1 3-SAT is Still Hard

We noted last time that general SAT is NP-complete. How hard is SAT if we restrict its inputs to only 3-CNF formulas?

- Will consider the 3-CNF-SAT, or just 3-SAT, problem.
- Given a 3-CNF formula $\phi$, is there a satisfying assignment to $\phi$?
- Is 3-CNF-SAT NP-complete?
- If we can solve SAT, we can trivially solve 3-CNF-SAT.
- But this only shows 3-CNF-SAT $\leq_p$ SAT.
- “Soln to a hard problem can be used to solve an easy problem” – duh.
- We need to prove the other direction.

Claim: 3-CNF-SAT is NP-complete.

- **Pf:** first, must show that 3-CNF-SAT is in NP.
- I’ll just reuse NP-ness proof for SAT; same certificate and verification scheme works.
- Still must show NP-hardness.
- Will prove that SAT $\leq_p$ 3-CNF-SAT.
- Given a formula $\phi$, we will construct a 3-CNF formula $\psi = f(\phi)$.
- Will show that $\psi$ is satisfiable iff $\phi$ is satisfiable.

OK, here’s the reduction, given $\phi$.

- **Step 1:** push all $\neg$s to inside to create an equivalent formula $\phi_1$ using only ands, ors, and literals.
- **Example:** if $\phi = (x \land y) \lor \neg(x \land z)$, then
  $$\phi_1 = (x \land y) \lor (\neg x \lor \neg z).$$
- (Takes $O(|\phi|)$ transformations, each in time $O(|\phi|)$)
• **Step 2**: construct a *parse tree* for $\phi_1$.

• **Example:**

- Each leaf of parse tree contains a literal.
- Each internal node contains a binary connective.
- At most as many internal nodes as leaves, so tree has $O(|\phi|)$ nodes.
- Number all nodes of tree; for each node $j$, assign a variable $v_j$.

• **Draw $y$’s on above parse tree**

• **Step 3**: construct a formula $\psi_0$ from parse tree as follows.

- If node $j$ is a leaf with literal $\ell$, set
  \[ C_j = (v_j \leftrightarrow \ell). \]

- If node $j$ connects nodes $p$ and $q$ with connective $\otimes$, set
  \[ C_j = (v_j \leftrightarrow v_p \otimes v_q). \]

- Finally, if $v_1$ labels the root of parse tree $T$, set
  \[ \psi_0(T) = v_1 \land \bigwedge_j C_j. \]

• **Example:**

- (Note that $\psi_0(T)$ has size linear in $|T|$, and hence linear in $|\phi_1|$.)

• **Claim**: $\psi_0$ is satisfiable iff $\phi_1$ is satisfiable.

• **Pf** (sketch): suppose assignment $A$ satisfies $\psi_0$.  


• Can show inductively on structure of $T$ that $A$ makes $v_j$ true iff $A$'s assignment to literals satisfies subformula of $\phi_1$ corresponding to subtree rooted at node $j$.
• Since $A$ makes $v_1$ true, the whole formula $\phi_1$ must be satisfiable.
• Conversely, if assignment $A$ satisfies $\phi_1$, construct assignment $A'$ for $\psi_0$ from $A$ by setting each $v_j$ in bottom-up fashion to make its $C_j$ true. QED

We now want to turn $\psi_0$ into an equivalent 3-CNF formula.

• Each subformula $C_j$ has at most 3 variables but is not necessarily CNF.
• **Step 4**: turn each $C_j$ into equivalent CNF formula $\alpha_j$ of at most constant size.
  
  First, write down the truth table for $C_j$.
  
  Then, build an equivalent *disjunctive* formula giving all the 0s of $C_j$.
  
  Finally, negate the result to achieve a CNF formula for $C_j$.
• **Example**: suppose we have formula $\neg(x \oplus y)$ ($\oplus$ means “exclusive-or”).

• Transformation turns $C_j$ into at most $2^3$ clauses, each with at most 3 variables.
• Hence, $|\alpha_j|$ is $O(1)$.
• Let $\psi_1 = \bigwedge_j \alpha_j$.
• Observe that $|\psi_1| = O(|\psi_0|)$, and that $\psi_1$ is satisfiable iff $\psi_0$ is.

Almost done.

• Formally, a 3-CNF formula must have *exactly* three literals per clause.
• Our $\psi_1$ might have only 1 or 2 literals.
• **Step 5**: transform each $\alpha_j$ into a valid 3-CNF formula.
  
  Let $p$ and $q$ be dummy variables.
  
  Replace each 2-literal clause $\ell_1 \lor \ell_2$ by
  
  \[(p \lor \ell_1 \lor \ell_2) \land (\neg p \lor \ell_1 \lor \ell_2)\]
  
  Above is logically eqv to $(p \land \neg p) \lor (\ell_1 \lor \ell_2)$, so is satisfied precisely when $\ell_1 \lor \ell_2$ is satisfied (for any $p$).
• Replace each 1-literal clause $\ell$ by

$$(\neg p \lor \neg q \lor \ell) \land (\neg p \lor q \lor \ell) \land (p \lor \neg q \lor \ell) \land (p \lor q \lor \ell)$$

• Above is logically eqv to $(p \land \neg p) \lor (q \land \neg q) \lor \ell$, so is satisfied precisely when $\ell$ is satisfied (for any $p$, $q$).

• Let $\psi$ be formula obtained by $\psi_1$ by above transformation.

• Each clause of $\psi_1$ expands to at most 4 clauses, so $|\psi| = O(|\psi_1|)$.

• Moreover, $\psi$ is logically equivalent to $\psi_1$.

• Hence, $\psi$ is satisfied iff $\phi$ is satisfied!

• Moreover, $|\psi| = O(|\phi|)$, and $\psi$ can be built from $\phi$ in time $O(|\phi|^2)$.

• Conclude that SAT $\leq_p$ 3-CNF-SAT! QED

2 Another Hard Problem – SUBSET-SUM

Because 3SAT is NP-complete, we now know from reductions that we’ve seen that INDEPENDENT-SET (and hence VERTEX-COVER), which are clearly in NP, are also NP-complete. Now, let’s study yet another hard problem!

• **Input**: a set $S$ of positive integers, and a target $t$

• **Problem**: does $S$ contain subset $S'$ whose members add to exactly $t$?

• This is called the SUBSET-SUM problem.

• **Ex**: $S = \{2, 5, 3, 6, 9\}$: true for $(S, 14)$, but not for $(S, 4)$.

Well, that doesn’t seem hard, does it?

• **Lemma**: SUBSET-SUM is NP-complete.

• **Pf**: First, will show it’s in NP.

• Given instance $(S, t)$, cert is subset $S'$.

• Surely, $|S'| \leq |S|$.

• Moreover, can check in time $O(|S||S'|)$ that all elts of $S'$ are in $S$, and that their sum is $t$.

So far, so good. But what about NP-hardness?

• Will reduce from 3-SAT (i.e. prove 3-SAT $\leq_p$ SUBSET-SUM)!

• Given a 3-CNF formula $\phi$, will construct an instance $(S, t)$ of SUBSET-SUM.

• Will show that $S$ has subset with sum $t$ iff $\phi$ is satisfiable.
• WLOG, will assume $\phi$ does not contain any clause with both literals $x$ and $\neg x$.
• (Any such clause is trivially satisfiable, so transducer function $f$ can delete it.)

Yikes! What’s the construction?

• Suppose $\phi$ has $n$ variables and $m$ clauses.
• Set $S$ will contain base-10 integers, each with $n + m$ digits.
• Label the digit positions $p_1 \ldots p_n, q_1 \ldots q_m$.
• For variable $x_i$, define integers $v_i$ and $\bar{v}_i$ as follows:
  1. Digit $p_i$ of both $v_i$ and $\bar{v}_i$ is 1.
  2. All other digits $p_j, j \neq i$, are 0 for both $v_i$ and $\bar{v}_i$.
  3. Digit $q_k$ of $v_i$ is 1 iff $x_i$ appears in clause $C_k$ of $\phi$.
  4. Digit $q_k$ of $\bar{v}_i$ is 1 iff $\neg x_i$ appears in clause $C_k$ of $\phi$.
• For each clause $C_k$, define integers $y_k$ and $z_k$ as follows:
  1. All digits $p_j$ of both $y_k$ and $z_k$ are zero.
  2. Digit $q_k$ of $y_k$ is 1; digit $q_k$ of $z_k$ is 2.
  3. All other digits $q_\ell, \ell \neq k$, of both $y_k$ and $z_k$ are zero.
• Finally, define target $t$ as follows:
  1. Every digit $p_j$ of $t$ is 1.
  2. Every digit $q_k$ of $t$ is 4.
• Example:

That’s really funky (but at least it’s polynomial-time). Why does it work?

• Observation 0: $S$ really is a set – $v_i$ and $\bar{v}_i$ differ because $x_i$ and $\neg x_i$ do not occur together in any clause, and all other numbers are pairwise different.
• Observation 1: if we add up any subset of values in $S$, no digit position causes a carry.
• (No position \( p_j \) exceeds 2, and no position \( q_k \) exceeds 6.)
• Hence, we can consider each position separately.
• **Claim 1**: If \( \phi \) is satisfiable, \((S,t)\) is solvable.
  • **Pf**: let \( A \) be a satisfying assignment to \( \phi \).
  • Construct \( S' \) as follows.
  • If \( A \) makes variable \( x_i \) true, add \( v_i \) to \( S' \); otherwise, add \( \bar{v}_i \).
  • This alone guarantees that posn \( p_i \) sums to 1 for \( 1 \leq i \leq n \).
  • For clause \( C_k \), let \( a_k \) be the sum in posn \( q_k \) of all values in \( S' \) so far.
  • Note that \( a_k \) must be at least 1 (since \( \phi \) is satisfied) and at most 3.
    - If \( a_k \) is 3, add \( y_k \) to \( S' \).
    - If \( a_k \) is 2, add \( z_k \) to \( S' \).
    - If \( a_k \) is 1, add both \( y_k \) and \( z_k \) to \( S' \).
  • This ensures that position \( q_k \) sums to 4 over \( S' \).
• Conclude that total of all values in \( S' \) is exactly \( t \).

Halfway there...

• **Claim 2**: If \((S,t)\) is solvable, \( \phi \) is satisfiable.
  • **Pf**: Let \( S' \) be a valid solution to \((S,t)\).
  • Construct assignment \( A \) for \( \phi \) as follows:
    - If \( v_i \in S' \), set \( x_i \) true.
    - If \( \bar{v}_i \in S' \), set \( x_i \) false.
  • Note first that every valid solution to \((S,t)\) includes exactly one of \( v_i \) and \( \bar{v}_i \).
  • (Only way to make sum in posn \( p_i \) equal to 1.)
  • Hence, \( A \) assigns every \( x_i \) a unique truth value.
  • Suppose clause \( C_k \) contains literals \( \ell_1 \), \( \ell_2 \), and \( \ell_3 \).
  • **Notn**: for a literal \( \ell \), let \( v(\ell) \) be the value in \( S \) corresponding to \( \ell \).
  • Every valid solution to \((S,t)\) must contain at least one of the values \( v(\ell_1) \), \( v(\ell_2) \), \( v(\ell_3) \).
  • (We cannot make 4 in posn \( q_k \) using only \( y_k \) and \( z_k \).)
  • But then \( A \) makes at least one of the three literals true!
  • Conclude that \( A \) satisfies every clause of \( \phi \). QED