Quantum Information (ELEC-C9440) Lecture 5-6

Matti Raasakka

Spring 2023

Aalto University

Simple quantum algorithms

Big-O notation

In our study of quantum algorithms, it will be useful to have a way of quantifying (roughly) how much resources (time, space) an algorithm needs. The big-$\mathcal{O}$ notation is used to give upper bounds on the asymptotic behavior of functions.

Definition (Big-$\mathcal{O}$ notation)
Let $f(n)$ and $g(n)$ be functions on non-negative integers. Then "$f(n)$ is $\mathcal{O}(g(n))$" if there exist constants c and n_0 such that

$$f(n) \leq cg(n) \ , \tag{1}$$

for all $n > n_0$.

Example: $f(n) = 4n^3 + 2n$ is $\mathcal{O}(n^3)$.
Example: In the big-$\mathcal{O}$ notation, asympotics need not be tight:
$f(n) = 4n^3 + 2n$ is also $\mathcal{O}(n^4)$ and $\mathcal{O}(2^n)$.

Decision problems, P

Here we give a few concepts from the theory of computational complexity that are most important in quantum computation.

Definition (Decision problem)
A problem that can be posed as a yes-no question.

Example: Is 7 a prime number?

Example: Is the list $\{-3, 2, 5, 7, 4, 8, 10\}$ sorted?

Definition (Polynomial time)
We say that a problem can be solved in polynomial time if there exists an algorithm which solves the problem on an input of size n and runs in time $\mathcal{O}(n^c)$ for some c.

Definition (The class P)
The complexity class P is the set of all decision problems which can be solved by a classical algorithm which runs in polynomial time.

Example: Multiplying two numbers is in P.

Example: Deciding whether a number is prime is in P.

NP, BQP

Definition (The class NP)
The complexity class NP is the set of all decision problems for which solutions can be verified in polynomial time on a classical computer.

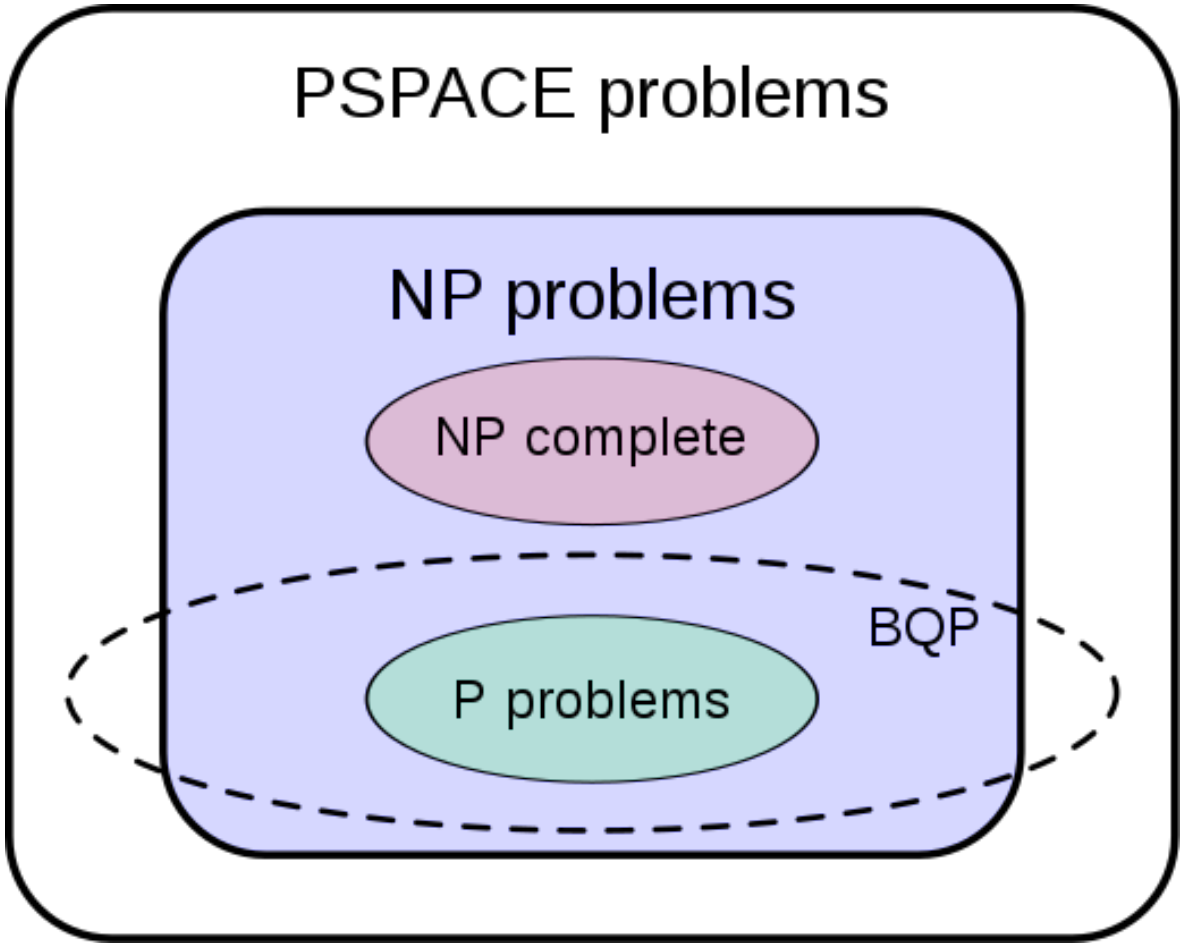
So the difference to P is that it is not necessary to solve the problem in polynomial time, only to check a solution if you are given one.
Example: Factoring integers is in NP.
Example: All problems in P are also in NP.

Definition (The class BQP)
The complexity class BQP consists of all decision problems which can be solved in polynomial time on a quantum computer with error probability at most 1/3.

You can see that BQP is the quantum version of P. In quantum computing, we are interested in finding problems which are in BQP but are not in P.
Example: Factoring integers is in BQP.
Example: All problems in P are also in BQP.

Complexity zoo

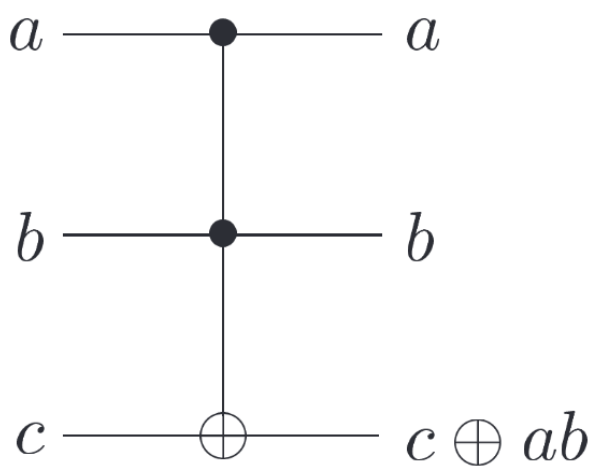
The classes P, NP, and BQP are the most relevant for our purposes. There is a large amount of other classes as well. For example:



Classical algorithms on a quantum computer

Any classical algorithm can in principle be run on a quantum computer. How? Turns out that all classical, irreversible logical circuits can be made reversible by simulating them with a Toffoli gate. The Toffoli gate is reversible, because it is its own inverse. It is also unitary and therefore a valid quantum gate if we define the quantum version to act on computational basis states of three qubits the exact same way Toffoli gate acts on three classical bits.

This is only a proof that *theoretically* quantum computers are at least as powerful as classical computers. Of course it wouldn't make practically much sense to run classical algorithms on quantum hardware. The complexity theoretic lesson is that P is contained in BQP.



Quantum parallelism

Let $f : \{0,1\}^n \mapsto \{0,1\}^m$ be a function from n bits to m bits. Suppose we have a quantum circuit which implements f

$$U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle \ , \tag{2}$$

where $|x\rangle$ is the data register with n qubits and $|y\rangle$ is the target register with m qubits. We can use U_f to compute f on many inputs simultaneously:

1. Initialize registers to $|0\rangle |0\rangle$.
2. Apply Hadamards to the data register $(H^{\otimes n} |0\rangle) |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle |0\rangle$.
3. Apply U_f: $\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} U_f |x\rangle |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle |f(x)\rangle$.

In some sense we computed $f(x)$ on all 2^n inputs with only one call to U_f, this is "quantum parallelism". Note that this is not immediately useful: to get information about the quantum state you have to measure it! If you measure the data register in the computational basis you simply get one randomly chosen $|x\rangle |f(x)\rangle$. Not very impressive, this could be easily done on a classical computer too. Quantum parallelism is still a recurring theme in quantum algorithms, though we have to be smarter about how to use it.

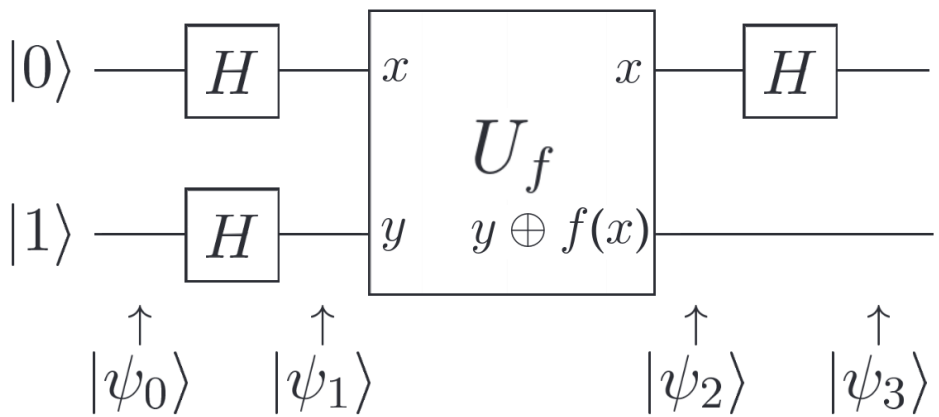
Deutsch's algorithm

Consider the following problem. Given a function $f : \{0, 1\} \mapsto \{0, 1\}$, determine whether f is *balanced* or *constant*.

$$\text{Balanced:} \quad f(0) \neq f(1) \tag{3}$$

$$\text{Constant:} \quad f(0) = f(1) \tag{4}$$

Classically, you would need to evaluate f *twice*: compute both $f(0)$ and $f(1)$ and check whether it is balanced or not. Using quantum computation, you can solve this with only *one* evaluation of U_f which implements f!



Deutsch's algorithm

Let's study what happens to the quantum state at different steps in the circuit. We start with $|\psi_0\rangle = |0\rangle\,|1\rangle$. After the initial Hadamards the state is

$$|\psi_1\rangle = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) . \tag{5}$$

Observe that

$$U_f\,|x\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = |x\rangle\left(\frac{|f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}}\right) \tag{6}$$

$$= \begin{cases} |x\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) & \text{if } f(x) = 0 \\ |x\rangle\left(\frac{|1\rangle - |0\rangle}{\sqrt{2}}\right) & \text{if } f(x) = 1 \end{cases} \tag{7}$$

$$= (-1)^{f(x)}\,|x\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) . \tag{8}$$

Therefore

$$|\psi_2\rangle = U_f \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \tag{9}$$

$$= \begin{cases} \pm\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0) = f(1) \\ \pm\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0) \neq f(1) \end{cases} \tag{10}$$

that is, the relative phase in the first qubit changes if the f is balanced. After the final Hadamard

$$|\psi_3\rangle = \begin{cases} \pm|0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0) = f(1) \\ \pm|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0) \neq f(1) \end{cases} \tag{11}$$

Now that if we measure the first qubit, we'll find $|0\rangle$ if and only if f is constant. If we find $|1\rangle$, then we know that f is balanced. So in other words we found a global property of f with only one application of U_f, whereas classically we would need to evaluate f twice.

Demo: Deutsch's algorithm

We already saw how this algorithm works. Now we'll implement it in Qiskit as a demo.

Deutsch-Jozsa algorithm

Let's study next a generalization of Deutsch's algorithm where the quantum speedup is more dramatic.
Consider now the following problem. Given a function $f : \{0,1\}^n \mapsto \{0,1\}$ with the *promise* that it is either balanced or constant, how to determine which is the case using as few calls to f as possible?

$$\text{Balanced:} \quad \begin{cases} f(x) = 0 \text{ for exactly half of all possible inputs } x \in [0, 2^n - 1] \\ f(x) = 1 \text{ for all other inputs} \end{cases} \tag{12}$$

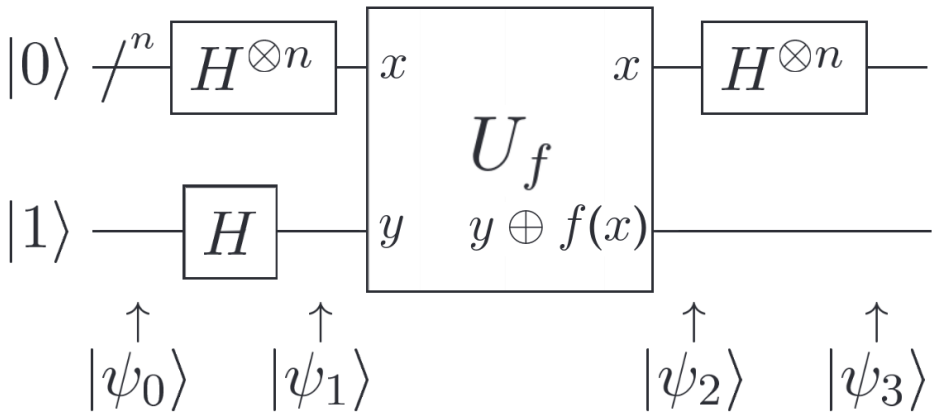
$$\text{Constant:} \quad f(x) = f(y) \text{ for all } x, y \in [0, 2^n - 1] \tag{13}$$

Classically the best way to solve this is to simply compute $f(0), f(1), \ldots, f(2^{n-1})$ and stop immediately if two different values are encountered, then f must be balanced. However, in the worst case you would have to compute f $2^{n-1} + 1$ times to be sure that the function is constant. Thus the classical solution runs in time exponential in n.

$$\text{Worst case } f(x)\text{: } n = 3 \quad \{0,0,0,0,1,1,1,1\} \tag{14}$$

Deutsch-Jozsa algorithm

The quantum solution is essentially the same as for Deutsch's problem and requires only one evaluation of U_f! The algorithm is



Looks very familiar. Let's again track the state throughout the computation. After the Hadamards on the first wire bundle

$$|\psi_1\rangle = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) , \tag{15}$$

where the summation is over all n-bit strings.

Deutsch-Jozsa algorithm

Previously we saw that

$$U_f |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) = (-1)^{f(x)} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) . \tag{16}$$

Therefore

$$|\psi_2\rangle = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \tag{17}$$

In order to proceed, we need to compute $H^{\otimes n} |x\rangle$. Let's write it out explicitly

$$\begin{aligned} H^{\otimes n} |x\rangle &= (H |x_1\rangle)(H |x_2\rangle) \dots (H |x_n\rangle) && (18) \\ &= \frac{1}{2^{n/2}} (|0\rangle + (-1)^{x_1} |1\rangle) (|0\rangle + (-1)^{x_2} |1\rangle) \dots (|0\rangle + (-1)^{x_n} |1\rangle) && (19) \\ &= \frac{1}{2^{n/2}} \sum_z (-1)^{x \cdot z} |z\rangle , && (20) \end{aligned}$$

where $x \cdot z = x_1 z_1 + \dots x_n z_n$.

Deutsch-Jozsa algorithm

It is easier to get a feel for that formula by writing out everything for fixed n, say $n = 2$:

$$H^{\otimes n}|0\rangle = H^{\otimes n}|0\rangle|0\rangle = \frac{1}{2}(|0\rangle|0\rangle + |0\rangle|1\rangle + |1\rangle|0\rangle + |1\rangle|1\rangle) \tag{21}$$

$$H^{\otimes n}|1\rangle = H^{\otimes n}|0\rangle|1\rangle = \frac{1}{2}(|0\rangle|0\rangle - |0\rangle|1\rangle + |1\rangle|0\rangle - |1\rangle|1\rangle) \tag{22}$$

$$H^{\otimes n}|2\rangle = H^{\otimes n}|1\rangle|0\rangle = \frac{1}{2}(|0\rangle|0\rangle + |0\rangle|1\rangle - |1\rangle|0\rangle - |1\rangle|1\rangle) \tag{23}$$

$$H^{\otimes n}|3\rangle = H^{\otimes n}|1\rangle|1\rangle = \frac{1}{2}(|0\rangle|0\rangle - |0\rangle|1\rangle - |1\rangle|0\rangle + |1\rangle|1\rangle)\ . \tag{24}$$

So, now we can compute $|\psi_3\rangle$,

$$|\psi_3\rangle = \frac{1}{2^{n/2}}\sum_z\sum_x(-1)^{x\cdot z+f(x)}|z\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)\ . \tag{25}$$

We can use this superposition to find whether f is balanced or not.

Deutsch-Jozsa algorithm

Notice that the probability amplitude of $|0\rangle$ of the data register is $(z = 0)$

$$\frac{1}{2^{n/2}} \sum_x (-1)^{f(x)} \ . \tag{26}$$

If f is constant, that is either $f(x) = 0$ or $f(x) = 1$, then this amplitude is simply $+1$ or -1. This means that the first register is in state $|0\rangle$ with probability 1 and all other amplitudes vanish.
If f is balanced, then the above sum has exactly as many $+1$ terms as -1 terms which cancel each other. So in this case the amplitude of $|0\rangle$ vanishes.
Therefore, when we measure the first register in the very end, the result is all zeros if and only if f is constant. Otherwise f is balanced. Therefore we have an *exponential quantum speedup* over the deterministic classical algorithm. This is an impressive demonstration of the power of quantum computation but initial excitement is somewhat deflated by noting that the problem we solved is not very practical. Also note that if we allow for probabilistic algorithms, one could randomly pick elements x and evaluate $f(x)$ on a classical computer to find whether f is constant or balanced with high probability.

Quantum Fourier transform

Quantum Fourier transform

The most famous discovery in quantum computing is Shor's algorithm for factoring integers. It runs in polynomial time in the size of the number being factored. The size of the number is the number of bits n needed to describe it. Shor's algorithm gives an exponential speedup compared to the best known classical factoring algorithm, the number field sieve.
The *quantum Fourier transform* (QFT) is the key ingredient in Shor's algorithm and in many other algorithms as well:

QFT $\rightarrow$

- Integer factoring
- Phase estimation
- Discrete logarithm problem
- Quantum counting
- ...

Quantum Fourier transform

The *discrete Fourier transform* is a transformation which takes a vector of complex numbers $\{x_0, \ldots, x_{N-1}\}$ and outputs a different vector of complex numbers defined by

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi ijk/N} \ . \tag{27}$$

The *quantum Fourier transform* is the same transformation but for computational basis states $\{|0\rangle, \ldots, |N-1\rangle\}$

$$|j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle \ . \tag{28}$$

This is transformation is unitary and acts on arbitrary quantum states as

$$\sum_{j=0}^{N-1} x_j |j\rangle \mapsto \sum_{k=0}^{N-1} y_k |k\rangle \ , \tag{29}$$

so effectively it applies a discrete Fourier transformation on the basis state amplitudes.

Quantum Fourier transform

Unitarity of QFT as we defined it is not immediately clear so let's check it:

$$\langle j' | U^\dagger U | j \rangle = \frac{1}{N} \sum_{k'=0}^{N-1} \sum_{k=0}^{N-1} e^{-2\pi i j' k'/N} e^{2\pi i j k/N} \langle k' | k \rangle \tag{30}$$

$$= \frac{1}{N} \sum_{k'=0}^{N-1} \sum_{k=0}^{N-1} e^{-2\pi i j' k'/N} e^{2\pi i j k/N} \delta_{k',k} \tag{31}$$

$$= \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi i (j-j')k/N} \ . \tag{32}$$

If $j = j'$, then clearly $\langle j | U^\dagger U | j \rangle = 1$. If $j \neq j'$, then define $e^{2\pi i (j-j')/N} = r$

$$\langle j' | U^\dagger U | j \rangle = \frac{1}{N} \sum_{k=0}^{N-1} r^k = \frac{1}{N} \frac{1-r^N}{1-r} = \frac{1}{N} \frac{1-e^{2\pi i (j-j')}}{1-r} = 0 \ , \tag{33}$$

where we used the formula $\sum_{k=0}^{N-1} r^k = (1-r^N)/(1-r)$ for the geometric series and in the last step noted that $e^{2\pi i (j-j')} = 1$ because $j - j'$ is a integer. Therefore $U^\dagger U = \mathbb{I} \to U$ is unitary.

Quantum Fourier transform

Next we would like to figure out which circuit realizes QFT. First we write QFT in the so-called *product representation* which is extremely helpful when we are coming up with a circuit for QFT. Now we consider n qubits, so there are $N = 2^n$ basis states. It is convenient to use the binary representations $j = j_1 j_2 \dots j_n$. More explicitly

$$j = j_1 2^{n-1} + j_2 2^{n-2} + \dots + j_n 2^0 \ . \tag{34}$$

We'll also use *binary fractions*

$$0.j_l j_{l+1} \dots j_m = \frac{j_l}{2} + \frac{j_{l+1}}{4} + \dots \frac{j_m}{2^{m-l+1}} \ . \tag{35}$$

Now we can derive the product representation of QFT:

$$|j\rangle \mapsto \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi i j k/2^n} \, |k\rangle \tag{36}$$

$$= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \dots \sum_{k_n=0}^{1} e^{2\pi i j \left(\sum_{l=1}^{n} k_l 2^{-l}\right)} \, |k_1 \dots k_n\rangle \quad \text{(binary repr for } k\text{)} \tag{37}$$

Quantum Fourier transform

$$|j\rangle \mapsto \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \bigotimes_{l=1}^{n} e^{2\pi i j k_l 2^{-l}} |k_l\rangle \tag{38}$$

$$= \frac{1}{2^{n/2}} \bigotimes_{l=1}^{n} \left(\sum_{k_l=0}^{1} e^{2\pi i j k_l 2^{-l}} |k_l\rangle \right) \tag{39}$$

$$= \frac{1}{2^{n/2}} \bigotimes_{l=1}^{n} \left(|0\rangle + e^{2\pi i j 2^{-l}} |1\rangle \right) \tag{40}$$

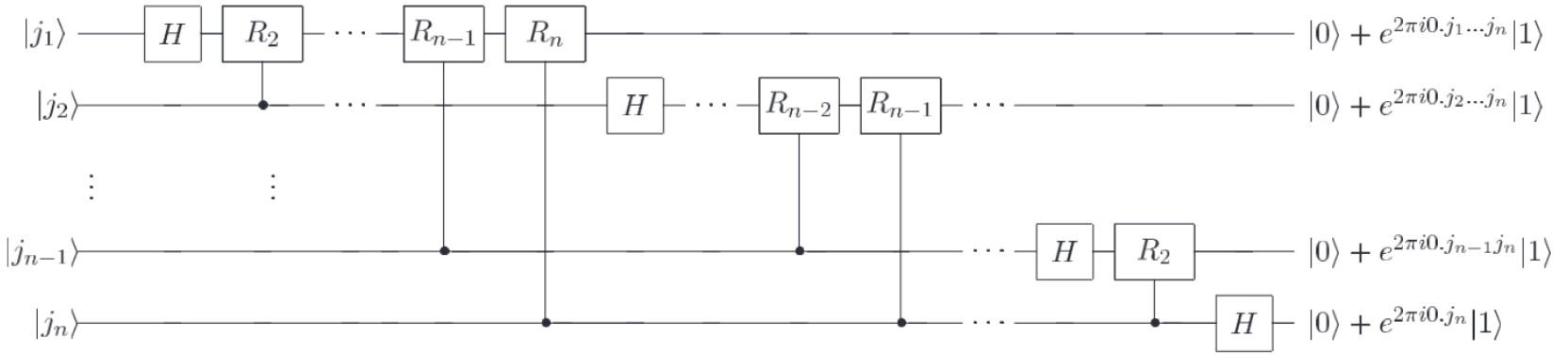
$$= \frac{(|0\rangle + e^{2\pi i 0.j_n} |1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1} j_n} |1\rangle) \ldots (|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n} |1\rangle)}{2^{n/2}} \tag{41}$$

This is the product representation for QFT. Now to the quantum circuit. We'll utilize the R_k gate

$$R_k \equiv \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix} . \tag{42}$$

Quantum Fourier transform

The circuit diagram is



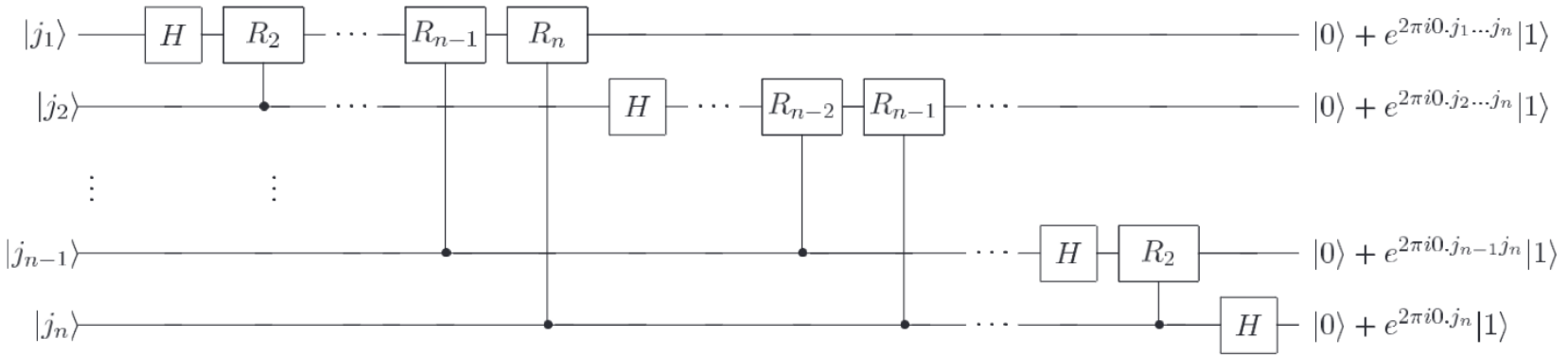
Let's study how this circuit operates on the state $|j_1 j_2 \ldots j_n\rangle$. The Hadamard on the first wire produces

$$\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{j_1}|1\rangle)\,|j_2 \ldots j_n\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.j_1}|1\rangle)\,|j_2 \ldots j_n\rangle \ , \qquad (43)$$

because if $j_1 = 0$ then $e^{2\pi i 0.j_1} = 1$ and if $j_1 = 1$ then $e^{2\pi i 0.j_1} = -1$. After the controlled-R_2

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.j_1 j_2}|1\rangle\right)|j_2 \ldots j_n\rangle \ . \qquad (44)$$

Quantum Fourier transform



We see that each controlled-R_k gate adds some relative phase between $|0\rangle$ and $|1\rangle$ of the target qubit. After all controlled-R_k gates targeting the first qubit, we have the state

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}\,|1\rangle\right)|j_2 \ldots j_n\rangle \ . \tag{45}$$

Then to the second qubit. After the initial Hadamard we have

$$\frac{1}{2}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}\,|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_2}\,|1\rangle\right)|j_3 \ldots j_n\rangle \ . \tag{46}$$

The controlled-R_k gates work exactly the same for the second qubit. Only difference is that there is one R_k gate less:

$$\frac{1}{2}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_2 \ldots j_n}|1\rangle\right)|j_3 \ldots j_n\rangle \ . \tag{47}$$

After repeating this analysis for all the remaining qubits, the quantum state becomes

$$\frac{(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle)(|0\rangle + e^{2\pi i 0.j_2 \ldots j_n}|1\rangle)\ldots(|0\rangle + e^{2\pi i 0.j_n}|1\rangle)}{2^{n/2}} \ . \tag{48}$$

This is almost the product representation of QFT! Only the qubits are in the wrong order. We finish by using SWAP-gates to reverse qubit order. So, altogether

$$|j_1 j_2 \ldots j_n\rangle \mapsto \tag{49}$$

$$= \frac{(|0\rangle + e^{2\pi i 0.j_n}|1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1} j_n}|1\rangle)\ldots(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle)}{2^{n/2}} \ . \tag{50}$$

Quantum Fourier transform

The circuit we presented is efficient:

- The j^{th} wire has $n - j + 1$ gates:
  $n + (n-1) + \ldots + 1 = n(n+1)/2 = \mathcal{O}(n^2)$ gates in total
- At most $n/2$ swaps are needed, each requiring $\mathcal{O}(1)$ controlled-not gates.

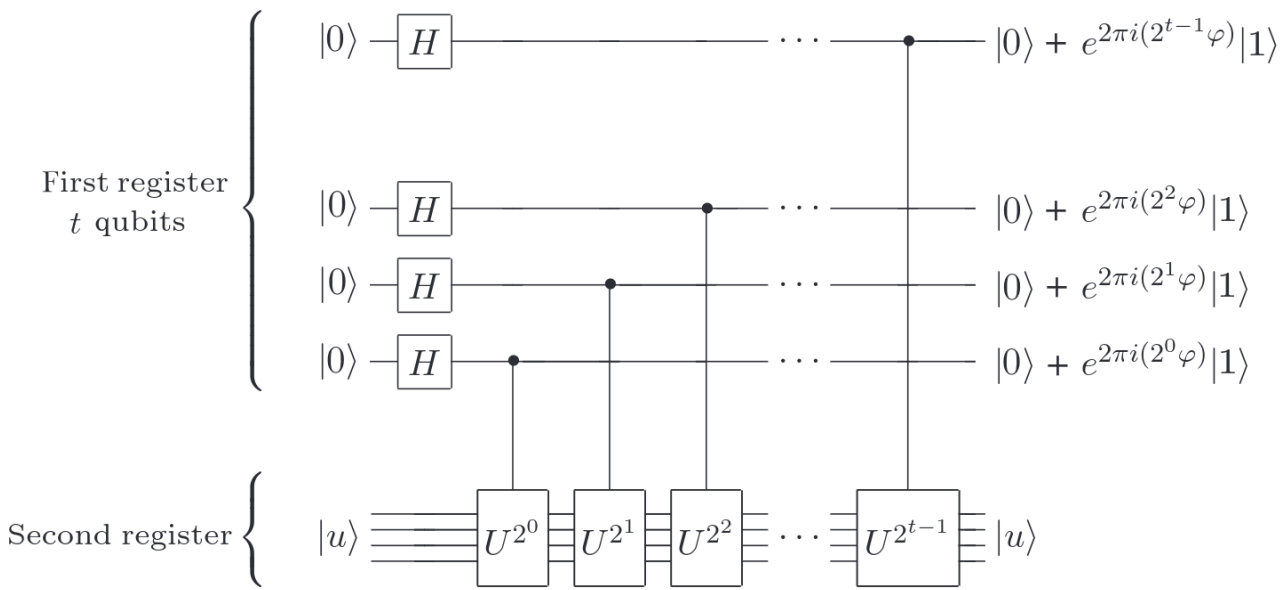
Therefore this algorithm runs in $\mathcal{O}(n^2)$ time.
Best classical algorithms for discrete Fourier transform run in time $\mathcal{O}(n2^n)$ so the quantum algorithm is exponentially faster. Unfortunately, we cannot use QFT to speed up classical Fourier transforms because QFT outputs a quantum state that we can only extract information from by measuring it. Measurement collapses the quantum state and doesn't tell us the Fourier transformed amplitudes of the original state.
Fortunately, there are other ways to use QFT to our advantage.

Phase estimation

Suppose we have a quantum state $|u\rangle$ which is an eigenstate of some unitary U with eigenvalue $e^{2\pi i\varphi}$. Additionally we have a *black box* capable of performing controlled-U^{2^j} operations for non-negative integers j. The goal of *phase estimation* is to estimate the value of φ. Here is how it works:

Registers: We use two quantum registers: first with t qubits initially in the state $|0\rangle$, the second consists of as many qubits as is needed to store $|u\rangle$.

Step 1: Run the following circuit:



Phase estimation

The initial Hadamards transform the first register to an uniform superposition of computational basis states. Then the controlled-U operations simply pick up powers of the $e^{2\pi i\varphi}$, the eigenvalue of $|u\rangle$ as the phase multiplying $|1\rangle$-terms. Now the first register is in state

$$\frac{1}{2^{t/2}}\left(|0\rangle + e^{2\pi i 2^{t-1}\varphi}|1\rangle\right)\left(|0\rangle + e^{2\pi i 2^{t-2}\varphi}|1\rangle\right)\ldots\left(|0\rangle + e^{2\pi i 2^{0}\varphi}|1\rangle\right) . \quad (51)$$

Let's assume that we can write the phase $\varphi = 0.\varphi_1\varphi_2\ldots\varphi_t$. The above state becomes

$$\frac{1}{2^{t/2}}\left(|0\rangle + e^{2\pi i 0.\varphi_t}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.\varphi_{t-1}\varphi_t}|1\rangle\right)\ldots\left(|0\rangle + e^{2\pi i 0.\varphi_1\varphi_2\ldots\varphi_t}|1\rangle\right) . \quad (52)$$

This looks a lot like the product representation of QFT!

Step 2: Apply the *inverse* QFT. The computer is then in state

$$|\varphi_1\varphi_2\ldots\varphi_t\rangle\,|u\rangle \ . \quad (53)$$

Step 3: Measure the first register in the computational basis to obtain $\varphi = 0.\varphi_1\varphi_2\ldots\varphi_t$.

Phase estimation

Above we assumed that φ can be written exactly as a binary fraction with t digits. One can show that if this is not possible, then the procedure outlined above will yield a good approximation to φ with high probability. We won't prove this now but you can see Nielsen-Chuang for details.

The bottom line is that QFT can be used in the phase estimation algorithm to find eigenvalues of given states $|u\rangle$. Next, we look at a notable application of phase estimation: order finding.

Order finding

Order finding is important because it turns out to be equivalent to the problem of factoring integers. The integer factoring algorithm can then be used to break the RSA cryptosystem.

Definition (Order)
Let x and N, $x < N$ be positive integers with no common factors. The order of x modulo N is the least positive integer such that $x^r = 1 \ mod \ N$.

Example: The order of $x = 5$ modulo $N = 21$ is 6, because:

r	x^r	$x^r \bmod N$
1	5	5
2	25	4
3	125	20
4	625	16
5	3125	17
6	15625	1

Order finding, integer factoring and discrete logarithms

The *order finding problem* is to find the order r when given x and N. Order finding is believed to be a hard problem classically: it cannot be solved in time polynomial in L, where L is the number of bits needed to store N. We can use phase estimation for order finding if we use the operator

$$U|y\rangle \equiv |xy \bmod N\rangle \ . \tag{54}$$

We don't have time to cover all the details (can be found in NC 5.3) but the point is that:

- We can use phase estimation to efficiently find r when we are given x and N.
- Integer factoring can be reduced to order finding.

Therefore, we can solve integer factoring efficiently by using a quantum computer (QFT $\rightarrow$ phase estimation $\rightarrow$ order finding $\rightarrow$ integer factoring)! Also we can nearly identically use phase estimation to solve the so-called *period-finding* problem which in turn can be used to solve the *discrete logarithm* problem. These are important problems because popular cryptosystems rely on the presumed hardness of integer factoring (RSA) and discrete logarithms (elliptic curve cryptography).

Quantum search

Quantum search

Suppose you are given an unordered collection of items $\{x_1, x_2, \ldots, x_N\}$ and you are tasked to find the item which satisfies some condition $f(x_i) = 1$. On a classical computer you would solve this by going through all x_i in order and checking whether the condition is satisfied. Clearly this will take $\mathcal{O}(N)$ evaluations of f.

Surprisingly, there is a quantum algorithm which solves this problem in only $\mathcal{O}(\sqrt{N})$ steps! The algorithm is called *Grover's algorithm* or simply the *quantum search algorithm*. In principle, we can use this algorithm so speed up solutions to any problems which need an exhaustive search of some search space. This class includes many interesting, hard optimization problems such as the traveling salesman problem.

Quantum oracle

The description of the quantum search algorithm is very general and not dependent on any specifics of a given search problem. This generality is achieved using a *quantum oracle* O. The only job of this oracle is to *recognize* solutions to the search problem. The function f was the oracle in the classical case.

$$f(x) = \begin{cases} 1 & \text{if } x \text{ is a solution} \\ 0 & \text{if } x \text{ is not a solution} \end{cases} \tag{55}$$

In the quantum case we have a n qubit register $|x\rangle$ which holds the search space of size $N = 2^n$. Additionally the quantum oracle uses a ancilla/oracle qubit $|q\rangle$ which records whether $|x\rangle$ is a solution. The oracle is defined as

$$O\,|x\rangle\,|q\rangle = |x\rangle\,|q \oplus f(x)\rangle \ . \tag{56}$$

So in other words, the ancilla qubit is flipped if x is a solution we are looking for. The oracle O is a black box, its inner workings are not important for the search algorithm (those details depend on the specific search problem we are solving).

Quantum oracle

Often we omit the ancilla qubit when discussing the quantum search algorithm. The reason is that if we prepare the ancilla qubit in state

$$|q\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \ , \tag{57}$$

the action of the oracle is

$$O\,|x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = \begin{cases} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) & \text{if } x \text{ is not a solution} \\ |x\rangle \left(\frac{|1\rangle - |0\rangle}{\sqrt{2}}\right) & \text{if } x \text{ is a solution} \ . \end{cases} \tag{58}$$

Therefore we notice that

$$O\,|x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = (-1)^{f(x)}\,|x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \ . \tag{59}$$

So the state of the ancilla does not change during the computation. Therefore, effectively the action of the oracle is

$$O\,|x\rangle = (-1)^{f(x)}\,|x\rangle \ . \tag{60}$$

We say that the oracle *marks* the solution by shifting its phase while leaving non-solutions invariant.

Recognizing solutions is often easy

The job of the oracle is to *recognize* a solution, not know how to solve the problem. It is important to note that often the former is much easier than the latter.

Example: Say the we are given a large integer $n = pq$ where $p, q \gg 1$ are integers. If n is very large, then finding its factors will take a very long time on a classical computer (solving the problem). On the other hand, if you are given p and q, it is very easy to check that $pq = n$ (recognizing the solution).

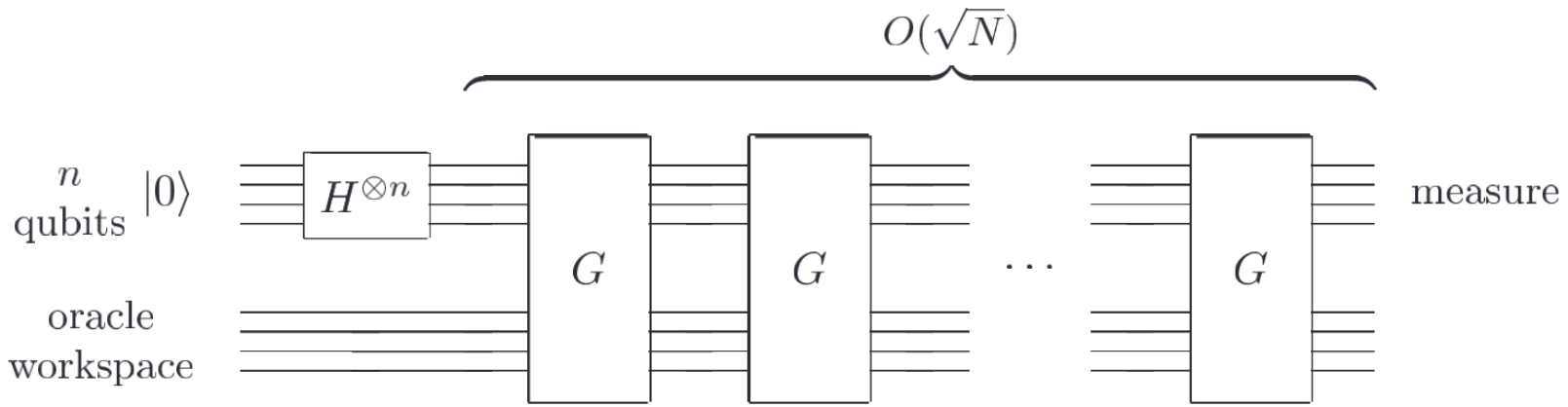
Such problems are very common in computer science. In the theory of *computational complexity* the complexity class NP ("nondeterministic polynomial time") is the class of problems where solutions can be checked easily but finding the solution may be very difficult.

Example: Not every problem is in NP. Consider playing chess and asking: given a board configuration, will white win assuming both players play perfectly? Solving this seems hard. It is even intractable to check solutions: if you were given a series of "perfect" moves for both players, how would you check that those moves are actually the best possible?

Quantum search algorithm

We'll now state the algorithm and then see how it works:

1. Prepare the equal superposition in the data register $|\psi\rangle = N^{-1/2} \sum_x |x\rangle$
2. Repeat the *Grover iteration/operator*
   - 2.1 Apply the oracle O
   - 2.2 Apply the Hadamard transform $H^{\otimes n}$
   - 2.3 Perform a conditional phase shift $|x\rangle \mapsto -(-1)^{\delta_{x,0}} |x\rangle$
   - 2.4 Apply the Hadamard transform $H^{\otimes n}$
3. Measure the data register



Above G denotes the Grover operator which performs steps 2.1-2.4.

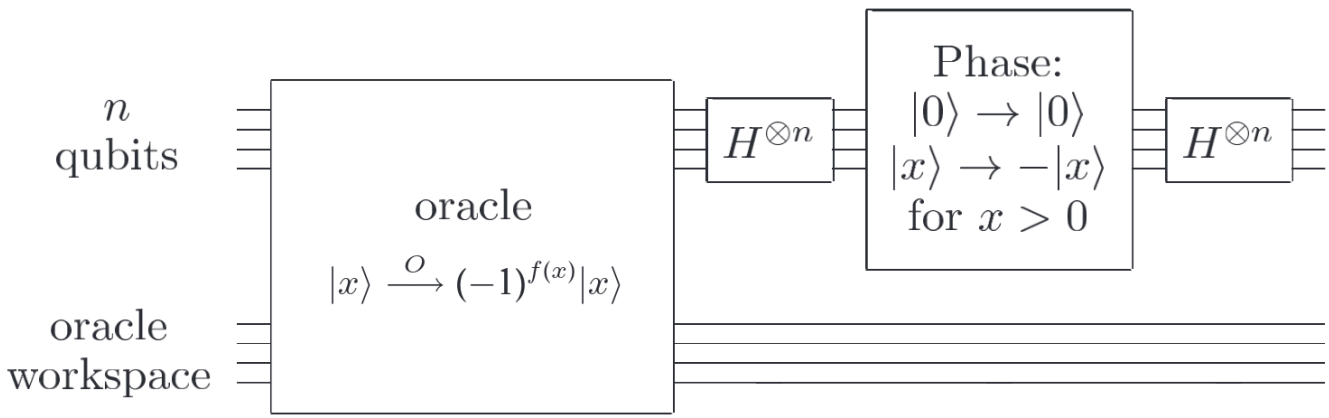
Grover iteration

The Grover iteration can be implemented efficiently using quantum gates. We don't care right now about the inner workings of the oracle but in applications it should also have an efficient implementation. The Grover iteration can be written

$$G = H^{\otimes n}(2\,|0\rangle\langle 0| - \mathbb{I})H^{\otimes n}O \tag{61}$$

$$= (2\,|\psi\rangle\langle\psi| - \mathbb{I})O\ , \tag{62}$$

where $|\psi\rangle = N^{-1/2}\sum_x |x\rangle$.



From now on we will assume that the search problem has M solutions and the search space has size N.

Grover iteration

The Grover iteration can be understood geometrically as a rotation in a certain two-dimensional subspace of the total Hilbert space. To see this, we define the two states

$$|\alpha\rangle \equiv \frac{1}{\sqrt{N-M}} \sum_x{}'' |x\rangle \tag{63}$$

$$|\beta\rangle \equiv \frac{1}{\sqrt{M}} \sum_x{}' |x\rangle \ , \tag{64}$$

where $\sum_x{}'$ sums over all x which are solutions to the search problem. Similarly $\sum_x{}''$ sums over all x which are not solutions to the search problem. Before the Grover iteration, the data register is in state $|\psi\rangle$. We can write this as a superposition of $|\alpha\rangle$ and $|\beta\rangle$

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle = \frac{1}{\sqrt{N}} \sum_x{}'' |x\rangle + \frac{1}{\sqrt{N}} \sum_x{}' |x\rangle \tag{65}$$

$$= \sqrt{\frac{N-M}{N}} |\alpha\rangle + \sqrt{\frac{M}{N}} |\beta\rangle \ . \tag{66}$$

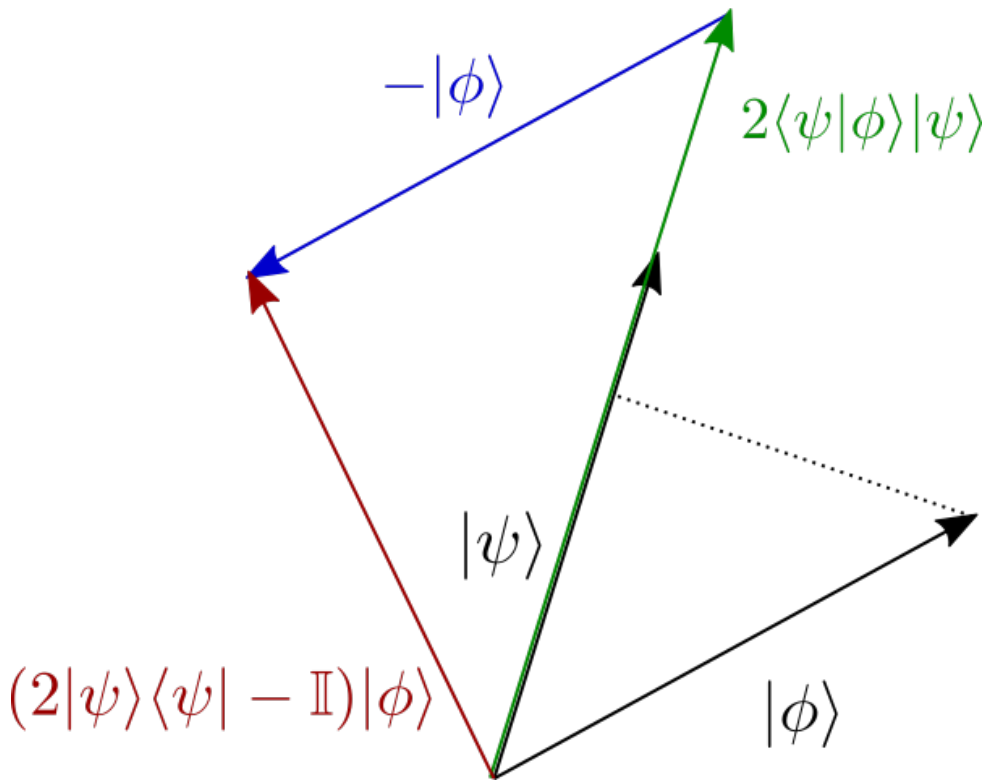
It turns out that when applying Grover operators to $|\psi\rangle$ the quantum state stays in this subspace spanned by $|\alpha\rangle$ and $|\beta\rangle$.

Grover iteration

The Grover iteration $G = (2\,|\psi\rangle\langle\psi| - \mathbb{I})O$ starts with a oracle call. Since the oracle phase shifts only solutions, it effectively performs a reflection about the vector $|\alpha\rangle$ in the plane of $|\alpha\rangle$ and $|\beta\rangle$:

$$O(a\,|\alpha\rangle + b\,|\beta\rangle) = a\,|\alpha\rangle - b\,|\beta\rangle \quad . \tag{67}$$

The next step $(2\,|\psi\rangle\langle\psi| - \mathbb{I})$ performs a reflection about the vector $|\psi\rangle$:



Grover iteration

So, G is the product of two reflections. Therefore it is a rotation in the $|\alpha\rangle$, $|\beta\rangle$ -plane (see the Fig). Let's define the rotation angle θ by

$$\cos\frac{\theta}{2} = \sqrt{\frac{N-M}{N}} \tag{68}$$

$$\rightarrow |\psi\rangle = \cos\frac{\theta}{2}|\alpha\rangle + \sin\frac{\theta}{2}|\beta\rangle \ . \tag{69}$$

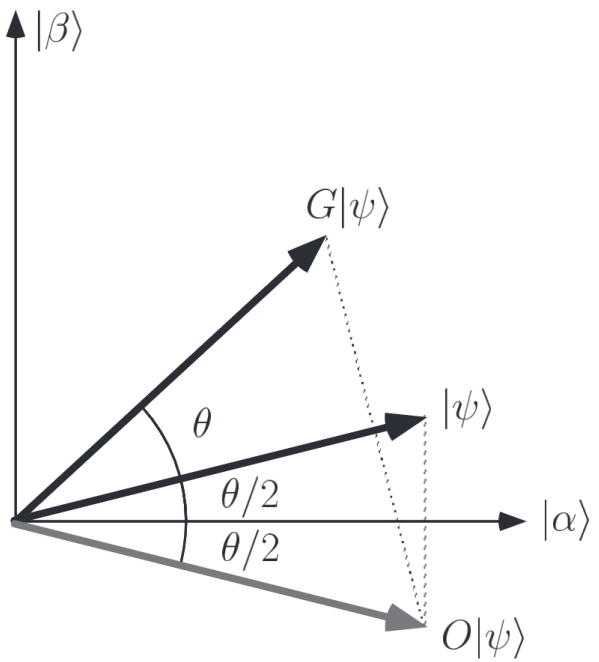
The figure shows that after one Grover iteration we have

$$G|\psi\rangle = \cos\frac{3\theta}{2}|\alpha\rangle + \sin\frac{3\theta}{2}|\beta\rangle \ . \tag{70}$$

So each time the state is rotated by θ. After k iterations

$$G^k|\psi\rangle = \cos\left(\frac{2k+1}{2}\theta\right)|\alpha\rangle \tag{71}$$

$$+ \sin\left(\frac{2k+1}{2}\theta\right)|\beta\rangle \ . \tag{72}$$



Grover iteration

The fundamental idea is then to use Grover iterations to rotate the quantum state as close to $|\beta\rangle$ as possible. If the data register is then measured, the outcome will be a solution to the search problem with high probability. How many times should we apply G to get as close as possible to $|\beta\rangle$? The picture tells us that the angle between $|\psi\rangle$ and $|\beta\rangle$ is $\arccos\sqrt{M/N}$. Therefore we should find the multiple of θ which is closest to $\arccos\sqrt{M/N}$. Therefore the number R of Grover iterations is

$$R = CI\left(\frac{\arccos\sqrt{M/N}}{\theta}\right) , \tag{73}$$

where $CI(x)$ gives the closest integer to x with the convention that halves are rounded down, $CI(3.5) = 3$. For now we'll assume $M \leq N/2$. Note that the angle $\arccos\sqrt{\frac{M}{N}} \leq \pi/2$ and therefore $R \leq \lceil \pi/2\theta \rceil$. Then because $\theta/2 = \arccos\sqrt{\frac{N-M}{N}} \geq \sqrt{\frac{M}{N}}$, we have

$$R \leq \left\lceil \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rceil . \tag{74}$$

So, remarkably we need only $R = \mathcal{O}(\sqrt{N/M})$ oracle calls to search a set of N items!

Quantum counting

In the quantum search algorithm we assumed that the number of solutions M is known in advance. What if we don't know M?

The number of solutions M can be found by using the phase estimation algorithm with the Grover operator G. The Grover operator has eigenvalues $e^{\pm i\theta}$. The phase θ can be estimated using phase estimation which requires $\mathcal{O}(\sqrt{N})$ oracle calls. When θ is known, we can solve for M. This procedure is known as *quantum counting*.[1] Therefore we can use quantum counting to find the number of solutions M and then run quantum search, all using only $\mathcal{O}(\sqrt{N})$ oracle calls.

Quantum counting and search can in principle be used to speed up the finding of solutions to NP-hard problems. By definition of NP, solutions to those problems can be checked efficiently and made into an oracle. This is not necessarily practical because the speedup over brute-force classical search is only quadratic.

[1] For details, see NC 6.3.

Summary: Quantum search algorithm

Summary of Grover search:

1. Initialize quantum registers: $|0\rangle^{\otimes n} |0\rangle$
2. Apply $H^{\otimes n}$ on the first and HX on the second register: $N^{-1/2} \sum_x |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$
3. Apply Grover iteration R times: $\approx |\beta\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$
4. Measure the first register in the computational basis to get one solution to the search problem