TECHNISCHE UNIVERSITÄT DRESDEN

International Center for Computational Logic

COMPLEXITY THEORY

Lecture 25: Quantum Computing (2)

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 27th Jan 2025

More recent versions of this slide deck might be available.
For the most current version of this course, see
https://iccl.inf.tu-dresden.de/web/Complexity_Theory/en

Quantum computing

Quantum computing currently is our main hope for building physical computers that may in some cases perform exponentially better than deterministic Turing machines.

- Quantum computers exploit the rules of Quantum Mechanics
- Constructing a real quantum computer is an open engineering challenge of high complexity
- The properties of such computers, however, can be studied with relatively simple mathematical tools, and without any of the underlying physical interpretations of the theory

Review: The Quantum World

The following mathematical abstraction delineates the basis for using Quantum Mechanics in computation:

- Quantum systems that have a measurable attribute can represent one bit of information
- Single bits can be combined into larger quantum registers of many states (representing many possible combinations of bits)
- The state of a quantum system is described by a probability distribution over its possible discrete values
- Probabilities p are represented by complex amplitudes q such that $q^2 = p$
- Amplitude vectors of quantum states must therefore be unit vectors in the Euclidian norm (2-norm)
- Quantum systems are manipulated by linear, norm-preserving mappings on their amplitude vectors
- Such mappings can be represented by unitary matrices

The rest is linear algebra.

Computing with qubits

We have already defined the basic approach for representing data in registers of m qubits.

Which operations can we use on such data?

- Mathematically, any unitary $2^m \times 2^m$ matrix can be applied
- However, operations that manipulate an arbitrary number of bits in one step are too powerful (both for practical implementation and for theoretical definition of computational power)

We therefore restrict to “local” operations, that only affect some bits within a large register.

Computing with qubits

We have already defined the basic approach for representing data in registers of m qubits.

Which operations can we use on such data?

- Mathematically, any unitary $2^m \times 2^m$ matrix can be applied
- However, operations that manipulate an arbitrary number of bits in one step are too powerful (both for practical implementation and for theoretical definition of computational power)

We therefore restrict to “local” operations, that only affect some bits within a large register.

Making local operations global: Given a $2^m \times 2^m$ matrix A that modifies m qubits, we obtain an operation on the qubits number $i+1, \ldots, i+m$ in an n qubit register as the Kronecker product $I_i \otimes A \otimes I_{n-i-m}$, where I_k is the $2^k \times 2^k$ identity matrix (cf. the calculations we did for the EPR Paradox).

In practice, local operations may be truly local, as they are performed on a subset of the particles of some system only. The expansion of the matrix to the state of the whole system is purely conceptual and has no physical analogy.

Common local operations (1)

- Flipping a bit (negation) can be achieved with a matrix

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Common local operations (1)

- Flipping a bit (negation) can be achieved with a matrix

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

- Rotating a bit by θ is achieved by

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Common local operations (1)

- Flipping a bit (negation) can be achieved with a matrix

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

- Rotating a bit by θ is achieved by

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

- Exchanging two bits (swap) is achieved by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Common local operations (2)

- Copying a bit

Common local operations (2)

- Copying a bit is impossible (no-cloning theorem)!

Common local operations (2)

- Copying a bit is impossible (no-cloning theorem)!
- Copying a bit to a new qubit that was initialised to $|0\rangle$ is possible by implementing an operation $|xy\rangle \mapsto |x(x \oplus y)\rangle$, called controlled not (CNOT):

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Common local operations (2)

- Copying a bit is impossible (no-cloning theorem)!
- Copying a bit to a new qubit that was initialised to $|0\rangle$ is possible by implementing an operation $|xy\rangle \mapsto |x(x \oplus y)\rangle$, called controlled not (CNOT):

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- The Hadamard operation is given by the matrix:

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

  Hadamard gates are used to create superposition, e.g., applying Hadamard to $|0\rangle$ yields $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, a state where 0 and 1 are equally probable.

Common local operations (3)

- Setting a bit to the conjunction of two bits

Common local operations (3)

- Setting a bit to the conjunction of two bits is impossible (unitary mappings are always reversible)!

Common local operations (3)

- Setting a bit to the conjunction of two bits is impossible (unitary mappings are always reversible)!
- Setting an unused bit to the conjunction of two bits can be achieved by an operation $|abc\rangle \mapsto |ab(c \oplus (a \wedge b))\rangle$. This is called a Toffoli gate:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

This is also called the controlled-controlled-not (CCNOT) gate.

Quantum circuits

To define computation, we assemble circuits from quantum gates:

Definition 25.1: A quantum operation is called elementary or a quantum gate if it operates on at most three bits of a quantum register.

Quantum circuits

To define computation, we assemble circuits from quantum gates:

Definition 25.1: A quantum operation is called elementary or a quantum gate if it operates on at most three bits of a quantum register.

Definition 25.2: A quantum circuit is a direct acyclic graph where all non-input nodes are labelled with quantum gates, and where the out-degree of all gates equals their in-degree, and the out-degree of all inputs equals 1. We allow special workspace inputs that are hard-wired to $|0\rangle$.

Note:

- Input nodes can only have one wire for each output due to the No-Cloning Theorem
- The number of input and output wires in gates must be equal since unitary mappings are reversible
- The overall operation of a quantum circuit is therefore reversible, too

Universal sets of gates

Can we restrict to a small set of universal quantum gates?

Problem:

- There are uncountably many quantum gates (for different complex number factors)
- Only countably many circuits can be built from a finite set of gates

Universal sets of gates

Can we restrict to a small set of universal quantum gates?

Problem:

- There are uncountably many quantum gates (for different complex number factors)
- Only countably many circuits can be built from a finite set of gates

However, there are (many) sets of gates that are universal up to approximation:

Theorem 25.3: Up to any precision ε, any $n \times n$ unitary matrix U can be approximated by a product of ℓ matrices $U_1, \ldots, U_\ell$, in the sense that

$$\left|U_{i,j} - (U_1 \cdots U_\ell)_{i,j}\right| < \varepsilon$$

where $\ell \leq 100(n \log(1/\varepsilon))^3$, and each matrix U_i corresponds to the application of a Toffoli gate, a Hadamard gate, or a phase-shift gate $\begin{pmatrix} 1 & 0 \\ 0 & \mathrm{i} \end{pmatrix}$ to at most three qubits.

(without proof)

Note: quantum computing retains the same power even without the phase shift gates.

BQP

We can use quantum circuits to define a complexity class:

Definition 25.4: A language $\mathbf{L} \subseteq \{0,1\}^*$ is in BQP (Bounded-Error Quantum Polynomial Time) if there is a polynomially time-bounded DTM that computes, on input 1^n the description of a quantum circuit C_n, such that:

- C_n has n inputs and, in addition, a polynomial number of constant $|0\rangle$ input gates (the latter are called ancilla bits);
- C_n has one designated main output;
- Measuring the value v of the main output of C_n on input $|w\rangle|0\cdots0\rangle$ obeys $\Pr[v = 1 \text{ iff } w \in \mathbf{L}] \geq \frac{2}{3}$.

BQP

We can use quantum circuits to define a complexity class:

Definition 25.4: A language $\mathbf{L} \subseteq \{0,1\}^*$ is in BQP (Bounded-Error Quantum Polynomial Time) if there is a polynomially time-bounded DTM that computes, on input 1^n the description of a quantum circuit C_n, such that:

- C_n has n inputs and, in addition, a polynomial number of constant $|0\rangle$ input gates (the latter are called ancilla bits);
- C_n has one designated main output;
- Measuring the value v of the main output of C_n on input $|w\rangle|0\cdots0\rangle$ obeys $\Pr[v = 1 \text{ iff } w \in \mathbf{L}] \geq \frac{2}{3}$.

In other words:

$(C_i)_i$ is a polynomial-time uniform family of quantum circuits with error probability $< \frac{1}{3}$.

Notes on the definition of BQP

- We can safely restrict to, e.g., Toffoli and Hadamard gates
- It is therefore not necessary that the DTM writes out numbers in gate matrices in decimal (which would not be possible)[1]
- As with BPP, the error probability of $\frac{1}{3}$ is not essential; other constant values $< \frac{1}{2}$ can be picked
- As with Boolean circuits, we can view quantum circuits as "straight line programs" where in each step, we apply one elementary operation to up to three inputs of an n-qubit register

[1] We generally assume that the numbers that occur in any matrix of the chosen basic set of gates can be computed to arbitrary precision

Simulating Boolean Circuits in Quantum Computers

We have seen quantum analogues of classical operations:

- Flip negates a qubit
- The Toffoli gate computes a logical AND, that can be stored onto a third ancilla qubit
- Similarly, one can define a quantum gate for a reversible version of OR

Simulating Boolean Circuits in Quantum Computers

We have seen quantum analogues of classical operations:

- Flip negates a qubit
- The Toffoli gate computes a logical AND, that can be stored onto a third ancilla qubit
- Similarly, one can define a quantum gate for a reversible version of OR

Combining these gates, we get:

Lemma 25.5: If $f : \{0,1\}^n \to \{0,1\}^m$ is computable by a Boolean circuit using s gates, then there is a quantum circuit of size linear in s that computes the mapping $|w\rangle|0^{m+O(s)}\rangle \to |w\rangle|f(w)\rangle|v\rangle$, for words $w \in \{0,1\}^n$, $f(w) \in \{0,1\}^m$, and $v \in \{0,1\}^{O(s)}$.

Note: The quantum circuit must use the same number of inputs and outputs, and requires zeroed ancilla bits for (in the worst case) each gate of the Boolean circuit; after use, the $O(s)$ ancilla bits are returned in the qubit values v.

Note: Linear overhead in s stems from the fact that classical circuits allow output bits of gates and the input bits of the circuit to be used by more than one gate as input. Use ancilla bits to create respective copies (CNOT gates).

Collecting garbage

The values of ancilla bits after computation may be problematic:

- Used up ancilla bits cannot be reused if the circuit is intended as a subroutine in a larger algorithm
- Ancilla bits might be entangled with the result, and possibly affect its measurement

Collecting garbage

The values of ancilla bits after computation may be problematic:

- Used up ancilla bits cannot be reused if the circuit is intended as a subroutine in a larger algorithm
- Ancilla bits might be entangled with the result, and possibly affect its measurement

Solution: clean up any such garbage by uncomputation:

Theorem 25.6: If $f : \{0,1\}^n \to \{0,1\}^m$ is computable by a Boolean circuit of size s, then there is a quantum circuit of size linear in $s+m+n$ that computes the mapping $|w\rangle|0^{2m+s}\rangle \to |w\rangle|f(w)\rangle|0^{m+s}\rangle$, for words $w \in \{0,1\}^n$ and $f(w) \in \{0,1\}^m$.

Collecting garbage

The values of ancilla bits after computation may be problematic:

- Used up ancilla bits cannot be reused if the circuit is intended as a subroutine in a larger algorithm
- Ancilla bits might be entangled with the result, and possibly affect its measurement

Solution: clean up any such garbage by uncomputation:

Theorem 25.6: If $f : \{0,1\}^n \to \{0,1\}^m$ is computable by a Boolean circuit of size s, then there is a quantum circuit of size linear in $s+m+n$ that computes the mapping $|w\rangle|0^{2m+s}\rangle \to |w\rangle|f(w)\rangle|0^{m+s}\rangle$, for words $w \in \{0,1\}^n$ and $f(w) \in \{0,1\}^m$.

Proof: The quantum circuit proceeds in steps:

(1) Use s gates as before to obtain a state $|w\rangle|f(w)\rangle|0^m\rangle|v\rangle$ with ancilla bits $v \in \{0,1\}^s$

(2) Copy $f(w)$ to the m unused qubits using CNOT gates, leading to state $|w\rangle|f(w)\rangle|f(w)\rangle|v\rangle$

(3) Apply the inverse operation to all gates used in step (1), in reverse order, using the first n and the final $m+s$ qubits.

Step (3) reverses the computation of $f(w)$ and of v, resulting in $|w\rangle|f(w)\rangle|0^{m+s}\rangle$ □

Classes below BQP

We have shown that deterministic polytime TMs can be simulated by logspace-uniform Boolean circuits (Theorem 19.7)

Using the simulation of Boolean circuits by quantum circuits, we immediately get:

Corollary 25.7: $\mathrm{P} \subseteq \mathrm{BQP}$.

Classes below BQP

We have shown that deterministic polytime TMs can be simulated by logspace-uniform Boolean circuits (Theorem 19.7)

Using the simulation of Boolean circuits by quantum circuits, we immediately get:

Corollary 25.7: $\mathrm{P} \subseteq \mathrm{BQP}$.

How about simulating random computation?

Classes below BQP

We have shown that deterministic polytime TMs can be simulated by logspace-uniform Boolean circuits (Theorem 19.7)

Using the simulation of Boolean circuits by quantum circuits, we immediately get:

Corollary 25.7: $\mathsf{P} \subseteq \mathsf{BQP}$.

How about simulating random computation?

We can use a Hadamard gate to produce "random bits" $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ that can simulate coin flips. Representing PTMs by deterministic TMs with a random string as certificate, one easily gets:

Corollary 25.8: $\mathsf{BPP} \subseteq \mathsf{BQP}$.

Classes above BQP

Can we simulate quantum computation on classical computers?

Classes above BQP

Can we simulate quantum computation on classical computers?

- Yes, if we represent complex numbers up to limited precision (this suffices)
- In particular, classical computers can do anything quantum computers can
- However, the quantum state of an n qubit register may involve 2^n complex amplitudes

$\rightsquigarrow$ A direct simulation needs exponential time

Classes above BQP

Can we simulate quantum computation on classical computers?

- Yes, if we represent complex numbers up to limited precision (this suffices)
- In particular, classical computers can do anything quantum computers can
- However, the quantum state of an n qubit register may involve 2^n complex amplitudes

$\rightsquigarrow$ A direct simulation needs exponential time

It turns out that one can do better:

Theorem 25.9: BQP $\subseteq$ PP ($\subseteq$ PSpace).

Proof idea for BQP $\subseteq$ PSpace: We can do a "backwards" computation of individual amplitudes in the final state by (1) considering each possible final base state $\{0,1\}^n$ in separation, and (2) tracing each of them back to the at most eight predecessor states that could contribute to its amplitude. The argument is similar to the evaluation of Boolean circuits space that is polynomial in their depth. □

BQP and NP

The relationship of BQP and NP is unknown.

- The best known BQP algorithm for NP problems offers a quadratic speedup (Grover's search algorithm)
- "Most researchers" believe NP $\not\subseteq$ BQP
- Conversely, there is an artificial problem (Recursive Fourier Sampling) in BQP that is not known to be in the Polynomial Hierarchy
- "Many researchers" believe BQP $\not\subseteq$ NP

Corollary 25.10: It is not known or expected that quantum computing yields exponential runtime improvements for NP-complete problems!

Quantum algorithms

There are a few known algorithms where quantum computing is expected to offer significant speed-ups over classical computers, for example:

Quantum algorithms

There are a few known algorithms where quantum computing is expected to offer significant speed-ups over classical computers, for example:

- Grover's search: Solving Boolean circuit satisfiability

  Theorem 25.11: There is a quantum algorithm that, given a Boolean circuit C of size n, runs in time $O(\text{poly}(n)2^{n/2})$, and returns an input on which C evaluates to 1.

Quantum algorithms

There are a few known algorithms where quantum computing is expected to offer significant speed-ups over classical computers, for example:

- Grover's search: Solving Boolean circuit satisfiability

  Theorem 25.11: There is a quantum algorithm that, given a Boolean circuit C of size n, runs in time $O(\text{poly}(n)2^{n/2})$, and returns an input on which C evaluates to 1.

- Shor's algorithm: Polytime Integer Factorization

  Theorem 25.12: There is a quantum algorithm that, given a natural number N, runs in time $O(\text{poly}(\log N))$ and outputs the prime factors of N.

Shor's algorithm

Basic ideas:

Shor's algorithm

Basic ideas:

- It suffices to find a single prime factor. One can then divide by this factor and repeat the algorithm to find the rest.

Shor's algorithm

Basic ideas:

- It suffices to find a single prime factor. One can then divide by this factor and repeat the algorithm to find the rest.
- One can solve this by computing, for random numbers A, the smallest r such that $A^r = 1 \mod N$. Such r is called the order of $A \mod N$.

Shor's algorithm

Basic ideas:

- It suffices to find a single prime factor. One can then divide by this factor and repeat the algorithm to find the rest.
- One can solve this by computing, for random numbers A, the smallest r such that $A^r = 1 \mod N$. Such r is called the order of $A \mod N$.
  - With high probability, r is even and $A^{r/2} - 1$ has non-trivial common factors with N
  - We can compute such factors classically, e.g., with a greatest-common divisor computation

Shor's algorithm

Basic ideas:

- It suffices to find a single prime factor. One can then divide by this factor and repeat the algorithm to find the rest.
- One can solve this by computing, for random numbers A, the smallest r such that $A^r = 1 \mod N$. Such r is called the order of $A \mod N$.
  - With high probability, r is even and $A^{r/2} - 1$ has non-trivial common factors with N
  - We can compute such factors classically, e.g., with a greatest-common divisor computation
- The mapping $A \mapsto A^k \mod N$ is computable in poly$(\log N)$ time classically (fast exponentiation)

Shor's algorithm

Basic ideas:

- It suffices to find a single prime factor. One can then divide by this factor and repeat the algorithm to find the rest.
- One can solve this by computing, for random numbers A, the smallest r such that $A^r = 1 \mod N$. Such r is called the order of $A \mod N$.
  - With high probability, r is even and $A^{r/2} - 1$ has non-trivial common factors with N
  - We can compute such factors classically, e.g., with a greatest-common divisor computation
- The mapping $A \mapsto A^k \mod N$ is computable in poly$(\log N)$ time classically (fast exponentiation)

At this point, the algorithm becomes somewhat complicated ...

- The problem of finding the order is further related to the period of a periodic number series that can be expressed in a quantum state
- The period is determined using a technique known as Quantum Fourier Transform (QFT), used in many quantum algorithms

Shor's Algorithm: Discussion

- Shor's algorithm is of practical interest since it would allow us to break RSA-type asymmetric encryption
- However, it won't happen overnight . . .
- . . . and there are quantum algorithms for encryption that would be able to safe us
- Moreover, symmetric encryption methods would mostly stay safe

Summary and Outlook

Quantum computing is an exciting alternative theory of computation that might become practice in some future

We know that $P \subseteq BPP \subseteq BQP \subseteq PP \subseteq PSpace$, but little more

There are many further topics on quantum computing not discussed here – algorithms, encryption, error correction, etc.

What's next?

- Interactive Proof Systems
- Summary & consultation
- Examinations