CS 3510 Algorithms	11/01/2023

Lecture 15: NP-completeness

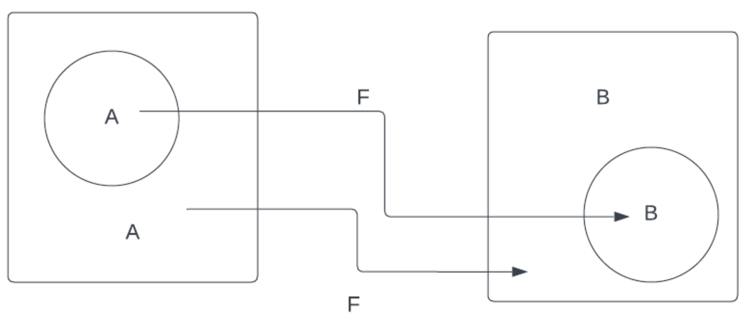
Lecturer: Abrahim Ladha — *Scribe(s): Jiaxuan Chen*

1 Introduction

Let's review what we did last time. We began our discussion on NP-completeness. To prove some problem B is NP-complete, you should:

1. Prove $B \in NP$ by showing it is verifiable in polynomial time.
2. Prove B is NP-hard. That is, $A \leq_p B$.
   - Choose some known A which is NP-complete.
   - Give some reduction f computable in polynomial time such that, for every $x \in A$:

$$x \in A(\text{is good}) \iff f(x) \in B(\text{is good}),$$
$$x \notin A(\text{is bad}) \implies f(x) \notin B(\text{is bad}).$$



In order to prove a problem is NP-complete, this depends on some other known NP-complete problem existing. Cook and Levin independently did this. They proved SAT is NP-complete without a predecessor. That is, $\forall A \in \mathsf{NP}, A \leq_p \text{SAT}$. Note that, this is true for every problem in NP. But, what is SAT?

A variable is one of $x_1, x_2, \ldots, x_n$.
A literal is a variable or its negation x_i or $\neg x_i$.
A clause is an OR of several literals.
A formula in CNF form is an AND of several clauses.

For example:

$$(x_1) \wedge (\neg x_1)$$

is unsatisfiable. Another example might be satisfiable:

$$(x \lor y \lor z) \land (x \lor z \lor w) \land \dots$$

SAT

SAT is extremely universal. Most constraint problems can be made to look like SAT. Each clause is a constraint: every constraint must be satisfied, but they can be satisfied in a number of ways.
Let's say you have to feed everyone. You want either a burger, a gyro or a cheeseburger. My buddy only wants a cheeseburger. Each of us is a constraint. We have variables like you order a burger (b) or gyro (g). Our SAT formula is like:

$$(b \lor g \lor c) \land c$$

The formula $(x_1 \lor \neg y_1) \land (x_2 \lor y_2) \land \dots \land (x_n \lor \neg y_n)$ is satisfiable only when $x_1 = y_1, x_2 = y_2, \dots, x_n = y_n$. A SAT formula for string equality.
To be clear, an assignment is a selection of variables $x_i \in \{0, 1\}$. An assignment satisfies a given boolean constraint, an assignment satisfies I.

SAT Definition: $\Phi \in SAT$ such that Φ is a formula in CNF form and is satisfiable.

Recall Cook and Levin proved $L \in \mathsf{NP} \implies L \leq_p$ SAT. So if $SAT \in \mathsf{P} \implies \mathsf{NP} \subseteq \mathsf{P} \implies \mathsf{P} = \mathsf{NP}$. SAT is like an elected representative of the entire class of NP. This is also why we don't believe there exists a polynomial time algorithm for SAT.

$kSAT$ definition: $\exists\Phi$ such that Φ is a formula in CNF, satisfiable, each clause has at most k literals.

3SAT

We prove that $3SAT$ is NP-complete by reduction. First, we show $3SAT \in \mathsf{NP}$. Our witness is simply the assignment of variables for the problem instance solution. All these computations can be done in polynomial time. For all $\Phi(C_1, \dots, C_m)$, check if $\Phi(C_1, \dots, C_m) = 1$ or not.

Now we prove $SAT \leq_p 3SAT$. For a general SAT formula, we convert it to a $3SAT$ instance such that Φ is satisfiable ($\in SAT$) if and only if $F(\Phi)$ is satisfiable ($\in 3SAT$). We describe our reduction F as follows: For an input Φ of every SAT formula has some max clause size k. If $k \leq 3$ then Φ is both in SAT and $3SAT$. Now suppose Φ has max clause size $k > 3$. We convert a clause of size $k > 3$ to a pair of clauses, one of size $k - 1$ and the other of size 3. We add a variable z as follows:

$$(x_1 \lor x_2 \lor \dots \lor x_{k-1} \lor x_k) \iff (x_1 \lor x_2 \lor \dots \lor x_{k-1} \lor z) \land (\neg x_k \lor \neg z)$$

Where each x_i is a literal. Note, if the k clause is true, at least one of its literals is true, so there is a selection of z to make the two clauses true. If the k clause is always false, the two clauses are also always false for any selection of z. Note, it is important this conversion does not change the satisfiability of Φ. Repeat this process, adding dummy variables, until Φ only has clauses of size 3.

- Note: Since this does not alter satisfiability, $\Phi \in SAT$ if and only if $F(\Phi) \in 3SAT$. reduction F occurs in polynomial time.
- This reduction F occurs in polynomial time.
- We conclude: $SAT \leq_p 3SAT$ and so, $3SAT$ is NP-complete.

Note that since Cook-Levin showed us SAT $\in \mathsf{NP}$, 3SAT $\leq_p$ SAT, and we found 3SAT $\in \mathsf{NP}$, 3SAT $\leq_p$ 3SAT. This implies 3SAT is NP-complete without having to repeat the entire SAT proof. A simple reduction suffices. It is possible to repeat this reduction for 4SAT, 5SAT, $\ldots$, kSAT for any $k \geq 3$.

What about 2SAT? Actually, 2SAT $\in \mathsf{P}$, so if 3SAT $\leq_p$ 2SAT, SAT $\in \mathsf{P}$ and $\mathsf{NP} = \mathsf{P}$. Surely, we don't believe should happen. Recall $(p \Rightarrow q) \Leftrightarrow (\neg p \vee q)$. So every 2SAT clause of size two is an implication.

$$(a \vee b) \Leftrightarrow (\neg a \Rightarrow b), \quad (\neg a \vee b) \Leftrightarrow (a \Rightarrow b), \quad (a \vee \neg b) \Leftrightarrow (\neg a \Rightarrow \neg b), \quad (\neg a \vee \neg b) \Leftrightarrow (a \Rightarrow \neg b).$$

Create a graph two vertices for each literal, two edges for each clause. If $(a \vee b)$ a clause, add edge $\neg a \to b$, $\neg b \to a$. Recall implication is transitive, and a formula is unsatisfiable if and only if $\forall x, (x \Rightarrow \neg x)$ or $(\neg x \Rightarrow x)$ so $\exists$ a path in our graph from x to $\neg x$ and from $\neg x$ to x.

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

$$\begin{array}{ccc} x & \to & \neg y \\ \neg y & \to & x \\ \neg x & \to & \neg y \\ \neg y & \to & \neg x \\ x & \to & y \\ y & \to & \neg x \end{array}$$

If $x = 0 \Rightarrow y = 1$, if $y = 1 \Rightarrow x = 0$.

CircuitSAT

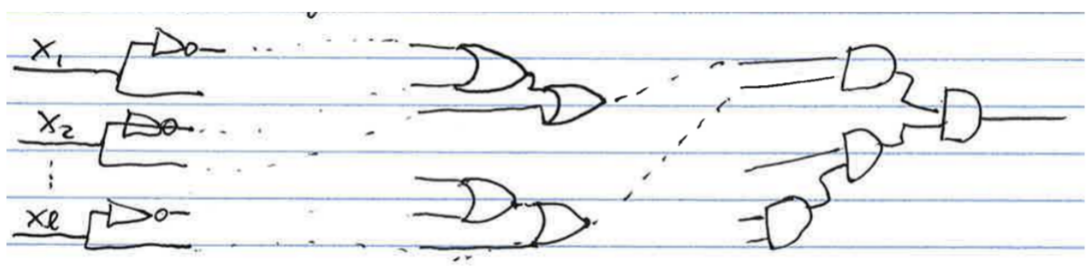
Let circuitSAT be defined as the set:

circuitSAT = $\{C \mid C$ a boolean circuit with AND/OR/NOT gates and a way to bring output to 1$\}$

We prove that circuitSAT is NP-complete.

- First, we show that circuitSAT $\in$ NP. The verifier V takes as input $\langle C \rangle$ and a witness of n bits, and runs $\langle C \rangle$ on the inputs. The size of the input is obviously polynomial (increasing depth or more gates).

- Now, we show that $3SAT \leq_p$ circuitSAT. Let Φ be a $3SAT$ formula. We create a boolean circuit with variables $x_1, \ldots, x_k$ and additional input wires for negated literals. We add one root gate on the next layer. For each clause, add a sub-circuit for the appropriate three. Then, add an "AND" gate to AND the clauses together.



- If $\Phi \in 3SAT$, this circuit $C = F(\Phi) \in$ circuitSAT.

- If $\Phi \notin 3SAT$, this circuit is also unsatisfiable.

- Construction of this circuit obviously takes polynomial time.

We conclude that $3SAT \leq_p$ circuitSAT so circuitSAT is NP-complete.