CSCI 341–Fall 2024: Lecture Notes
Set 20: Proof of the Cook-Levin Theorem

Edward Talmage

December 6, 2024

Polynomial-time reductions are great for showing that more problems are NP-Hard, but we need a starting point, a first language we prove is NP-Hard. We will prove that SAT is NP-Complete, so that we can use it to prove other problems are NP-Complete and -Hard more easily.

Theorem 1. *SAT is NP-Complete.*

> **Exercise:** What are the two conditions for a problem to be NP-Complete?

Proof. First, we must show that SAT is in NP.

> **Exercise:** Prove that $SAT \in NP$.

We can non-deterministically try all truth assignments to the variables in polynomial time, by branching into a True option and a False option for every variable in the expression, then evaluate the entire expression.

Now, we must show that every NP problem is poly-time reducible to SAT. To do this, let A be any NP problem, with a poly-time NTM M. For any string w, we use $\langle M, w\rangle$ to build a Boolean formula ϕ for which M accepts w iff ϕ is satisfiable. Define k s.t. M's complexity is at most n^k steps.

We want to represent the computation of each branch of the NTM M running on w. To do this, we will build tables (the book calls them "tableaus"), where each row is a configuration of M. Recall that a configuration is a string uqw, where q is the current state, uw is the tape contents, and the $Read/Write$ head points to the first element of w. The first row of each table will be the start configuration, the second row is the second configuration on this execution branch, and so on. Note that each table is $n^k \times n^k$, since a branch may take up to n^k steps, and thus may use up to n^k memory. If any row is an accepting configuration, we say that this is an accepting table. All we need to do now is determine whether there is an accepting table which represents a valid branch of M's computation on w.

Example:

#	q_0	w_1	w_2	w_3	...	w_n	⊔	⊔	...	⊔	#
#	w_1'	q_1	w_2	w_3	...	w_n	⊔	⊔	...	⊔	#
#					...						#
		⋮							⋮		
#					...						#

(Note that the #'s are just delimiters for convenience.)

Now, we reduce A to SAT using these tables. Given A and an input w, we need to construct a formula that is satisfiable iff A accepts w. That is, if there is an accepting table. Let Q and Γ be the state set and tape alphabet of A, respectively. Let $C = Q \cup \Gamma \cup \{\#\}$. Each variable in our expression will be of the form $X_{i,j,s}$, where $1 \leq i, j \leq n^k$ and $s \in C$. Such a variable is true if there is an s in the table cell at index $[i, j]$.

> **Exercise:** How many variables do we have?

Our formula ϕ now just has to check that we have a valid, accepting table, so consider what properties must hold for this to be true:

1. The first row must be a valid start configuration:

$$h_s = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge \cdots \wedge x_{1,n^k,\#}$$

2. We need exactly one variable per cell:

$$h_c = \bigwedge_{1 \le i,f \le n^k} \left(\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{s,t \in C, s \neq t} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right)$$

3. We need to have an accept state somewhere in the table:

$$h_a = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_q}$$

4. We need each row to be the result of a valid step from the previous row. Note that the only changes between two rows occur in a 2×3 "window" with the current state in the center of the top row. Those changes look like one of the following:

If $\delta(q, b) \ni (q', c, R)$:

a	q	b
a	c	q'

If $\delta(q, b) \ni (q', c, L)$:

a	q	b
q'	c	b

All windows away from the current *Read/Write* head will be unchanged. Then we have

$$h_m = \bigwedge_{1 \le i,j \le n^k} (2 \times 3 \text{ window with upper corner at cell } [i, j] \text{ is valid})$$

If we let $a_1, \ldots, a_6$ be the six cells of a window, we can express the legality of a window as follows:

$$h_m = \bigwedge_{1 \le i,j, \le n^k} \left(\bigvee_{a_1,\ldots,a_6 \text{ is legal}} (x_{i,j,a_1} \wedge x_{i,j+1,a_2} \wedge x_{i,j+2,a_3} \wedge x_{i+1,j,a_4} \wedge x_{i+1,j+1,a_5} \wedge x_{i+1,j+2,a_6}) \right)$$

Note that the non-determinism of M is handled in the $\bigvee$, since different non-deterministic choices will yield different valid windows.

Let $\phi = h_s \wedge h_c \wedge h_1 \wedge h_m$. ϕ is satisfiable iff there is an accepting table for $M(w)$.

We now have a reduction from A to SAT. All we need to do is argue that constructing ϕ takes only polynomial time in the input. We have $(n^k)(n^k)(|C|) = O(n^{2k})$ variables, since $|C|$ is constant. We can consider the length of each of our subexpressions:

1. $|h_s| = O(n^k)$, since it has a variable for each cell in row 1 of the table.
2. $|h_c| = O(n^{2k})$, since it has a constant number of variables for each cell in the table.
3. $|h_a| = O(n^{2k})$, as for h_c (one variable per cell)
4. $|h_m| = O(n^{2k})$, as for h_c (six variables per cell times the constant number of non-deterministic branches)

There is then an extra $\log(n)$ factor for each, since indices could take $\log(n)$ bits, but we get a total length $|\phi| = O(n^{2k} \log N)$, which is polynomial. The work to generate each term is a small polynomial (looking up information in the input), so total work is a product of polynomials, which is polynomial. □

Theorem 2. *3-SAT is NP-Complete.*

Sketch. First, note that 3-SAT is in NP, since we can solve it non-deterministically by trying all possible assignments.

Next, we can tweak the argument that $SAT \in NPC$ to make ϕ be in 3-CNF. Of the four subexpressions we used in the proof of the Cook-Levin Theorem, only h_m is not already in CNF. Instead, it is in Disjunctive Normal Form, meaning it is an OR of ANDs. Distributing the OR gives us an expression in CNF, which may be longer than h_m by a constant factor dependent on M.

Exercise: Convert the DNF expression $(a \wedge b) \vee (c \wedge d)$ to CNF.

Now, we have $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_t$ in CNF, and we need to convert each clause C_i to have 3 or fewer terms. There are four cases, each of which increases the number of clauses by at most 3:

1. C_i is a single literal: $C_i = (a)$. Replace with $(a \vee a \vee a)$.
2. C_i is the OR of two literals: $C_i = (a \vee b)$. Replace with $(a \vee b \vee b)$.
3. C_i is the OR of three literals: Do nothing.
4. C_i is the OR of four or more literals: $C_i = (a_1 \vee a_2 \vee \cdots \vee a_k)$. Add new variables $z_1, z_2, \ldots, z_{k-3}$ and replace C_i with
$$D_i = (a_1 \vee a_2 \vee z_1) \wedge (\overline{z_1} \vee a_3 \vee z_2) \wedge (\overline{z_2} \vee a_4 \vee z_3) \wedge \cdots \wedge (\overline{z_{k-3}} \wedge a_{k-1} \wedge a_k)$$
Note that D_i is satisfiable iff C_i is satisfiable, as C_i is True only when at least one a_i is True, and we can than set all the z_i to make the other clauses True. If C_i is unsatisfiable, then all a_i are False, and we cannot set the z_i's to make D_i True.

Now, ϕ is the AND of many clauses, each with exactly 3 variables, and is thus in 3-CNF, so we have reduced A to 3-SAT. The reduction is still polynomial, since we have increased ϕ by only a constant factor from the proof of the Cook-Levin Theorem. □

Now, we will use this to prove, by reduction, that another language is NP-Complete.

Claim 1. *$SUBSET\_SUM$ is NP-Complete.*

Recall that $SUBSET\_SUM = \{\langle S, t\rangle \mid S \text{ a set of numbers, } t \text{ a number, there is a subset of } S \text{ summing to } t\}$.

Proof. We previously showed that $SUBSET\_SUM$ is in NP. We will show that it is NP-Hard by reducing 3-SAT to $SUBSET\_SUM$.

Given a Boolean expression in 3-CNF, we need to construct a set S and target t s.t. S has a subset summing to t iff the expression is satisfiable. Let f be a Boolean formula in 3-CNF with variables $x_1, \ldots, x_h$ and clauses $c_1, \ldots c_k$. We generate S with two numbers for each variable and two numbers for each clause. We then choose t carefully.

1. For each variable x_i, add to S y_i which is 10^i plus a fractional portion where y_i has a 1 in the jth decimal place if x_i is in clause c_j.
2. For each variable x_i, add to S y_i which is 10^i plus a fractional portion where y_i has a 1 in the jth decimal place if $\overline{x_i}$ is in clause c_j.
3. For each clause c_i, add $g_i = h_i = 10^{-i}$.
4. Set $t = 11 \ldots 1.333 \ldots 3$, where there are h 1's and k 3's.

Example: Consider the formula
$$(a \vee b \vee c) \wedge (a \vee \overline{b} \vee d) \wedge (\overline{a} \vee \overline{c} \vee \overline{d})$$

We have $X = [a, b, c, d]$, $C = [c_1, c_2, c_3]$. We add the following values to S:

$$\begin{aligned}
y_1 &= \quad 1.110 \quad (a)\\
y_2 &= \quad 10.100 \quad (b)\\
y_3 &= 100.100 \quad (c)\\
y_4 &= 1000.010 \quad (d)
\end{aligned}$$

$$\begin{aligned}
z_1 &= \quad 1.001 \quad (\overline{a})\\
z_2 &= \quad 10.010 \quad (\overline{b})\\
z_3 &= 100.001 \quad (\overline{c})\\
z_4 &= 1000.001 \quad (\overline{d})
\end{aligned}$$

$$\begin{aligned}
g_1 &= \quad .1 \quad (c_1)\\
h_1 &= \quad .1 \quad (c_1)\\
g_2 &= \quad .01 \quad (c_2)\\
h_2 &= \quad .01 \quad (c_2)\\
g_3 &= \quad .001 \quad (c_3)\\
h_4 &= \quad .001 \quad (c_3)
\end{aligned}$$

$$t = 1111.333$$

$a, b, \overline{c}, d$ is a satisfying assignment, which corresponds to the subset $\{y_1, y_2, z_3, y_4, g_1, g_2, g_3, h_3\}$.

Back to the proof: We will know prove that S has a subset summing to t if and only if the Boolean formula was satisfiable.

Suppose f is satisfiable and fix one satisfying assignment. Construct a subset S' of S which sums to t as follows:

- If x_i is True in the satisfying assignment, then y_i is in S'.
- If x_i is False in the satisfying assignment, then z_i is in S'.

This matches the integer portion of t. Further, each of the decimal places is between 1 and 3, since we took at least one literal from each clause, but not more than 3, as there are not more than 3. Select whatever of the g_i's and g_i's are needed to bring each decimal place up to 3. There are two 1's per decimal place, so this is always feasible.

In the other direction, if the formula has no satisfying assignment, we need to show that S has no subset summing to t. We prove this by contrapositive: Suppose that S has a subset S' summing to t. We will then construct a satisfying assignment for the formula.

First, note that no selection of S' can cause a carry to the next decimal place, since there are at most five 1's in each decimal place among the elements of S. Further, to get a 1 in each integer place, S' must contain either y_i or z_i, but not both, for each i. Now, we construct the assignment. If S' contains y_i, set $x_i = True$. If S' contains z_i, set $x_i = False$. This is a satisfying assignment because in the fractional places, at most two of the total 3 in each decimal place can come from g_i's and h_i's, so there must be a y_i or z_i with a 1 in each decimal place, which corresponds to a True literal in each clause. Thus, all clauses are True, and the expression is satisfied.

The size of S is approximately $(h + k)^2$, and calculation of each element of S takes polynomial time, so this is a poly-time reduction.

□