CS 599 P1: Introduction to Quantum Computation — Boston University, Fall 2025
Instructor: Alexander Poremba — **Scribe:** Jiaqian Li

LECTURE # 14: QUANTUM COMPLEXITY THEORY

Before we dive into quantum complexity theory, we will begin with a brief introduction to *classical* computational complexity theory—a framework for reasoning about computational problems. The goal of complexity theory is to classify problems according to the amount of computational resources required to solve them; for example, time, space, randomness, or interaction.

In this lecture, we focus on time as the primary resource and briefly touch on randomness. We introduce several foundational complexity classes and illustrate them through canonical examples.

1 Introduction to Computational Complexity

1.1 Computational Resources and Problem Types

A *computational resource* quantifies the cost of executing an algorithm. The most common examples are:

- **Time:** the number of computation steps required as a function of input size.
- **Space:** the amount of memory used during computation.
- **Randomness:** the number of random bits an algorithm consumes.
- **Interaction:** the number of rounds or messages exchanged in a protocol between a (possibly efficient) verifier and one or more provers.

Other important but more specialized resources include the number of queries made to an oracle (*query complexity*) and the amount of communication between parties (*communication complexity*). In this lecture, we focus on time and randomness, the two primary measures relevant to the study of both classical and quantum algorithms.

1.2 Decision Problems

In computational complexity, it is convenient to restrict attention to *decision problems*, which are problems with simple YES or NO answers.

Definition 1.1 (Decision problem)**.** *A* decision problem *is a function*

$$f : \{0,1\}^* \to \{0,1\},$$

*where each input instance $x \in \{0,1\}^*$ encodes a mathematical object (e.g., a number, a graph, or a Boolean formula). The output $f(x) = 1$ corresponds to the answer "*YES*" and $f(x) = 0$ to "*NO*."*

Remark 1 (Search problems and decision-to-search)**.** *Many computational tasks are naturally search problems, where the goal is to produce a witness/solution (e.g., a satisfying assignment, a factor of an integer). For a wide range of problems, it is convenient to study their decision versions, because decision formulations fit cleanly into complexity classes and allow uniform comparisons.*

Formally, a search problem can be specified by a polynomially decidable relation $R(x, w)$ with polynomially bounded witnesses $|w| \leq \mathrm{poly}(|x|)$. The associated decision language is

$$L_R = \{x : \exists w \text{ with } R(x, w) = 1\}.$$

In many canonical cases, access to a decision oracle for L_R suffices to recover a witness with only polynomial overhead. We will see two generic ways to achieve this later.

1.3 Examples of Decision Problems

We now examine three canonical decision problems that will serve as running examples. In the latter two, we also record a simple decision-to-search reduction, instantiating the two generic patterns mentioned above.

Problem 1: PRIMES

Given an integer N (represented in binary), output YES if N is prime and NO otherwise.

This problem was long believed to require randomization to solve efficiently, until the discovery of the deterministic AKS primality test [AKS04], which showed that PRIMES lies in class $\mathbf{P}$.

Problem 2: SAT

Given a Boolean formula φ over variables $x_1, \ldots, x_n$, output YES if there exists a satisfying assignment $(x_1, \ldots, x_n) \in \{0, 1\}^n$ such that $\varphi(x_1, \ldots, x_n) = 1$, and NO otherwise.

The satisfiability problem (SAT) is the first known $\mathbf{NP}$-complete problem and plays a central role in complexity theory.

Claim 1.2 (Decision-to-search for SAT via self-reduction)**.** *Given oracle access to the decision problem $\mathcal{O}_{\mathrm{SAT}}(\cdot)$, there is a polynomial-time procedure that, on input φ, either returns a satisfying assignment for φ or correctly concludes that φ is unsatisfiable. The procedure makes at most $n + 1$ oracle queries for $n = \#\mathit{vars}(\varphi)$.*

Proof. Query $\mathcal{O}_{\mathrm{SAT}}(\varphi)$. If NO, output "unsat." Otherwise, for $i = 1, \ldots, n$ do: fix $x_i \leftarrow 0$ obtaining $\varphi_{i,0}$; if $\mathcal{O}_{\mathrm{SAT}}(\varphi_{i,0}) =$ YES, keep $x_i = 0$ and set $\varphi \leftarrow \varphi_{i,0}$; else set $x_i = 1$ and update φ accordingly. Each step preserves satisfiability, so after n steps we obtain a full satisfying assignment. The time overhead is polynomial and the number of oracle calls is $O(n)$. □

Problem 3: FACTORING

Given two integers N and K, output YES if N has a nontrivial factor less than K, and NO otherwise.

Although FACTORING is often stated as a search task ("find a nontrivial factor of N"), the following shows that the decision form above suffices to recover a factor efficiently.

Claim 1.3 (Decision-to-search for FACTORING via threshold queries)**.** *Let $\mathcal{O}_{\mathrm{FAC}}(N, K)$ be the decision oracle that answers whether N has a nontrivial factor $\leq K$. There is a polynomial-time procedure that, on input N, finds a nontrivial factor of N (or correctly concludes that N is prime) using $O(\log N)$ oracle queries.*

Proof. If desired, first run a deterministic primality test to handle prime N outright; alternatively, note that if no factor $\leq \lfloor\sqrt{N}\rfloor$ exists then N is prime. To find a factor when one exists, binary search over $K \in [2, \lfloor\sqrt{N}\rfloor]$ using the monotone predicate

$$P(K) := \big[N \text{ has a nontrivial factor} \leq K\big].$$

Set $L \leftarrow 2$, $R \leftarrow \lfloor\sqrt{N}\rfloor$. While $L < R$, let $M = \lfloor(L + R)/2\rfloor$; if $\mathcal{O}_{\mathrm{FAC}}(N, M) = \textsc{yes}$ set $R \leftarrow M$, else set $L \leftarrow M{+}1$. At termination, L is the smallest nontrivial factor; verify $L \mid N$ and return L (optionally also N/L). The search uses $O(\log N)$ queries and polynomial-time arithmetic. □

Claims 1.2 and 1.3 instantiate the two generic patterns from the remark above: *self-reduction* for SAT and *threshold (monotone) queries* for FACTORING.

2 Important Complexity Classes

Before turning to quantum models, we briefly organize the classical landscape. A *complexity class* is specified by four ingredients: (i) a uniform algorithmic model (e.g., deterministic/probabilistic Turing machines), (ii) a resource budget as a function of input length (e.g., time, space), (iii) an acceptance criterion (deterministic acceptance, nondeterministic witnessing, bounded error, etc.), and (iv) a fixed encoding convention for inputs. Throughout, we treat polynomial time as the threshold for "efficient," and we work with decision problems unless otherwise stated.

2.1 The Class P

The first major complexity class is the class of problems solvable in deterministic polynomial time.

Definition 2.1 (Class $\mathbf{P}$)**.** *A decision problem f belongs to $\mathbf{P}$ if there exists a deterministic algorithm that computes $f(x)$ in time at most $O(|x|^c)$ for some constant $c > 0$, where $|x|$ denotes the length of the input.*

Intuitively, $\mathbf{P}$ contains all problems that can be solved efficiently by a classical deterministic computer. Examples include PRIMES, graph connectivity, and sorting.

2.2 The Class NP

Next, we consider problems whose solutions may be hard to find but easy to verify.

Definition 2.2 (Class $\mathbf{NP}$)**.** *A decision problem f belongs to $\mathbf{NP}$ if, whenever $f(x) = 1$, there exists a polynomial-length* witness *w that can be verified in polynomial time by a deterministic algorithm.*

Equivalently, $\mathbf{NP}$ is the class of problems for which YES instances admit efficiently verifiable certificates. The FACTORING problem lies in $\mathbf{NP}$: if N has a nontrivial factor $d < K$, the witness $w = d$ can be verified in polynomial time by checking that $d \mid N$ and $1 < d < K$. Similarly, SAT is in $\mathbf{NP}$: a satisfying assignment serves as a witness that can be checked in time polynomial in the size of the formula.

2.3 NP-Completeness

Within $\mathbf{NP}$, some problems are particularly important because they capture the full difficulty of the class.

Definition 2.3 (NP-completeness)**.** *A decision problem A is* NP-complete *if:*

1. $A \in \mathbf{NP}$*, and*
2. *every problem $B \in \mathbf{NP}$ can be reduced to A by a polynomial-time reduction.*

NP-complete problems are the hardest problems in $\mathbf{NP}$. If any NP-complete problem were shown to be solvable in polynomial time, then $\mathbf{P} = \mathbf{NP}$ would follow. The canonical NP-complete problem is SAT, as established by Cook's theorem.

2.4 The Class BPP

Finally, we introduce a complexity class that captures efficient computation using randomness.

Definition 2.4 (Class $\mathbf{BPP}$)**.** *A decision problem f belongs to* $\mathbf{BPP}$ *(*Bounded-Error Probabilistic Polynomial Time*) if there exists a probabilistic polynomial-time algorithm A such that for every input x:*

$$\Pr[A(x) = f(x)] \geq \tfrac{2}{3}.$$

The constant $2/3$ is arbitrary; by repeating the algorithm and taking a majority vote, the success probability can be amplified exponentially close to 1.

Polynomial Identity Testing

The *Polynomial Identity Testing (PIT)* problem is a canonical example in $\mathbf{BPP}$. Given two multivariate polynomials $p(x_1, \ldots, x_n)$ and $q(x_1, \ldots, x_n)$ presented as arithmetic circuits, the task is to decide whether $p \equiv q$. A simple randomized algorithm evaluates both polynomials on random inputs and declares equality if all evaluations match, achieving bounded error by the Schwartz–Zippel lemma.

2.5 Summary

A quick word on MA. $\mathbf{MA}$ (Merlin–Arthur) is a one-round proof/verification model: a prover (Merlin) sends a polynomial-length string (witness) w; then a polynomial-time randomized verifier (Arthur) checks it with bounded error (say completeness $\geq 2/3$, soundness $\leq 1/3$). In particular,

$$\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{MA} \text{ and } \mathbf{P} \subseteq \mathbf{NP}.$$

Where is BPP relative to NP? We do not currently know whether $\mathbf{BPP} \subseteq \mathbf{NP}$ or $\mathbf{NP} \subseteq \mathbf{BPP}$. Sufficiently strong pseudorandom generators [NW94, IW01] imply

$$\mathbf{P} = \mathbf{BPP}.$$

Importantly, any unconditional proof that resolves $\mathbf{P}$ vs.$\mathbf{BPP}$ cannot be *relativizing*: in the relativized world both behaviors occur – there exist oracles A with $\mathbf{P}^A = \mathbf{BPP}^A$ and oracles B with $\mathbf{P}^B \neq \mathbf{BPP}^B$ [BF99].

3 Quantum Complexity: The Class BQP

We now introduce the first quantum complexity class. Informally, $\mathbf{BQP}$ is the quantum analogue of $\mathbf{BPP}$, obtained by replacing probabilistic polynomial-time algorithms by polynomial-size quantum circuits with bounded two-sided error.

Definition 3.1 (Class $\mathbf{BQP}$)**.** *A decision problem $f : \{0,1\}^* \to \{0,1\}$ is in $\mathbf{BQP}$ if there exists a family of polynomial-size, uniformly generated quantum circuits $\{Q_n\}_{n\geq 1}$ such that for every input $x \in \{0,1\}^n$,*

$$\Pr[Q_n(x) \textit{outputs } f(x)] \;\geq\; \tfrac{2}{3}.$$

Here the probability is over the outcome of measuring a designated output qubit at the end of the computation. As usual, the constant $\frac{2}{3}$ can be amplified to $1 - 2^{-\Omega(n)}$ by independent repetition and majority vote.

FACTORING $\in \mathbf{BQP}$

By Shor's algorithm, integer factorization (and, in particular, the decision version described earlier) admits a polynomial-time quantum algorithm. Consequently, FACTORING lies in $\mathbf{BQP}$. This provides a concrete example of a natural number-theoretic problem believed to be outside $\mathbf{P}$ but efficiently solvable on a quantum computer.

3.1 Relationships

Quantum computation subsumes classical deterministic and randomized computation:

$$\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP}.$$

The first inclusion is immediate; the second follows because a quantum circuit can sample classical random bits and simulate a probabilistic Turing machine with only polynomial overhead.

Surprisingly, $\mathbf{BQP}$ is known to be simulable in polynomial *space* (though it is not believed to be simulable in polynomial time):

Theorem 3.2 (Theorem 8.4 of [BV97])**.** $\mathbf{BQP} \subseteq \mathbf{PSPACE}$.

At a high level, the proof evaluates acceptance probabilities of polynomial-size quantum circuits to sufficient precision via a dynamic-programming traversal of the computation, storing only polynomially many amplitudes at a time.

While unconditional separations between major classes remain elusive, there is compelling *relativized* evidence about $\mathbf{BQP}$:

Theorem 3.3 (Oracle separations; folklore after [BV97]; [BBBV97])**.** *There exist oracles A, B such that*

$$\mathbf{BQP}^A \not\subseteq \mathbf{NP}^A \quad \textit{and} \quad \mathbf{NP}^B \not\subseteq \mathbf{BQP}^B.$$

Interpretation and limits. The first oracle exhibits a relativized world where quantum computation solves problems beyond any nondeterministic polynomial-time verifier; the second shows that, with another oracle, quantum speedups still fail to capture all of $\mathbf{NP}$. Taken together, these point to the prevailing view that $\mathbf{BQP}$ and $\mathbf{NP}$ are incomparable in general.

A complementary line of evidence comes from black-box lower bounds for search. Grover's algorithm provides a quadratic improvement for unstructured search – reducing 2^n brute-force queries to $O(2^{n/2})$ quantum queries – and the Bennett-Bernstein-Brassard-Vazirani bound shows this is optimal in the oracle (query) model [BBBV97]. Thus, absent exploitable structure, quantum speedups alone are insufficient to place $\mathbf{NP}$-complete problems (e.g., SAT) in $\mathbf{BQP}$. By contrast, structured number-theoretic problems such as FACTORING are in $\mathbf{BQP}$ due to the algebraic structure leveraged by Shor's algorithm.

3.2 Where We Stand

Collecting the above,

$$\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{PSPACE},$$

and relativized worlds indicate that $\mathbf{BQP}$ and $\mathbf{NP}$ neither contain one another.

References

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004. 2

[BBBV97] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997. 5, 6

[BF99] Harry Buhrman and Lance Fortnow. One-sided versus two-sided error in probabilistic computation. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 100–109. Springer, 1999. 4

[BV97] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997. 5

[IW01] Russell Impagliazzo and Avi Wigderson. Randomness vs time: Derandomization under a uniform assumption. *Journal of Computer and System Sciences*, 63(4):672–688, 2001. 4

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of computer and System Sciences*, 49(2):149–167, 1994. 4