

Quantum Computing (CST Part II)

Lecture 12: Quantum Complexity

By any objective standard, the theory of computational complexity ranks as one of the greatest intellectual achievements of humankind.

Scott Aaronson

Recap: Church-Turing

The Church-Turing thesis states:

A function on the natural numbers can be calculated by an effective method if and only if it is computable by a Turing machine.

We know that quantum computing does not violate the Church-Turing thesis, which concerns **computability**, however we have reason to suspect that it may violate the Strong Church-Turing thesis:

Any algorithmic process can be simulated efficiently using a probabilistic Turing machine,

which concerns **complexity** (here “efficiently” is taken to mean, with only a polynomial time overhead).

Quantum complexity: the big picture

- We have already seen that quantum mechanics enables information processing tasks that cannot be achieved classically, for example superdense coding, and quantum key distribution.
- Furthermore, we have seen that quantum mechanics definitely allows some computational tasks to be achieved more quickly than in the classical case, for example Grover search...
- ...and in some cases, this speed-up is apparently exponential, for example *Shor's algorithm*.
- Indeed, the nature of entangled spaces seems to be such that quantum mechanical systems fundamentally have exponentially more computational power than classical systems do.

All this begs the question of whether there is some fundamental complexity class separation between tasks that can be achieved efficiently (in polynomial time) quantumly and those that can be classically.

Finite automata

A finite automaton consists of:

- A set of n_s states.
- An input alphabet of size n_a .
- A set of state transitions: usually represented in the form of a $n_s \times n_s$ matrix for each of the n_a letters.
- An initial “start” state.
- An “accept” state (marked by a black circle).

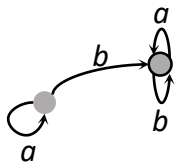
The *accepted language* is the set of strings of letters from the alphabet, such that the final state is the “accept” state.

Note that in general there could be multiple accept states (but in this lecture we only deal with examples with a single accept state).

Deterministic finite automata

Deterministic automata have the property that **for each state-letter pair, there is only one outgoing arrow**. The transition matrices are such that, if $|i\rangle$ is the start state, and $s_1 s_2 \cdots s_n$ is input the string, then the final state is $|f\rangle = M_{s_n} M_{s_{n-1}} \cdots M_{s_1} |i\rangle$ (where M_{s_i} is the transition matrix for the i th symbol, s_i). It follows that **the transition matrices of deterministic finite automata have exactly one 1 in each column**.

Example:



With the left-hand state, $|0\rangle$, as the starting state (and $|1\rangle$ is the accept state).

The transition matrices are:

$$M_a = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} ; M_b = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

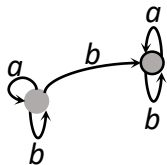
It can be seen that the accepted language is the set of all strings containing at least one "b", e.g., if the string abb is read we get, $M_b M_b M_a$ multiplied by the start state:

$$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Nondeterministic finite automata

Nondeterministic finite automata can have any number of outgoing arrows for each state-letter pair, so the transition matrices are general binary matrices. A string is part of the accepted language if there is *some* path finishing in the accepted state.

Example:



With the left-hand state, $|0\rangle$, as the starting state (and $|1\rangle$ is the accept state).

In this example, the transition matrices are:

$$M_a = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and

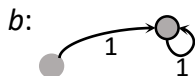
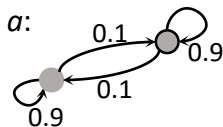
$$M_b = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

And it is still the case that the accepted language is the set of all strings containing at least one "b".

Probabilistic automata

Probabilistic automata are essentially *Markov chains*, with state transitions being probabilistic, and thus **the transition matrices contain fractional values, such that each column sums to one**. The accepted language can be defined either as the set of all strings that end in the final state with certainty, or with probability above some threshold.

Example:



With the left-hand state, $|0\rangle$, as the starting state (and $|1\rangle$ is the accept state).

In this example, the transition matrices are:

$$M_a = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}$$

and

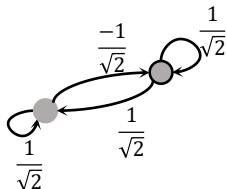
$$M_b = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

So all strings (with at least one symbol) are accepted with some probability, and all strings ending with a “ b ” are accepted with certainty.

Quantum automata

In **quantum automata**, the transition matrices are unitary matrices consisting of positive and negative complex numbers. A special case of quantum automata are **reversible automata**, where the transition matrices are **binary permutation matrices** (exactly one 1 in each column and row).

Example:



With the left-hand state, $|0\rangle$, as the starting state (and $|1\rangle$ is the accept state).

This single-letter alphabet automaton has transition matrix:

$$M_a = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

We can see that:

$$M_a M_a = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

So, if we start in state $|0\rangle$ there are two paths of length 2 from $|0\rangle$ back to itself ($|0\rangle \rightarrow |0\rangle \rightarrow |0\rangle$ and $|0\rangle \rightarrow |1\rangle \rightarrow |0\rangle$), but these **interfere** in such a way that there is actually zero chance of ending up in $|0\rangle$.

Turing machines

Turing machines are deterministic finite automata, equipped with an infinitely long read-write tape, upon which the input string is initially written. At any time a “head” is over one space on the tape, and can read the symbol written there (initially the head is at the left-hand end of the tape). The action of a Turing machine is thus:

At a given time, the DFA is in a certain state, and the head is over a symbol which it reads. Given this state-symbol pair, a transition function determines:

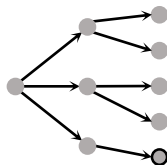
- Which symbol to overwrite on the current space on the tape.
- Whether to move the head left or right.
- Which next state the DFA moves to.

A Turing machine accepts the input if it halts in an accept state.

The Turing machine is a sufficiently general model for computation to capture entirely that which can reasonably be thought of as mathematically computable. The class of problems that can be decided in polynomial-time on a Turing machine is denoted P.

Nondeterministic Turing machines

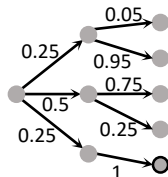
If instead of a single action (overwritten symbol, move left or right and state transition) we allow a set of possible actions, then we get the *nondeterministic Turing machine*. This can be thought of as a tree, where each branching process represents the variety of possible actions at a given time. A string is accepted if there is some path through the tree to an accept state.



If the height of the tree is bounded by a polynomial in the length of the input string, then the language is in **NP**. Clearly $P \subseteq NP$, as each decision point could just consist of one single branch.

Probabilistic Turing machines

Probabilistic Turing machines are similar in appearance to nondeterministic Turing machines, but now the branches represent a probability distribution over possible next actions:



The complexity class **BPP** is the set of languages, L , for which there is a probabilistic Turing machine, M , running in polynomial time with:

$$P(M \text{ accepts } w) = \begin{cases} > \frac{2}{3} & \text{if } w \in L \\ < \frac{1}{3} & \text{if } w \notin L \end{cases}$$

Note $\frac{2}{3}$ is arbitrary – all that is required is that we have a constant fraction greater than $\frac{1}{2}$ (and similarly one less than $\frac{1}{2}$ for the $\frac{1}{3}$ term). Clearly each probability distribution in a probabilistic Turing machine could consist of a single deterministic branch, so $P \subseteq BPP$.

Quantum Turing machines

Quantum Turing machines are like probabilistic Turing machines, but now complex amplitudes are associated with each possible next move. It is also necessary that the linear transformation defined by the machine is unitary.

The complexity class **BQP** is the set of languages, L , for which there is a quantum Turing machine, M , running in polynomial time with:

$$P(M \text{ accepts } w) = \begin{cases} > \frac{2}{3} & \text{if } w \in L \\ < \frac{1}{3} & \text{if } w \notin L \end{cases}$$

It has been shown that the quantum Turing machine generalises the probabilistic Turing machine, so **BPP** \subseteq **BQP**.

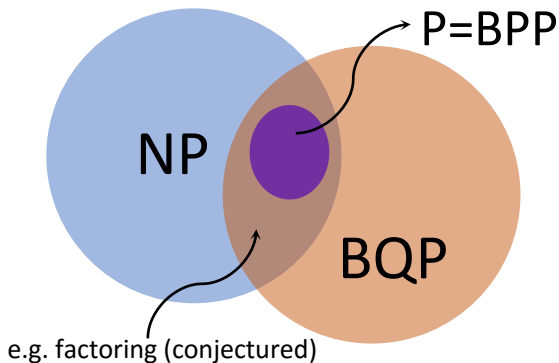
Relationships between complexity classes

Many complexity class inclusions remain open problems in theoretical computer science. We have already seen that $P \subseteq NP$ and $P \subseteq BPP \subseteq BQP$, but are these proper subsets?

- $P = NP$? is the most famous open problem in theoretical computer science, and the vast majority of theorists believe $P \neq NP$.
- We do not know whether BPP is a subset of NP or vice versa, but it is conjectured that $P = BPP$.
- As factoring is widely believed to be super-polynomial classically (even with a probabilistic Turing machine), the existence of Shor's algorithm is taken as evidence that $BPP \neq BQP$.
- It is widely believed that NP -complete problems cannot be solved in polynomial time on a quantum computer (unless $P = NP$), so $NP \not\subseteq BQP$.
- ...but it is also believed that there are problems outside of NP which *can* be solved in polynomial time on a quantum computer, so $BQP \not\subseteq NP$.

Complexity class inclusions

A Venn diagram of complexity class inclusions as we conjecture them:



How important is the question $BPP \stackrel{?}{=} BQP$

The most spectacular results in quantum computing concern exponential speed-ups over the best-known classical algorithms. Therefore we may ask how significant it is that the exponentially superior power of quantum computing is only conjectured, and not known for certain.

- Without question, were it to be proved that $BPP = BQP$ this would send shock-waves not only through the quantum computing community, but through theoretical computer science in general. But not many theorists believe this is even remotely likely.
- It is, however, worth noting that the exponential speed-up is not the only reason for pursuing quantum computing: algorithms with guaranteed polynomial speed-ups, such as Grover's are increasingly being seen as compelling use-cases for quantum computing.
- More speculatively, we do know of an exponential separation between quantum and classical computational power when *query* complexity is the figure of merit.
- Finally, using quantum computers to manipulate quantum states will enable other advantageous applications of quantum information processing such as QKD and superdense coding.

Sampling and quantum supremacy

- In general, when we measure a quantum state the outcome is not deterministic and hence we are **sampling a random variable**.
- Therefore a different approach to demonstrating the superior power of quantum computing is to show that the measurement statistics from such a random process are hard to replicate classically.
- Most notably, if we measure the quantum state prepared by a class of circuits termed “IQP circuits”, we obtain samples from a probability distribution that cannot be efficiently prepared classically unless the *polynomial hierarchy* collapses to its third level (which is considered to be extremely unlikely to happen).
- Most *quantum supremacy* experiments (i.e., experiments aiming to demonstrate the exponentially superior power of quantum computing) are based on sampling as, with a bit of ingenuity, they can be performed on relatively small and noisy quantum computers. By contrast, aiming to demonstrate quantum supremacy by performing some algorithm (say Shor’s) would require an enormous, error-corrected quantum computer.

Summary

In this lecture we have covered:

- The bigger picture of quantum computability and complexity.
- Various finite automata, including quantum automata.
- Turing machines and complexity classes.
- The practical relevance of $BPP \stackrel{?}{=} BQP$.
- Sampling and quantum supremacy.