NP and NP-completeness

$L \in NP$ if exists $M_L(x,y)$ such that
a) M_L runs in polynomial time
b) $x \in L \Leftrightarrow$ Exists y, $|y| = poly(|x|)$, such that $M_L(x,y) = 1$

L is **NP-HARD** if for all $L' \in NP$ it holds that $L' \leq_p L$

L is **NP-COMPLETE** if
a) $L \in NP$
b) L is NP-hard

COOK-LEVIN THEOREM
SAT and 3-SAT are NP-complete

First proof idea

(i) Define Boolean function

$$f_x^L(y) = \begin{cases} 1 & \text{if } M_L(x,y) = 1 \\ 0 & \text{otherwise} \end{cases}$$

(ii) Represent f_x^L as CNF formula

This formula has all the right properties, except it is unfortunately exponentially large

Define instead SNAPSHOTS $z_i = \langle a_i, b_i, q_i \rangle$ of Turing machine computation

- a_i read from input tape
- b_i read from work tape
- q_i Turing machine state

} at time i

Can check correctness of z_i locally for

- z_{i-1} for q_i
- inputbits(i) for a_i
- $z_{prev(i)}$ for b_i

Consistency check (given x and M_L) is Boolean function of constant number of bits

Write CNF Formula F_x with variables for

$$y, z_1, z_2, \ldots, z_T$$

enforcing

a) starting state z_1 correct
b) z_i is locally correct
c) z_T is final state of accepting computation

Assignment to variables encode

- witness y
- snapshots of accepting computation for $M_L(x, y)$

Possible if and only if $x \in L$

Two observations

① If M_L runs in time $T(|x|)$, then formula F_x (and time to compute it) can be made very small $O(T \log T)$

② From satisfying assignment to F_x, can read off witness y for x

This is called a LEVIN REDUCTION

> To prove language L NP-complete
> (i) Show $L \in NP$ (usually easy)
> (ii) Reduce from SAT, 3-SAT or other NP-complete language to L

Will see some such reductions
But first some more definitions

DEFINITION (COMPLEMENT CLASS)
For $L \subseteq \Sigma^*$, the COMPLEMENT LANGUAGE of L is $\overline{L} = \Sigma^* \setminus L$

DEFINITION (coNP)

$$coNP = \{ L \mid \overline{L} \in NP \}$$

Technical aside: Usually exclude syntactically incorrect inputs from both L and $\overline{L}$
(Does not really matter — why?)

Note that coNP is not the complement of NP
The intersection is non-empty!

$$P \subseteq NP \cap coNP$$

TAUTOLOGY is in coNP — if F is not a tautology, then this is witnessed by an assignment falsifying the formula

UNSAT $= \{ F \mid F$ is an unsatisfiable CNF formula $\}$
is also in coNP

Both of these languages are coNP-complete — any other language L in coNP can be efficiently reduced to them

Proof sketch (for UNSAT) Given $L \in coNP$, run Cook-Levin reduction on $\bar{L}$
$x \in L \Leftrightarrow x \notin \bar{L} \Leftrightarrow F_x$ unsatisfiable

How is coNP related to NP?
Could there be short certificates for UNSAT that somehow "compress information about exponentially many failing assignments"?

Most researchers believe

$$NP \neq coNP$$

but this is a wide-open problem

THEOREM | 0-1 LINEAR PROGRAMMING IS NP-complete

Proof: Containment in NP easy - just plug in solution and check

NP-hardness: Reduce from SAT

Translate clauses to linear inequalities

$$x \mapsto x$$

$$\bar{x} \mapsto 1 - x$$

$$x \vee \bar{y} \vee z \mapsto x + (1-y) + z \geq 1$$

$$\text{or} \quad x - y + z \geq 0$$

linear inequality satisfied precisely when clause satisfied

clearly polynomial-time reduction

Interesting counterfactual question: What would complexity theory have looked like if the first NP-complete problem would have been 0-1 linear programming rather than Boolean satisfiability?

$$3\text{-COLOURING} = \{G \mid \text{Undirected graph } G \text{ has 3-colouring}\}$$

3-colouring: Colour all vertices with 3 colours

$$\chi : V \to \{\text{red}, \text{blue}, \text{green}\}$$

Colouring χ valid if for every edge $(u,v) \in E(G)$ it holds that $\chi(u) \neq \chi(v)$

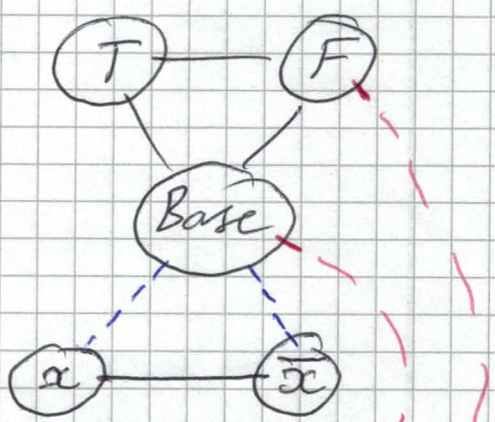
THEOREM: 3-COLOURING is NP-complete

Proof: Let witness be colouring

Containment in NP – obvious

NP-hardness: Reduce from 3-SAT

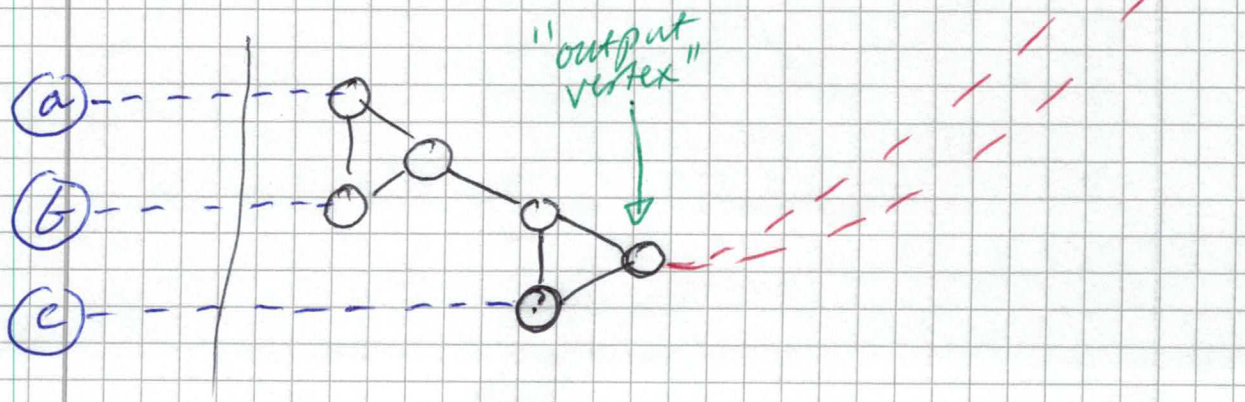
Gadgets: (1) Assignment triangle



(2) For every variable x

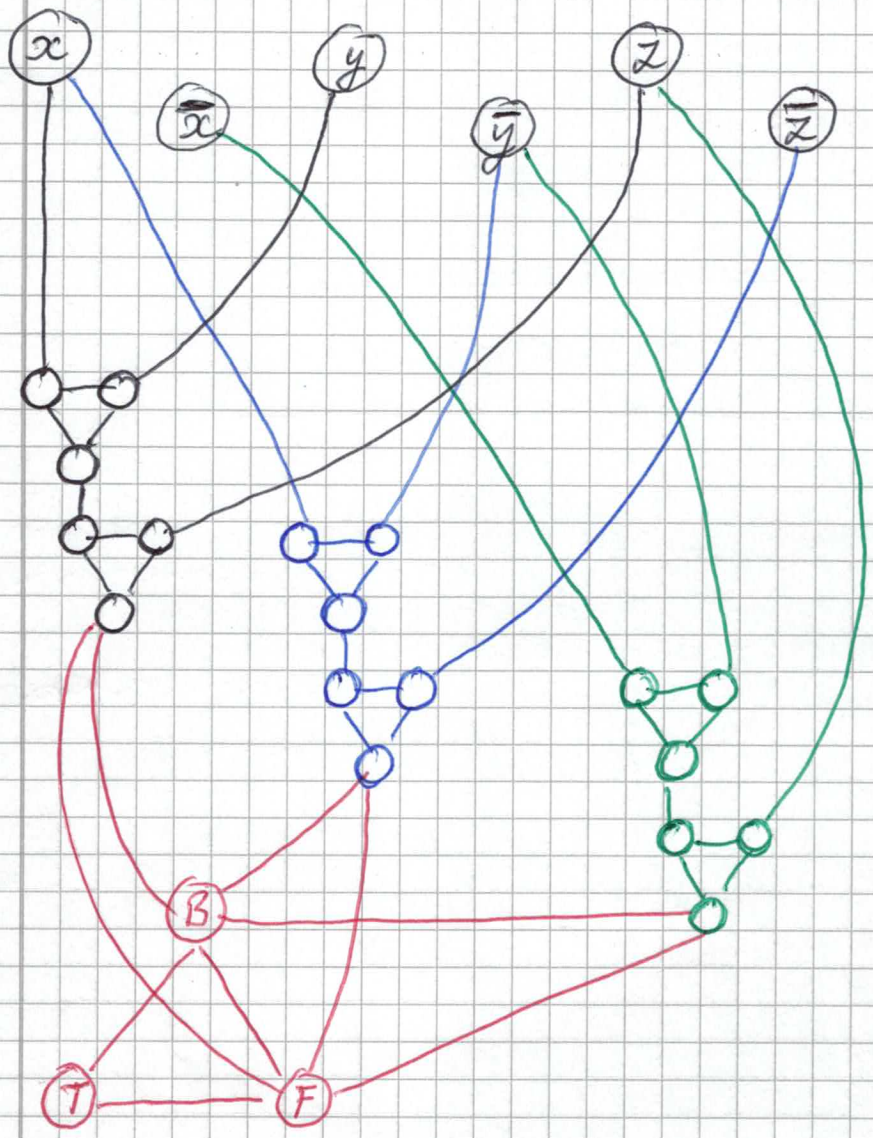
(3) For every clause

$$a \vee b \vee c$$



Example

$$\left.\begin{array}{l}(x \vee y \vee z) \\ \wedge\ (x \vee \bar{y} \vee \bar{z}) \\ \wedge\ (\bar{x} \vee \bar{y} \vee z)\end{array}\right\} F$$



G_F

Need to prove

($\Rightarrow$) F satisfiable $\Rightarrow$ G_F 3-colourable

($\Leftarrow$) G_F 3-colourable $\Rightarrow$ F satisfiable

(It is clear that G_F can be constructed from F in polynomial time.)

($\Rightarrow$) Let α be satisfying assignment for F

Colours (T) - green
(F) - red
(Base) - blue

true literals - green
false literals - red

If a clause contains a true literal, then "output vertex" can be coloured green / true

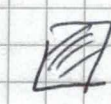
($\Leftarrow$) Given colouring of G_F
Suppose w.l.o.g (by permuting colours if necessary that (T) - green
(F) - red
(Base) - blue

Note that no literal is coloured blue because of variable gadget.

For literal l, define $\alpha(l) = \begin{cases} 1 & \text{if } \chi(l) = \text{green} \\ 0 & \text{if } \chi(l) = \text{red} \end{cases}$

Defines truth value assignment in view of variable gadget

Every output vertex in clause gadget coloured true / green. Only possible if some literal in clause coloured true — hence α satisfies all clauses



$$\text{CLIQUE} = \{ \langle G, k \rangle \mid G \text{ is an undirected graph containing a } k\text{-clique} \}$$

A **k-clique** in G is a set of k pairwise connected vertices $v_1, v_2, \ldots, v_k \in V(G)$
That is, all edges (v_i, v_j), $1 \le i < j \le k$ are in $E(G)$

THEOREM CLIQUE is NP-complete

We will show
(i) CLIQUE $\in$ NP
(ii) 3-SAT $\le_p$ CLIQUE (CLIQUE is NP-hard)

For (i), let witness be the k vertices in a clique
Verifier checks that all edges between these vertices are in graph.
Can clearly be done in polynomial time on reasonable computer with reasonable programming language (which can be efficiently simulated by Turing machine).

For (ii), given a 3-CNF formula F, we need to construct a graph G_F and choose a parameter k_F such that

F satisfiable $\Leftrightarrow$ G_F has k_F-clique

And construction should be computable in polynomial time (in size of F)

Construction of G_F from $F = \bigwedge_{i=1}^{m} C_i$

For every clause $C_r = l_{r,1} \vee l_{r,2} \vee l_{r,3}$
create 3 vertices $v_{r,1}, v_{r,2}, v_{r,3}$

Add edge $(v_{r,i}, v_{s,j})$ iff

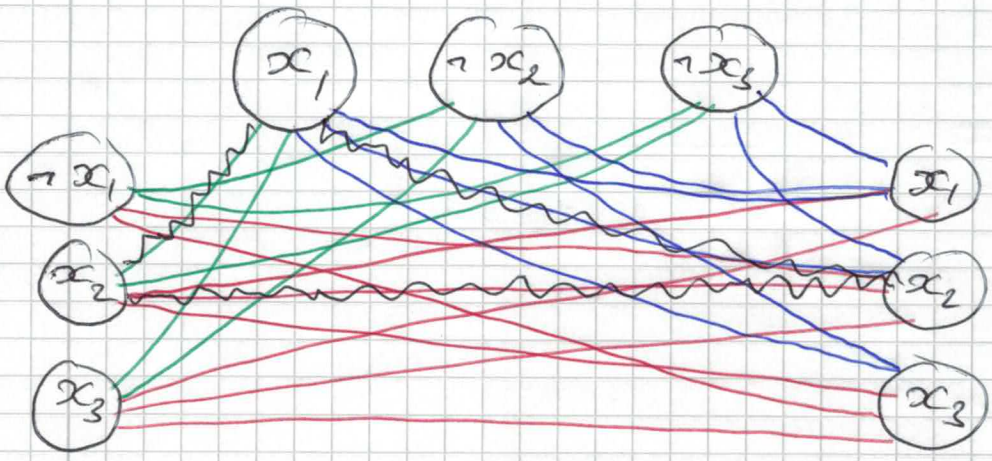
a) $r \neq s$ (vertices come from different clauses)

b) $l_{r,i}$ and $l_{s,j}$ are <u>not</u> negations of each other

Set $k_F = m$ = # clauses in F

<u>Example</u>

$$F = (x_1 \vee \neg x_2 \vee \neg x_3) \quad C_1$$
$$\wedge (\neg x_1 \vee x_2 \vee x_3) \quad C_2$$
$$\wedge (x_1 \vee x_2 \vee x_3) \quad C_3$$



Clearly possible to build G_F & k_F from F in polynomial time

Need to show

($\Rightarrow$) F satisfiable $\Rightarrow$ G_F has k_F-clique

($\Leftarrow$) F satisfiable $\Leftarrow$ G_F has k_F-clique

($\Rightarrow$) Let α be satisfying assignment
α satisfies at least one literal $l_{r,i}$ per clause C_r
Pick one such literal per clause.
Since literals are all true, none is negation of other, so clique
One vertex per clause $\Rightarrow$ k_F - clique

<u>Example</u> $\alpha = \{x_1 \mapsto 1,\ x_2 \mapsto 1,\ x_3 \mapsto 0\}$

Pick x_1 from C_1, x_2 from C_2, x_2 from C_3

($\Leftarrow$) Suppose G_F has k_F - clique.
Then at least one (precisely one) vertex $v_{r,i}$ per clause C_r
All literals $l_{r,i}$ can be assigned true, since there are edges between them.
Yields partial truth value assignment satisfying all clauses.
(Assign any remaining variables arbitrarily)

Vertex cover of $G=(V,E)$ is subset $V' \subseteq V$ such that every edge $(u,v) \in E$ has an endpoint in V' (i.e., $\{u,v\} \cap V' \neq \emptyset$)

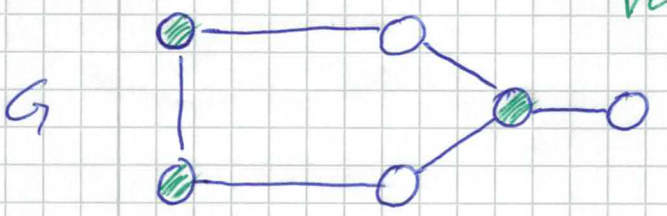
$$\text{VERTEXCOVER} = \{\langle G, k\rangle \mid G \text{ has vertex cover of size } k\}$$

THEOREM VERTEXCOVER is NP-complete

We will show
(i) VERTEXCOVER $\in$ NP (Already known to be NP-complete, so can reduce from it!)
(ii) CLIQUE $\leq_p$ VERTEXCOVER (VERTEXCOVER is NP-hard)

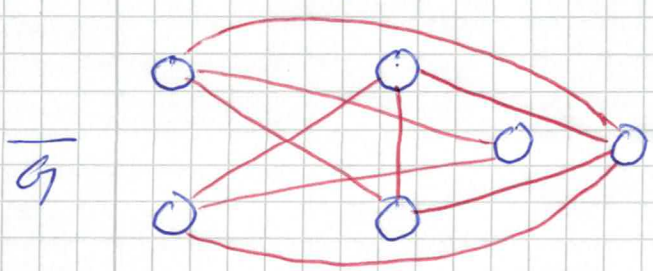
For (i), suitable witness consists of $V' = (v_1, v_2, \ldots, v_k)$
Verifier checks for all edges (u,v) that $\{u,v\} \cap \{v_1, \ldots, v_k\} \neq \emptyset$
Can clearly be done in polynomial time

Example



Vertex cover of size 3

For (ii), translate $\langle G, k\rangle$ to $\langle \overline{G}, |V| - k\rangle$ where $\overline{G}$ is complement graph with same vertex set but edges $(V \times V) \setminus E$
(u,v) is edge in $\overline{G}$ $\Leftrightarrow$ (u,v) is not edge in G



$\overline{G}$ can clearly be constructed from G in polynomial time

Need to show:

$\langle G, k \rangle \in$ CLIQUE $\iff \langle \overline{G}, |V| - k \rangle \in$ VERTEXCOVER

($\Rightarrow$) Suppose $C \subseteq V$ is clique in G of size $|C| = k$

$u, v \in C, u \neq v \Rightarrow (u, v)$ edge in G

$\Rightarrow (u, v)$ <u>not</u> edge in $\overline{G}$

So C contains <u>no edges</u> in $\overline{G}$

$\Rightarrow$ all edges in $\overline{G}$ have an endpoint in $V \setminus C$, which is a vertex cover of size $|V| - k$.

($\Leftarrow$) Suppose C^* vertex cover in $\overline{G}$

Then $V \setminus C^*$ contains no edges in $\overline{G}$

Means that <u>all edges</u> in $V \setminus C^*$ are present in G

Hence if C^* vertex cover in $\overline{G}$ of size $|V| - k$, then $V \setminus C^*$ clique in G of size

$|V| - (|V| - k) = k$.

Subset sum

(think of subset S' as indices of numbers)

Given positive integers $A_1, A_2, \ldots, A_n, T$

Is there subset $S' \subseteq \{1, 2, \ldots, n\}$ such that

$$\sum_{i \in S'} A_i = T \quad ?$$

$$\text{SUBSETSUM} = \left\{ \langle A_1, \ldots, A_n, T \rangle \;\middle|\; \exists S' \text{ s.t. } \sum_{i \in S'} A_i = T \right\}$$

THEOREM SUBSETSUM is NP-complete.

We will show

(i) SUBSETSUM $\in$ NP

(ii) 3-SAT $\leq_p$ SUBSETSUM (SUBSETSUM is NP-hard)

For (i), let witness be set of indices S'

Verifier computes $\sum_{i \in S'} A_i$ and checks if this sum equals T

Can clearly be done in polynomial time

For (ii), suppose we have 3-CNF formula $F = \bigwedge_{i=1}^{m} C_i$ over variables $x_1, \ldots, x_n$

Create subset sum instance

$$\{ A_1^T, A_1^F, A_2^T, A_2^F, \ldots, A_n^T, A_n^F, B_1^1, B_1^2, \ldots, B_m^1, B_m^2 \}$$

and T as follows

- Each number has $n+m$ decimal digits
- n most significant digits associated with $x_1, \ldots, x_n$
- m least significant digits associated with $C_1, \ldots, C_m$

Assume W.L.O.G.

- No clause contains both a variable and its negation (if so, remove this clause)
- All variable appear in F (remove & renumber otherwise)

Construction of numbers

A_i^T: digit x_i is 1
digit C_j is 1 if $x_i \in C_j$
all other digits 0

A_i^F: digit x_i is 1
digit C_j is 1 if $\neg x_i \in C_j$
all other digits 0

B_j^1 digit C_j is 1
all other digits 0

B_j^2 digit C_j is 2
all other digits 0

T all digits x_i are 1
all digits C_j are 4

Clearly possible to construct such a subset sum instance in polynomial time in the size of F

($\Rightarrow$)
Suppose F satisfied by assignment α
- If $\alpha(x_i) = 1$ pick A_i^T, otherwise A_i^F
- If α satisfies
  - 1 literal in C_j — pick B_j^1 and B_j^2
  - 2 literals in C_j — pick B_j^2
  - 3 literals in C_j — pick B_j^1

This sums to T

($\Leftarrow$)
Given solution S' to subset sum instance
Let α set $x_i = 1$ iff A_i^T chosen in S'
Why is this a satisfying assignment?

Example

$$F = (x_1 \vee \neg x_2 \vee \neg x_3) \quad C_1$$
$$\wedge\ (\neg x_1 \vee x_2 \vee x_3) \quad C_2$$
$$\wedge\ (x_1 \vee x_2 \vee x_3) \quad C_3$$

Solution corresponding to $x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 0$

$n = 3$

$m = 3$

$A_1^T = 100101$ *

$A_1^F = 100010$

$A_2^T = 0\,10011$ *

$A_2^F = 010100$

$A_3^T = 001011$

$A_3^F = 001100$ *

First n digits of T enforce that exactly one of x_i and $\neg x_i$ chosen

For last m digits, can only reach 4 if at least one literal in clause is chosen to be true.

$B_1^1 = 000100$

$B_1^2 = 000200$ * C_1

$B_2^1 = 000010$ * C_2

$B_2^2 = 000020$ * C_2

$B_3^1 = 000001$

$B_3^2 = 000002$ * C_3

$T = 111444$

DECISION VS. SEARCH

THEOREM

If $P = NP$, then for every $L \in NP$ we can find a polynomial-time Turing machine M that finds certificates for $x \in L$ (i.e., "solves x")

Proof sketch

Start with SAT.

Given CNF formula $F(x_1, \ldots, x_n)$

set $x_1 = 1$, and simplify F to

$F|_{x=1}$ - remove satisfied clauses

- then remove falsified literals

Check if $F|_{x=1}$ is satisfiable; if so set $b = 1$

Else set $x_1 = 0$ and simplify F to $F|_{x=0}$

Check if $F|_{x=0}$ satisfiable; if so set $b = 0$

otherwise report UNSATISFIABLE

Remember $x_1 = b$ and make recursive call on $F|_{x_1=b}$

Return satisfying assignment to $x_2, \ldots, x_n$ for $F|_{x_1=b}$ plus $x_1 = b$

For general $L \in NP$, run reduction $x \mapsto F_x$ in proof of Cook-Levin

Find satisfying assignment to F_x

Read off witness y for x from this satisfying assignment. ☐

SAT is **DOWNWARD SELF-REDUCIBLE**

Given efficient algorithm that solves SAT on input size $< n$ can solve also instances of size n

All NP-complete problems/languages have this property (follows from Cook-Levin)

NONDETERMINISTIC EXPONENTIAL TIME AND PADDING

DEFINITION $\boxed{NEXP = \bigcup_{k \in \mathbb{N}} NTIME(2^{n^k})}$

Clearly $P \subseteq NP \subseteq EXP \subseteq NEXP$

Why care about such classes?

THEOREM $\boxed{\text{If } EXP \neq NEXP, \text{ then } P \neq NP}$

Proof By contraposition. Suppose $P = NP$;
prove $EXP = NEXP$

Consider any language $L \in NTIME(2^{n^k})$

There is a nondeterministic TM M deciding L.

Let $L_{pad} = \{ \langle x, 1^{2^{|x|^k}} \rangle \mid x \in L \}$

Claim: $L_{pad} \in NP$

- First check input is on correct form
- Then run M

This can be done in polynomial time in the size of the input, since we have padded the input to be exponentially large

By assumption, $L_{pad} \in NP \Rightarrow L_{pad} \in P$, decided by M^*, say.

But then $L \in EXP$

- Given x, construct $\langle x, 1^{2^{|x|^k}} \rangle$.
- Then run M^* on this input, and answer as M^* does

Runs in exponential time ∎

PADDING: Useful technique to "scale up" or "scale down" results between weaker and stronger complexity classes.

> What does all of this mean? Section 2.7 in Arora-Barak is highly recommended reading

SO FAR: Focus on P, NP, and NP-completeness

You should have seen most of this before (though perhaps not presented in this way and in this level of detail)

GOING FORWARD

New material (most likely)

- Can we separate complexity classes? Is P distinct from EXP?
- Can we say anything about the landscape between P and NP (if $P \neq NP$)