1 My First NP-Complete Problem

We need to start somewhere.

- How can you prove that a problem $Q$ is as hard as any problem in NP?
- Must show a reduction from an arbitrary problem in NP to $Q$.
- Idea: we will identify a problem whose solution can be used to simulate a general-purpose computer.

On to the problem definition.

- Consider the set of all propositional Boolean formulas $\phi$ over the connectives $\land$, $\lor$, and $\neg$.
- Example:
  \[ \phi = (x \land y) \lor (\neg x \land z) \]
- Each propositional variable may be assigned a value of true or false.
- Depending on values assigned to vars, formula may be true or false.
- If assignment $A$ of values to variable makes formula $\phi$ true, we say that $A$ satisfies $\phi$.
- Example: if $x = \text{false}$, $y = \text{false}$, and $z = \text{true}$, then $\phi$ is true.
- Not every formula has a satisfying assignment!
- Example:
  \[ \psi = ((x \land y) \lor (\neg x \land z)) \land \neg(y \lor z) \]
  is unsatisfiable.
- Problem (SAT): given a Boolean formula $\phi$ on variable set $X = \{x_1 \ldots x_n\}$, does there exist an assignment to $X$ that satisfies $\phi$?
2 SAT is NP-Complete, Part 1

**Thm:** SAT is NP-complete.

- First, let’s check that SAT is in NP.
- A certificate for formula $\phi$ is a satisfying assignment $A$ to $\phi$’s variables!
- If $\phi$ uses all its variables, it certainly includes $\Omega(|A|)$ symbols, so $A$ has size polynomial in $\phi$.
- Moreover, we can verify $\phi$ by plugging in the assignment and evaluating the formula!
- Can be done in time proportional to size of $\phi$: evaluate logical expressions from the inside out, taking constant time per logical operation in $\phi$. QED

OK, but why is SAT complete for NP?

- **Claim:** let $L$ be any problem in NP. Then $L \leq_p \text{SAT}$.
- **Pf:** Start from definition of NP.
  - For every $x \in L$, there exists a certificate $c$ of size polynomial in $x$.
  - Also, there exists a verification algorithm $V(x,c)$ that validates that $x \in L$ in time polynomial in $|x|$ and $|c|$.
- **Idea:** given an input $z$ to problem $L$, reduction $f$ constructs a formula $\phi(z)$ of size $\text{poly}(|z|)$.
  - $\phi(z)$ will be satisfiable iff there exists a polynomial-sized certificate $c$, such that $V(z,c)$ returns true.
  - But by defn of NP, $c$ exists for $z$ iff $z \in L$.
  - Hence, we will have that $z \in L$ iff $\phi(z) \in \text{SAT}$.

Now all we have to do is to construct $\phi(z)$! But how?

- Algorithm $V$ terminates in a polynomial number of steps in its input size when run on a computer.
- What the heck does “run on a computer” mean?
  - (Formally, run on a Turing machine, but let’s not go there.)
  - A reasonable computer has the following properties:
    1. A computer runs a program composed of machine instructions that read and write a *store* (memory, registers, and PC, if you like)
    2. Each step of an algorithm is one of a fixed, finite set of instructions.
    3. For some constant $m$, a single instruction reads and writes at most $m$ bits of store.
4. Instruction executed each cycle is defined by a program counter, which is part of the store.

5. When a computer runs \( V \) on input \( z, c \) this input is written at the beginning of the store (first \( c \), then \( z \)). All other bits are initially set to 0.

6. When \( V \) terminates, a certain bit in the store, say the \( i^* \)th bit, contains its result, which is “true” or “false.”

   - This computational model is simple enough for our needs.
   - \( \phi(z) \) will describe execution of \( V \) on such a computer given input \( z \) and a certificate of size at most polynomial in \( |z| \).

Before we begin, some important facts to bound running time and space of \( V \).

- \( V(z, c) \) terminates in at most \( |z|^a |c|^b \) steps, for some constants \( a \), and \( b \).
  (Follows because \( V \) runs in time polynomial in its input size)

- If there exists a certificate \( c \) for which \( V(z, c) \) answers true, then there exists such a \( c \) of size at most \( r = |z|^d \), for some constant \( d \).
  (Follows because \( c \) is of size polynomial in \( |z| \).)

- Hence, if \( V(z, c) \) answers true for any \( c \), it does so for some \( c \) in at most \( n = |z|^{a+bd} \) steps.

- Finally, in this many steps, a computer running \( V \) can read and write at most \( mn \) total bits of store.

- WLOG, assume that complete state of computation is described by first \( mn \) bits of store.

- Remember: by our defns, both \( n \) and \( mn \) are \( \text{poly}(|z|) \).

3 SAT is NP-Complete, Part 2

On to construction of \( \phi(z) \! 

- Define Boolean variable \( b_{i,t} \) to be the value of bit \( i \) of the store after \( t \) steps of computation.

- To describe entire state of store at time \( t \), we need to specify values of bits \( b_{1,t} \ldots b_{mn,t} \).

- Value of any bit at time \( t \) depends on state of store at time \( t-1 \).

- There exists some boolean function \( f_{i,t} \) such that
  \[
  b_{i,t} = f_{i,t}(b_{1,t-1} \ldots b_{mn,t-1})
  \]

- We record this fact in the following formula \( F_{i,t} \):
  \[
  b_{i,t} \iff f_{i,t}(b_{1,t-1} \ldots b_{mn,t-1}).
  \]
• If we specify store at time $t - 1$, formula $F_{i,t}$ is satisfied iff bit $i$ at time $t$ is consistent with what $V$ would compute given the store at time $t - 1$.

• We claim that size of function $f_{i,t}$, and hence of formula $F_{i,t}$, is polynomial in $|z|$.

• Indeed, a fixed instruction’s modification of bit $i$ depends on only a constant number $(m)$ of bits of the store, so we can write a constant-sized description $f_{i,t}^j$ assuming we execute the $j$th instruction of program $V$ at time $t$.

• Moreover, which instruction we execute at time $t$ is determined by the program counter, and there cannot be more than $n (= \text{poly}(|z|))$ possible instructions if, as we assumed above, we do not run $V$ for more than $n$ steps.

• Conclude that $f_{i,t}$ can be written as a disjunction of at most $n$ formulas $f_{i,t}^j$ selected among based on the PC value $j$, and so it has size $\text{poly}(|z|)$.

• To describe evolution of entire store from $t - 1$ to $t$, we write a formula $E_t$ that describes what happens to each of its bits:

$$E_t = F_{1,t} \land \ldots \land F_{mn,t}$$

Note that $E_t$ has size $O(mn)$, which is poly$(|z|)$.

• Finally, we set

$$\alpha = E_1 \land \ldots \land E_n \land b_{i^*n}.$$  

• When is $\alpha$ satisfiable?

• Precisely when there are values for all $b_{i,0}$, the initial state of the store, such that after $n$ steps, $V$’s output bit is true.

• Note that $\alpha$ has size $O(mn^2)$, which is still poly$(|z|)$.

We’re almost there.

• At beginning of computation, we want to specify that store contains the string $\langle c \rangle \cdot \langle z \rangle \cdot 00000 \ldots$, that is, alleged certificate $c$, followed by input $z$, followed by zeros.

• First $r$ bits of initial store, $b_{1,0} \ldots b_{r,0}$, specify $c$.

• Let $X = \langle z \rangle \cdot 00000 \ldots$

• For each remaining bit $j$, $r + 1 \leq j \leq mn$, let

$$L_j = \begin{cases}  b_{j,0} & \text{if bit } j - r \text{ of } X \text{ is 1} \\  -b_{j,0} & \text{if bit } j - r \text{ of } X \text{ is 0} \end{cases}$$

• Set $S_0(z) = L_{r+1} \land \ldots \land L_{mn}$.

• Finally, set

$$\phi(z) = \alpha \land S_0(z).$$

• Note that $\phi(z)$ has size $O(mn^2 + mn) = O(mn^2)$, which is poly$(|z|)$.
• Given \( z \), can generate \( \phi(z) \) in time \( \text{poly}(|z|) \) by computing \( S_0(z) \), then adjoining the fixed formula \( \alpha \) to it.

• When is \( \phi(z) \) satisfiable?

• \( S_0(z) \) forces initial bits of store, other than input \( c \), to be \( z \) followed by 0’s.

• Hence, \( \phi(z) \) is satisfiable precisely when there exist values for remaining input bits \( b_{1,0} \ldots b_{r,0} \) that cause \( b_{i^*,n} \) to become true!

• In other words, \( \phi(z) \) is satisfiable iff there exists a certificate \( c \) of length at most \( r \) for which \( V(z,c) \) is true!

• To conclude, \( \phi(z) \in \text{SAT} \) iff \( z \in L \). QED