The Cook-Levin Theorem (Cook 1971, Levin 1973)

1. SAT is NP-complete
2. 3SAT is NP-complete

SAT was the first problem shown to be NP-complete. 3SAT was shown to be NP-complete by reducing SAT to 3SAT using a polynomial time reduction (i.e., $SAT \leq_p 3SAT$).

Subsequently Karp showed 21 other problems - from different domains (e.g., graph theory, math prog, etc.) NP-complete via reduction chains from 3SAT.

Preparation for the proof:

(1) The input to SAT is a boolean formula in Conjunctive Normal Form (CNF).

Example: $(\bar{x}_1 \vee x_2 \vee \bar{x}_5 \vee x_7) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_6 \vee \bar{x}_7 \vee x_8)$

This boolean formula contains 3 clauses and 8 variables $\{x_1, x_2, \ldots, x_8\}$. The clauses contain literals connected by $\vee$s and the clauses themselves are connected by $\wedge$s.

This instance of SAT is satisfiable if there exists a truth assignment to the variables $\{x_1, x_2, \ldots, x_8\}$ satisfying the boolean formula (alternately, satisfying all 3 boolean clauses).

[redacted]:

(2) Expressing equality of 2 binary strings $x = x_1 x_2 \ldots x_n$ and $y = y_1 y_2 \ldots y_n$.

It is easy to see that

$x_1 = y_1$ iff $(x_1 \vee \bar{y}_1) \wedge (\bar{x}_1 \vee y_1)$ is T

Therefore,

$x_1 = y_1, x_2 = y_2, \ldots, x_n = y_n$ iff

CNF → $(x_1 \vee \bar{y}_1) \wedge (\bar{x}_1 \vee y_1) \wedge (x_2 \vee \bar{y}_2) \wedge (\bar{x}_2 \vee y_2) \wedge \ldots$
$\ldots \wedge (x_n \vee \bar{y}_n) \wedge (\bar{x}_n \vee y_n)$.

For two boolean formula ϕ_1, ϕ_2, we will use $\phi_1 = \phi_2$ as a shorthand for $(\phi_1 \vee \bar{\phi}_2) \wedge (\bar{\phi}_1 \vee \phi_2)$.

(3) CLAIM: For any boolean function $f : \{0,1\}^l \to \{0,1\}$, there is an [redacted] l-variable boolean formula φ in CNF of size $l \cdot 2^l$ such that $\varphi(u) = f(u)$ $\forall u \in \{0,1\}^l$. Here, the size of φ is the number of $\vee$ and $\wedge$ symbols in φ.

PROOF: For any $v \in \{0,1\}^l$, there is a clause C_v on l variables such that $C_v(v) = 0$ and

$C_v(u) = 1 \quad \forall u \in \{0,1\}^\ell : u \neq v$. For example, if $v = \langle 1,0,0,1,1 \rangle$ then $C_v = \bar{u}_1 \vee u_2 \vee u_3 \vee \bar{u}_4 \vee \bar{u}_5$. Clearly, $C_v(v) = 0$ & $C_v(u) = 1$ for any $u \neq v$.

For any boolean function $f : \{0,1\}^\ell \to \{0,1\}$, let

$$\varphi = \bigwedge_{v : f(v)=0} C_v(z_1, z_2, \ldots, z_\ell)$$

(i.e., φ is the "and" of clauses C_v over all ℓ-vectors v that make f 0).

Then for all $u \in \{0,1\}^\ell : f(u) = 0$, φ contains the clause C_u. Since $C_u(u) = 0$, we get that $\varphi(u) = 0$.

For any $u \in \{0,1\}^\ell$; $f(u) \neq 0$, φ does not contain the clause C_u. Therefore, $\varphi(u) = 1$.

Thus, $\varphi(u) = f(u) \quad \forall u \in \{0,1\}^\ell$. Each clause has $\ell - 1$ $\vee$s and there are 2^ℓ clauses, at most. Hence, size of φ is at most $\ell \cdot 2^\ell$. □

PROOF OF THE COOK-LEVIN THEOREM:

To show that SAT $\in$ NP-complete we need to show that (i) SAT $\in$ NP & (ii) SAT $\in$ NP-hard. We have already seen that SAT $\in$ NP. We will show that

SAT $\in$ NP-hard, by showing that for all $L \in NP$, $L \leq_p$ SAT.

Consider an arbitrary $L \in NP$. We will show a polynomial-time computable function $f: \{0,1\}^* \to$ set of binary encodings of boolean formulae such that

$$x \in L \quad \text{iff} \quad f(x) \in SAT$$

In other words, f is a function that transforms arbitrary binary strings into boolean formulae such that if the binary string x is in L, then the boolean formula $f(x)$ is satisfiable & if the binary string $x \notin L$, then the boolean formula $f(x)$ is not satisfiable.

Only thing we can rely on

Since $L \in NP$, $\exists$ a (deterministic) poly-time TM M and a polynomial p such that:

$$x \in L \quad \text{iff} \quad \exists u \in \{0,1\}^{p(|x|)} : M(x,u) = 1.$$

One (incorrect) approach to constructing a boolean formula

Fix an input $x \in \{0,1\}^*$. For this input M can be viewed as a boolean function $g_x : \{0,1\}^{p(|x|)} \to \{0,1\}$. We can then rewrite the definition of $L \in NP$ as:

$$x \in L \quad \text{iff} \quad \exists u \in \{0,1\}^{p(|x|)} : g_x(u) = 1$$

We know from the earlier claim that corresponding to g_x there is a boolean formula (in CNF) φ_x. Therefore,

$$x \in L \text{ iff } \exists u \in \{0,1\}^{p(|x|)}.\ \varphi_x(u) \text{ is TRUE}$$

$$\Rightarrow x \in L \text{ iff } \varphi_x \text{ is satisfiable.}$$

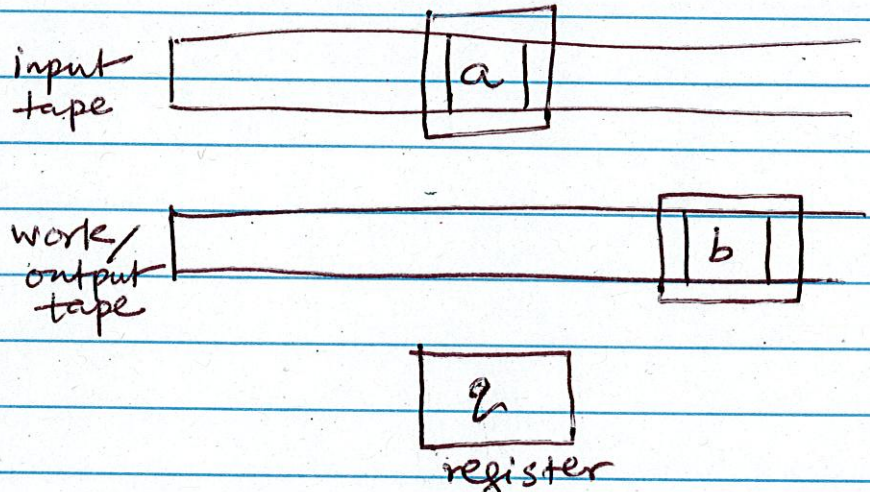
Thus we have transformed each instance $x \in \{0,1\}^*$ into a boolean CNF formula φ_x such that $x \in L$ iff φ_x is satisfiable. Are we done?? No, because φ_x is too large! The earlier claim told us that the size of φ_x was bounded above by $p(|x|) \cdot 2^{p(|x|)}$. Hence, what we have demonstrated is <u>not</u> a polynomial time reduction.

But we can construct a "short" boolean formula by examining the behavior of M more closely. Without loss of generality we assume that:

(i) M is a 2-tape TM - one input tape & one work/output tape.

(ii) M is an <u>oblivious</u> TM. This means that M takes exactly the same amount of time on all inputs of a particular length & for every i the location of M's heads at the i^{th} step depends only on i & the length of the input.

Denote by Q the set of M's possible states and by Γ the alphabet of M. The snapshot of M's execution on some input y, just before step i is the triple $(a, b, q) \in \Gamma \times \Gamma \times Q$ where a = symbol read by M's read head, b = symbol read by M's read/write head, q = M's state just before the i^{th} step. The snapshot can be encoded as a binary string of length c, where c depends only on $|Q|$ & $|\Gamma|$. (and not on the input length). (So c is a constant)



Snapshot (a, b, q)

We will use z_i to denote the snapshot just before step i.

Key idea

It is clear that z_i is a function (which depends on M's transition function) of $z_1, z_2, \ldots, z_{i-1}$. In other words, if you gave me $z_1, z_2, \ldots, z_{i-1}$ & z_i & M's transition table, I can determine

if z_i is "correct" or not. However, it turns out that z_i just depends on two snapshots from earlier time periods.

To check if z_i is "correct" we only need to know:

(a) z_{i-1} : snapshot just before step $i-1$.

Using this snapshot we can figure out what state M should be in z_i.

(b) $y_{inputpos(i)}$: y is the input $\langle x, u \rangle$ and inputpos(i) is the position of M's read head just before step i.

This is the input symbol that M should be reading just before step i.

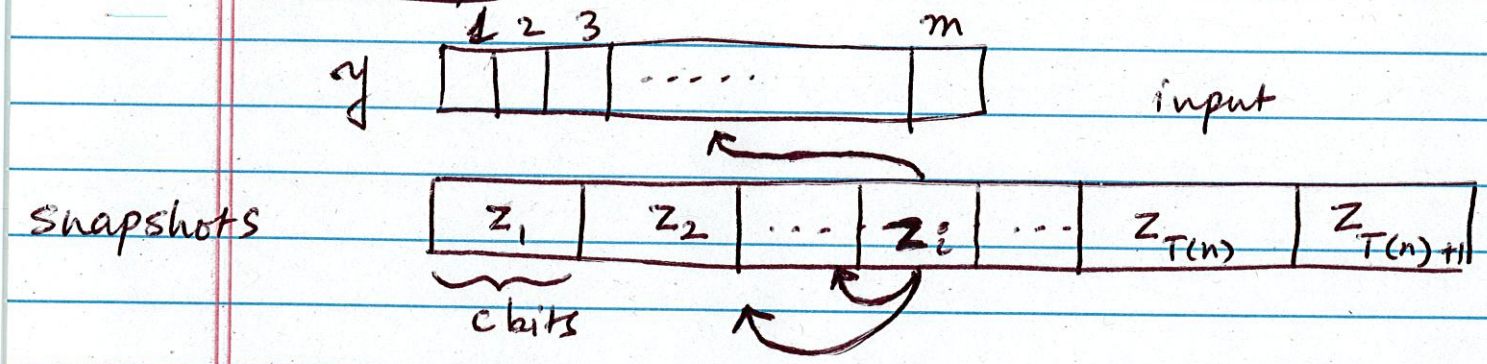
(c) $z_{prev(i)}$: prev(i) is the last step before i that M's read/write head was on the same cell as M's read/write head is currently on.

Thus $z_{prev(i)}$ is the most recent snapshot (prior to the current one) that may cause M to write to the cell that M's read/write head is currently on. Using M's transition function we can figure out the symbol that should be in the cell that M's read/write head is currently on.

Thus, there is a function $F : \{0,1\}^{2c+1} \to \{0,1\}^c$ that maps $(z_{i-1}, y_{inputpos(i)}, z_{prev(i)})$ to the "correct" snapshot just before step i.

Note: z_{i-1}'s encoding needs c bits
$z_{prev(i)}$'s " " " " } $2c+1$ bits
$y_{inputpos(i)}$ is 1 bit

Pictorially



Thus given string $x \in \{0,1\}^*$ we transform it into a boolean formula φ_x that encodes the following conditions:

(i) z_1 should encode the snapshot $(\triangleright, \triangleright, q_{start})$

(ii) For each $i \in \{2, 3, \ldots, T(n)+1\}$,
$$z_i = F(z_{i-1}, y_{inputpos(i)}, z_{prev(i)}).$$

a snapshot that is in an accepting state.

(iii) $z_{T(n)+1}$ should be ~~such that from this snapshot~~ ~~M transitions to an accepting state.~~

(iv) The first n bits of y equal x.

The last condition is critical in guaranteeing that we get a boolean formula φ_x that depends on x.

Because of the construction it should be clear that

$$\exists u \in \{0,1\}^{p(|x|)} : \varphi_x(u) = 1 \text{ iff}$$

$$\exists u \in \{0,1\}^{p(|x|)} : M(x,u) = 1.$$

We will now estimate the size of φ_x and the time it takes to construct φ_x. This is important to ensuring that f is a poly-time computable function.

Condition (i) can be expressed as a boolean formula of size at most $4(2c+1)$ using the "equality" construction.

Condition (ii) can be expressed, for each i, as a boolean formula of size at most $(2c+1)\cdot(2c+1)\cdot 2^{(2c+1)}$. Hence, this is a boolean formula of size $(2c+1)^2 \cdot 2^{(2c+1)} \cdot T(n)$.

Condition (iii) can be expressed as a boolean formula of size at most $4(2c+1)$.

Condition (iv) can be expressed as a boolean formula of size $4n$.

The size of φ_x is therefore $O(T(n)+n)$ and the time to construct φ_x is also $O(T(n)+n)$. Since $T(n) = \text{poly}(n)$, we have shown that the transformation f is computable in polynomial time. □

Theorem: $SAT \leq_p 3SAT$

PROOF: Suppose we are given an instance φ of SAT. Consider a clause C in φ with $k > 3$ literals in it.

Example: $C = x_1 \vee \bar{x}_3 \vee x_5 \vee \bar{x}_6 \vee x_9$

C can be replaced by

$C' = (x_1 \vee \bar{x}_3 \vee x_5 \vee z) \wedge (\bar{x}_6 \vee x_9 \vee \bar{z})$

If C is true then C' is true.

If C is false then one of the two clauses in C' is false & hence C' is false.

Then C can be replaced by 2 clauses, one of size $(k-1)$ and one of size 3. Continuing in this manner on all clauses of size >3 we see that by adding at most $m \cdot n$ clauses, we can reduce all clauses to size ≤ 3.

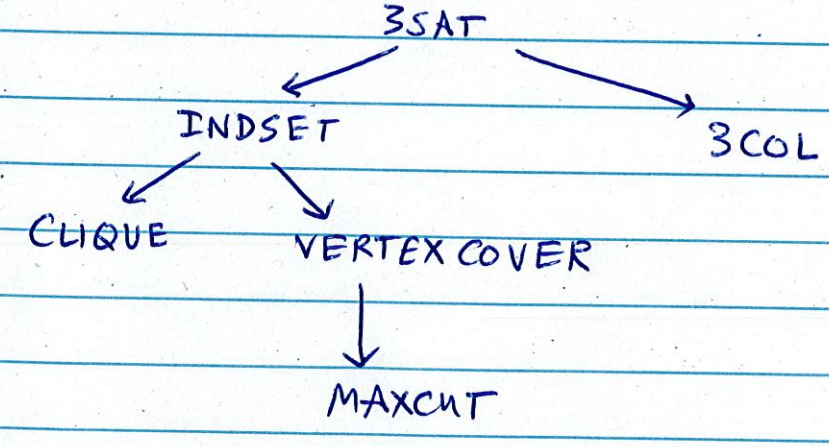
Your textbook defines 3SAT as having instances with CNF every clause having ≤ 3 literals. Most definitions I have seen (e.g., CLRS - Introduction to Algorithms) defines 3SAT as having clauses with exactly 3 literals per clause. Depending on our definition we are either done with the proof or have a little more work to do. Specifically, if we require that every clause has exactly 3 literals, then we have to convert clauses with one or two literals into clauses with 3 literals by adding dummy variables. □

SAT/

In praise of 3SAT :

SAT/

Not only is 3SAT the root of an enormous tree of poly-time reductions & etc.,



but lot of problems are directly reduced from

SAT/3SAT. It comes with minimal combinatorial structure and is therefore quite flexible. When trying to show that a problem is NP-hard first look for NP-hard problems similar to reduce from. If this fails try a reduction from SAT/3SAT.

Variants of SAT/3SAT that are useful for reductions:

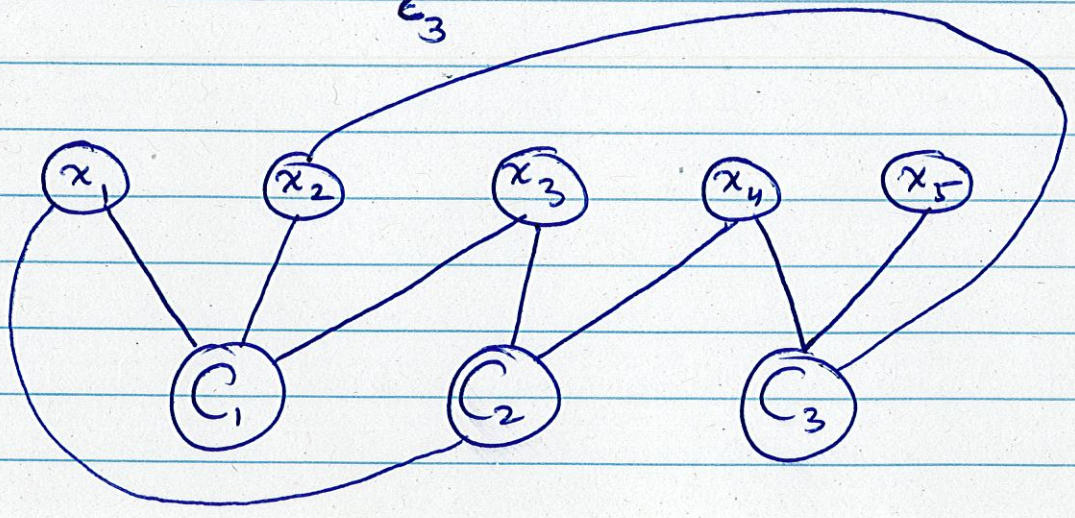
PLANAR-3SAT

INPUT: A boolean formula φ in CNF with clauses containing exactly 3 literals. Furthermore, it is required that the clause-variable graph be planar.

QUESTION: Is the given boolean formula satisfiable?

Example:

$$\varphi = \overbrace{(x_1 \vee \bar{x}_2 \vee x_3)}^{C_1} \wedge \overbrace{(x_1 \vee \bar{x}_3 \vee x_4)}^{C_2} \vee \overbrace{(\bar{x}_2 \vee \bar{x}_4 \vee x_5)}^{C_3}$$



One can show that 3SAT $\leq_p$ PLANAR-3SAT and hence PLANAR-3SAT $\in$ NP-hard. Clearly, PLANAR-3SAT $\in$ NP & hence PLANAR-3SAT $\in$ NP-complete.

Various problems in computational geometry can be shown to be NP-complete via PLANAR-3SAT.

(2) NAE-SAT / NAE-3SAT
("Not all equal"-SAT & "Not all equal"-3SAT):
NAE-SAT
INPUT: A boolean formula in CNF
QUESTION: Is there a satisfying assignment to the boolean formula in which no clause has all of its literals set to TRUE?

Closing remarks (for Chapter 2)

- Computability vs Complexity Theory

Turing, Church, Gödel are all usually associated with Computability theory - whether or not a function can be computed.

Gödel wrote a letter to von Neumann (1956) in which he essentially asked about the polynomial solvability of a problem related to

SAT (that is NP-complete). Gödel's "lost letter" was discovered in the 70's. Gödel was thinking about complexity theory well before Cook, Levin, Karp, & Edmonds.

(2) NP = coNP ?

Most researchers believe that NP ≠ coNP. In other words, there are problems that have "short refutations," but not "short certificates."

Example: $\overline{3SAT}$ (also called $\overline{UNSAT}$):

INPUT: A boolean formula φ in CNF with 3 literals per clause.

QUESTION: Is there no satisfying truth assignment to φ?

$\overline{3SAT}$ is clearly in coNP because SAT ∈ NP. But it is generally believed that $\overline{3SAT}$ does not have a short certificate.

(3) Are there problems in NP that are not NP-complete?

Most likely, GRAPH-ISOMORPHISM, FACTORING etc. are likely candidates.

UNIQUE GAMES

LABELING problem

FINDING NASH EQUI in 2-pers

Next chapter:

Ladner's Theorem (1975)

If $P \neq NP$ then there exists languages in $NP \setminus P$ that are not NP-complete.

(Called "NP intermediate" languages)

(4) Coping with NP-hardness

- Better modeling of problems.
- Approximation Algorithms
- Average case complexity
- Heuristics

(5) Finer explorations of time complexity

OK, problems such as INDSET are unlikely to have poly-time algorithms. But could these problems have subexponential-time algorithms?

n^c, const. $c \geq 1$, $2^{(\log n)^2}$, $2^{\sqrt{n}}$, 2^{n^c}, const. $c >$

polynomial — neither poly-, nor expo. — exponential

Could INDSET live here? ↑

Most researchers believe that the fastest algo. for INDSET will take time $2^{\Omega(n)}$. Such a conjecture is of course stronger than the $P \neq NP$ conjecture.

THE END