① Complexity Theory

⊛ What is the complexity of a computation or algorithm?

⊛ We usually quantify this in terms of time and space resources.

↳ Specifically: How do the time and space requirements scale as a function of the size of the input?

↳ "time" ⟷ number of basic operations in a given computing architecture.

⊛ The exact requirements could strongly depend on the physical architecture! That is why we just look at how the complexity scales.
(i.e., we ignore constants). $f(n) = \alpha n^2 + \beta n + \cdots$

(a) Big-O notation

- $f(n) = O(g(n))$ (or $f(n) \in O(g(n))$) if there exists constants c, N such that

$$|f(n)| \leq c|g(n)| \quad \forall\, n \geq N.$$

Example: $15n^4 + n^3 + 10000n^2 - n + 17 = O(n^4)$

$O(1) \rightarrow$ "constant"

$O(\log(n)) \rightarrow$ "logarithmic"

$O(n) \rightarrow$ "linear"

$O((\log(n))^k) \rightarrow$ "polylogarithmic"

$O(n^2) \rightarrow$ "quadratic"

$O(n^k) \rightarrow$ "polynomial"

$O(c^n)$, $c > 0 \rightarrow$ "exponential"

2^n

- $f(n) = \Omega(g(n))$ (or $f(n) \in \Omega(g(n))$) if there exists constants c, N such that $|f(n)| \geq c|g(n)|$ $\forall\, n \geq N$.

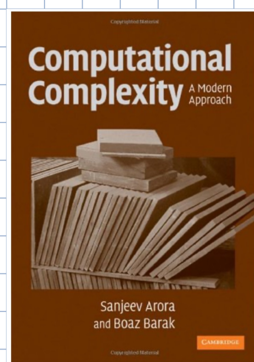
- $f(n) = \Theta(g(n))$ (or $f(n) \in \Theta(g(n))$) if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

  (or $f(n) = O(g(n))$ and $g(n) = O(f(n))$).

  ⊛ Essentially, up to constants, f and g are the same asymptotically.

  Example: $f(n) = 5n^3 + 2n^2 - 7$, $g(n) = n^3$

(b) Another abstraction is to reason in terms of <u>Turing Machines</u> (but we will not get into this).



(c) <u>Basic Complexity Classes</u> (⊛ All definitions informal!)

⊛ All of these definitions apply to <u>decision problems</u>: problems in which the answer is "yes" or "no".

⊛ More generally, <u>promise problems</u>:

- Let $A_{yes} \subseteq \{0,1\}^*$ (all finite bit strings), $A_{no} \subseteq \{0,1\}^*$
- $A_{yes} \cap A_{no} = \emptyset$ (the "yes" and "no" sets are disjoint)
- Problem: given $x \in A_{yes} \cup A_{no}$ (this is the promise), determine if $x \in A_{yes}$ or $x \in A_{no}$.

(e.g., $A_{yes} = \{ N \in \mathbb{Z}_{>1} : N \text{ is composite} \}$
$A_{no} = \{ N \in \mathbb{Z}_{>1} : N \text{ is prime} \}$)

- P : problems solvable (deterministically) in polynomial time

⊛ These are often referred to as "efficient" and/or "easy" problems.

- BPP : problems solvable (probabilistically) in polynomial time (correct on every input with probability 2/3).

- PSPACE : problems solvable (deterministically) using polynomial space ("space" ≡ memory).

- EXP(TIME): problems solvable (deterministically) in exponential time.

- NP : problems solvable (non-deterministically) in polynomial time

  This refers to a non-deterministic Turing Machine — not the same as probabilistic Turing machine

  ⊛ Equivalently: problems verifiable (deterministically) in polynomial time.

  ⊛ If verification is probabilistic, then we get the class MA ("Merlin-Arthur")

  ⊛ Examples

  - Factoring → we can efficiently multiply two proposed factors to verify that they are indeed factors.

  - Boolean Satisfiability problem (aka SAT):

    - A Boolean variable $x \in \{\underbrace{\text{TRUE}}_{1}, \underbrace{\text{FALSE}}_{0}\}$;

    - $\neg x \equiv \text{NOT } x \equiv \bar{x}$ (negation) $\neg 1 = 0$, $\neg 0 = 1$

    - $x \wedge y$ (AND) $\left(\begin{matrix} 0 \wedge 0 = 0 & 1 \wedge 0 = 0 \\ 0 \wedge 1 = 0 & 1 \wedge 1 = 1 \end{matrix}\right)$

- $x \vee y$ (OR) $\left(\begin{array}{ll} 0 \vee 0 = 0 & 1 \vee 0 = 1 \\ 0 \vee 1 = 1 & 1 \vee 1 = 1 \end{array}\right)$

- A (propositional logic) formula:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1$$

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- SAT problem: Given a formula, is it satisfiable: does there exist an assignment of variables such that the formula evaluates to 1?

(*) There is no known poly-time algorithm for SAT (in the worst case — many efficient solvers can efficiently solve large, structured instances of SAT that arise in practical applications.

(*) The SAT problem is NP-complete: (1) SAT $\in$ NP; (2) SAT is NP-hard.

(*) If a problem P is "X-hard", for some complexity class X, it means that every problem in X can be reduced to P in polynomial time (i.e., a solution to P can be used as a subroutine to efficiently solve the given problem).

In other words: P can be used to solve any problem in X!

<u>Note</u>: P itself might not be an easy problem! But it as least as hard as any problem in X.

⊛ P is "X-complete" means that P is a representative of the hardest problems in X.

⊛ <u>An efficient (i.e, poly-time) algorithm for SAT would imply that P = NP!</u> (People generally believe that P ≠ NP, but there is no proof.)

SIAM J. COMPUT.
Vol. 26, No. 5, pp. 1411–1473, October 1997

© 1997 Society for Industrial and Applied Mathematics
007

QUANTUM COMPLEXITY THEORY*

ETHAN BERNSTEIN† AND UMESH VAZIRANI‡

② Quantum Complexity Classes

⊛ We define decision/promise problems as before.

(a) <u>BQP</u> : "efficient"/"easy" for quantum computers. (Bounded-Error Quantum polynomial time)

More formally: problems solvable in polynomial time on a quantum computer (correct on every input with probability $2/3$).

⊛ Basically a quantum version of BPP.

- <u>Note</u>: $BQP \subseteq PSPACE$
- <u>Note</u>: $P \subseteq BQP$ (all classical computations can be performed on a q. computer).
- <u>Note</u>: Factoring $\in$ BQP → But we don't know if factoring $\in$ P!

  ⇒ we don't know if BQP is strictly larger than P!

  ⊛ We have some <u>oracle separations</u> (e.g, Deutsch-Jozsa, Simon's).

- Other problems that could demonstrate a separation:
  - Sampling from random quantum circuits (e.g, Boson Sampling).
  - Simulating the evolution of highly entangling dynamics.

↳ Here also there are no rigorous proofs of classical hardness!

- Oracle separations are not "ideal", b/c they can be used to prove seemingly odd results

  (*) e.g., there are oracles with respect to which we have both P=NP and P≠NP!

SIAM J. Comput.
Vol. 4, No. 4, December 1975

RELATIVIZATIONS OF THE $\mathcal{P} =? \mathcal{NP}$ QUESTION*

THEODORE BAKER†, JOHN GILL‡ AND ROBERT SOLOVAY¶

Abstract. We investigate relativized versions of the open question of whether every language accepted nondeterministically in polynomial time can be recognized deterministically in polynomial time. For any set X, let $\mathcal{P}^X$ (resp. $\mathcal{NP}^X$) be the class of languages accepted in polynomial time by deterministic (resp. nondeterministic) query machines with oracle X. We construct a recursive set A such that $\mathcal{P}^A = \mathcal{NP}^A$. On the other hand, we construct a recursive set B such that $\mathcal{P}^B \neq \mathcal{NP}^B$. Oracles X are constructed to realize all consistent set inclusion relations between the relativized classes $\mathcal{P}^X$, $\mathcal{NP}^X$, and co $\mathcal{NP}^X$, the family of complements of languages in $\mathcal{NP}^X$. Several related open problems are described.

(b) Proof that $P \subseteq BQP$: Make any classical computation (even a non-linear one) reversible!

- Take a computation $x \mapsto f(x)$. The Boolean function f might not be reversible (e.g., $f(x_1, x_2) = x_1 \wedge x_2$ is not reversible).

- Make it reversible by adding extra bits and keeping track of the input:

  $(x, y) \mapsto (x, y \oplus f(x)) \longrightarrow$ This transformation is reversible!

  Let $g(x,y) = (x, y \oplus f(x))$. Then $g(x, y \oplus f(x)) = (x, y \oplus f(x) \oplus f(x)) = (x, y)$
  So g is invertible, and the inverse is itself!

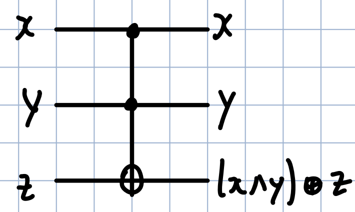
- Making the circuit reversible incurs only polynomial overhead!

C. H. Bennett
(Nov. 1973)

Logical Reversibility of Computation*

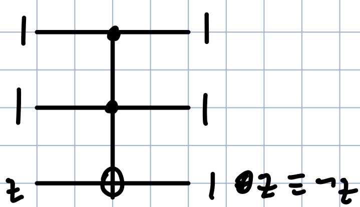
- The function g can be implemented efficiently using <u>Toffoli gates</u> only!

- <u>Toffoli gate</u> : the CCNOT gate

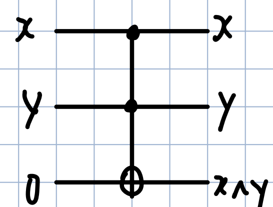


x	y	z	x'	y'	z'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

⊛ Toffoli can be used to simulate the universal gate set {AND, OR, NOT} in a reversible way!



→ NOT gate!

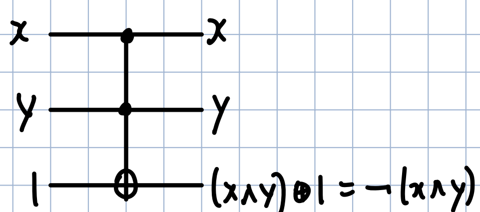


→ AND gate

(For OR gate, use $x \vee y = \neg(\neg x \wedge \neg y)$)

⊛ The NAND gate alone is universal!

$\mathrm{NAND}(x,y) = \neg(x \wedge y)$



- The map $|x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle$ is unitary, so we can in principle implement it on a quantum computer.

  ⊛ In particular, Toffoli has this form, so it can be implemented as a unitary.

- Transform the given (irreversible) Boolean circuit to a reversible one with extra bits and Toffoli gates; then replace all Toffoli gates with its quantum version.

(b) QMA : (Quantum Merlin-Arthur) (Quantum version of NP/MA).

- Recall NP : problems verifiable (deterministically) in polynomial time.

⊛ If verification is probabilistic, then we get the class MA ("Merlin – Arthur")

Also called "prover"

Also called "verifier"

- Prover (Merlin) is unbounded/all-powerful
- The verifier (Arthur) is computationally bounded (can do poly-time computations)
- The prover can provide any witness to the verifier to convince them that a given problem instance is true.
- A promise problem $A = (A_{yes}, A_{no})$ is in MA if there is a poly-time verifier such that:

  - Completeness: $x \in A_{yes} \Rightarrow \exists w \in poly(|x|)$ s.t. $Pr[\text{verifer accepts } x, w] \geq \frac{2}{3}$

    This is the witness

    Verifier concludes that x is a yes instance.

  - Soundness : $x \in A_{no} \Rightarrow \forall w \in poly(|x|)$, $Pr[\text{verifier accepts } x, w] \leq \frac{1}{3}$.

- Example: factoring $\rightarrow$ $A_{yes} = \{ N \in \mathbb{Z}_{\geq 1} : N \text{ is composite} \}$
  $A_{no} = \{ N \in \mathbb{Z}_{\geq 1} : N \text{ is prime} \}$

  - An instance $x \in A_{yes} \cup A_{no}$ is any $N \in \mathbb{Z}_{\geq 1}$.
  - A witness would be some other $w \in \mathbb{Z}_{\geq 1}$ that should be a non-trivial factor of N $\rightarrow$ but the verifier can efficiently check this.

- In QMA, the prover and verifier are quantum, and the prover can send a quantum state. (Verifier can do poly-time quantum circuits.)

- **Completeness**: $x \in A_{yes} \Rightarrow \exists\, |\psi_x\rangle \in (\mathbb{C}^2)^{\otimes poly(|x|)}$ s.t. $\Pr[V_x(|\psi_x\rangle \otimes |\vec{0}\rangle) = 1] \geq \frac{2}{3}$.

- **Soundness**: $x \in A_{no} \Rightarrow \forall\, |\psi_x\rangle \in (\mathbb{C}^2)^{\otimes poly(|x|)}$, $\Pr[V_x(|\psi_x\rangle \otimes |\vec{0}\rangle) = 1] \leq \frac{1}{3}$

- The canonical QMA-complete problem: k-local Hamiltonian problem

(※ Basically, a quantum version of the SAT problem.).

- **Given**: a Hamiltonian $H = \sum_{i=1}^{m} H_i$ acting on n qubits.

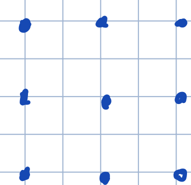
$i\hbar \frac{\partial}{\partial t}|\psi_t\rangle = H|\psi_t\rangle$

↳ a Hermitian operator describing the energy of a many-body system.

Each H_i acts on at most k qubits, and $\|H_i\|_\infty \leq 1$. (k is a constant).
Also given: $a, b \in \mathbb{R}$ s.t. $a - b \geq 1/poly(n)$.

$1/n^2$

- **Problem**: $A = (A_{yes}, A_{no})$

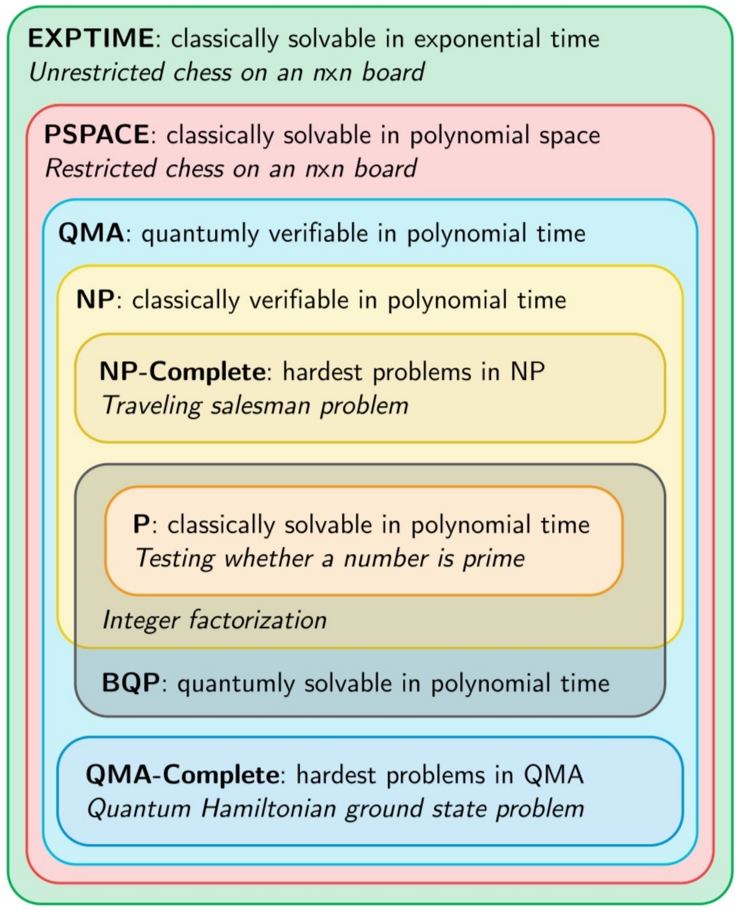


$$A_{yes} = \{(H, a, b) : \lambda_{min}(H) \leq a\}$$

$$A_{no} = \{(H, a, b) : \underbrace{\lambda_{min}(H)}_{} \geq b\}.$$

↳ smallest eigenvalue of H ⟺ "ground-state energy"

③ Summary



https://arxiv.org/abs/2101.08448