

Recap: Polynomial Time, Nondeterminism, and NP-completeness

Computational Complexity 2026/2027 — Massimo Lauria

We recall deterministic time classes, define nondeterministic machines and the classes NP and coNP, and set up the theory of NP-completeness up to the Cook–Levin theorem.

READING. [AB, Chapter 1, Sections 2.1, 2.2, 2.3 (skip 2.3.4), 2.6, 3.1, 6.1, 6.5]. Everything in these notes is in [AB]; the pointers tell you where.

Summary of complexity classes

$$L \subseteq NL = \text{coNL} \subseteq P \begin{matrix} \subseteq \text{NP} \\ \subseteq \text{coNP} \end{matrix} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXP} \begin{matrix} \subseteq \text{NEXP} \\ \subseteq \text{coNEXP} \end{matrix}$$

Every inclusion is believed to be strict, but the only separations we can prove are those forced by hierarchy theorems: $L \subsetneq \text{PSPACE}$, $P \subsetneq \text{EXP}$ and $\text{NP} \subsetneq \text{NEXP}$. In particular at least one inclusion in the chain $P \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXP}$ is strict, and we do not know which. The two equalities are Savitch’s theorem and Immerman–Szelepcsényi [AB, Thm. 4.14, Thm. 4.20]; we prove them in the space complexity lecture.

Time

DEFINITION 1 (DTIME, P, EXP). For $T : \mathbb{N} \rightarrow \mathbb{N}$, a language L is in $\text{DTIME}(T(n))$ if there is a Turing machine deciding L that halts within $c \cdot T(|x|)$ steps on every input x , for some constant c . Then

$$P = \bigcup_{c \geq 1} \text{DTIME}(n^c), \quad \text{EXP} = \bigcup_{c \geq 1} \text{DTIME}(2^{n^c}).$$

The constant c makes the definition robust: linear speed-up and the simulation overhead between machine models (number of tapes, alphabet size, random access) is polynomial, so P and EXP do not depend on the model.

EXAMPLES OF PROBLEMS IN P : reachability in a graph, shortest paths, maximum matching, greatest common divisor, minimum spanning tree, 2-SAT, linear programming over the rationals, and primality testing. The last two are non-obvious.

PROBLEMS IN EXP and not known to be in P : SAT and essentially every NP-complete problem (brute force over 2^n candidates), TQBF, and generalised board games such as chess or go on an $n \times n$ board, which are genuinely EXP-complete and hence provably outside P .

THEOREM 2 (Time hierarchy). If f, g are time-constructible and $f(n) \log f(n) = o(g(n))$, then $\text{DTIME}(f(n)) \subsetneq \text{DTIME}(g(n))$.

[AB, Section 1.6 and Section 2.6].

This robustness is the real reason to use P as the proxy for “efficient”, not the belief that an n^{100} algorithm is practical. See the discussion in [AB, Section 1.6.1].

Linear programming: Khachiyan’s ellipsoid method (1979), then Karmarkar (1984). Primality: Agrawal, Kayal and Saxena, *PRIMES is in P*, Annals of Math. 160 (2004).

Fraenkel and Lichtenstein, *Computing a perfect strategy for $n \times n$ chess requires time exponential in n* , JCTA 31 (1981).

[AB, Thm. 3.1]. Proved by diagonalisation, requires the notion of a universal Turing machine; the log factor is the cost of simulating a machine with a universal machine.

COROLLARY 3. $P \subsetneq EXP$.

Proof. Take $f(n) = n^{\log n}$ and $g(n) = 2^n$: then $P \subseteq DTIME(f) \subsetneq DTIME(g) \subseteq EXP$. $\square$

So we *can* prove unconditional lower bounds, as long as the gap in resources is large enough. All the hard open problems concern separations between classes that are “close” in resources, or that differ in the *kind* of resource rather than the amount.

Non deterministic Turing Machines (NDTM)

DEFINITION 4 (Nondeterministic Turing machine). A nondeterministic Turing machine is a Turing machine with two transition functions δ_0, δ_1 and distinguished states q_{accept} and q_{reject} . At each step the machine applies either δ_0 or δ_1 ; a *computation branch* is a choice of one of the two at every step. The machine *accepts* x if *some* branch reaches q_{accept} , and *rejects* x if *no* branch does. It runs in time $T(n)$ if every branch halts within $T(|x|)$ steps on every input x of length n .

[AB, Section 2.1.2].

If a (deterministic) Turing Machine transition from a state to the next, in a NDTM execution every step causes the computation to branch into two different paths. After t steps there are up to 2^t branches.

Note the asymmetry between accepting and rejecting: acceptance is an existential statement about branches, rejection a universal one. A single accepting branch is a *proof* that the input is in the language. A rejecting path does not prove anything. This single asymmetry is the source of everything in this lecture, including the fact that NP and coNP look nothing alike.

DEFINITION 5 (NTIME, NP, coNP). $L \in \text{NTIME}(T(n))$ if some NDTM decides L in time $c \cdot T(n)$. Then $\text{NP} = \bigcup_c \text{NTIME}(n^c)$ and $\text{NEXP} = \bigcup_c \text{NTIME}(2^{n^c})$, and $\text{coNP} = \{L : \bar{L} \in \text{NP}\}$.

[AB, Section 2.1.2 and Section 2.6].

DEFINITION 6 (NP via verifiers). $L \in \text{NP}$ if there are a polynomial p and a polynomial-time machine R (the *verifier*) such that for every $x \in \{0, 1\}^*$

$$x \in L \iff \exists w \in \{0, 1\}^{p(|x|)} R(x, w) = 1.$$

Such a w is a *witness* (or certificate, or proof) for x .

Since $|w|$ is required to be polynomial in $|x|$, asking R to run in time polynomial in $|x|$ or in $|x| + |w|$ is the same thing.

[AB, Def. 2.1]. This is the definition you should have in mind when you want to *show* that a problem is in NP.

EXERCISE 7. Prove that Definition 4 and Definition 6 describe the same class. One direction records the branch taken at each step as the witness; the other guesses the witness bit by bit and then runs R .

NP IS THE CLASS OF PUZZLES. A sudoku is hard to solve and trivial to check; that is exactly the shape of Definition 6. Three things are worth separating:

- witnesses are *short* — polynomial in the input;
- witnesses are *easy to check* — polynomial time;
- witnesses may be *very hard to find* — nothing in the definition says otherwise.

TAKE AWAY. $x \in L$ has a short easy-to-check proof; $x \notin L$ need not have one. Whether the second asymmetry can be removed is exactly the question $NP \stackrel{?}{=} coNP$, which is the question of whether every propositional tautology has a short proof. Or more generally whether an existential and a universal statement about polynomial time computation are of fundamentally different type.

Cook and Reckhow, *The relative efficiency of propositional proof systems*, JSL 44 (1979). We take this up in the proof complexity part of the course, [EB, Chapter 1].

Some problems in NP

Independent Set Given G and k , is there a set of k pairwise non-adjacent vertices? Witness: the set.

3-Coloring Given G , can the vertices be coloured with 3 colours with no monochromatic edge? Witness: the colouring.

SAT Given a CNF formula, is it satisfiable? Witness: the assignment.

3-SAT SAT restricted to clauses of width at most 3.

2-SAT SAT with clauses of width at most 2. In P: a 2-clause is an implication, and the formula is unsatisfiable iff some variable and its negation lie in the same strongly connected component of the implication graph.

Circuit-Eval Given a Boolean circuit C and an input a , is $C(a) = 1$? In P: evaluate gate by gate in topological order.

Circuit-SAT Given a Boolean circuit C , is there a with $C(a) = 1$?

Graph Isomorphism Given G_1, G_2 , is there a bijection of the vertices mapping one to the other? In NP, not known to be in P, and *probably not NP-complete*: if it were, the polynomial hierarchy would collapse. We will see why when we do interactive proofs.

Linear Programming Is a system of linear inequalities over $\mathbb{Q}$ feasible? In P.

Integer Programming The same with the variables constrained to $\{0, 1\}$ (or to $\mathbb{Z}$). This constraint makes the problem NP-complete.

Subset Sum Given numbers $a_1, \dots, a_n$ in binary and a target value T , is there $S \subseteq [n]$ with $\sum_{i \in S} a_i = T$? Since the numbers are given in binary: hence the obvious dynamic program is exponential in the input length.

Composite Is N composite? In P by AKS. Long before it was known to be in P, it was known that the problem belongs to $NP \cap coNP$: it is in NP for the trivial reason (exhibit a factor), and in coNP thanks to a way to certify primality.

Factor Given N, L, U , does N have a prime factor in $[L, U]$? In $NP \cap coNP$ — for the coNP side, the witness is the full prime factorisation of N , together with primality certificates. Not believed to be NP-complete: a problem in $NP \cap coNP$ that is NP-complete would give $NP = coNP$.

Aspvall, Plass and Tarjan, IPL 8 (1979).

Best known algorithm: Babai's quasipolynomial time algorithm (STOC 2016).

Assume that in Subset Sum, $|a_i| \leq n^{O(1)}$ for all i , then give a polynomial time algorithm to solve this problem.

AKS algorithm for testing primality: Agrawal, Kayal and Saxena, *PRIMES is in P*, Annals of Math. 160 (2004).

EXERCISE 8. Show that each problem above is in NP, and verify that

the ones claimed to be in P are indeed decidable in polynomial time.

EXERCISE 9. Show the obvious inclusions

$$\text{DTIME}(t(n)) \subseteq \text{NTIME}(t(n)) \quad \text{NTIME}(t(n)) \subseteq \text{DTIME}(2^{O(t(n))})$$

and how this implies $P \subseteq NP$ and $NP \subseteq EXP$.

P vs NP and the theory of completeness

The question is whether finding a witness is essentially as easy as checking one. We have no idea. What we do have is a way of comparing problems, which lets us identify the hardest problems inside NP: if any of them is in P , then all of NP is.

A FIRST REDUCTION. Restricting the width of clauses does not make SAT easier, even though restricting it to 2 does.

LEMMA 10. SAT is not harder than 3-SAT.

Proof. Replace a clause $(\ell_1 \vee \dots \vee \ell_k)$ with $k > 3$ by

$$(\ell_1 \vee \ell_2 \vee y_1) \wedge (\bar{y}_1 \vee \ell_3 \vee y_2) \wedge \dots \wedge (\bar{y}_{k-3} \vee \ell_{k-1} \vee \ell_k),$$

with fresh variables $y_1, \dots, y_{k-3}$. Any satisfying assignment of the original clause extends to one of the chain (set $y_i = 1$ up to the satisfied literal and 0 afterwards), and conversely if all ℓ_j are false the chain forces $y_1, \dots, y_{k-3}$ to be simultaneously 1 and 0. The size grows linearly and the map is computable in polynomial time. $\square$

Observe that this trick needs width 3: with width 2 there is no room for both a literal and two chaining variables. This is not a coincidence, since 2-SAT is in P .

[AB, Section 2.3]. The concept of reduction and the notation $\text{SAT} \leq_p 3\text{-SAT}$ will be defined later.

DEFINITION 11 (Karp reduction). Given two language A and B , we write $A \leq_p B$ if there is a polynomial-time computable f with $x \in A \iff f(x) \in B$ for all x . We say that A reduces to B in polynomial time.

[AB, Def. 2.7].

PROPOSITION 12. If $A \leq_p B$ and $B \in P$ then $A \in P$. Moreover $\leq_p$ is transitive.

The proof is one line — run f , then the algorithm for B — but it uses two facts that are easy to take for granted: polynomials are closed under composition, and $|f(x)|$ is polynomial in $|x|$ because f has no time to write more.

REMARK 13. The polynomial bound on the reduction is what makes the notion useful, not a technicality. If we allowed exponential-time reductions, then *any* two non-trivial languages in EXP would be interreducible — the reduction could simply decide x itself and output a fixed yes- or no-instance — and the relation would tell us nothing. A reduction must be weaker than the problems it compares.

DEFINITION 14 (Hardness and completeness). B is NP-hard if $A \leq_p B$ for every $A \in \text{NP}$. It is NP-complete if in addition $B \in \text{NP}$.

EXERCISE 15. Prove [AB, Thm. 2.8]: if L is NP-hard and $L \in \text{P}$ then $\text{P} = \text{NP}$; if L is NP-complete then $L \in \text{P}$ iff $\text{P} = \text{NP}$. Then check that L is NP-complete iff $\bar{L}$ is coNP-complete, and that an NP-complete L in coNP would give $\text{NP} = \text{coNP}$.

That NP-complete problems exist at all is cheap: the language of triples $\langle M, x, 1^t \rangle$ such that the NDTM M accepts x within t steps is NP-complete by definition. The content of the next theorem is that a completely natural, purely combinatorial problem is complete.

[AB, Thm. 2.9]. This is the TMSAT problem. It is just syntactical formalization and carries no combinatorial value.

The Cook–Levin theorem

THEOREM 16 (Cook 1971, Levin 1973). The following are NP-complete:

1. SAT;
2. 3-SAT;
3. Circuit-SAT.

[AB, Thm. 2.10] and [AB, Section 6.5]. Karp (1972) then showed that 21 natural problems are complete, which is what turned the theorem into a research programme.

Since the three problems are in NP, we only need to show that they are NP-hard.

- 3-SAT is a special case of SAT, so $3\text{-SAT} \leq_p \text{SAT}$ trivially, and $\text{SAT} \leq_p 3\text{-SAT}$ by Lemma 10. By transitivity, either both are NP-complete or neither is. So it is enough to prove hardness for one of them.
- $\text{Circuit-SAT} \leq_p 3\text{-SAT}$ [AB, Lem. 6.11]: introduce a variable z_g for every gate g , and for each gate write down the constant number of clauses of width ≤ 3 expressing $z_g \leftrightarrow \text{op}(z_{g_1}, z_{g_2})$, plus a unit clause asserting the output. Again the construction produces clauses of width 3 and cannot be pushed down to 2-SAT.
- Circuit-SAT is NP-hard: given $L \in \text{NP}$ with verifier R and witness length $p(n)$, use the simulation of a time- T machine by a circuit of size $O(T \log T)$ [AB, Thm. 6.6] to build a circuit computing the function $(x, u) \mapsto R(x, u)$. Once we hardwire input x , the resulting circuit is satisfiable iff u is a witness for $x \in L$. We saw the construction of the circuit in the recap on Turing machines.

This is the Tseitin encoding. Note that the naive alternative — distributing the circuit into a CNF over the input variables only — can blow up exponentially; the extra variables are what keeps it linear. Circuits and their normal forms: [AB, Section 6.1], [JK, Chapter 1].

Chaining these gives hardness for 3-SAT and hence for SAT, and all three problems are clearly in NP. $\square$

REMARK 17. Going through circuits instead of directly encoding a tableau of machine configurations [AB, Lem. 2.11] is a matter of taste, and it pays off later: the same simulation is what gives $\text{P} \subseteq \text{P/poly}$ and what makes circuit lower bounds a plausible attack on P vs NP .

REMARK 18. Nondeterminism reappears in every model we will study, always with the same existential/universal asymmetry: nondeterministic communication complexity is the logarithm of the number of monochromatic rectangles needed to cover the 1s, and the gap

with the co-nondeterministic version is a theorem rather than an open problem there.

[RY, Chapter 2]. Comparing what we can and cannot prove in the two settings is a good way to calibrate how hard P vs NP really is.

Homework

- [AB] Sections 2.1, 2.2, 2.3 (skip 2.3.4), 2.6, 6.1.
- [AB] Exercises 2.1, 2.2, 2.4, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, 2.15, 2.16, 2.17, 2.18, 6.1.
- Prove the equivalence of Definition 4 and Definition 6.
- Prove [AB, Thm. 2.8].
- Show that if $P = NP$ then $NP = coNP$, and that the converse is not known.
- Where does the proof of Lemma 10 break for 2-SAT, and where does the proof of Proposition 12 break for exponential-time reductions?

References

- Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge, 2009. ISBN 978-0-521-42426-4. DOI: 10.1017/CBO9780511804090.
- Eli Ben-Sasson. *Expansion in Proof Complexity*. PhD thesis, Hebrew University of Jerusalem, Jerusalem, 2001. URL <http://www.math.ias.edu/~avi/STUDENTS/ebsthesis.ps>.
- Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, Berlin, Heidelberg, 2012. ISBN 978-3-642-24507-7. DOI: 10.1007/978-3-642-24508-4.
- Anup Rao and Amir Yehudayoff. *Communication Complexity and Applications*. Cambridge University Press, Cambridge, 2020. ISBN 978-1-108-49798-5. DOI: 10.1017/9781108671644.