

# Complexity Theory

Alessandro Barenghi

Dipartimento di Elettronica, Informazione e Bioingegneria  
Politecnico di Milano

7 giugno 2021

# Complexity theory: purpose and subjects

- Complexity theory is used to classify problems according to how expensive in time/space is solving them
- We will deal with problems which are:
  - With both domain and range over the naturals  $\mathbb{N}$
  - Corresponding to the computation of a *total computable function*: each problem has a **finitely described algorithm** solving it in **finite** time for any input
- A bit of problem taxonomy:
  - **Search problem**: given an input  $x \in \mathbb{N}$  to a problem corresponding to the computation of  $f(x)$ , find  $y = f(x), y \in \mathbb{N}$ . Example: compute the square root of  $x$ .
  - **Decision problem**: given an input  $x \in \mathbb{N}$  decide if  $x$  abides to some property  $f(x) : \mathbb{N} \rightarrow \{\top, \perp\}$ . Ex. Is  $x$  a perfect square?

# Complexity classes

- Complexity of computing the solution to a problem as a function of the input length  $n$  in base  $b > 1$
- Define the class (=set of problems)  $\text{DTIME}(f(n))$  as the ones for which a **deterministic TM** takes  $f(n)$  moves to compute the solution
- $\text{NTIME}(f(n))$  class: a **nondeterministic TM** takes  $f(n)$  moves to compute the solution
- In general, relations between  $\text{DTIME}$  and  $\text{NTIME}$  are not well understood
  - Exception:  $\text{DTIME}(O(n)) \subset \text{NTIME}(O(n))$
- Analog classes exist for space complexity  $\text{DSPACE}(f(n))$ ,  $\text{NSPACE}(f(n))$

# Complexity classes

- Given  $f(n)$ , can always build a problem not in  $\text{DTIME}(f(n))$ .
- $\forall k \geq 1$  there is a problem  $\in \text{DTIME}(n^k)$  and  $\notin \text{DTIME}(n^{k-1})$
- Some notable time complexity classes are:
  - $P = \bigcup_{i \geq 1} \text{DTIME}(n^i), i \in \mathbb{N}$ : “practically treatable” for any  $n$
  - $\text{NP} = \bigcup_{i \geq 1} \text{NTIME}(n^i), i \in \mathbb{N}$
  - $\text{EXP} = \bigcup_{i \geq 1} \text{DTIME}(2^{n^i}), i \in \mathbb{N}$
  - $\text{PSPACE} = \bigcup_{i \geq 1} \text{DSPACE}(n^i), i \in \mathbb{N}$ : NOT practically tractable in general,  $\text{NP} \subseteq \text{PSPACE}$
- $P \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXP}$ , but  $P \subsetneq \text{EXP}$
- Open questions:  $P \stackrel{?}{=} \text{NP}$ ,  $P \stackrel{?}{=} \text{PSPACE}$ . Likely answer: No.

# NP - An alternate definition

- It is possible to define a problem to be in NP in two ways
  - ① There is a nondeterministic TM which computes the solution to it in polynomial time
  - ② There is a **deterministic** TM which verifies that a solution for the problem is an actual solution in polynomial time
    - For decision problems: there is a deterministic TM which, given an element which makes the ND-TM accept, tests that is actually one of the elements which should make the ND-TM accept
- Example: Exiting a binary branching labyrinth without a map:
  - ① A nondeterministic TM will find the poly-length exiting path, if any, taking all the branches in parallel
  - ② A deterministic TM, given the path, will verify that it actually exits from the labyrinth through walking through it

# NP and complement classes

- P and NP defined on decision problems, for simplicity.
- A decision problem on the naturals = test if a natural belongs to some defined set = test if a string belongs to a language
- What about testing if the integer does **not** belong to a set?
  - If recognizing if it belongs to the set  $\in P$  the problem is still  $\in P$ : swap accepting/rejecting states in the recognizing TM.
  - If recognizing if it belongs to the set  $\in NP$ , the problem is  $\in coNP$ : the class of problems for which (deciding if an elements belongs to) the complement of a given set is in nondet-poly time.
  - The “swap accepting/rejecting” does not work anymore: a TM terminates on a single accepting state, or when all paths reject
    - If I swap states, I'll only check **one** of the old rejecting paths
  - NP = coNP? Open question; Highly likely answer: no.

# Computational reductions

- A need tool to compare the complexity of two problems: computational (aka Turing) reduction
- Given two problems  $A$  and  $B$ ,  $A$  reduces to  $B$  if given an oracle for  $B$  I can solve  $A$ . Thus
  - $A$  cannot be harder to solve than  $B$
  - $B$  can be harder to solve than  $A$
  - Therefore,  $A \leq^T B$  (where  $T$  reminds it's a Turing reduction)
- If  $A \leq^T B$  and I make only a poly number of calls to the oracle for  $B$  when solving  $A$ , plus extra poly-time computations
  - $A$  reduces polynomially (or, Cook-reduces) to  $B$
  - $A \leq_p^T B$ , since solving  $B$  also solves  $A$  with extra poly effort

# CLASS-hardness and CLASS-completeness

- Given a complexity class CLASS and a generic problem  $B$ ,  $B$  is CLASS-hard iff:
  - $\forall A \in \text{CLASS}, A \leq_p^T B$
- In other words, solving a CLASS-hard problem solves with extra poly effort all the problems in CLASS.
- Given a complexity class CLASS and a problem  $B$ ,  $B$  is CLASS-complete iff:
  - $B \in \text{CLASS}$  and  $B$  is CLASS-hard
- CLASS-complete problems are the “computationally hardest” within a class



# A taxonomy in NP – Assuming $P \neq NP$

- NP-Hard problem: any problem such that I can poly reduce to it any problem in NP. Note, there may be non-decision problems in this set.
- NP-Complete class: problems such that all the problems in NP can be poly reduced to any of them. Conventionally formulated as decision problems only
  - 3-SAT, Graph coloring, Subgraph isomorphism, decoding random linear codes, syndrome decoding
- NP-intermediate: essentially the complement of NPC to  $NP \setminus P$ . Some problems here have sub-exponential solutions
  - Factoring  $\in \text{DTIME}(\mathcal{O}(e^{(c+o(1))n^{\frac{1}{3}} \log(n)^{\frac{2}{3}}}))$ , Discrete Logarithms
  - Graph isomorphism (proven in 2015, disproved in '17, reproven in '17 later on, should be  $\mathcal{O}(2^{(\log(n)^3)})$ , this time, for real)

# Quantum Computing Model

- A transition of a classical TM computes a constant time operation on a finite set of symbols from a finite alphabet
- An abstract quantum computing machine is a machine applying unitary transformations (gates) assumed to be constant time to a finite set of qubits
- Complexity evaluated as a function of the (classical) input length  $n$ :
  - Time: evaluated as either the number of sequential gates (depth) or the total number of gates ( $\mathcal{O}(\text{depth} \times \text{qubits})$ )
  - Space: evaluated as number of involved qubits
- A measurement of the result may not yield the same classical value if the computation is repeated → **probabilistic computation model**

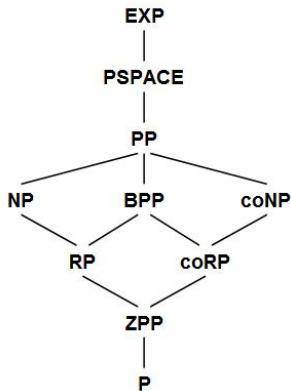
# A primer on probabilistic computation

- What is a randomized algorithm? We may randomize:
  - **running time**: Algorithm  $A$  runs in probabilistic poly time, i.e., with a certain probability it terminates in poly time
  - **correctness**: regardless of the running time, the algorithm returns the correct answer with probability  $p$
- We can solve problems, in expectation, if:
  - The running time is either deterministically or probabilistically polynomial, with  $Pr(\text{poly time}) \gg \frac{1}{2}$
  - The algorithm provides a correct solution with a satisfactory probability, that is, either :
    - $p \approx 1$
    - $p$  is large enough to allow us to query the algo poly times and do majority voting.  $p = \frac{1}{2} + \frac{1}{2^n}$  would need exp. no. calls

## Probabilistic computation classes (for TMs)

- PP class: the problem is solved in  $\text{DTIME}(\text{poly}(n))$  by an algorithm outputting the correct answer with  $\text{Pr} > \frac{1}{2}$ 
  - Not necessarily tractable:  $\frac{1}{2} + \frac{1}{2^n} > \frac{1}{2}$
- BPP class: the problem is solved in  $\text{DTIME}(\text{poly}(n))$  by an algorithm outputting the correct answer with  $\text{Pr} > \frac{1}{2} + k$ 
  - Tractable, for reasonable values of constant  $k$ , Typ.  $\text{Pr} = \frac{2}{3}$
- RP class: the problem is solved in  $\text{DTIME}(\text{poly}(n))$  by an algorithm outputting accept with  $\text{Pr} > \frac{1}{2} + k$ , if the solution is accept, but *never* accepting when it has to reject
- ZPP class: the problem is solved in  $\text{DTIME}(\text{poly}(n))$  by an algorithm which: gives a correct answer with  $\text{Pr} = \frac{1}{2}$ , or answers “I don’t know” with  $\text{Pr} = \frac{1}{2}$ .
  - Equivalent to an algo running on *expected* poly time, always giving correct answers (not straightforward)

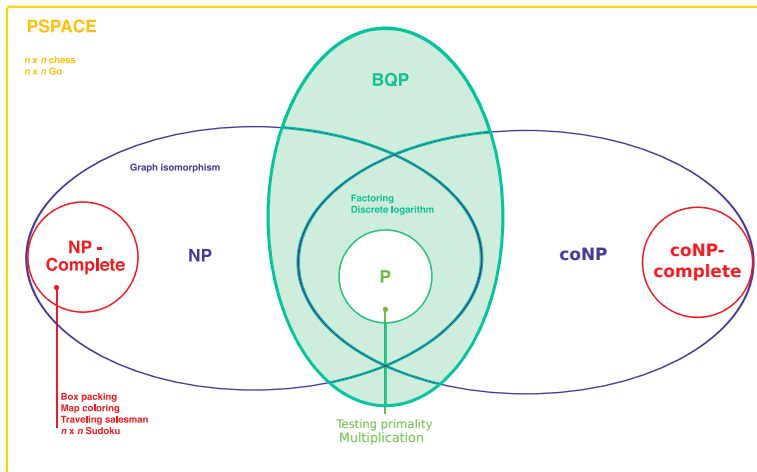
# Probabilistic computation classes (for TMs)




# Probabilistic computation classes (for quantum computers)

- Finally, we can define what is tractable by a QC
  - We consider the former abstract quantum machine model and define complexity as a function of the classical input length  $n$
- BQP class: the problem is solved in  $\text{DTIME}(\text{poly}(n))$  by a **quantum machine** which leaves a final state on which a measure yields the **classical correct answer** with  $\text{Pr} > \frac{1}{2} + k$
- Functionally analogous to BPP, but on a quantum computer
- We know  $\text{BPP} \subseteq \text{BQP}$ : a QC emulates a classical probabilistic TM on poly-time algs, with poly-time overhead
- No sense in defining QP: cannot have a deterministic QC!

# Relations between BQP and the other classes



 Problems with potential exponential speedup

# Which speedups can we achieve?

- Anything lying in  $NP \setminus P$ , and in BQP, or in  $coNP \setminus P$ , and in BQP  $\rightarrow$  likely to gain **exponential speedup**
  - We have no general det. poly algorithm for  $A \in NP \setminus P$
- Anything in  $P \rightarrow$  likely not worth it: already poly time
  - Usually, solving problems in  $P$  on a QC is slower (due probabilistic computation and reversibility)
- Outside BQP, inside PSPACE  $\rightarrow$  no exponential speedup, but subexponential gains are possible
  - E.g. complexity goes down from  $\mathcal{O}(2^n)$  to  $\mathcal{O}(2^{\frac{n}{2}})$
- Strong belief:  $NP\text{-complete} \cap BQP = \emptyset$ 
  - **A quantum computer is not a nondeterministic TM!**
  - 3SAT, SubGI are **not** getting an exponential speedup