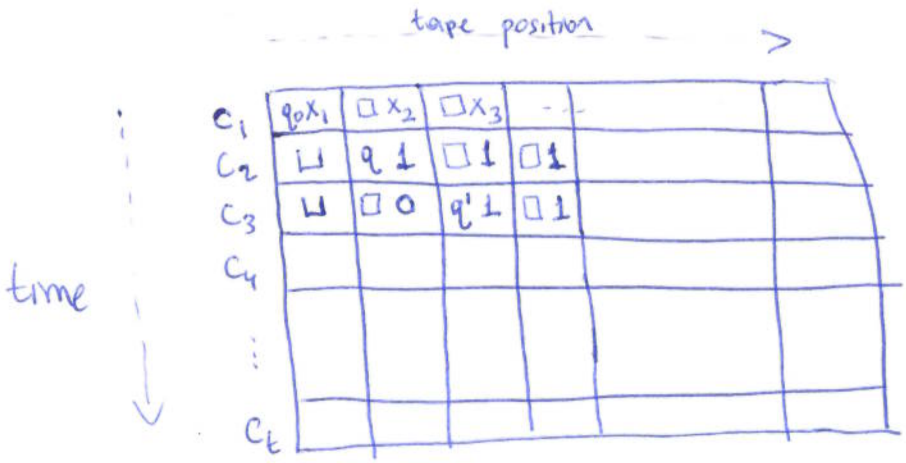
Fix n and input x with $|x| = n$.

We know M goes through configurations

$$c_1, c_2, c_3, \ldots, c_t \qquad t = O(T(n))$$

Recall each configuration is of the form $c_i = u q v \quad u, v \in \Gamma^*, q \in Q$

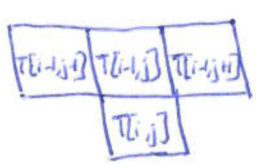
Consider a t by t table where ~~each~~ row i corresponds to c_i



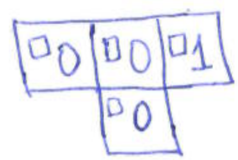
Each cell contains: (a state name or NONE [if no state is given]) and (a symbol from Γ)

Let's call this table T (not to be confused with the running time)

Observation: The contents of a cell $T[i,j]$ is determined by the contents of $T[i-1, j-1]$, $T[i-1, j]$, $T[i-1, j+1]$



e.g.



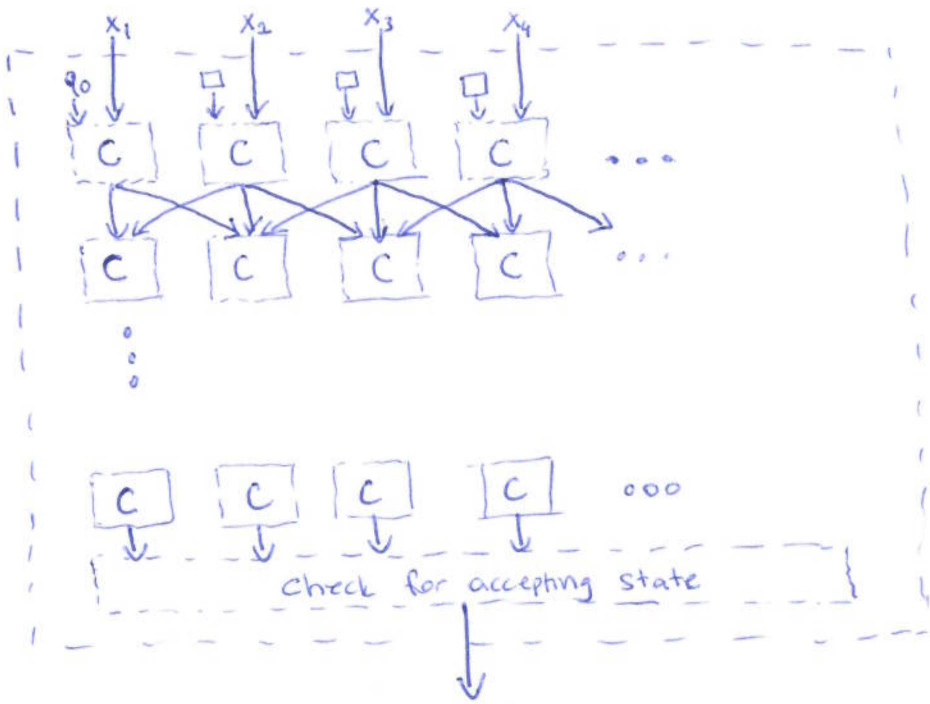
The transition function of M governs this transformation

Let's say each cell encodes k bits of information.

k is a constant because $|Q|$ and $|\Gamma|$ are constant.

So the transition function $\{0,1\}^{3k} \to \{0,1\}^k$ can be implemented by a circuit of constant size. Let's call this circuit C.

Now we can build a circuit that computes the answer given by M:



This circuit has size $\leq c \cdot t^2$.

$\square$

Using this result, we can now show Circuit-SAT is NP-complete.

Thm: Circuit-SAT is NP-complete.

Proof:

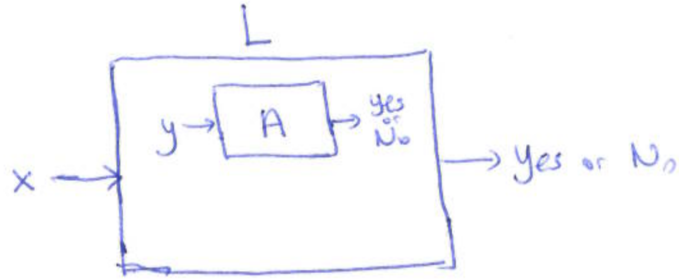
We have to show Circuit-SAT $\in$ NP and Circuit-SAT is NP-hard.

Circuit-SAT $\in$ NP because we can take a satisfying assignment to the variables as "proof". We can check in poly-time that this proof is correct, i.e., it indeed satisfies the circuit.

To show Circuit-SAT is NP-hard, we need to show that for every $L \in NP$, $L \leq_T^P$ Circuit-SAT.

i.e., if we can solve Circuit-SAT efficiently, then we can solve L efficiently.

Let A be an algorithm that solves Circuit-SAT efficiently.



Since $L \in NP$, we know there is a poly-time verifier TM V s.t.

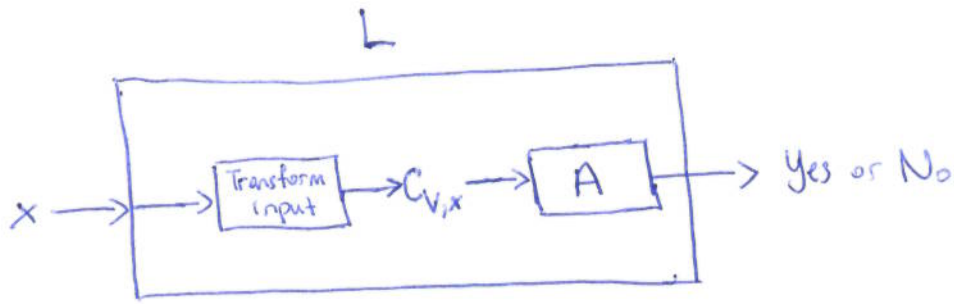
$x \in L$ iff $\exists u, |u| = |x|^k$ s.t. $V(x,u) = 1$

We know V has a corresponding circuit C_V of poly-size.

Then

$x \in L$ iff $C_{V,x}$ is a Yes instance of Circuit-SAT

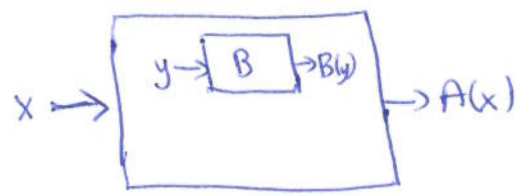
C_V with x-variables fixed to x.



This reduction clearly works correctly.

It is polynomial time because $\to \boxed{\text{Transform input}} \to$ is polynomial time:
If V runs in time n^k, we can build $C_{V,x}$ in time $O(n^{2k})$. $\square$

<u>Note 1</u>: When you do a reduction $A \le_T^P B$,



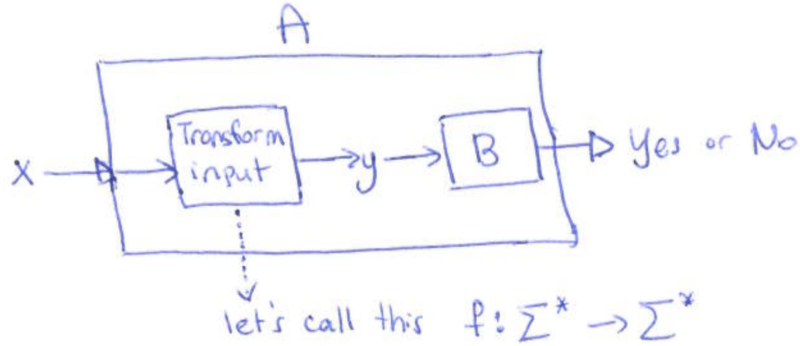
(A has a poly-time algorithm provided B has a poly-time alg.)

You are in good shape as long as the computation done outside of B is polynomial time.

Why? Suppose B runs in time $|y|^k$, for some constant k.
Let $n = |x|$. The length of y can be at most n^r, for some constant r.
Then B will run for $(n^r)^k = n^{rk}$ time, still polynomial.

Note 2:

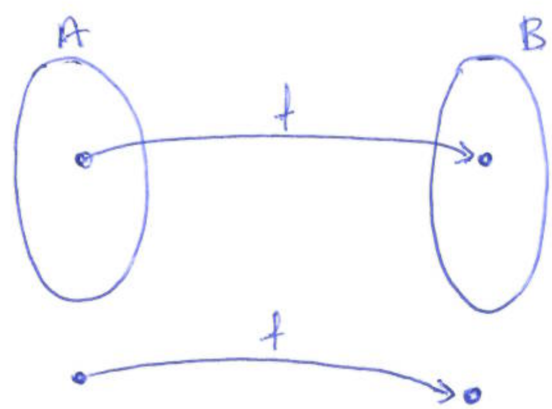
Most reductions $A \leq_T^P B$ you'll encounter will be of the form



These kinds of reductions are called mapping reductions.

With these reductions, all you have to do is come up with a function f that is computable in poly-time such that

$$x \in A \quad \text{iff} \quad f(x) \in B \qquad (x \in A \text{ iff } y \in B)$$



You have to always argue that

1. f is poly-time
2. $x \in A \Rightarrow f(x) \in B$
3. $f(x) \in B \Rightarrow x \in A$ (sometimes more convenient to argue $x \notin A \Rightarrow f(x) \notin B$)

Thm: SAT is NP-complete

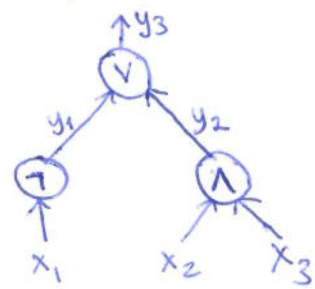
Proof (sketch): It is clear that SAT $\in$ NP.

We'll show SAT is NP-hard by showing Circuit-SAT $\leq$ SAT. (dropping T and P from $\leq_T^P$)

For illustration purposes, we'll do a proof by example, but the example is really without loss of generality.

Our reduction will be a mapping reduction. So we'll convert an instance x of Circuit-SAT to an instance y of SAT so that $x \in$ CircuitSAT iff $y \in$ SAT.

Suppose we are given the circuit



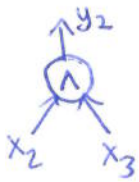
(These x_i's and y_j's have nothing to do with the x and y above.)

Here we have labeled each wire with a variable corresponding to the bit value it carries.

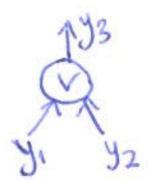
From this circuit we'll build a SAT formula with variables $x_1, x_2, x_3, y_1, y_2, y_3$.

Observe that:

$y_1 = \bar{x}_1 \iff (x_1 \vee y_1) \wedge (\bar{x}_1 \vee \bar{y}_1)$ (call this C_1)



$y_2 = x_2 \wedge x_3 \iff (\bar{y}_2 \vee x_2) \wedge (\bar{y}_2 \vee x_3) \wedge (y_2 \vee \bar{x}_2 \vee \bar{x}_3)$ (call this C_2)



$y_3 = y_1 \vee y_2 \iff (y_3 \vee \bar{y}_1) \wedge (y_3 \vee \bar{y}_2) \wedge (\bar{y}_3 \vee y_1 \vee y_2)$ (call this C_3)

The SAT formula corresponding to the circuit is $C_1 \wedge C_2 \wedge C_3 \wedge y_3$ $\square$

3SAT Problem

Input: A Boolean formula in CNF (conjunctive normal form) such that each clause contains exactly 3 literals.

e.g. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$

Output: Yes if the formula is satisfiable.
No otherwise.

Thm: 3SAT is NP-complete.

Proof: Adapt previous proof so that each clause has 3 literals. □

Thm: 2SAT is in P.

CLIQUE Problem

Input: An undirected graph $G=(V,E)$ and a number k.

Output: Yes if G contains a clique of size at least k.
No otherwise.

Thm: CLIQUE is NP-complete.

Proof: CLIQUE is in NP because a proof that a given graph has a clique of size at least k is the set of vertices of size at least k that are all connected to each other. We can verify in polynomial time that this set indeed has size at least k, and that every pair of vertices in the set has an edge between them.

To show CLIQUE is NP-hard, we show 3SAT $\leq$ CLIQUE.

Given a 3SAT instance φ, we will transform it into a CLIQUE instance $\langle G, k \rangle$ so that φ is satisfiable iff G has a clique of size k.

Let $\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_m \vee b_m \vee c_m)$

where each a_i, b_i, c_i is a literal.

(Notice that φ is satisfiable means that we can set values to the variables so that each clause has at least one literal set to 1.)

We build the graph G as follows. For each clause, we create 3 vertices corresponding to the literals in that clause. So in total, our graph has $3m$ vertices.

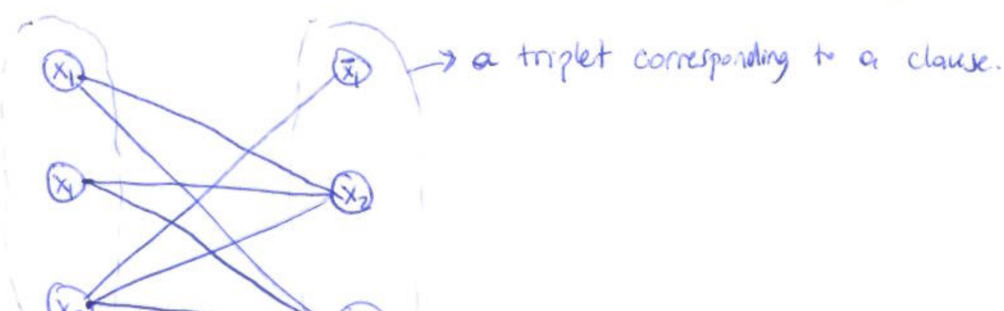
Vertices corresponding to the same clause are not connected to each other with an edge.

Vertices corresponding to contradictory labels (e.g. x_2 and $\overline{x_2}$) are not connected to each other with an edge.

Every other pair of vertices are connected with an edge.

<u>Example</u> If $\varphi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee x_2 \vee x_2)$, then

$G =$



This is the construction of G. And we let $k = m$.

We claim that φ is satisfiable iff $\langle G, \underset{\substack{\| \\ m\, =\, \#\text{clauses}}}{k} \rangle$ has a clique of size k.

If φ is satisfiable, then there is an assignment to the variables so that at least one literal from each clause is set to 1. The vertices corresponding to these literals form a clique of size m:

The only way two of these vertices do not share an edge is if they are x_i and $\overline{x_i}$ for some variable x_i. But a satisfying assignment cannot assign 1 to both x_i and $\overline{x_i}$. So the literals that were picked cannot contain both x_i and $\overline{x_i}$.

For the reverse direction, suppose the constructed G has a clique of size m. These m vertices have to come from different triplets. (~~a trip~~ the vertices in a triplet that correspond to a clause do not share an edge.)

So these m vertices correspond to a choice of one literal from each clause of φ. These literals can be simultoneously set to 1: The only way we could not have done this is if this set of literals contained two literals of the form x_i and $\overline{x_i}$, but we know these literals could not be in a clique as they are not connected by an edge.

Since we can set these literals simultoneously to 1, and there is one literal from each clause, the formula φ is satisfiable.

This completes the proof of correctness of the reduction.

It is easy to check that the reduction can be done in poly-time.

□