Introduction to Quantum Computation

These lecture notes are based on the *Introduction to Quantum Computation* course given at EPFL for the academic year 2024/2025 by Prof. Olivier Lévêque and Prof. Rüdiger Urbanke. The content of the course was originally created by Prof. Nicolas Macris and is based on his lecture notes.

Department of Computer Science - École Polytechnique Fédérale de Lausanne

Author: Arthur Aimone

2024/2025

Introduction to Quantum Computation

Chapter: Quantum Complexity Theory

Draft lecture notes

Chapter 9

Quantum Complexity Theory

Quantum algorithms such as Simon's algorithm, Shor's algorithm, and Grover's algorithm show that quantum computers can sometimes be much faster than classical computers. But they do not tell us the full story. Which problems can quantum computers solve efficiently? Which problems remain hard? What does it even mean to compare the power of classical and quantum computation?

Complexity theory gives a mathematical language for such questions. It classifies computational problems according to the resources needed to solve them: time, memory, randomness, nondeterminism, interaction, and, for us, quantumness. In this chapter our goal is not to prove the deepest theorems of the field, but to build a useful map of the basic classes and inclusions, and to explain what is known and what remains open.

9.1 Decision problems and resources

Complexity theory usually focuses on **decision problems**, where the output is a single bit: yes or no. This is less restrictive than it may first appear, because many search and optimization problems can be converted into related decision problems.

When we mention a search problem such as factoring, we will mean either the search task (find a prime factorization of the integer N), or a suitable associated decision problem (for example, given integer N, a and b, decide whether N has a prime factor in the interval $[a, b]$). It is generally straightforward to solve the search problem with polynomially many calls to the decision procedure (here, we can determine the prime factors of N through repeated bisection of the initial interval $[2, N-1]$.

Definition 9.1.1 (Language)**.** *A **language** is a set $L \subseteq \{0,1\}^*$. The associated decision problem is:*

> *Given a binary string x, decide whether $x \in L$.*

*Inputs $x \in L$ are called **yes-instances**; inputs $x \notin L$ are called **no-instances**.*

9.2 Some classical complexity classes

We first recall the main classical classes that will serve as reference points.

9.2.1 P: deterministic polynomial time

Definition 9.2.1 (P)**.** P *is the class of languages decidable by a deterministic classical algorithm in time* $\mathrm{poly}(n)$*.*

In this definition, n refers to the input length $n = |x|$. An algorithm is called **efficient** if it uses at most $\mathrm{poly}(n)$ elementary steps. This is a robust notion: changing the precise model of classical computation usually changes running time only by (at most) a polynomial factor.

P is the traditional formalization of efficient classical computation. Examples include graph connectivity, shortest paths, primality testing, and linear programming.

Example 9.2.2 (Graph connectivity)**.** *Let* L_{conn} *be the set of binary encodings of connected graphs. The decision problem is: given a graph* G*, decide whether* G *is connected. This problem is in* P*: one can run breadth-first search or depth-first search.*

9.2.2 BPP: randomized polynomial time

Randomized algorithms are allowed to toss coins during the computation. They may sometimes make mistakes, but the probability of error must be bounded away from $1/2$ on every input.

Definition 9.2.3 (BPP)**.** *A language* L *is in* BPP *if there exists a randomized polynomial-time classical algorithm* A *such that, for every input* x*,*

$$x \in L \implies \mathbb{P}[A(x) = 1] \geq \frac{2}{3}, \tag{9.1}$$

$$x \notin L \implies \mathbb{P}[A(x) = 1] \leq \frac{1}{3}. \tag{9.2}$$

The probability is over the internal randomness of A*.*

The constants $2/3$ and $1/3$ are arbitrary. Any constants strictly separated from $1/2$ would define the same class.

Lemma 9.2.4 (Error reduction)**.** *Suppose an algorithm outputs the correct answer with probability at least* $2/3$ *on every input. By repeating it independently* k *times and taking the majority vote, the error probability becomes at most* $e^{-\Omega(k)}$*.*

Sketch of Proof. Let X_i be the indicator that the i-th run is wrong. Then $\mathbb{E}[X_i] \leq 1/3$. The majority vote is wrong only if $\sum_i X_i \geq k/2$. A Chernoff bound implies that this probability is exponentially small in k. □

Thus, with $k = O(\log(1/\varepsilon))$ repetitions, one can reduce the error to ε, while keeping the running time polynomial if ε is inverse-polynomial or exponentially small.

9.2.3 NP: efficiently checkable certificates

The class NP captures problems for which yes-instances have efficiently verifiable certificates.

Definition 9.2.5 (NP)**.** *A language L is in* NP *if there exists a deterministic polynomial-time algorithm V, called a* ***verifier****, and a polynomial p, such that*

$$x \in L \implies \exists y \in \{0,1\}^{p(|x|)} \text{ such that } V(x,y) = 1, \tag{9.3}$$

$$x \notin L \implies \forall y \in \{0,1\}^{p(|x|)},\ V(x,y) = 0. \tag{9.4}$$

The string y is called a ***witness*** *or* ***certificate****.*

A problem is **NP-complete** if it lies in NP and every problem in NP reduces to it in polynomial time. Informally, the NP-complete problems are the hardest problems in NP. Famous examples include SAT, 3-SAT, graph coloring, and the traveling salesperson decision problem.

Example 9.2.6 (Satisfiability)**.** *Let L_{SAT} be the set of satisfiable Boolean formulas. The decision problem is: given a formula φ, decide whether there exists an assignment of its variables that makes φ true. This problem is in* NP *(the certificate is a satisfying assignment), and is* NP*-complete.*

9.2.4 PSPACE and EXP

Time is not the only important resource. We can also measure memory.

Definition 9.2.7 (PSPACE)**.** PSPACE *is the class of languages decidable by a deterministic classical algorithm using* $\mathrm{poly}(n)$ *bits of memory. The running time may be much larger than polynomial.*

Definition 9.2.8 (EXP)**.** EXP *is the class of languages decidable by a deterministic classical algorithm in time* $2^{\mathrm{poly}(n)}$*.*

A polynomial-time algorithm uses at most polynomial space, because it cannot write more than polynomially many bits in polynomially many steps. Also, a polynomial-space algorithm has at most exponentially many configurations, so if it halts then it can be simulated in exponential time. Thus,

$$\mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}. \tag{9.5}$$

Moreover $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$. The inclusion $\mathsf{NP} \subseteq \mathsf{PSPACE}$ follows by trying all possible certificates one after another, reusing the same polynomial-size memory.

Known versus believed. We know by the time hierarchy theorem that $\mathsf{P} \neq \mathsf{EXP}$. However, many central separations remain open, including P versus NP, P versus PSPACE, and BPP versus P.

Class	Main resource	Informal interpretation
P	deterministic polynomial time	efficiently solvable classically
BPP	randomness plus polynomial time	efficiently solvable with small error
NP	polynomial-time verification	yes-instances have short certificates
PSPACE	polynomial memory	solvable using limited memory
EXP	exponential time	solvable by exhaustive exponential computation

Table 9.1: A quick reference table for the classical classes used in this chapter.

9.3 Quantum circuits and the class BQP

We now introduce the quantum analogue of BPP. In this chapter, a quantum computation is a polynomial-size quantum circuit acting on polynomially many qubits, followed by a measurement of one output qubit. The input x is written into the first n qubits as the computational-basis state $|x\rangle$, with extra work qubits initialized to $|0\rangle$.

There is one technical point. A single circuit has a fixed number of input qubits, so an algorithm for all input lengths is really a **circuit family** $\{Q_n\}_{n\geq 1}$, one circuit for each input length n.

Definition 9.3.1 (Uniform polynomial-size quantum circuit family)**.** *A family of quantum circuits $\{Q_n\}_{n\geq 1}$ is **polynomial-size** if there is a polynomial $q(n)$ (which may depend on the family) such that Q_n has at most $q(n)$ gates and acts on at most $q(n)$ qubits.*
*It is **uniform** if there is a deterministic classical polynomial-time algorithm which, on input 1^n, outputs a description of Q_n.*

Uniformity rules out hiding hard computations in the description of the circuits.

Definition 9.3.2 (BQP)**.** *A language L is in* BQP *if there exists a uniform polynomial-size quantum circuit family $\{Q_n\}$ such that, for every input $x \in \{0,1\}^n$,*

$$x \in L \implies \mathbb{P}[Q_n(x) \text{ outputs } 1] \geq \frac{2}{3}, \tag{9.6}$$

$$x \notin L \implies \mathbb{P}[Q_n(x) \text{ outputs } 1] \leq \frac{1}{3}. \tag{9.7}$$

The probability is over the final measurement outcome.

In this definition, by $\mathbb{P}[Q_n(x)$ outputs $b]$ (for some $b \in \{0,1\}$) we mean the probability that, by executing the circuit Q_n on input $|x\rangle|0\rangle$, where there are as many $|0\rangle$ qubits as needed, and measuring the designated output qubit, the outcome b is obtained.

As for BPP, the constants $2/3$ and $1/3$ are not important: the error can be reduced by independent repetitions and majority vote.

Remark 9.3.3 (Gate sets)**.** *One usually fixes a finite universal gate set, for example $\{H, T, \mathrm{CNOT}\}$. Changing from one reasonable universal gate set to another changes circuit sizes only by polynomial factors, provided gates are approximated to sufficiently high precision. Thus* BQP *is a robust class.*

Example 9.3.4 (Problems in BQP)**.** *The following problems are in* BQP*:*

(i) every problem in P*;*

(ii) every problem in BPP*;*

(iii) the standard decision versions of integer factoring and discrete logarithm, and the corresponding search problems, by Shor's algorithm;

(iv) certain quantum-simulation tasks, such as estimating observables of local quantum systems after polynomial-time evolution under suitable promises.

9.3.1 Why BPP $\subseteq$ BQP

Quantum computers can simulate randomized classical algorithms.

Proposition 9.3.5.

$$\mathsf{BPP} \subseteq \mathsf{BQP}. \tag{9.8}$$

Sketch of Proof. A BPP computation is a deterministic polynomial-time computation supplied with random bits. Suppose the randomized algorithm uses $r(n)$ random bits. A quantum circuit can create the uniform superposition

$$\frac{1}{\sqrt{2^{r(n)}}} \sum_{r \in \{0,1\}^{r(n)}} |r\rangle \tag{9.9}$$

by applying Hadamard gates to $r(n)$ fresh qubits. Equivalently, these qubits could be measured at the start to produce independent random bits.

The deterministic part of the classical algorithm can be implemented reversibly with polynomial overhead, using gates such as Toffoli gates. On input x and random string r, the reversible circuit computes the same answer $A(x,r)$ as the classical randomized algorithm. Measuring the output qubit therefore accepts with exactly the same probability as A. Hence every BPP language is in BQP. □

So we have the basic inclusions

$$\mathsf{P} \subseteq \mathsf{BPP} \subseteq \mathsf{BQP}. \tag{9.10}$$

The major open question is whether either inclusion is strict. We believe BQP contains problems outside BPP, but we do not know how to prove this!

9.4 Where does BQP live?

The following diagram is a useful mental picture. It should not be interpreted as a theorem about all separations: many of the separations suggested by the drawing are unproved.

According to the diagram, BQP is a subset of PSPACE. Why is this the case? We show it in the next subsections.

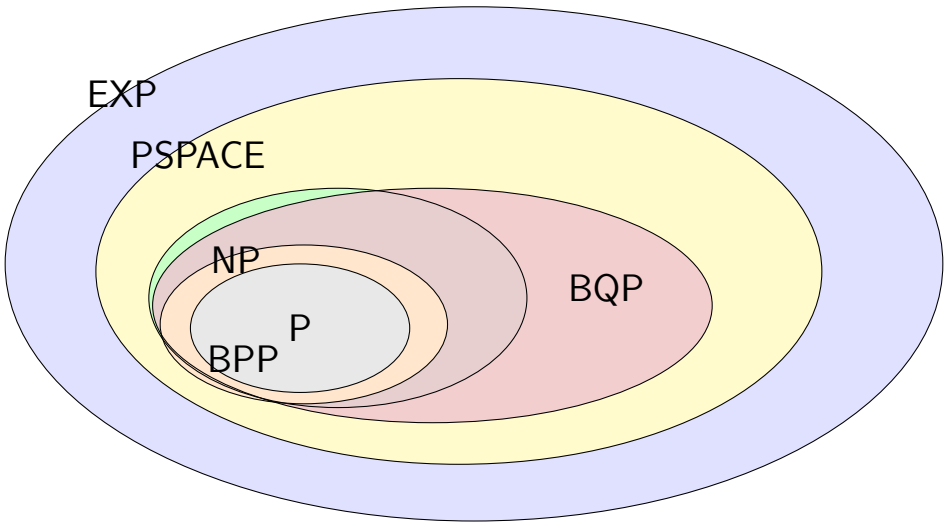

Figure 9.1: A schematic picture of some complexity classes. The proven containments $\mathsf{P} \subseteq \mathsf{BPP} \subseteq \mathsf{BQP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$ are the important message. The precise relationship between BQP and NP is not known.

9.4.1 A simple exponential-time simulation

If the total number of qubits is m, a quantum state is a vector in a 2^m-dimensional complex vector space:

$$|\psi\rangle = \sum_{x \in \{0,1\}^m} \alpha_x |x\rangle . \tag{9.11}$$

A direct classical simulation stores all 2^m amplitudes α_x. If the circuit has T one- and two-qubit gates, then applying each gate updates only local parts of the vector, but the vector still has length 2^m. This gives a simulation using roughly

$$O(T 2^m) \text{ time and } O(2^m) \text{ space.} \tag{9.12}$$

For a BQP computation we have $m, T = \mathrm{poly}(n)$, where n is the input length, so this proves the crude upper bound $\mathsf{BQP} \subseteq \mathsf{EXP}$.

9.4.2 Polynomial-space simulation by summing over paths

Surprisingly, one does not need to store the full state vector if the goal is only to compute a final acceptance probability. This gives a stronger upper bound.

Theorem 9.4.1.

$$\mathsf{BQP} \subseteq \mathsf{PSPACE}. \tag{9.13}$$

Sketch of Proof. Let

$$C = G_T G_{T-1} \cdots G_1 \tag{9.14}$$

be a quantum circuit on $m = \mathrm{poly}(n)$ qubits and $T = \mathrm{poly}(n)$ gates. Suppose the initial basis state is $|y_0\rangle$, which encodes the input x and some ancillas initialized to 0. For a final basis state $|z\rangle$, the amplitude of $|z\rangle$ after the circuit is

$$\langle z | C | y_0 \rangle = \langle z | G_T G_{T-1} \cdots G_1 | y_0 \rangle . \tag{9.15}$$

Insert the identity

$$I = \sum_{y \in \{0,1\}^m} |y\rangle \langle y| \tag{9.16}$$

between every pair of gates. We obtain

$$\langle z| C |y_0\rangle = \sum_{y_1,\ldots,y_{T-1} \in \{0,1\}^m} \langle z| G_T |y_{T-1}\rangle \langle y_{T-1}| G_{T-1} |y_{T-2}\rangle \cdots \langle y_1| G_1 |y_0\rangle . \tag{9.17}$$

The tuples $(y_1, \ldots, y_{T-1})$ are often called **Feynman paths**. There are exponentially many such paths, so this expression does not give a polynomial-time simulation. But it can be evaluated using only polynomial space: enumerate the paths one by one, keep the current path, the current product of transition amplitudes, and an accumulated sum.

The acceptance probability is a sum of squared magnitudes of such amplitudes over all accepting final basis states z, e.g.

$$\mathbb{P}[C(x) \text{ outputs } 1] = \sum_{z=(z_1,\ldots,z_m):\ z_1=1} \left| \langle z| C |y_0\rangle \right|^2 .$$

This outer sum also has exponentially many terms, but it can again be evaluated by enumeration using only polynomial space. For fixed gate sets such as $\{H, T, \mathrm{CNOT}\}$, every transition amplitude has an efficient classical description, and the enumeration can round accumulated sums to $\mathrm{poly}(n)$ bits of precision. Since the gap between acceptance probability at least $2/3$ and at most $1/3$ is constant, this much precision suffices to decide which case holds. □

The theorem $\mathsf{BQP} \subseteq \mathsf{PSPACE}$ says that quantum computation does not exceed polynomial space. It does *not* give an efficient classical simulation, because the time used by the path-sum algorithm is exponential.

9.5 Most problems are hard, even for quantum computers

Quantum computation is powerful, but it is not magic. A counting argument shows that most Boolean functions require exponentially many gates, even for quantum circuits over a fixed finite universal gate set.

Theorem 9.5.1 (Counting lower bound)**.** *For every sufficiently large n, all but a vanishing fraction of Boolean functions $f : \{0,1\}^n \to \{0,1\}$ require circuits of size $\Omega(2^n/n)$ to be computed with bounded error, over any fixed finite gate set.*

Sketch of Proof. There are

$$2^{2^n} \tag{9.18}$$

Boolean functions on n input bits. On the other hand, the number of circuits of size at most s over a fixed finite gate set is at most

$$2^{O(s \log s)}. \tag{9.19}$$

Indeed, each gate can be described by choosing its type and the wires it acts on, which takes $O(\log s)$ bits once the circuit has size s.

If circuits of size s computed all Boolean functions, we would need

$$2^{O(s \log s)} \geq 2^{2^n}, \tag{9.20}$$

which implies $s \log s = \Omega(2^n)$ and therefore $s = \Omega(2^n/n)$. □

This theorem is not specific to quantum computing. Its message is broader: most computational tasks have no small circuit at all. Complexity theory is about understanding the special structure that makes some problems solvable efficiently.

The counting lower bound is nonconstructive. It proves that hard functions exist in abundance, but it does not identify a natural explicit function, such as SAT, that requires exponential-size circuits.

9.6 What evidence do we have about the power of BQP?

We know several inclusions:

$$\mathsf{P} \subseteq \mathsf{BPP} \subseteq \mathsf{BQP} \subseteq \mathsf{PP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}. \tag{9.21}$$

We also know that $\mathsf{P} \neq \mathsf{EXP}$. But we do not know whether

$$\mathsf{BQP} \subseteq \mathsf{BPP} \tag{9.22}$$

or whether

$$\mathsf{NP} \subseteq \mathsf{BQP}. \tag{9.23}$$

Both questions are expected to have a negative answer.

9.6.1 Evidence that BQP is larger than BPP

The strongest examples of quantum speedups come from structured problems.

Example 9.6.1 (Factoring)**.** *Shor's algorithm solves the factoring search problem in time polynomial in* $\log N$ *on a quantum computer. Consequently, standard decision versions of factoring are in* BQP*. No polynomial-time classical factoring algorithm is known, so factoring is a candidate for a quantum speedup beyond* BPP*.*

It is important not to overinterpret this example. Factoring is not known, and is not believed, to be NP-complete; standard decision versions lie in $\mathsf{NP} \cap \mathsf{coNP}$. Shor's algorithm is therefore evidence for quantum speedups on special structured problems, not evidence that quantum computers solve arbitrary NP-complete problems.

This is not a proof that $\mathsf{BPP} \neq \mathsf{BQP}$, because we cannot prove that factoring is outside BPP. Complexity theory contains many such situations: we often have strong evidence for separations long before we can prove them.

Example 9.6.2 (Oracle evidence)**.** *Simon's problem gives an exponential quantum query speedup over classical randomized query algorithms. This can be converted into an oracle O such that $\mathsf{BQP}^O \not\subseteq \mathsf{BPP}^O$. Such results do not prove $\mathsf{BPP} \neq \mathsf{BQP}$ in the ordinary, non-oracle world, but they show that black-box access to information can be much more powerful quantumly than classically.*

9.6.2 Evidence that NP-complete problems are not in BQP

A common misconception is that a quantum computer solves an NP-complete problem by trying all possible witnesses in superposition and then measuring a correct one. The problem is that measurement does not reveal all branches of a superposition. A quantum algorithm must arrange interference so that wrong answers cancel and correct answers are amplified. This is possible for some highly structured problems, but not for arbitrary search.

The cleanest illustration is unstructured search. Suppose there is an unknown marked item in a list of size N, and the only way to learn about the list is to query an oracle.

Theorem 9.6.3 (Grover optimality, informal)**.** *Grover's algorithm finds a marked item using $O(\sqrt{N})$ queries. Every quantum algorithm for unstructured search requires $\Omega(\sqrt{N})$ queries.*

This lower bound is important. If the only approach to SAT were to search among 2^n possible assignments without exploiting additional structure, then Grover's algorithm would reduce the time from $O(2^n)$ to $O(2^{n/2})$, but not to polynomial time. Related oracle results show that purely black-box reasoning cannot prove that NP-complete problems are in BQP.

9.7 Quantum advantage

The term **quantum advantage** refers to a demonstration that a quantum device performs some computational task better than classical computers. It is a task-dependent statement: a device may show advantage for one sampling task while still being useless for another application. It can also be a moving target, because improved classical algorithms may later reduce or remove the observed gap. Different communities use the phrase in slightly different ways, so it is helpful to separate three meanings.

(i) **Asymptotic advantage.** A family of tasks has a quantum algorithm with better asymptotic scaling than any known classical algorithm. Shor's algorithm is the paradigmatic example.

(ii) **Provable query advantage.** In a query model, one proves that quantum algorithms use fewer oracle queries than classical algorithms. Simon's problem and Grover search are examples.

(iii) **Experimental sampling advantage.** A near-term quantum device samples from a distribution that appears hard for classical computers to sample from. Random circuit sampling is a typical example.

9.7.1 Sampling tasks

Many near-term demonstrations are not decision problems. They are **sampling problems**. Given a quantum circuit C on n qubits, define the output distribution

$$p_C(x) = |\langle x| C |0^n\rangle|^2, \qquad x \in \{0,1\}^n. \tag{9.24}$$

The task is to output samples x distributed approximately according to p_C.

This differs from computing the full distribution. Writing down all probabilities $p_C(x)$ already takes 2^n space. A quantum device samples by running the circuit and measuring; it never writes down the full distribution.

Example 9.7.1 (Random circuit sampling)**.** *Choose a random-looking quantum circuit C, run it on $|0^n\rangle$, and measure all qubits. The goal is to generate samples from the distribution p_C. For some idealized sampling models, efficient exact, or even approximate, classical sampling would imply unlikely consequences in complexity theory.*

There are two important caveats.

(i) **Verification is subtle.** For small enough circuits, a classical computer can compute some probabilities and compare with the quantum device. For larger circuits, full verification becomes difficult.

(ii) **Usefulness is separate from hardness.** A sampling task may be hard for classical computers but not immediately useful for an application. This is still scientifically meaningful: it probes the boundary between classical and quantum computation.

9.7.2 What makes a good quantum-advantage task?

A convincing near-term task should have the following features:

(i) **Implementability:** the quantum circuit can be run on available hardware with sufficiently low noise;

(ii) **Classical hardness evidence:** there are reasons to believe no efficient classical algorithm can perform the same task;

(iii) **Verifiability:** the output can be checked or benchmarked at least partially;

(iv) **Robustness:** the hardness evidence should tolerate realistic approximation and noise.

These requirements pull in different directions. A task that is easy to verify may also be easy to simulate. A task that is very hard to simulate may be hard to verify. Designing tasks that balance these requirements is one of the central challenges in quantum complexity and near-term quantum computing.

9.8 Summary of inclusions and open questions

Here is the main complexity map from this chapter:

$$\mathsf{P} \subseteq \mathsf{BPP} \subseteq \mathsf{BQP} \subseteq \mathsf{PP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}. \tag{9.25}$$

The most important inclusions are

$$\mathsf{P} \subseteq \mathsf{BPP} \subseteq \mathsf{BQP} \subseteq \mathsf{PSPACE}. \tag{9.26}$$

The first two inclusions are straightforward simulations. The inclusion $\mathsf{BQP} \subseteq \mathsf{PSPACE}$ follows from summing over exponentially many computational paths using only polynomial memory.

The main open questions include:

(i) Is $\mathsf{BPP} = \mathsf{BQP}$? Equivalently, can every efficient quantum computation be efficiently simulated classically?

(ii) Is $\mathsf{NP} \subseteq \mathsf{BQP}$? Equivalently, can quantum computers efficiently solve every problem with efficiently checkable certificates?

(iii) Are there practically useful quantum advantages before large-scale fault-tolerant quantum computers are available?