Quantum Information Processing: from Theory to Practice

Lecture 6: Classical vs Quantum Computation.
Grover's Algorithm and the Quantum Fourier Transform.

Outline

Basics of Computational Complexity Theory

Asymptotic Complexity Notation

Asymptotic notation[1] quickly describes a function's behavior as n grows large:

1. $f(n) = \mathcal{O}(g(n))$, if there exists an n_0 such that for any $n > n_0$, $f(n) \leq cg(n)$ with $c = \text{constant}$.
2. $f(n) = \Omega(g(n))$, if there exists an n_0 such that for any $n > n_0$, $f(n) \geq cg(n)$ with $c = \text{constant}$.
3. $f(n) = \theta(g(n))$, if $f(n)$ and $g(n)$ have the same asymptotic behavior up to a constant factor. If $f(n) = \mathcal{O}(g(n)) = \Omega(g(n))$ then $f(n) = \theta(g(n))$.

Examples:

- $2n$ is $\mathcal{O}(n^2)$
- 2^n is $\Omega(n^3)$
- $7n^2 + \sqrt{n}\log n$ is $\theta(n^2)$ since $7n^2 \leq 7n^2 + \sqrt{n}\log n \leq 8n^2$ for n sufficiently large
- if $g(n)$ is a polynomial of order k, $g(n) = \mathcal{O}(n^l)$ for $k \leq l$
- $n^{\log n}$ is superpolynomial: $n^k = \mathcal{O}(n^{\log n})$ for any k, but $n^{\log n} \neq \mathcal{O}(n^k)$ (not bounded above by any polynomial)

[1] Standard computer science style notation related to asymptotic complexity is used.

Complexity Theory

- Complexity theory is the branch of the theory of computation that deals with the resources required to solve a problem.
- The two most important types of resources are:
  - time, describing the number of computational steps used by an algorithm to solve a problem or to verify an answer
  - space, giving the amount of work space used by the algorithm
- Classically, the formal model underpinning the notion of an algorithm is the Turing machine[2].
- The informal approach is here considered where
  1. number of computational steps used by an algorithm $\rightarrow$ number of lines of the executed code to run it
  2. space usage $\rightarrow$ amount of memory (RAM) used by the algorithm.
- Computational complexity theory classifies problems in terms of their difficulty, that is, the resources required in order to solve them.

[2] A Turing machine is a theoretical model of computation describing an abstract machine that can perform any algorithm, no matter how complex it is, using a simple set of rules and an infinite tape. It was invented by Alan Turing in 1936, and it is considered the basis of modern computer science.

Complexity Theory

- Rather than looking at the complexity of algorithms for solving one particular instance of a problem, the theory considers asymptotics: given a family of problems, parameterized by their size n, the theory studies the resources used by the best possible algorithm for solving it.
- We can distinguish between two types of problems:
  - Easy problems: solution can be found in polynomial time, i.e. with a number of steps that is a polynomial function of the dimension n of the input. For example the addition of two n-bit integers scales as n.
  - Hard problems: superpolynomial in n. For example the factorization of an integer N requires a number of steps scaling as $\exp n$, if n is the number of digits of N.
- In computer science, superpolynomial functions describe the time complexity of some algorithms that are considered very inefficient or intractable.

Complexity Theory

A line of demarcation between efficient and inefficient algorithms is provided by the notion of polynomial-time computation, where an algorithm running in $\mathcal{O}(n^c)$ for some fixed c, is considered efficient.

Consider the following two problems:

> **Primality testing**
> Given an integer M expressed as n binary digits, is it a prime number?
>
> **Integer factorization**
> Given an integer M expressed as n binary digits, output the prime factors of M.

As the input is of size n, we would like to solve these problems using an algorithm which runs in polynomial time n.

- There are no known classical algorithms that solve the integer factorization problem in polynomial time (the situation is different for quantum algorithms).
- Surprisingly, there exists a classical algorithm in polynomial time for similar problem of primality testing.

Complexity Theory

- Decision problems are an important class of problems with a "yes" or "no" answer:

  Primality testing is a decision problem, whereas integer factorization is not!

- Actually, the integer factorization problem can be cast into a decision one without changing its underlying complexity significantly as follows:

> **Integer factorization (decision variant)**
> Given integers M and K expressed as n binary digits each, does M have a prime factor smaller than K?

- Solving the decision variant allows us to solve the search variant of the problem using binary search.
- Given an integer M, we can find its smallest prime factor F using $\mathcal{O}(\log M) = \mathcal{O}(n)$ calls to a decision factorization algorithm with varying K (e.g., $\lceil M/2 \rceil$, $\lceil M/4 \rceil$ or $\lceil 3M/4 \rceil$, etc.). Remaining factors are found by dividing M by F and repeating.
- What illustrated above is a simple example of a reduction, which consists in the conversion of one problem into another.

Example: Binary Search for the Smallest Prime Factor

Consider $M = 91$, and assume we have a decision algorithm (oracle) that tells us whether M has a prime factor less than or equal to K.

1. Initial call:
   Try $K = \lceil 91/2 \rceil = 46$. Assume the algorithm returns Yes (there is a factor ≤ 46).
2. Next, narrow down:
   Try $K = \lceil 91/4 \rceil = 23$. Suppose it says Yes again.
3. Try $K = \lceil 91/8 \rceil = 12$. Suppose it says Yes.
4. Try $K = \lceil 91/16 \rceil = 6$. Suppose it says No.
5. Try $K = \lceil 3 \cdot 91/32 \rceil = 9$. Suppose it says Yes.
6. Try $K = \lceil 5 \cdot 91/64 \rceil = 8$. Suppose it says Yes.

Now we know the smallest prime factor is 7. Then:

$$91/7 = 13 \Rightarrow \text{Next factor is } 13$$

So the factorization is 7×13, both primes!

Classical and Quantum Computational Complexity

Classical Computational Complexity

A natural way to compare the complexity of problems is via the notion of complexity classes, where a complexity class is simply a categorization into a set of problems with their related complexity.
Some important classical complexity classes are[3]:

- P: the class of decision problems that can be solved in polynomial time by a classical computer (easy problems).
- NP: the class of decision problems whose solution can be verified in polynomial time.
- NP-complete: any NP problem that can be reduced to an NP-complete problem in polynomial time and whose solution can still be verified in polynomial time.

Complexity classes are useful in classifying problems according to the amount of time (how many steps it takes to solve a problem) and space (how much memory it takes to solve a problem) needed to solve them and validate their solutions.

[3] P class refers to decision problems solvable by a deterministic machine in polynomial time. NP class includes decision problems solvable by a non-deterministic machine in polynomial time.

Quantum Computational Complexity

- The basic framework where quantum computation operates is that defined by a system of n qubits in which a basis $\{|0\rangle, |1\rangle\}$ is chosen for each of them.
- By taking tensor products, this gives the computational basis $\left\{|x\rangle : x \in \{0,1\}^n\right\}$ for the whole system, written as $|x\rangle$ for $|x_1\rangle |x_2\rangle \cdots |x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \cdots \otimes |x_n\rangle$ for conciseness.
- A quantum computation consists of the application of a unitary operator U to some initial state (usually $|0\rangle = |0\rangle^{\otimes n} = |0^n\rangle$, which we often just write as $|0\rangle$), followed by a measurement of k of the qubits in the computational basis, yielding some outcome $y \in \{0,1\}^k$, which is the output of the computation.
- If the state of a quantum computer is $\sum_{x\in\{0,1\}^n} \alpha_x |x\rangle$ for some coefficients α_x, and we measure all of the qubits, the output is x with probability $|\alpha_x|^2$.

Quantum Computational Complexity

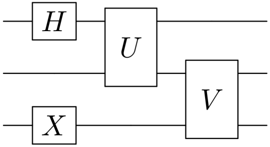
- The quantum circuit model is a framework used to measure the resource usage of a quantum algorithm running on a quantum computer.
- Although in quantum mechanics the evolution of a quantum system is described by unitary operators, not all unitary operators are equally easy to implement physically.
- We might imagine that, in a real quantum computing experiment, the operations that we can actually perform in the lab are:
  - Small, local ones on just a few qubits at a time;
  - More complicated unitary operators can be constructed by combining these small elementary local ones.

Quantum Computational Complexity

- By intuitive reasoning, a quantum computation that runs for T steps and uses space S corresponds to a unitary operation on S qubits (i.e., acting on $\mathcal{C}^{2^S}$), expressed as a product of T elementary operations selected from some family $\mathcal{F}$.
- Each elementary operation is assumed to take time $\mathcal{O}(1)$ and act nontrivially on $\mathcal{O}(1)$ qubits.
- That is, if U is one such elementary operation, we assume that it can be written as $U = U' \otimes I$, where U' acts on k qubits, for some small constant k (usually, $k \leq 3$). The nontrivial parts U' of such operations are called quantum gates, by analogy with logic gates in classical electronic circuits.
- The set of allowed quantum gates depends on the physical architecture, but most "reasonable" gate sets on $\mathcal{O}(1)$ qubits are universal, meaning any unitary operation on S qubits can be approximated as a product of these basic operations acting on different qubits.
- A sequence of quantum gates is known as a quantum circuit.

Quantum Computational Complexity

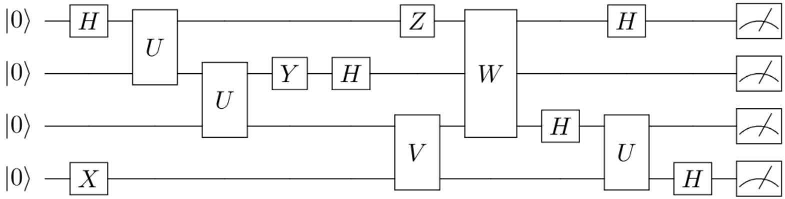
- A quantum circuit can be drawn as a diagram by associating each qubit with a horizontal wire, and drawing each gate as a box across the wires corresponding to the qubits on which it acts.
- The following circuit corresponds to the unitary operator $(I \otimes V)(U \otimes I)(H \otimes I \otimes X)$ on 3 qubits.



- Beware that a circuit is read left to right, with the starting input state on the far left, but the corresponding unitary operators act right to left!
- For convenience, in the diagram we have drawn multi-qubit gates as only acting on nearest-neighbour qubits, but this is not an essential restriction of the model.

Quantum Circuit Representation

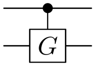
- The quantum circuit model also enables the representation of initial state preparation and final measurement of qubits in the computational basis.



- We are already familiar with:
  - Hadamard gate $H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$ and the gates X, Y, Z corresponding to the Pauli operators.
  - "Controlled-G" (CG) gates, which use an extra qubit to control whether a gate G is applied or not. That is,

$$CG\left|0\right\rangle\left|\psi\right\rangle = \left|0\right\rangle\left|\psi\right\rangle, \quad CG\left|1\right\rangle\left|\psi\right\rangle = \left|1\right\rangle G\left|\psi\right\rangle.$$

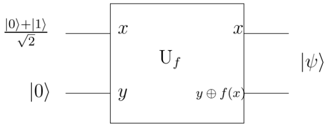
- In a circuit diagram, this is denoted using a filled circle on the control line:



Function Evaluation and Quantum Parallelism

- On a quantum computer it is possible to evaluate a function $f(x)$ on different values of x at the same time.
- The above feature, which is known as quantum parallelism, represents a fundamental characteristic of quantum circuits.
- Consider a Boolean function $f : \{0, 1\} \mapsto \{0, 1\}$.
- To calculate $f(x)$ by quantum computation one must define the transformation $f(x)$ as a unitary transformation U_f.
- This can be done by applying to the input state $|x, y\rangle$, called data register, an appropriate sequence of quantum logic gates transforming $|x, y\rangle$ in state $|x, y \oplus f(x)\rangle$, called the target register. If $y = 0$ then the final state of the second qubit will contain exactly the value of $f(x)$.

Function Evaluation and Quantum Parallelism



- The input to the circuit in the figure is

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |0\rangle ,$$

- Applying U_f to this data register gives

$$\frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}.$$

- This state contains information about both the $f(0)$ value and the $f(1)$ value. We therefore evaluated f simultaneously on $x = 0$ and $x = 1$. This type of parallelism is different from the classical one where multiple circuits (each of which calculates $f(x)$ for a single value of x) are executed simultaneously.

Classical and Reversible Circuits

Classical and Reversible Circuits

- In classical computation, mapping a bit-string (element of $\{0,1\}^n$) to another (element of $\{0,1\}^m$) is achieved through a sequence of logical operations, each acting on few bits (e.g., AND, OR, NOT), forming a classical circuit[4]
- Any classical circuit can be implemented as a quantum circuit, showing that quantum computation is at least as powerful as classical computation and highlighting the power of quantum computers.
- However, a difficulty arises: in quantum mechanics, to keep the system's state pure, the evolution must be unitary, and thus reversible.
- A fundamental difference between classical and quantum circuits is that classical logic gates might be irreversible (e.g. AND, XOR, NAND), while quantum logic gates are always unitary and therefore reversible.
- An alternative computation model should express at least all computations possible with the classical model, with the first objective being to represent classical computations as unitary transformations, i.e., quantum computations.

[4] Where "circuit" is understood in the sense of "electronic circuit".

Classical and Reversible Circuits

- Since unitary transformations are invertible (i.e., reversible), the first step is to transform every irreversible classical computation into a reversible one by ensuring the function is a bijection, so that each output uniquely determines its input.
- Any irreversible computation can be made reversible by ensuring the corresponding function is bi-univocal.
- For example, if we wish to compute an arbitrary classical operation

  $$f : \{0,1\}^n \to \{0,1\}^m$$

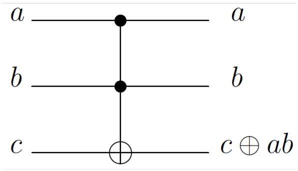
  and make it reversible, we can build

  $$\tilde{f} : \{0,1\}^{n+m} \mapsto \{0,1\}^{n+m},$$

  such that $\tilde{f}$ is bijective and calculates $(x, f(x))$ acting on the input $(x, 0^m)$, where 0^m denotes m bits initialized to 0.

Toffoli (Quantum CCNOT) Gate

- Every bijective function
$$\tilde{f} : \{0,1\}^k \mapsto \{0,1\}^k \,,$$
can be seen as a permutation on the k input bits or, equivalently, on the integers $0, 1, \ldots, 2^k - 1$, and consequently it describes a reversible classical computation.
- Any irreversible classical computation can be converted into an equivalent reversible one using the Toffoli gate (CCNOT, “controlled-controlled-not”), invented by Tommaso Toffoli in 1980.
- The corresponding circuit, shown in the figure, flips the target bit if both control bits are 1, as detailed in its truth table.



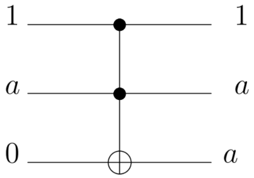
IN			OUT		
a	b	c	a’	b’	c’
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Reversibility and Universality of the Toffoli Gate

- The reversibility of the operation can be easily verified by observing that by applying the Toffoli gate twice consecutively the same starting result is obtained:

$$(a, b, c) \to (a, b, c \oplus ab) \to (a, b, c),$$

  where $\oplus$ is bitwise XOR, i.e. addition modulo 2. The Toffoli gate is universal for classical reversible computations, that is, every classical computation can be constructed reversibly using the Toffoli gate.
- This result follows from the universality of the operations of NAND and FANOUT (the classical bit copy operation) for classical computations and by the fact that both of these operations can be expressed using the Toffoli circuit.
- In fact, by applying the operation with $c = 1$, we obtain $a' = a$, $b' = b$, and $c' = 1 \oplus ab = \neg ab$, i.e. the NAND simulation as a reversible operation.
- The reversible FANOUT is obtained as shown in the figure



Reversible Circuit Construction with the Toffoli Gate

- As for the NAND and the FANOUT, the construction of a reversible circuit for any classical operation f using the Toffoli gate involves the use of service bits in input (ancilla bits) and in output (garbage).
- After eliminating these service bits, the resulting circuit implements:

$$(x, y) \mapsto (x, y \oplus f(x, y)),$$

  where x is the input of f and y is the register that contains the output. It can be considered as the standard reversible circuit to evaluate f.
- The ancilla register assists the main register (holding the input and output) and is used for tasks like creating entanglement, error correction, or implementing reversible logic. It is typically initialized to a known state (e.g., zero) and reset after the computation.

Reversible Computation and Universal Gates

- To compute an arbitrary classical operation $f : \{0,1\}^n \to \{0,1\}^m$ reversibly, we attach a so-called “ancilla” register of m bits, each originally set to 0, and modify f to give a new operation $f' : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n \times \{0,1\}^m$ to perform

  $$f'(x,y) = (x, y \oplus f(x,y)) .$$

  Then if we input $y = 0^m$, we get $(x, f(x))$, from which we can extract our desired output $f(x)$.
- To obtain universal deterministic classical computation, it is sufficient to implement the NOT and AND gates. The NOT gate is immediately reversible. Using the above construction for AND we get the map $(x_1, x_2, y) \mapsto (x_1, x_2, x_1 \wedge x_2)$ for $x_1, x_2, y \in \{0,1\}$.
- The Toffoli gate and the Hadamard gate are even sufficient for universal quantum computation

$$\Downarrow$$

any quantum computation can be approximately represented as a circuit of Toffoli and Hadamard gates

Matrix Representation of the Toffoli Gate

The matrix representation of the Toffoli gate is

$$CX = \begin{bmatrix}
\langle 000|CX|000\rangle & \langle 000|CX|001\rangle & \langle 000|CX|010\rangle & \langle 000|CX|011\rangle & \langle 000|CX|100\rangle & \langle 000|CX|101\rangle & \langle 000|CX|110\rangle & \langle 000|CX|111\rangle \\
\langle 001|CX|000\rangle & \langle 001|CX|001\rangle & \langle 001|CX|010\rangle & \langle 001|CX|011\rangle & \langle 001|CX|100\rangle & \langle 001|CX|101\rangle & \langle 001|CX|110\rangle & \langle 001|CX|111\rangle \\
\langle 010|CX|000\rangle & \langle 010|CX|001\rangle & \langle 010|CX|010\rangle & \langle 010|CX|011\rangle & \langle 010|CX|100\rangle & \langle 010|CX|101\rangle & \langle 010|CX|110\rangle & \langle 010|CX|111\rangle \\
\langle 011|CX|000\rangle & \langle 011|CX|001\rangle & \langle 011|CX|010\rangle & \langle 011|CX|011\rangle & \langle 011|CX|100\rangle & \langle 011|CX|101\rangle & \langle 011|CX|110\rangle & \langle 011|CX|111\rangle \\
\langle 100|CX|000\rangle & \langle 100|CX|001\rangle & \langle 100|CX|010\rangle & \langle 100|CX|011\rangle & \langle 100|CX|100\rangle & \langle 100|CX|101\rangle & \langle 100|CX|110\rangle & \langle 100|CX|111\rangle \\
\langle 101|CX|000\rangle & \langle 101|CX|001\rangle & \langle 101|CX|010\rangle & \langle 101|CX|011\rangle & \langle 101|CX|100\rangle & \langle 101|CX|101\rangle & \langle 101|CX|110\rangle & \langle 101|CX|111\rangle \\
\langle 110|CX|000\rangle & \langle 110|CX|001\rangle & \langle 110|CX|010\rangle & \langle 110|CX|011\rangle & \langle 110|CX|100\rangle & \langle 110|CX|101\rangle & \langle 110|CX|110\rangle & \langle 110|CX|111\rangle \\
\langle 111|CX|000\rangle & \langle 111|CX|001\rangle & \langle 111|CX|010\rangle & \langle 111|CX|011\rangle & \langle 111|CX|100\rangle & \langle 111|CX|101\rangle & \langle 111|CX|110\rangle & \langle 111|CX|111\rangle
\end{bmatrix}$$

$$= \sum_{n=1}^{8} |input_n\rangle \langle output_n| = |000\rangle \langle 000| + |001\rangle \langle 001| + \cdots + |101\rangle \langle 101| + |110\rangle \langle 111| + |111\rangle \langle 110|$$

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

Query Complexity and Oracle Models in Algorithm Analysis

Query Complexity and Oracle Models

- Time complexity is a practical measure of an algorithm's complexity, but it is hard to prove. One way to sidestep this is to use a model that is less realistic but cleaner and more mathematically tractable: the model of query complexity[5].
- In this model, we introduce the oracle as a powerful external resource that solves specific subproblems instantaneously, which may be difficult or impossible to compute in the given model.
- On a classical computer, we can think of the oracle as a function $f : \{0,1\}^n \to \{0,1\}^m$. We pass in inputs $x \in \{0,1\}^n$, and receive outputs $f(x) \in \{0,1\}^m$.
- The oracle is a "black box" that answers certain queries in constant time, allowing focus on algorithm structure and efficiency without dealing with the complexities of computing the oracle's outputs.
- How does this fit into physical reality? We imagine we are given access to the oracle as a circuit, which we can see, but for which it might be difficult to compute some properties directly.

[5] A query is a command used to retrieve information from a database, typically structured according to the relational model, enabling specific operations on the data.

Query Complexity and Oracle Models

- For example, given a description of a circuit computing some function $f\colon \{0,1\}^n \to \{0,1\}$, it might be hard to find an input x such that $f(x) = 1$.
- Sometimes, it is more intuitive to think of an oracle function f as a memory storing 2^n strings of m bits each, where any string can be retrieved with a single query.
- We can provide a quantum computer access to an oracle using the generic reversible computation construction presented in previous slides.
- That is, instead of having a function $f : \{0,1\}^n \to \{0,1\}^m$, we produce a unitary operator O_f which performs the map

$$O_f \left|x\right\rangle \left|y\right\rangle = \left|x\right\rangle \left|y \oplus f(x)\right\rangle .$$

- The unitary operator O_f is sometimes known as the bit oracle[6].

[6] We want to translate f into a quantum circuit, and since the output is a bit, we call it a bit oracle.

Grover’s Algorithm

Grover's Algorithm

- A simple example of a problem in the query complexity model is unstructured search[7] on a set of N elements, where the structure of the solution space is unknown and the goal is to find a unique marked element.
- In an unstructured search problem, we are given access to a function $f : \{0, 1, \dots, N-1\} \to \{0, 1\}$ with the promise that $f(x) = 1$ for a unique "marked" element ω. Our task is to identify ω.
- We can access f with a subroutine (sometimes called an oracle) in the form of a unitary operator U_ω that acts as

$$U_\omega |x\rangle = (-1)^{f(x)} |x\rangle \, .$$

- The best classical algorithm checks the condition on randomly chosen elements x, and intuitively, the unstructured search problem requires about N queries to solve classically. This is formalized in the following proposition:

Proposition

Let $\mathcal{A}$ be a classical algorithm which solves the unstructured search problem on a set of N elements. Then $\mathcal{A}$ makes $\mathcal{O}(N)$ queries in the worst case.

[7] In a structured search problem, knowledge of the structure enables the construction of efficient algorithms, such as binary search for solution spaces with a binary tree structure.

Grover's Algorithm

Theorem (Grover '97)

There is a quantum algorithm which solves the unstructured search problem using $\mathcal{O}\left(\sqrt{N}\right)$ queries.

- Grover's algorithm is a quantum algorithm that can search for a value or element in an unsorted set faster than any classical algorithm.
- The algorithm requires significantly fewer queries, with a complexity $\mathcal{O}\left(\sqrt{N}\right)$ evaluations.
- It is optimal for completely unstructured search problems and is particularly useful for obtaining faster solutions to NP-complete problems.

Grover's Algorithm

- U_ω is different from the standard quantum oracle for a function f. This standard oracle, denoted here as U_f, uses an ancillary qubit system.
- For some integer n, with $n \geq 1$, let $N = 2^n$ be the number of candidate solutions for the unstructured search problem (in concrete problems n is generally very large).
- We assume the existence of a function $f(x)$ that maps n bits to a single bit, where $f(x) = 1$ if x is a solution, and $f(x) = 0$ otherwise.
- Additionally, we assume an oracle exists, implemented on a quantum computer, which can perform the operation on the $n+1$-qubit state $|x\rangle |q\rangle$, where $|x\rangle$ is an n-qubit state and $|q\rangle$ is a single qubit:

  $$U_f |x\rangle |q\rangle = |x\rangle |q \oplus f(x)\rangle ,$$

  with $x = \{0,1\}^n$ and $|q\rangle$ is a single qubit.
  In particular

  $$U_f |x\rangle |0\rangle = |x\rangle |f(x)\rangle .$$

Grover's Algorithm

- In the quantum search algorithm, it is useful to prepare the 1-bit state $|q\rangle$ as:

$$|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

- The action of the oracle U_f is

$$U_f |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} = (-1)^{f(x)} |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}},$$

  which consists in inverting the amplitudes of the states $|x\rangle$ that are solutions and leave the others unchanged[8]. Since the 1-bit initial state $|q\rangle$ does not change, it can be omitted from the discussion.

- We can then consider the action of U_f as follows:

$$U_\omega |x\rangle = (-1)^{f(x)} |x\rangle \,,$$

  i.e., the oracle marks the solutions of the search problem with a minus sign (corresponding to $f(x) = 1$).

[8]We have

$$U_f |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) = \left(|x\rangle \frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} \right) = (U_\omega |x\rangle) \otimes |-\rangle \,.$$

Grover's Algorithm

The steps of Grover's algorithm are as follows:

1. Starting from the initial state $|0\rangle^{\otimes n}$, apply Hadamard gates to create an equiprobable superposition of all basis states:
$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x \in \{0,1\}^n} |x\rangle .$$
2. Apply the Grover operator a certain number of times:
$$G = H^{\otimes n} P_0 H^{\otimes n} U_\omega ,$$
where $P_0 = 2|0^n\rangle\langle 0^n| - I$ applies a phase shift of -1 to all computational states except $|0\rangle$:
$$P_0 : |x\rangle \mapsto \begin{cases} |x\rangle & \text{if } |x\rangle = |0\rangle \\ -|x\rangle & \text{if } |x\rangle \neq |0\rangle \end{cases} ,$$
for every $0 \leq x \leq N-1$. The operation $H^{\otimes n} P_0 H^{\otimes n}$, or "inversion around the mean," doubles the solution amplitudes relative to the average after sign flipping by the oracle.
3. Measure the resulting quantum state in the computational basis.

Grover's Algorithm

- The Grover operator G can be written as

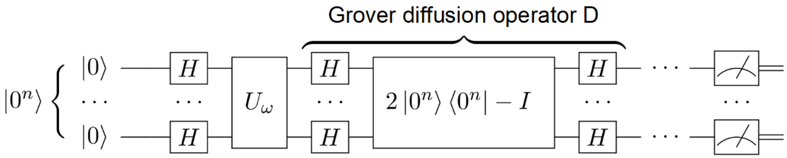
$$G = H^{\otimes n} P_0 H^{\otimes n} U_\omega = H^{\otimes n} \left(2 \left|0^n\right\rangle \left\langle 0^n\right| - I\right) H^{\otimes n} U_\omega = \left(2 \left|\psi\right\rangle \left\langle\psi\right| - I\right) U_\omega = D U_\omega,$$

where U_ω implements the phase inversion and

$$D = 2 \left|\psi\right\rangle \left\langle\psi\right| - I$$

defines the Grover diffusion operator.

- The circuit diagram form of the Grover's algorithm looks like



Example of Grover's Quantum Search Algorithm

- Grover's algorithm has a geometric interpretation, which we illustrate through a 2-qubit example, where the quantum state remains confined to a two-dimensional subspace after each step.
- Consider the plane spanned by $|\omega\rangle$ and $|\psi\rangle$, or equivalently, the space spanned by $|\omega\rangle$ and the orthogonal state $\left|\omega^{\perp}\right\rangle = \frac{1}{\sqrt{N-1}} \sum_{x\neq\omega} |x\rangle$.
- To describe the algorithm, we introduce the following unitary operators:

$$U_{|\omega\rangle} = I - 2\,|\omega\rangle\,\langle\omega|\,, \quad R_{|\psi\rangle} = D,$$

  where $|\psi\rangle$ is an arbitrary state.
- $U_{|\omega\rangle}$ can be seen as an "inversion about $|\omega\rangle$" operation, while $R_{|\psi\rangle}$ can be seen as a "reflection about $|\psi\rangle$" operation.

Grover's Algorithm: Geometric interpretation

- An arbitrary state $|\psi\rangle$ can be expanded as

$$|\psi\rangle = \alpha\,|\omega\rangle + \beta\left|\omega^{\perp}\right\rangle,$$

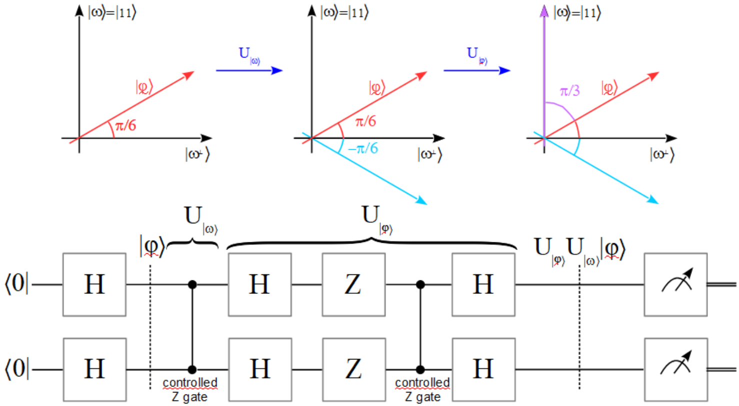
for some α and β, and some state $\left|\omega^{\perp}\right\rangle$ such that $\left\langle\omega\middle|\omega^{\perp}\right\rangle = 0$.

- Then

$$U_{|\omega\rangle}\,|\psi\rangle = -\alpha\,|\omega\rangle + \beta\left|\omega^{\perp}\right\rangle,$$

has flipped the phase of the component corresponding to $|\omega\rangle$, and left the component orthogonal to $|\omega\rangle$ unchanged. $R_{|\psi\rangle}$ has the opposite effect.

Grover's Algorithm: Geometric interpretation



The Quantum Fourier Transform

The Quantum Fourier Transform and Periodicity

- The Quantum Fourier transform (QFT) over $\mathbb{Z}_N$, the integers modulo N, is an important unitary transformation used in a number of different contexts in quantum information processing.
- The QFT is defined by the map

$$Q_N \left| x \right\rangle = \frac{1}{\sqrt{N}} \sum_{y \in \mathbb{Z}_N} \omega_N^{xy} \left| y \right\rangle ,$$

  where $\omega_N = e^{2\pi j/N}$ and xy is just the product of the two numbers x and y, thought of as integers.
- The QFT can be seen as a generalization of the Hadamard gate.

The Quantum Fourier Transform and Periodicity

- Examples of the QFT with respect to the computational basis:

$$Q_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad Q_3 = \frac{1}{\sqrt{3}}\begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{2\pi j/3} & e^{-2\pi j/3} \\ 1 & e^{-2\pi j/3} & e^{2\pi j/3} \end{bmatrix},$$

$$Q_4 = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix}.$$

- The QFT is unitary and it is exactly the same transformation as the Discrete Fourier Transform (DFT) used for classical computation and signal processing, up to the nonstandard normalisation of $1/\sqrt{N}$.
- One of the most important applications of the QFT is determining the period of a periodic function.
- Example of periodic sequence, with period 5, which is one-to-one on each period.

7	3	4	2	9	7	3	4	2	9	7	3	4	2	9	7	3	4	2	9

The Quantum Fourier Transform and Periodicity

- Imagine we are given access to an oracle function $f : \mathbb{Z}_N \to \mathbb{Z}_M$, for some integers N and M, such that:
  - f is periodic: there exists r such that r divides N and $f(x+r) = f(x)$ for all $x \in \mathbb{Z}_N$;
  - f is one-to-one on each period: for all pairs (x, y) such that $|x - y| < r$, $f(x) \neq f(y)$.

The goal is to determine r!

Periodicity determination algorithm

We are given access to a periodic function f with period r, which is one-to-one on each period. We start with the state $|0\rangle\,|0\rangle$, where the first register is dimension N, and the second dimension M.

1. Apply Q_N to the first register.
2. Apply O_f to the two registers.
3. Measure the second register.
4. Apply Q_N to the first register.
5. Measure the first register; let the answer be k.
6. Simplify the fraction k/N as far as possible and return the denominator.

The Quantum Fourier Transform and Periodicity

- The initial sequence of operations which occur during the algorithm is:

$$|0\rangle\,|0\rangle \overset{1}{\mapsto} \frac{1}{\sqrt{N}} \sum_{x \in \mathcal{Z}_N} |x\rangle\,|0\rangle \overset{2}{\mapsto} \frac{1}{\sqrt{N}} \sum_{x \in \mathcal{Z}_N} |x\rangle\,|f(x)\rangle$$

- When the second register is measured, we receive an answer, say z. By the periodic and one-to-one properties of f, all input values $x \in \mathcal{Z}_N$ for which $f(x) = z$ are of the form $x_0 + lr$ for some x_0 and integer l.
- The state therefore collapses to something of the form

$$\sqrt{\frac{r}{N}} \sum_{l=0}^{N/r-1} |x_0 + lr\rangle\,.$$

The Quantum Fourier Transform and Periodicity

- After we apply the QFT, we get the state

$$\frac{\sqrt{r}}{N}\sum_{l=0}^{N/r-1}\left(\sum_{y\in\mathcal{Z}_N}\omega_N^{y(x_0+lr)}\,|y\rangle\right)=\frac{\sqrt{r}}{N}\sum_{y\in\mathcal{Z}_N}\omega_N^{yx_0}\left(\sum_{l=0}^{N/r-1}\omega_N^{yl}\right)|y\rangle\,.$$

- The sum over l is 0 unless $y\equiv 0 \bmod N/r$ or, in other words if $y=lN/r$ for some integer l. So the state can be rewritten as

$$\frac{1}{\sqrt{r}}\sum_{l=0}^{r-1}\omega_N^{lx_0N/r}\,|lN/r\rangle\ \rightarrow$$

When we perform the final measurement, we receive an outcome $k=l_0N/r$, for some l_0 picked uniformly at random from $0,\ldots,r-1$.

Shor’s Algorithm: Example of QFT Application in Integer Factorization

- The main application of periodicity determination is Shor’s quantum algorithm for integer factorization. Given an n-digit integer N as input, this algorithm outputs a non-trivial factor of N (or that N is prime, if this is the case) with success probability $1 - \epsilon$, for any $\epsilon > 0$, in time $\mathcal{O}(n^3)$.
- Shor’s efficient factorization algorithm implies that this cryptosystem is insecure against attack by a large quantum computer. Although no large-scale quantum computer yet exists, this result has already had major implications for cryptography, as national security agencies start to move away from the RSA cryptosystem.

Shor's Algorithm: Example of QFT Application in Integer Factorization

Shor's Algorithm.

Let N denote the integer to be factorised. Assume that N is not even or a power of a prime.

1. Choose $1 < a < N$ uniformly at random.
2. Compute $b = GCD(a, N)$. If $b > 1$ output b and stop.
3. Apply a QFT.
4. Determine the order r of a modulo N. If r is odd, the algorithm has failed; terminate.
5. Compute $s = GCD\left(a^{r/2} - 1, N\right)$. If $s = 1$, the algorithm has failed; terminate.
6. Output s.