Chapter 6

Quantum Computation

6.1 Classical Circuits

The concept of a quantum computer was introduced in Chapter 1. Here we will specify our model of quantum computation more precisely, and we will point out some basic properties of the model. But before we explain what a quantum computer does, perhaps we should say what a classical computer does.

6.1.1 Universal gates

A classical (deterministic) computer evaluates a function: given n-bits of input it produces m-bits of output that are uniquely determined by the input; that is, it finds the value of

$$f : \{0,1\}^n \to \{0,1\}^m, \tag{6.1}$$

for a particular specified n-bit argument. A function with an m-bit value is equivalent to m functions, each with a one-bit value, so we may just as well say that the basic task performed by a computer is the evaluation of

$$f : \{0,1\}^n \to \{0,1\}. \tag{6.2}$$

We can easily count the number of such functions. There are 2^n possible inputs, and for each input there are two possible outputs. So there are altogether 2^{2^n} functions taking n bits to one bit.

The evaluation of any such function can be reduced to a sequence of elementary logical operations. Let us divide the possible values of the input

$$x = x_1 x_2 x_3 \ldots x_n, \tag{6.3}$$

into one set of values for which $f(x) = 1$, and a complementary set for which $f(x) = 0$. For each $x^{(a)}$ such that $f(x^{(a)}) = 1$, consider the function $f^{(a)}$ such that

$$f^{(a)}(x) = \begin{cases} 1 & x = x^{(a)} \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

Then

$$f(x) = f^{(1)}(x) \vee f^{(2)}(x) \vee f^{(3)}(x) \vee \ldots . \tag{6.5}$$

f is the logical OR ($\vee$) of all the $f^{(a)}$'s. In binary arithmetic the $\vee$ operation of two bits may be represented

$$x \vee y = x + y - x \cdot y; \tag{6.6}$$

it has the value 0 if x and y are both zero, and the value 1 otherwise.

Now consider the evaluation of $f^{(a)}$. In the case where $x^{(a)} = 111\ldots 1$, we may write

$$f^{(a)}(x) = x_1 \wedge x_2 \wedge x_3 \ldots \wedge x_n; \tag{6.7}$$

it is the logical AND ($\wedge$) of all n bits. In binary arithmetic, the AND is the product

$$x \wedge y = x \cdot y. \tag{6.8}$$

For any other $x^{(a)}$, $f^{(a)}$ is again obtained as the AND of n bits, but where the NOT ($\neg$) operation is first applied to each x_i such that $x_i^{(a)} = 0$; for example

$$f^{(a)}(x) = (\neg x_1) \wedge x_2 \wedge x_3 \wedge (\neg x_4) \wedge \ldots \tag{6.9}$$

if

$$x^{(a)} = 0110\ldots . \tag{6.10}$$

The NOT operation is represented in binary arithmetic as

$$\neg x = 1 - x. \tag{6.11}$$

We have now constructed the function $f(x)$ from three elementary logical connectives: NOT, AND, OR. The expression we obtained is called the "disjunctive normal form" of $f(x)$. We have also implicitly used another operation, COPY, that takes one bit to two bits:

$$\text{COPY} : x \to xx. \tag{6.12}$$

We need the COPY operation because each $f^{(a)}$ in the disjunctive normal form expansion of f requires its own copy of x to act on.

In fact, we can pare our set of elementary logical connectives to a smaller set. Let us define a NAND ("NOT AND") operation by

$$x \uparrow y = \neg(x \wedge y) = (\neg x) \vee (\neg y). \tag{6.13}$$

In binary arithmetic, the NAND operation is

$$x \uparrow y = 1 - xy. \tag{6.14}$$

If we can COPY, we can use NAND to perform NOT:

$$x \uparrow x = 1 - x^2 = 1 - x = \neg x. \tag{6.15}$$

(Alternatively, if we can prepare the constant $y = 1$, then $x \uparrow 1 = 1 - x = \neg x$.) Also,

$$(x \uparrow y) \uparrow (x \uparrow y) = \neg(x \uparrow y) = 1 - (1 - xy) = xy = x \wedge y, \tag{6.16}$$

and

$$\begin{aligned}(x \uparrow x) \uparrow (y \uparrow y) &= (\neg x) \uparrow (\neg y) = 1 - (1 - x)(1 - y) \\ &= x + y - xy = x \vee y.\end{aligned} \tag{6.17}$$

So if we can COPY, NAND performs AND and OR as well. We conclude that the single logical connective NAND, together with COPY, suffices to evaluate any function f. (You can check that an alternative possible choice of the universal connective is NOR:

$$x \downarrow y = \neg(x \vee y) = (\neg x) \wedge (\neg y).) \tag{6.18}$$

If we are able to prepare a constant bit ($x = 0$ or $x = 1$), we can reduce the number of elementary operations from two to one. The NAND/NOT gate

$$(x, y) \to (1 - x, 1 - xy), \tag{6.19}$$

computes NAND (if we ignore the first output bit) and performs copy (if we set the second input bit to $y = 1$, and we subsequently apply NOT to both output bits). We say, therefore, that NAND/NOT is a universal gate. If we have a supply of constant bits, and we can apply the NAND/NOT gates to any chosen pair of input bits, then we can perform a sequence of NAND/NOT gates to evaluate any function $f : \{0,1\}^n \to \{0,1\}$ for any value of the input $x = x_1x_2 \ldots x_n$.

These considerations motivate the circuit model of computation. A computer has a few basic components that can perform elementary operations on bits or pairs of bits, such as COPY, NOT, AND, OR. It can also prepare a constant bit or input a variable bit. A computation is a finite sequence of such operations, a *circuit*, applied to a specified string of input bits.[1] The result of the computation is the final value of all remaining bits, after all the elementary operations have been executed.

It is a fundamental result in the theory of computation that just a few elementary gates suffice to evaluate any function of a finite input. This result means that with very simple hardware components, we can build up arbitrarily complex computations.

So far, we have only considered a computation that acts on a particular fixed input, but we may also consider *families* of circuits that act on inputs of variable size. Circuit families provide a useful scheme for analyzing and classifying the *complexity* of computations, a scheme that will have a natural generalization when we turn to quantum computation.

6.1.2 Circuit complexity

In the study of complexity, we will often be interested in functions with a one-bit output

$$f : \{0,1\}^n \to \{0,1\}. \tag{6.20}$$

[1] The circuit is required to be *acyclic*, meaning that no *directed* closed loops are permitted.

Such a function f may be said to encode a solution to a "decision problem" — the function examines the input and issues a YES or NO answer. Often, a question that would not be stated colloquially as a question with a YES/NO answer can be "repackaged" as a decision problem. For example, the function that defines the FACTORING problem is:

$$f(x,y) = \begin{cases} 1 & \text{if integer } x \text{ has a divisor less than } y, \\ 0 & \text{otherwise;} \end{cases} \tag{6.21}$$

knowing $f(x,y)$ for all $y < x$ is equivalent to knowing the *least* nontrivial factor of y. Another important example of a decision problem is the HAMILTONIAN path problem: let the input be an ℓ-vertex graph, represented by an $\ell \times \ell$ adjacency matrix (a 1 in the ij entry means there is an edge linking vertices i and j); the function is

$$f(x) = \begin{cases} 1 & \text{if graph } x \text{ has a Hamiltonian path,} \\ 0 & \text{otherwise.} \end{cases} \tag{6.22}$$

(A path is Hamiltonian if it visits each vertex exactly once.)

We wish to gauge how hard a problem is by quantifying the resources needed to solve the problem. For a decision problem, a reasonable measure of hardness is the *size* of the smallest circuit that computes the corresponding function $f : \{0,1\}^n \to \{0,1\}$. By size we mean the number of elementary gates or components that we must wire together to evaluate f. We may also be interested in how much *time* it takes to do the computation if many gates are permitted to execute in parallel. The *depth* of a circuit is the number of time steps required, assuming that gates acting on distinct bits can operate simultaneously (that is, the depth is the maximum length of a directed path from the input to the output of the circuit). The *width* of a circuit is the maximum number of gates that act in any one time step.

We would like to divide the decision problems into two classes: easy and hard. But where should we draw the line? For this purpose, we consider infinite families of decision problems with variable input size; that is, where the number of bits of input can be any integer n. Then we can examine how the size of the circuit that solves the problem scales with n.

If we use the scaling behavior of a circuit family to characterize the difficulty of a problem, there is a subtlety. It would be cheating to hide the difficulty of the problem in the *design* of the circuit. Therefore, we should

restrict attention to circuit families that have acceptable "uniformity" properties — it must be "easy" to build the circuit with $n+1$ bits of input once we have constructed the circuit with an n-bit input.

Associated with a family of functions $\{f_n\}$ (where f_n has n-bit input) are circuits $\{C_n\}$ that compute the functions. We say that a circuit family $\{C_n\}$ is "polynomial size" if the size of C_n grows with n no faster than a power of n,

$$\text{size } (C_n) \le \text{poly } (n), \tag{6.23}$$

where poly denotes a polynomial. Then we define:

$$P = \{\text{decision problem solved by polynomial-size circuit families}\}$$

(P for "polynomial time"). Decision problems in P are "easy." The rest are "hard." Notice that C_n computes $f_n(x)$ for every possible n-bit input, and therefore, if a decision problem is in P we can find the answer even for the "worst-case" input using a circuit of size no greater than poly(n). (As noted above, we implicitly assume that the circuit family is "uniform" so that the *design* of the circuit can itself be solved by a polynomial-time algorithm. Under this assumption, solvability in polynomial time by a circuit family is equivalent to solvability in polynomial time by a universal Turing machine.)

Of course, to determine the size of a circuit that computes f_n, we must know what the elementary components of the circuit are. Fortunately, though, whether a problem lies in P does not depend on what gate set we choose, as long as the gates are universal, the gate set is finite, and each gate acts on a set of bits of bounded size. One universal gate set can *simulate* another.

The vast majority of function families $f : \{0,1\}^n \to \{0,1\}$ are not in P. For most functions, the output is essentially random, and there is no better way to "compute" $f(x)$ than to consult a look-up table of its values. Since there are 2^n n-bit inputs, the look-up table has *exponential* size, and a circuit that encodes the table must also have exponential size. The problems in P belong to a very special class — they have enough structure so that the function f can be computed efficiently.

Of particular interest are decision problems that can be answered by exhibiting an example that is easy to verify. For example, given x and $y < x$, it is hard (in the worst case) to determine if x has a factor less than y. But if someone kindly provides a $z < y$ that divides x, it is easy for us to check that z is indeed a factor of x. Similarly, it is hard to determine if a graph has a Hamiltonian path, but if someone kindly provides a path, it is easy to verify that the path really is Hamiltonian.

This concept that a problem may be hard to solve, but that a solution can be easily verified once found, can be formalized by the notion of a "nondeterministic" circuit. A nondeterministic circuit $\tilde{C}_{n,m}(x^{(n)}, y^{(m)})$ associated with the circuit $C_n(x^{(n)})$ has the property:

$$C_n(x^{(n)}) = 1 \text{ iff } \tilde{C}_{n,m}(x^{(n)}, y^{(m)}) = 1 \quad \text{for } \textit{some } y^{(m)}. \tag{6.24}$$

(where $x^{(n)}$ is n bits and $y^{(m)}$ is m bits.) Thus for a particular $x^{(n)}$ we can use $\tilde{C}_{n,m}$ to *verify* that $C_n(x^{(n)} = 1$, *if* we are fortunate enough to have the right $y^{(m)}$ in hand. We define:

> NP: {decision problems that admit a polynomial-size *nondeterministic* circuit family}

(NP for "nondeterministic polynomial time"). If a problem is in NP, there is no guarantee that the problem is easy, only that a solution is easy to check once we have the right information. Evidently $P \subseteq NP$. Like P, the NP problems are a small subclass of all decision problems.

Much of complexity theory is built on a fundamental conjecture:

$$\text{Conjecture} : P \neq NP; \tag{6.25}$$

there exist hard decision problems whose solutions are easily verified. Unfortunately, this important conjecture still awaits proof. But after 30 years of trying to show otherwise, most complexity experts are firmly confident of its validity.

An important example of a problem in NP is CIRCUIT-SAT. In this case the input is a circuit C with n gates, m input bits, and one output bit. The problem is to find if there is *any* m-bit input for which the output is 1. The function to be evaluated is

$$f(C) = \begin{cases} 1 & \text{if there exists } x^{(m)} \text{ with } C(x^{(m)}) = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{6.26}$$

This problem is in NP because, *given* a circuit, it is easy to *simulate* the circuit and evaluate its output for any particular input.

I'm going to state some important results in complexity theory that will be relevant for us. There won't be time for proofs. You can find out more

by consulting one of the many textbooks on the subject; one good one is *Computers and Intractability: A Guide to the Theory of NP-Completeness*, by M. R. Garey and D. S. Johnson.

Many of the insights engendered by complexity theory flow from Cook's Theorem (1971). The theorem states that *every* problem in NP is *polynomially reducible* to CIRCUIT-SAT. This means that for any PROBLEM $\in NP$, there is a polynomial-size circuit family that maps an "instance" $x^{(n)}$ of PROBLEM to an "instance" $y^{(m)}$ of CIRCUIT-SAT; that is

$$\text{CIRCUIT} - \text{SAT } (y^{(m)}) = 1 \text{ iff PROBLEM } (x^{(n)}) = 1. \tag{6.27}$$

It follows that if we had a magical device that could efficiently solve CIRCUIT-SAT (a CIRCUIT-SAT "oracle"), we could couple that device with the polynomial reduction to efficiently solve PROBLEM. Cook's theorem tells us that if it turns out that CIRCUIT-SAT $\in P$, then $P = NP$.

A problem that, like CIRCUIT-SAT, has the property that every problem in NP is polynomially reducible to it, is called *NP-complete (NPC)*. Since Cook, many other examples have been found. To show that a PROBLEM $\in NP$ is NP-complete, it suffices to find a polynomial reduction to PROBLEM of another problem that is already known to be NP-complete. For example, one can exhibit a polynomial reduction of CIRCUIT-SAT to HAMILTONIAN. It follows from Cook's theorem that HAMILTONIAN is also NP-complete.

If we assume that $P \neq NP$, it follows that there exist problems in NP of intermediate difficulty (the class NPI). These are neither P nor NPC.

Another important complexity class is called co-NP. Heuristically, NP decision problems are ones we can answer by exhibiting an *example* if the answer is YES, while co-NP problems can be answered with a *counter-example* if the answer is NO. More formally:

$$\{C\} \in NP : C(x) = 1 \text{ iff } C(x, y) = 1 \text{ for some } y \tag{6.28}$$
$$\{C\} \in \text{co}-NP : C(x) = 1 \text{ iff } C(x, y) = 1 \text{ for all } y. \tag{6.29}$$

Clearly, there is a symmetry relating the classes NP and co-NP — whether we consider a problem to be in NP or co-NP depends on how we choose to frame the question. ("Is there a Hamiltonian circuit?" is in NP. "Is there no Hamiltonian circuit?" is in co-NP). But the interesting question is: is a problem in *both* NP and co-NP? If so, then we can easily verify the answer (once a suitable example is in hand) regardless of whether the answer is YES or NO. It is believed (though not proved) that $NP \neq$ co$-NP$. (For example, we can show that a graph has a Hamiltonian path by exhibiting an example, but we don't know how to show that it has *no* Hamiltonian path that way!) *Assuming* that $NP \neq$ co$-NP$, there is a theorem that says that no co-NP problems are contained in NPC. Therefore, problems in the intersection of NP and co-NP, if not in P, are good candidates for inclusion in NPI.

In fact, a problem in $NP \cap$ co$-NP$ that is believed not in P is the FACTORING problem. As already noted, FACTORING is in NP because, if we are offered a factor of x, we can easily check its validity. But it is also in co-NP, because it is known that if we are given a prime number then (at least in principle), we can efficiently verify its primality. Thus, if someone tells us the prime factors of x, we can efficiently check that the prime factorization is right, and can *exclude* that any integer less than y is a divisor of x. Therefore, it seems likely that FACTORING is in NPI.

We are led to a crude (conjectured) picture of the structure of $NP \cup$ co$-NP$. NP and co-NP do not coincide, but they have a nontrivial intersection. P lies in $NP \cap$ co$-NP$ (because $P =$ co$-P$), but the intersection also contains problems not in P (like FACTORING). Neither NPC nor co-NPC intersects with $NP \cap$ co$-NP$.

There is much more to say about complexity theory, but we will be content to mention one more element that relates to the discussion of quantum complexity. It is sometimes useful to consider *probabilistic* circuits that have access to a random number generator. For example, a gate in a probabilistic circuit might act in either one of two ways, and flip a fair coin to decide which action to execute. Such a circuit, for a single fixed input, can sample many possible computational paths. An algorithm performed by a probabilistic circuit is said to be "randomized."

If we attack a decision problem using a probabilistic computer, we attain a probability distribution of outputs. Thus, we won't necessarily always get the right answer. But if the probability of getting the right answer is larger than $\frac{1}{2}+\delta$ for every possible input ($\delta > 0$), then the machine is useful. In fact, we can run the computation many times and use majority voting to achieve an error probability less than ε. Furthermore, the number of times we need to repeat the computation is only polylogarithmic in ε^{-1}.

If a problem admits a probabilistic circuit family of polynomial size that always gives the right answer with probability larger than $\frac{1}{2}+\delta$ (for any input, and for fixed $\delta > 0$), we say the problem is in the class BPP ("bounded-error probabilistic polynomial time"). It is evident that

$$P \subseteq BPP, \tag{6.30}$$

but the relation of NP to BPP is not known. In particular, it has not been proved that BPP is contained in NP.

6.1.3 Reversible computation

In devising a model of a quantum computer, we will generalize the circuit model of classical computation. But our quantum logic gates will be unitary transformations, and hence will be invertible, while classical logic gates like the NAND gate are not invertible. Before we discuss quantum circuits, it is useful to consider some features of reversible classical computation.

Aside from the connection with quantum computation, another incentive for studying reversible classical computation arose in Chapter 1. As Landauer observed, because irreversible logic elements erase information, they are necessarily dissipative, and therefore, require an irreducible expenditure of power. But if a computer operates reversibly, then in principle there need be no dissipation and no power requirement. We can compute for free!

A reversible computer evaluates an invertible function taking n bits to n bits

$$f : \{0,1\}^n \to \{0,1\}^n, \tag{6.31}$$

the function must be invertible so that there is a unique input for each output; then we are able in principle to run the computation backwards and recover the input from the output. Since it is a 1-1 function, we can regard it as a permutation of the 2^n strings of n bits — there are $(2^n)!$ such functions.

Of course, any irreversible computation can be "packaged" as an evaluation of an invertible function. For example, for any $f : \{0,1\}^n \to \{0,1\}^m$, we can construct $\tilde{f} : \{0,1\}^{n+m} \to \{0,1\}^{n+m}$ such that

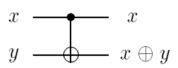
$$\tilde{f}(x; 0^{(m)}) = (x; f(x)), \tag{6.32}$$

(where $0^{(m)}$ denotes m-bits initially set to zero). Since $\tilde{f}$ takes each $(x; 0^{(m)})$ to a distinct output, it can be extended to an invertible function of $n+m$ bits. So for any f taking n bits to m, there is an invertible $\tilde{f}$ taking $n+m$ to $n+m$, which evaluates $f(x)$ acting on $(x, 0^{(m)})$

Now, how do we build up a complicated reversible computation from elementary components — that is, what constitutes a universal gate set? We will see that one-bit and two-bit reversible gates do not suffice; we will need three-bit gates for universal reversible computation.

Of the four 1-bit $\rightarrow$ 1-bit gates, two are reversible; the trivial gate and the NOT gate. Of the $(2^4)^2 = 256$ possible 2-bit $\rightarrow$ 2-bit gates, $4! = 24$ are reversible. One of special interest is the controlled-NOT or reversible XOR gate that we already encountered in Chapter 4:

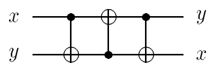
$$\text{XOR}: \ (x, y) \mapsto (x, x \oplus y), \tag{6.33}$$



This gate flips the second bit if the first is 1, and does nothing if the first bit is 0 (hence the name controlled-NOT). Its square is trivial, that is, it inverts itself. Of course, this gate performs a NOT on the second bit if the first bit is set to 1, and it performs the copy operation if y is initially set to zero:

$$\text{XOR}: \ (x, 0) \mapsto (x, x). \tag{6.34}$$

With the circuit



constructed from three X0R's, we can swap two bits:

$$(x, y) \rightarrow (x, x \oplus y) \rightarrow (y, x \oplus y) \rightarrow (y, x). \tag{6.35}$$

With these swaps we can shuffle bits around in a circuit, bringing them together if we want to act on them with a particular component in a fixed location.

To see that the one-bit and two-bit gates are nonuniversal, we observe that all these gates are *linear*. Each reversible two-bit gate has an action of the form

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x' \\ y' \end{pmatrix} = \mathcal{M} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a \\ b \end{pmatrix}, \tag{6.36}$$

where the constant $\binom{a}{b}$ takes one of four possible values, and the matrix $\mathcal{M}$ is one of the six invertible matrices

$$\mathcal{M} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}. \tag{6.37}$$

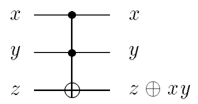
(All addition is performed modulo 2.) Combining the six choices for $\mathcal{M}$ with the four possible constants, we obtain 24 distinct gates, which exhausts all the reversible $2 \rightarrow 2$ gates.

Since the linear transformations are closed under composition, any circuit composed from reversible $2 \rightarrow 2$ (and $1 \rightarrow 1$) gates will compute a linear function

$$x \rightarrow \mathcal{M}x + a. \tag{6.38}$$

But for $n \geq 3$, there are invertible functions on n-bits that are nonlinear. An important example is the 3-bit *Toffoli gate* (or controlled-controlled-NOT) $\theta^{(3)}$

$$\theta^{(3)} : (x, y, z) \rightarrow (x, y, z \oplus xy); \tag{6.39}$$



it flips the third bit if the first two are 1 and does nothing otherwise. Like the XOR gate, it is its own inverse.

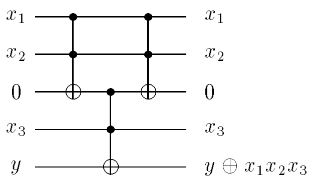
Unlike the reversible 2-bit gates, the Toffoli gate serves as a universal gate for Boolean logic, if we can provide fixed input bits and ignore output bits. If z is initially 1, then $x \uparrow y = 1 - xy$ appears in the third output — we can perform NAND. If we fix $x = 1$, the Toffoli gate functions like an XOR gate, and we can use it to copy.

The Toffoli gate $\theta^{(3)}$ is universal in the sense that we can build a circuit to compute any reversible function using Toffoli gates alone (if we can fix input bits and ignore output bits). It will be instructive to show this directly, without relying on our earlier argument that NAND/NOT is universal for Boolean functions. In fact, we can show the following: From the NOT gate and the Toffoli gate $\theta^{(3)}$, we can construct any invertible function on n bits, provided we have one extra bit of scratchpad space available.

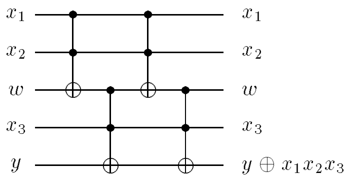
The first step is to show that from the three-bit Toffoli-gate $\theta^{(3)}$ we can construct an n-bit Toffoli gate $\theta^{(n)}$ that acts as

$$(x_1, x_2, \dots x_{n-1}, y) \to (x_1, x_2, \dots, x_{n-1} y \oplus x_1 x_2 \dots x_{n-1}). \tag{6.40}$$

The construction requires one extra bit of scratch space. For example, we construct $\theta^{(4)}$ from $\theta^{(3)}$'s with the circuit



The purpose of the last $\theta^{(3)}$ gate is to reset the scratch bit back to its original value zero. Actually, with one more gate we can obtain an implementation of $\theta^{(4)}$ that works irrespective of the initial value of the scratch bit:

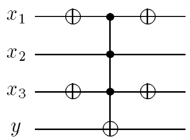


Again, we can eliminate the last gate if we don't mind flipping the value of the scratch bit.

We can see that the scratch bit really is necessary, because $\theta^{(4)}$ is an odd permutation (in fact a transposition) of the 24 4-bit strings — it transposes 1111 and 1110. But $\theta^{(3)}$ acting on any three of the four bits is an even permutation; *e.g.*, acting on the last three bits it transposes 0111 with 0110,

and 1111 with 1110. Since a product of even permutations is also even, we cannot obtain $\theta^{(4)}$ as a product of $\theta^{(3)}$'s that act on four bits only.

The construction of $\theta^{(4)}$ from four $\theta^{(3)}$'s generalizes immediately to the construction of $\theta^{(n)}$ from two $\theta^{(n-1)}$'s and two $\theta^{(3)}$'s (just expand x_1 to several control bits in the above diagram). Iterating the construction, we obtain $\theta^{(n)}$ from a circuit with $2^{n-2} + 2^{n-3} - 2$ $\theta^{(3)}$'s. Furthermore, just one bit of scratch space is sufficient.[2]) (When we need to construct $\theta^{(k)}$, any available extra bit will do, since the circuit returns the scratch bit to its original value. The next step is to note that, by conjugating $\theta^{(n)}$ with NOT gates, we can in effect modify the value of the control string that "triggers" the gate. For example, the circuit



flips the value of y if $x_1x_2x_3 = 010$, and it acts trivially otherwise. Thus this circuit transposes the two strings 0100 and 0101. In like fashion, with $\theta^{(n)}$ and NOT gates, we can devise a circuit that transposes any two n-bit strings that differ in only one bit. (The location of the bit where they differ is chosen to be the *target* of the $\theta^{(n)}$ gate.)

But in fact a transposition that exchanges any two n-bit strings can be expressed as a product of transpositions that interchange strings that differ in only one bit. If a_0 and a_s are two strings that are Hamming distance s apart (differ in s places), then there is a chain

$$a_0, a_1, a_2, a_3, \dots, a_s, \tag{6.41}$$

such that each string in the chain is Hamming distance one from its neighbors. Therefore, each of the transpositions

$$(a_0a_1), (a_1a_2), (a_2a_3), \dots (a_{s-1}a_s), \tag{6.42}$$

[2]With more scratch space, we can build $\theta^{(n)}$ from $\theta^{(3)}$'s much more efficiently — see the exercises.

can be implemented as a $\theta^{(n)}$ gate conjugated by NOT gates. By composing transpositions we find

$$\begin{aligned}(a_0 a_s) = (a_{s-1}a_s)(a_{s-2}a_{s-1})\ldots(a_2a_3)(a_1a_2)(a_0a_1)(a_1a_2)(a_2a_3)\\ \ldots(a_{s-2}a_{s-1})(a_{s-1}a_s);\end{aligned} \tag{6.43}$$

we can construct the Hamming-distance-s transposition from $2s-1$ Hamming-distance-one transpositions. It follows that we can construct (a_0a_s) from $\theta^{(n)}$'s and NOT gates.

Finally, since every permutation is a product of transpositions, we have shown that every invertible function on n-bits (every permutation on n-bit strings) is a product of $\theta^{(3)}$'s and NOT's, using just one bit of scratch space.

Of course, a NOT can be performed with a $\theta^{(3)}$ gate if we fix two input bits at 1. Thus the Toffoli gate $\theta^{(3)}$ is universal for reversible computation, if we can fix input bits and discard output bits.

6.1.4 Billiard ball computer

Two-bit gates suffice for universal irreversible computation, but three-bit gates are needed for universal reversible computation. One is tempted to remark that "three-body interactions" are needed, so that building reversible hardware is more challenging than building irreversible hardware. However, this statement may be somewhat misleading.

Fredkin described how to devise a universal reversible computer in which the fundamental interaction is an elastic collision between two billiard balls. Balls of radius $\frac{1}{\sqrt{2}}$ move on a square lattice with unit lattice spacing. At each integer valued time, the center of each ball lies at a lattice site; the presence or absence of a ball at a particular site (at integer time) encodes a bit of information. In each unit of time, each ball moves unit distance along one of the lattice directions. Occasionally, at integer-valued times, 90^o elastic collisions occur between two balls that occupy sites that are distance $\sqrt{2}$ apart (joined by a lattice diagonal).

The device is programmed by nailing down balls at certain sites, so that those balls act as perfect reflectors. The program is executed by fixing initial positions and directions for the moving balls, and evolving the system according to Newtonian mechanics for a finite time. We read the output by observing the final positions of all the moving balls. The collisions are nondissipative, so that we can run the computation backward by reversing the velocities of all the balls.

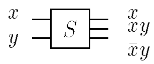
To show that this machine is a universal reversible computer, we must explain how to operate a universal gate. It is convenient to consider the three-bit *Fredkin gate*

$$(x, y, z) \to (x, xz + \bar{x}y, xy + \bar{x}z), \tag{6.44}$$

which swaps y and z if $x = 0$ (we have introduced the notation $\bar{x} = \neg x$). You can check that the Fredkin gate can simulate a NAND/NOT gate if we fix inputs and ignore outputs.

We can build the Fredkin gate from a more primitive object, the *switch gate*. A switch gate taking two bits to three acts as

$$(x, y) \to (x, xy, \bar{x}y). \tag{6.45}$$



The gate is "reversible" in that we can run it backwards acting on a constrained 3-bit input taking one of the four values

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \tag{6.46}$$

Furthermore, the switch gate is itself universal; fixing inputs and ignoring outputs, it can do NOT ($y = 1$, third output) AND (second output), and COPY ($y = 1$, first and second output). It is not surprising, then, that we can compose switch gates to construct a universal reversible $3 \to 3$ gate. Indeed, the circuit

builds the Fredkin gate from four switch gates (two running forward and two running backward). Time delays needed to maintain synchronization are not explicitly shown.

In the billiard ball computer, the switch gate is constructed with two reflectors, such that (in the case $x = y = 1$) two moving balls collide twice. The trajectories of the balls in this case are:

A ball labeled x emerges from the gate along the same trajectory (and at the same time) regardless of whether the other ball is present. But for $x = 1$, the position of the other ball (if present) is shifted down compared to its final position for $x = 0$ — this is a switch gate. Since we can perform a switch gate, we can construct a Fredkin gate, and implement universal reversible logic with a billiard ball computer.

An evident weakness of the billiard-ball scheme is that initial errors in the positions and velocities of the ball will accumulate rapidly, and the computer will eventually fail. As we noted in Chapter 1 (and Landauer has insistently pointed out) a similar problem will afflict any proposed scheme for dissipationless computation. To control errors we must be able to compress the phase space of the device, which will necessarily be a dissipative process.

6.1.5 Saving space

But even aside from the issue of error control there is another key question about reversible computation. How do we manage the scratchpad space needed to compute reversibly?

In our discussion of the universality of the Toffoli gate, we saw that in principle we can do any reversible computation with very little scratch space. But in practice it may be impossibly difficult to figure out how to do a particular computation with minimal space, and in any case economizing on space may be costly in terms of the run time.

There is a general strategy for simulating an irreversible computation on a reversible computer. Each irreversible NAND or COPY gate can be simulated by a Toffoli gate by fixing inputs and ignoring outputs. We accumulate and save all "garbage" output bits that are needed to reverse the steps of the computation. The computation proceeds to completion, and then a copy of the output is generated. (This COPY operation is logically reversible.) Then the computation runs in reverse, cleaning up all garbage bits, and returning all registers to their original configurations. With this procedure the reversible circuit runs only about twice as long as the irreversible circuit that it simulates, and all garbage generated in the simulation is disposed of without any dissipation and hence no power requirement.

This procedure works, but demands a huge amount of scratch space – the space needed scales linearly with the length T of the irreversible computation being simulated. In fact, it is possible to use space far more efficiently (with only a minor slowdown), so that the space required scales like $\log T$ instead of T. (That is, there is a general-purpose scheme that requires space $\propto \log T$; of course, we might do even better in the simulation of a particular computation.)

To use space more effectively, we will divide the computation into smaller steps of roughly equal size, and we will run these steps backward when possible during the course of the computation. However, just as we are unable to perform step k of the computation unless step $k-1$ has already been completed, we are unable to run step k *in reverse* if step $k-1$ has previously been executed in reverse.[3] The amount of space we require (to store our garbage) will scale like the maximum value of the number of forward steps minus the number of backward steps that have been executed.

The challenge we face can be likened to a game — the *reversible pebble game*.[4] The steps to be executed form a one-dimension directed graph with sites labeled $1, 2, 3 \ldots T$. Execution of step k is modeled by placing a pebble on the kth site of the graph, and executing step k in reverse is modeled as removal of a pebble from site k. At the beginning of the game, no sites are covered by pebbles, and in each turn we add or remove a pebble. But we cannot place a pebble at site k (except for $k = 1$) unless site $k-1$ is already covered by a pebble, and we cannot remove a pebble from site k (except for $k = 1$) unless site $k-1$ is covered. The object is to cover site T (complete the computation) without using more pebbles than necessary (generating a minimal amount of garbage).

In fact, with n pebbles we can reach site $T = 2^n - 1$, but we can go no further.

We can construct a recursive procedure that enables us to reach site $T = 2^{n-1}$ with n pebbles, leaving only one pebble in play. Let $F_1(k)$ denote placing a pebble at site k, and $F_1(k)^{-1}$ denote removing a pebble from site k. Then

$$F_2(1,2) = F_1(1)F_1(2)F_1(1)^{-1}, \tag{6.47}$$

leaves a pebble at site $k = 2$, using a maximum of two pebbles at intermediate

[3] We make the conservative assumption that we are not clever enough to know ahead of time what portion of the output from step $k-1$ might be needed later on. So we store a complete record of the configuration of the machine after step $k-1$, which is not to be erased until an updated record has been stored after the completion of a subsequent step.

[4] as pointed out by Bennett. For a recent discussion, see M. Li and P. Vitanyi, `quant-ph/9703022`.

stages. Similarly

$$F_3(1,4) = F_2(1,2)F_2(3,4)F_2(1,2)^{-1}, \qquad (6.48)$$

reaches site $k = 4$ using a maximum of three pebbles, and

$$F_4(1,8) = F_3(1,4)F_3(5,8)F_3(1,4)^{-1}, \qquad (6.49)$$

reaches $k = 8$ using four pebbles. Evidently we can construct $F_n(1, 2^{n-1})$ which uses a maximum of n pebbles and leaves a single pebble in play. (The routine

$$F_n(1,2^{n-1})F_{n-1}(2^{n-1}+1, 2^{n-1}+2^{n-2}) \ldots F_1(2^n - 1), \qquad (6.50)$$

leaves all n pebbles in play, with the maximal pebble at site $k = 2^n - 1$.)

Interpreted as a routine for executing $T = 2^{n-1}$ steps of a computation, this strategy for playing the pebble game represents a simulation requiring space scaling like $n \sim \log T$. How long does the simulation take? At each level of the recursive procedure described above, two steps forward are replaced by two steps forward and one step back. Therefore, an irreversible computation with $T_{\rm irr} = 2^n$ steps is simulated in $T_{\rm rev} = 3^n$ steps, or

$$T_{\rm rev} = (T_{\rm irr})^{\log 3/\log 2}, = (T_{\rm irr})^{1.58}, \qquad (6.51)$$

a modest power law slowdown.

In fact, we can improve the slowdown to

$$T_{\rm rev} \sim (T_{\rm irr})^{1+\varepsilon}, \qquad (6.52)$$

for any $\varepsilon > 0$. Instead of replacing two steps forward with two forward and one back, we replace ℓ forward with ℓ forward and $\ell - 1$ back. A recursive procedure with n levels reaches site ℓ^n using a maximum of $n(\ell - 1) + 1$ pebbles. Now we have $T_{\rm irr} = \ell^n$ and $T_{\rm rev} = (2\ell - 1)^n$, so that

$$T_{\rm rev} = (T_{\rm irr})^{\log(2\ell-1)/\log \ell}; \qquad (6.53)$$

the power characterizing the slowdown is

$$\frac{\log(2\ell-1)}{\log \ell} = \frac{\log 2\ell + \log\left(1 - \frac{1}{2\ell}\right)}{\log \ell} \simeq 1 + \frac{\log 2}{\log \ell}, \qquad (6.54)$$

and the space requirement scales as

$$S \simeq n\ell \simeq \ell \frac{\log T}{\log \ell}. \qquad (6.55)$$

Thus, for any fixed $\varepsilon > 0$, we can attain S scaling like $\log T$, and a slowdown no worse than $(T_{\rm irr})^{1+\varepsilon}$. (This is not the optimal way to play the Pebble game if our objective is to get as far as we can with as few pebbles as possible. We use more pebbles to get to step T, but we get there faster.)

We have now seen that a reversible circuit can simulate a circuit composed of irreversible gates efficiently — without requiring unreasonable memory resources or causing an unreasonable slowdown. Why is this important? You might worry that, because reversible computation is "harder" than irreversible computation, the classification of complexity depends on whether we compute reversibly or irreversibly. But this is not the case, because a reversible computer can simulate an irreversible computer pretty easily.

6.2 Quantum Circuits

Now we are ready to formulate a mathematical model of a quantum computer. We will generalize the circuit model of classical computation to the quantum circuit model of quantum computation.

A classical computer processes bits. It is equipped with a finite set of gates that can be applied to sets of bits. A quantum computer processes qubits. We will assume that it too is equipped with a discrete set of fundamental components, called *quantum gates*. Each quantum gate is a unitary transformation that acts on a fixed number of qubits. In a quantum computation, a finite number n of qubits are initially set to the value $|00\ldots 0\rangle$. A circuit is executed that is constructed from a finite number of quantum gates acting on these qubits. Finally, a Von Neumann measurement of all the qubits (or a subset of the qubits) is performed, projecting each onto the basis $\{|0\rangle, |1\rangle\}$. The outcome of this measurement is the result of the computation.

Several features of this model require comment:

(1) It is implicit but important that the Hilbert space of the device has a preferred decomposition into a tensor product of low-dimensional spaces, in this case the two-dimensional spaces of the qubits. Of course, we could have considered a tensor product of, say, qutrits instead. But

anyway we assume there is a natural decomposition into subsystems that is respected by the quantum gates — which act on only a few subsystems at a time. Mathematically, this feature of the gates is crucial for establishing a clearly defined notion of quantum complexity. Physically, the fundamental reason for a natural decomposition into subsystems is *locality*; feasible quantum gates must act in a bounded spatial region, so the computer decomposes into subsystems that interact only with their neighbors.

(2) Since unitary transformations form a continuum, it may seem unnecessarily restrictive to postulate that the machine can execute only those quantum gates chosen from a discrete set. We nevertheless accept such a restriction, because we do not want to invent a new physical implementation each time we are faced with a new computation to perform.

(3) We might have allowed our quantum gates to be superoperators, and our final measurement to be a POVM. But since we can easily simulate a superoperator by performing a unitary transformation on an extended system, or a POVM by performing a Von Neumann measurement on an extended system, the model as formulated is of sufficient generality.

(4) We might allow the final measurement to be a collective measurement, or a projection into a different basis. But any such measurement can be implemented by performing a suitable unitary transformation followed by a projection onto the standard basis $\{|0\rangle, |1\rangle\}^n$. Of course, complicated collective measurements can be transformed into measurements in the standard basis only with some difficulty, but it is appropriate to take into account this difficulty when characterizing the complexity of an algorithm.

(5) We might have allowed measurements at intermediate stages of the computation, with the subsequent choice of quantum gates conditioned on the outcome of those measurements. But in fact the same result can always be achieved by a quantum circuit with all measurements postponed until the end. (While we can postpone the measurements in principle, it might be very useful in practice to perform measurements at intermediate stages of a quantum algorithm.)

A quantum gate, being a unitary transformation, is reversible. In fact, a classical reversible computer is a special case of a quantum computer. A

classical reversible gate

$$x^{(n)} \to y^{(n)} = f(x^{(n)}), \tag{6.56}$$

implementing a permutation of n-bit strings, can be regarded as a unitary transformation that acts on the "computational basis $\{|x_i\rangle\}$ according to

$$U : |x_i\rangle \to |y_i\rangle. \tag{6.57}$$

This action is unitary because the 2^n strings $|y_i\rangle$ are all mutually orthogonal. A quantum computation constructed from such classical gates takes $|0\ldots 0\rangle$ to one of the computational basis states, so that the final measurement is deterministic.

There are three main issues concerning our model that we would like to address. The first issue is *universality*. The most general unitary transformation that can be performed on n qubits is an element of $U(2^n)$. Our model would seem incomplete if there were transformations in $U(2^n)$ that we were unable to reach. In fact, we will see that there are many ways to choose a discrete set of *universal quantum gates*. Using a universal gate set we can construct circuits that compute a unitary transformation that comes as close as we please to any element in $U(2^n)$.

Thanks to universality, there is also a machine independent notion of *quantum complexity*. We may define a new complexity class BQP — the class of decision problems that can be solved, with high probability, by polynomial-size quantum circuits. Since one universal quantum computer can simulate another efficiently, the class does not depend on the details of our hardware (on the universal gate set that we have chosen).

Notice that a quantum computer can easily simulate a probabilistic classical computer: it can prepare $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and then project to $\{|0\rangle, |1\rangle\}$, generating a random bit. Therefore BQP certainly contains the class BPP. But as we discussed in Chapter 1, it seems to be quite reasonable to expect that BQP is actually larger than BPP, because a probabilistic classical computer cannot easily simulate a quantum computer. The fundamental difficulty is that the Hilbert space of n qubits is huge, of dimension 2^n, and hence the mathematical description of a typical vector in the space is exceedingly complex. Our second issue is to better characterize the resources needed to simulate a quantum computer on a classical computer. We will see that, despite the vastness of Hilbert space, a classical computer can simulate an n-qubit quantum computer even if limited to an amount of memory space that is polynomial in n. This means the BQP is contained in the complexity class $PSPACE$, the decision problems that can be solved with polynomial space, but may require exponential time. (We know that NP is also contained in $PSPACE$, since checking if $C(x^{(n)}, y^{(m)}) = 1$ for each $y^{(m)}$ can be accomplished with polynomial space.[5]

The third important issue we should address is *accuracy*. The class BQP is defined formally under the idealized assumption that quantum gates can be executed with perfect precision. Clearly, it is crucial to relax this assumption in any realistic implementation of quantum computation. A polynomial size quantum circuit family that solves a hard problem would not be of much interest if the quantum gates in the circuit were required to have exponential accuracy. In fact, we will show that this is not the case. An idealized T-gate quantum circuit can be simulated with acceptable accuracy by noisy gates, provided that the error probability per gate scales like $1/T$.

We see that quantum computers pose a serious challenge to the strong Church–Turing thesis, which contends that any physically reasonable model of computation can be simulated by probabilistic classical circuits with at worst a polynomial slowdown. But so far there is no firm proof that

$$BPP \neq BQP. \tag{6.58}$$

Nor is such a proof necessarily soon to be expected.[6] Indeed, a corollary would be

$$BPP \neq PSPACE, \tag{6.59}$$

which would settle one of the long-standing and pivotal open questions in complexity theory.

It might be less unrealistic to hope for a proof that $BPP \neq BQP$ follows from another standard conjecture of complexity theory such as $P \neq NP$. So far no such proof has been found. But while we are not yet able to prove that quantum computers have capabilities far beyond those of conventional computers, we nevertheless might uncover evidence suggesting that $BPP \neq BQP$. We will see that there are problems that seem to be hard (in classical computation) yet can be efficiently solved by quantum circuits.

[5]Actually there is another rung of the complexity hierarchy that may separate BQP and $PSPACE$; we can show that $BQP \subseteq P^{\#P} \subseteq PSPACE$, but we won't consider $P^{\#P}$ any further here.

[6]That is, we ought not to expect a "nonrelativized proof." A separation between BPP and BQP "relative to an oracle" will be established later in the chapter.

Thus it seems likely that the classification of complexity will be different depending on whether we use a classical computer or a quantum computer to solve a problem. If such a separation really holds, it is the quantum classification that should be regarded as the more fundamental, for it is better founded on the physical laws that govern the universe.

6.2.1 Accuracy

Let's discuss the issue of accuracy. We imagine that we wish to implement a computation in which the quantum gates $\boldsymbol{U}_1, \boldsymbol{U}_2, \ldots, \boldsymbol{U}_T$ are applied sequentially to the initial state $|\varphi_0\rangle$. The state prepared by our ideal quantum circuit is

$$|\varphi_T\rangle = \boldsymbol{U}_T \boldsymbol{U}_{T-1} \ldots \boldsymbol{U}_2 \boldsymbol{U}_1 |\varphi_0\rangle. \tag{6.60}$$

But in fact our gates do not have perfect accuracy. When we attempt to apply the unitary transformation $\boldsymbol{U}_t$, we instead apply some "nearby" unitary transformation $\tilde{\boldsymbol{U}}_t$. (Of course, this is not the most general type of error that we might contemplate – the unitary $\boldsymbol{U}_t$ might be replaced by a *superoperator*. Considerations similar to those below would apply in that case, but for now we confine our attention to "unitary errors.")

The errors cause the actual state of the computer to wander away from the ideal state. How far does it wander? Let $|\varphi_t\rangle$ denote the ideal state after t quantum gates are applied, so that

$$|\varphi_t\rangle = \boldsymbol{U}_t |\varphi_{t-1}\rangle. \tag{6.61}$$

But if we apply the actual transformation $\tilde{\boldsymbol{U}}_t$, then

$$\tilde{\boldsymbol{U}}_t |\varphi_{t-1}\rangle = |\varphi_t\rangle + |E_t\rangle, \tag{6.62}$$

where

$$|E_t\rangle = (\tilde{\boldsymbol{U}}_t - \boldsymbol{U}_t)|\varphi_{t-1}\rangle, \tag{6.63}$$

is an unnormalized vector. If $|\tilde{\varphi}_t\rangle$ denotes the actual state after t steps, then we have

$$\begin{aligned} |\tilde{\varphi}_1\rangle &= |\varphi_1\rangle + |E_1\rangle, \\ |\tilde{\varphi}_2\rangle &= \tilde{\boldsymbol{U}}_2 |\tilde{\varphi}_1\rangle = |\varphi_2\rangle + |E_2\rangle + \tilde{\boldsymbol{U}}_2 |E_1\rangle, \end{aligned} \tag{6.64}$$

and so forth; we ultimately obtain

$$|\tilde{\varphi}_T\rangle = |\varphi_T\rangle + |E_T\rangle + \tilde{\boldsymbol{U}}_T|E_{T-1}\rangle + \tilde{\boldsymbol{U}}_T\tilde{\boldsymbol{U}}_{T-1}|E_{T-2}\rangle + \ldots + \tilde{\boldsymbol{U}}_T\tilde{\boldsymbol{U}}_{T-1}\ldots\tilde{\boldsymbol{U}}_2|E_1\rangle. \tag{6.65}$$

Thus we have expressed the difference between $|\tilde{\varphi}_T\rangle$ and $|\varphi_T\rangle$ as a sum of T remainder terms. The worst case yielding the largest deviation of $|\tilde{\varphi}_T\rangle$ from $|\varphi_T\rangle$ occurs if all remainder terms line up in the same direction, so that the errors interfere constructively. Therefore, we conclude that

$$\| \, |\tilde{\varphi}_T\rangle - |\varphi_T\rangle \, \| \leq \| \, |E_T\rangle \, \| + \| \, |E_{T-1}\rangle \, \| + \ldots + \| \, |E_2\rangle \, \| + \| \, |E_1\rangle \, \|, \tag{6.66}$$

where we have used the property $\| \boldsymbol{U}|E_i\rangle \| = \| |E_i\rangle \|$ for any unitary $\boldsymbol{U}$.

Let $\| \boldsymbol{A} \|_{\text{sup}}$ denote the sup norm of the operator $\boldsymbol{A}$ — that is, the maximum modulus of an eigenvalue of $\boldsymbol{A}$. We then have

$$\| \, |E_t\rangle \, \| = \| \left(\tilde{\boldsymbol{U}}_t - \boldsymbol{U}_t\right) |\varphi_{t-1}\rangle \, \| \leq \| \, \tilde{\boldsymbol{U}}_t - \boldsymbol{U}_t \, \|_{\text{sup}} \tag{6.67}$$

(since $|\varphi_{t-1}\rangle$ is normalized). Now suppose that, for each value of t, the error in our quantum gate is bounded by

$$\| \, \tilde{\boldsymbol{U}}_t - \boldsymbol{U}_t \, \|_{\text{sup}} < \varepsilon. \tag{6.68}$$

Then after T quantum gates are applied, we have

$$\| \, |\tilde{\varphi}_T\rangle - |\varphi_T\rangle \, \| < T\varepsilon; \tag{6.69}$$

in this sense, the accumulated error in the state grows linearly with the length of the computation.

The distance bounded in eq. (6.68) can equivalently be expressed as $\| \boldsymbol{W}_t - \boldsymbol{1} \|_{\text{sup}}$, where $\boldsymbol{W}_t = \tilde{\boldsymbol{U}}_t\boldsymbol{U}_t^\dagger$. Since $\boldsymbol{W}_t$ is unitary, each of its eigenvalues is a phase $e^{i\theta}$, and the corresponding eigenvalue of $\boldsymbol{W}_t - \mathit{1}$ has modulus

$$|e^{i\theta} - 1| = (2 - 2\cos\theta)^{1/2}, \tag{6.70}$$

so that eq. (6.68) is the requirement that each eigenvalue satisfies

$$\cos\theta > 1 - \varepsilon^2/2, \tag{6.71}$$

(or $|\theta| \lesssim \varepsilon$, for ε small). The origin of eq. (6.69) is clear. In each time step, $|\tilde{\varphi}\rangle$ rotates relative to $|\varphi\rangle$ by (at worst) an angle of order ε, and the distance between the vectors increases by at most of order ε.

How much accuracy is good enough? In the final step of our computation, we perform an orthogonal measurement, and the probability of outcome a, in the ideal case, is

$$P(a) = |\langle a|\varphi_T\rangle|^2. \tag{6.72}$$

Because of the errors, the actual probability is

$$\tilde{P}(a) = |\langle a|\tilde{\varphi}_T\rangle|^2. \tag{6.73}$$

If the actual vector is close to the ideal vector, then the probability distributions are close, too. If we sum over an orthonormal basis $\{|a\rangle\}$, we have

$$\sum_a |\tilde{P}(a) - P(a)| \leq 2 \, \| \, |\tilde{\varphi}_T\rangle - |\varphi_T\rangle \, \|, \tag{6.74}$$

as you will show in a homework exercise. Therefore, if we keep $T\varepsilon$ fixed (and small) as T gets large, the error in the probability distribution also remains fixed. In particular, if we have designed a quantum algorithm that solves a decision problem correctly with probability greater $\frac{1}{2} + \delta$ (in the ideal case), then we can achieve success probability greater than $\frac{1}{2}$ with our noisy gates, if we can perform the gates with an accuracy $T\varepsilon < O(\delta)$. A quantum circuit family in the BQP class can really solve hard problems, as long as we can improve the accuracy of the gates linearly with the computation size T.

6.2.2 BQP $\subseteq$ PSPACE

Of course a classical computer can simulate any quantum circuit. But how much memory does the classical computer require? Naively, since the simulation of an n-qubit circuit involves manipulating matrices of size 2^n, it may seem that an amount of memory space exponential in n is needed. But we will now show that the simulation can be done to acceptable accuracy (albeit very slowly!) in polynomial space. This means that the quantum complexity class BQP is contained in the class PSPACE of problems that can be solved with polynomial space.

The object of the classical simulation is to compute the probability for each possible outcome a of the final measurement

$$\text{Prob}(a) = |\langle a|\boldsymbol{U}_T|0\rangle|^2, \tag{6.75}$$

where

$$\boldsymbol{U}_T = \boldsymbol{U}_T \boldsymbol{U}_{T-1} \ldots \boldsymbol{U}_2 \boldsymbol{U}_1, \tag{6.76}$$

is a product of T quantum gates. Each $\boldsymbol{U}_t$, acting on the n qubits, can be represented by a $2^n \times 2^n$ unitary matrix, characterized by the complex matrix elements

$$\langle y|\boldsymbol{U}_t|x\rangle, \tag{6.77}$$

where $x, y \in \{0, 1 \ldots, 2^n - 1\}$. Writing out the matrix multiplication explicitly, we have

$$\langle a|\boldsymbol{U}_T|0\rangle = \sum_{\{x_t\}} \langle a|\boldsymbol{U}_T|x_{T-1}\rangle\langle x_{T-1}|\boldsymbol{U}_{T-1}|x_{T-2}\rangle \ldots$$
$$\ldots \langle x_2|\boldsymbol{U}_2|x_1\rangle\langle x_1|\boldsymbol{U}_1|0\rangle. \tag{6.78}$$

Eq. (6.78) is a sort of "path integral" representation of the quantum computation – the probability amplitude for the final outcome a is expressed as a coherent sum of amplitudes for each of a vast number ($2^{n(T-1)}$) of possible computational paths that begin at 0 and terminate at a after T steps.

Our classical simulator is to add up the $2^{n(T-1)}$ complex numbers in eq. (6.78) to compute $\langle a|\boldsymbol{U}_T|0\rangle$. The first problem we face is that finite size classical circuits do integer arithmetic, while the matrix elements $\langle y|\boldsymbol{U}_t|x\rangle$ need not be rational numbers. The classical simulator must therefore settle for an approximate calculation to reasonable accuracy. Each term in the sum is a product of T complex factors, and there are $2^{n(T-1)}$ terms in the sum. The accumulated errors are sure to be small if we express the matrix elements to m bits of accuracy, with m large compared to $n(T-1)$. Therefore, we can replace each complex matrix element by pairs of signed integers, taking values in $\{0, 1, 2, \ldots, 2^{m-1}\}$. These integers give the binary expansion of the real and imaginary part of the matrix element, expressed to precision 2^{-m}.

Our simulator will need to compute each term in the sum eq. (6.78) and accumulate a total of all the terms. But each addition requires only a modest amount of scratch space, and furthermore, since only the accumulated subtotal need be stored for the next addition, not much space is needed to sum all the terms, even though there are exponentially many.

So it only remains to consider the evaluation of a typical term in the sum, a product of T matrix elements. We will require a classical circuit that

evaluates

$$\langle y|\boldsymbol{U}_t|x\rangle; \tag{6.79}$$

this circuit accepts the $2n$ bit input (x, y), and outputs the $2m$-bit value of the (complex) matrix element. Given a circuit that performs this function, it will be easy to build a circuit that multiplies the complex numbers together without using much space.

Finally, at this point, we appeal to the properties we have demanded of our quantum gate set — the gates from a discrete set, and each gate acts on a bounded number of qubits. Because there are a fixed (and finite) number of gates, there are only a fixed number of gate subroutines that our simulator needs to be able to call. And because the gate acts on only a few qubits, nearly all of its matrix elements vanish (when n is large), and the value $\langle y|\boldsymbol{U}|x\rangle$ can be determined (to the required accuracy) by a simple circuit requiring little memory.

For example, in the case of a single-qubit gate acting on the first qubit, we have

$$\langle y_1 y_2 \ldots y_n|\boldsymbol{U}|x_1 x_2 \ldots x_n\rangle = 0 \text{ if } x_2 x_3 \ldots x_n \neq y_2 y_3 \ldots y_n. \tag{6.80}$$

A simple circuit can compare x_2 with y_2, x_3 with y_3, *etc.*, and output zero if the equality is not satisfied. In the event of equality, the circuit outputs one of the four complex numbers

$$\langle y_1|\boldsymbol{U}|x_1\rangle, \tag{6.81}$$

to m bits of precision. A simple circuit can encode the $8m$ bits of this 2×2 complex-valued matrix. Similarly, a simple circuit, requiring only space polynomial in n and m, can evaluate the matrix elements of any gate of fixed size.

We conclude that a classical computer with space bounded above by poly(n) can simulate an n-qubit universal quantum computer, and therefore that BQP $\subseteq$ PSPACE. Of course, it is also evident that the simulation we have described requires exponential time, because we need to evaluate the sum of $2^{n(T-1)}$ complex numbers. (Actually, most of the terms vanish, but there are still an exponentially large number of nonvanishing terms.)

6.2.3 Universal quantum gates

We must address one more fundamental question about quantum computation; how do we construct an adequate set of quantum gates? In other words, what constitutes a universal quantum computer?

We will find a pleasing answer. Any generic two-qubit gate suffices for universal quantum computation. That is, for all but a set of measure zero of 4×4 unitary matrices, if we can apply that matrix to any pair of qubits, then we can construct an n-qubit circuit that computes a transformation that comes as close as we please to any element of $U(2^n)$.

Mathematically, this is not a particularly deep result, but physically it is very interesting. It means that, in the quantum world, as long as we can devise a generic interaction between two qubits, and we can implement that interaction accurately between any two qubits, we can compute anything, no matter how complex. Nontrivial computation is ubiquitous in quantum theory.

Aside from this general result, it is also of some interest to exhibit particular universal gate sets that might be particularly easy to implement physically. We will discuss a few examples.

There are a few basic elements that enter the analysis of any universal quantum gate set.

(1) Powers of a generic gate

Consider a "generic" k-qubit gate. This is a $2^k \times 2^k$ unitary matrix $\boldsymbol{U}$ with eigenvalues $e^{i\theta_1}, e^{i\theta_2}, \ldots e^{i\theta_{2^k}}$. For all but a set of measure zero of such matrices, each θ_i is an irrational multiple of π, and all the θ_i's are incommensurate (each θ_i/θ_j is also irrational). The positive integer power $\boldsymbol{U}^n$ of $\boldsymbol{U}$ has eigenvalues

$$e^{in\theta_1}, e^{in\theta_2}, \ldots, e^{in\theta_{2^k}}. \tag{6.82}$$

Each such list of eigenvalues defines a point in a 2^k-dimensional torus (the product of 2^k circles). As n ranges over positive integer values, these points densely fill the whole torus, if $\boldsymbol{U}$ is generic. If $\boldsymbol{U} = e^{iA}$, positive integer powers of $\boldsymbol{U}$ come as close as we please to $\boldsymbol{U}(\lambda) = e^{i\lambda A}$, for any real λ. We say that any $\boldsymbol{U}(\lambda)$ is *reachable* by positive integer powers of $\boldsymbol{U}$.

(2) Switching the leads

There are a few (classical) transformations that we can implement just by switching the labels on k qubits, or in other words, by applying the gate $\boldsymbol{U}$ to the qubits in a different order. Of the $(2^k)!$ permutations of the length-k strings, $k!$ can be realized by swapping qubits. If a gate applied to k qubits with a standard ordering is $\boldsymbol{U}$, and P is a permutation implemented by swapping qubits, then we can construct the gate

$$\boldsymbol{U}' = P\boldsymbol{U}P^{-1}, \tag{6.83}$$

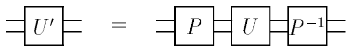
just by switching the leads on the gate. For example, swapping two qubits implements the transposition

$$P : |01\rangle \leftrightarrow |10\rangle, \tag{6.84}$$

or

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{6.85}$$

acting on basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. By switching leads, we obtain a gate



We can also construct any positive integer power of $\boldsymbol{U}'$, $(P\boldsymbol{U}P^{-1})^n = P\boldsymbol{U}^nP^{-1}$.

(3) Completing the Lie algebra

We already remarked that if $\boldsymbol{U} = e^{iA}$ is generic, then powers of $\boldsymbol{U}$ are dense in the torus $\{e^{i\lambda A}\}$. We can further argue that if $\boldsymbol{U} = e^{iA}$ and $\boldsymbol{U}' = e^{iB}$ are generic gates, we can compose them to come arbitrarily close to

$$e^{i(\alpha A + \beta B)} \text{ or } e^{-\gamma[A,B]}, \tag{6.86}$$

for any real α, β, γ. Thus, the "reachable" transformations have a closed *Lie algebra*. We say that $\boldsymbol{U} = e^{iA}$ is generated by A; then if A and B are both generic generators of reachable transformations, so are real linear combinations of A and B, and (i times) the commutator of A and B.

We first note that

$$\lim_{n\to\infty} (e^{i\alpha A/n} e^{i\beta B/n})^n = \lim_{n\to\infty} \left(1 + \frac{i}{n}(\alpha A + \beta B)\right)^n$$
$$= e^{i(\alpha A + \beta B)}. \tag{6.87}$$

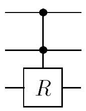
Therefore, any $e^{i(\alpha A+\beta B)}$ is reachable if each $e^{i\alpha A/n}$ and $e^{i\beta B/n}$ is. Furthermore

$$\lim_{n\to\infty} \left(e^{iA/\sqrt{n}} e^{iB/\sqrt{n}} e^{-iA/\sqrt{n}} e^{-iB/\sqrt{n}}\right)^n$$
$$= \lim_{n\to\infty} \left[1 - \frac{1}{n}(AB - BA)\right]^n = e^{-[A,B]}, \tag{6.88}$$

so $e^{-[A,B]}$ is also reachable.

By invoking the observations (1), (2), and (3) above, we will be able to show that a generic two-qubit gate is universal.

Deutsch gate. It was David Deutsch (1989) who first pointed out the existence of a universal quantum gate. Deutsch's three-bit universal gate is a quantum cousin of the Toffoli gate. It is the controlled-controlled-$\boldsymbol{R}$ transformation



that applies $\boldsymbol{R}$ to the third qubit if the first two qubits have the value 1; otherwise it acts trivially. Here

$$\boldsymbol{R} = -i\boldsymbol{R}_x(\theta) = (-i)\exp\left(i\frac{\theta}{2}\boldsymbol{\sigma}_x\right) = (-i)\left(\cos\frac{\theta}{2} + i\boldsymbol{\sigma}_x \sin\frac{\theta}{2}\right) \tag{6.89}$$

is, up to a phase, a rotation by θ about the x-axis, where θ is a particular angle incommensurate with π.

The nth power of the Deutsch gate is the controlled-controlled-$\boldsymbol{R}^n$. In particular, $\boldsymbol{R}^4 = \boldsymbol{R}_x(4\theta)$, so that all one-qubit transformations generated by $\boldsymbol{\sigma}_x$ are reachable by integer powers of $\boldsymbol{R}$. Furthermore the $(4n+1)$st power is

$$(-i)\left[\cos\frac{(4n+1)\theta}{2} + i\boldsymbol{\sigma}_x \sin\frac{(4n+1)\theta}{2}\right], \tag{6.90}$$

which comes as close as we please to $\boldsymbol{\sigma}_x$. Therefore, the Toffoli gate is reachable with integer powers of the Deutsch gate, and the Deutsch gate is universal for classical computation.

Acting on the three-qubit computational basis

$$\{|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle\}, \tag{6.91}$$

the generator of the Deutsch gate transposes the last two elements

$$|110\rangle \leftrightarrow |111\rangle. \tag{6.92}$$

We denote this 8×8 matrix as

$$(\boldsymbol{\sigma}_x)_{67} = \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & \boldsymbol{\sigma}_x \end{array}\right). \tag{6.93}$$

With Toffoli gates, we can perform any permutation of these eight elements, in particular

$$P = (6m)(7n), \tag{6.94}$$

for any m and n. So we can also reach any transformation generated by

$$P(\boldsymbol{\sigma}_x)_{67}P = (\boldsymbol{\sigma}_x)_{mn}. \tag{6.95}$$

Furthermore,

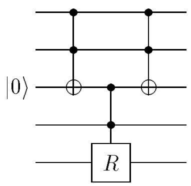
$$[(\boldsymbol{\sigma}_x)_{56}, (\boldsymbol{\sigma}_x)_{67}] = \left[\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}\right] = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix} = i(\boldsymbol{\sigma}_y)_{57}, \tag{6.96}$$

and similarly, we can reach any unitary generated by $(\boldsymbol{\sigma}_y)_{mn}$. Finally

$$[(\boldsymbol{\sigma}_x)_{mn}, (\boldsymbol{\sigma}_y)_{mn}] = i(\boldsymbol{\sigma}_z)_{mn}, \tag{6.97}$$

So we can reach any transformation generated by a linear combination of the $(\boldsymbol{\sigma}_{x,y,z})_{mn}$'s. These span the whole $SU(8)$ Lie Algebra, so we can generate any three-qubit unitary (aside from an irrelevant overall phase).

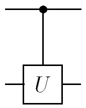
Now recall that we have already found that we can construct the n-bit Toffoli gate by composing three-bit Toffoli gates. The circuit



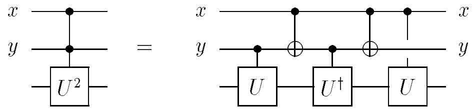
uses one scratch bit to construct a four-bit Deutsch gate ((controlled)^3-$\boldsymbol{R}$) from the three-bit Deutsch gate and two three-bit Toffoli gates, and a similar circuit constructs the n-bit Deutsch gate from a three-bit Deutsch gate and two $(n-1)$-bit Toffoli gates. Once we have an n-bit Deutsch gate, and universal classical computation, exactly the same argument as above shows that we can reach any transformation in $SU(2^n)$.

Universal two-qubit gates. For reversible classical computation, we saw that three-bit gates are needed for universality. But in quantum computation, two-bit gates turn out to be adequate. Since we already know that the Deutsch gate is universal, we can establish this by showing that the Deutsch gate can be constructed by composing two-qubit gates.

In fact, if



denotes the controlled-$\boldsymbol{U}$ gate (the 2×2 unitary $\boldsymbol{U}$ is applied to the second qubit if the first qubit is 1; otherwise the gate acts trivially) then a controlled-controlled-$\boldsymbol{U}^2$ gate is obtained from the circuit



the power of $\boldsymbol{U}$ applied to the third qubit is

$$y - (x \oplus y) + x = x + y - (x + y - 2xy) = 2xy. \tag{6.98}$$

Therefore, we can construct Deutsch's gate from the controlled-$\boldsymbol{U}$, controlled $\boldsymbol{U}^{-1}$ and controlled-NOT gates, where

$$\boldsymbol{U}^2 = -i\boldsymbol{R}_x(\theta); \tag{6.99}$$

we may choose

$$\boldsymbol{U} = e^{-i\frac{\pi}{4}}\boldsymbol{R}_x\left(\frac{\theta}{2}\right). \tag{6.100}$$

Positive powers of $\boldsymbol{U}$ came as close as we please to $\boldsymbol{\sigma}_x$ and $\boldsymbol{U}^{-1}$, so from the controlled-$\boldsymbol{U}$ alone we can construct the Deutsch gate. Therefore, the controlled-$\left(e^{-i\frac{\pi}{4}}\boldsymbol{R}_x\left(\frac{\theta}{2}\right)\right)$ is itself a universal gate, for θ/π irrational.

(Note that the above construction shows that, while we cannot construct the Toffoli gate from two-bit reversible classical gates, we *can* construct it from a controlled "square root of NOT" — a controlled-$\boldsymbol{U}$ with $\boldsymbol{U}^2 = \boldsymbol{\sigma}_x$.)

Generic two-bit gates. Now we have found particular two-bit gates (controlled rotations) that are universal gates. Therefore, for universality, it is surely sufficient if we can construct transformations that are dense in the $U(4)$ acting on a pair of qubits.

In fact, though, any generic two-qubit gate is sufficient to generate all of $U(4)$. As we have seen, if $e^{i\boldsymbol{A}}$ is a generic element of $U(4)$, we can reach any transformation generated by $\boldsymbol{A}$. Furthermore, we can reach any transformations generated by an element of the minimal Lie algebra containing $\boldsymbol{A}$ and

$$\boldsymbol{B} = P\boldsymbol{A}P^{-1} \tag{6.101}$$

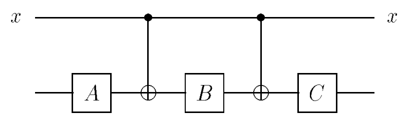
where P is the permutation ($|01\rangle \leftrightarrow |10\rangle$) obtained by switching the leads.

Now consider a general $\boldsymbol{A}$, (expanded in terms of a basis for the Lie algebra of $U(4)$), and consider a particular scheme for constructing 16 elements of the algebra by successive commutations, starting from $\boldsymbol{A}$ and $\boldsymbol{B}$. The elements so constructed are linearly independent (and it follows that any transformation in $U(4)$ is reachable) if the determinant of a particular 16×16 matrix vanishes. Unless this vanishes identically, its zeros occur only on a submanifold of vanishing measure. But in fact, we can choose, say

$$\boldsymbol{A} = (\alpha I + \beta\sigma_x + \gamma\sigma_y)_{23}, \tag{6.102}$$

(for incommensurate α, β, γ), and show by explicit computation that the entire 16-dimension Lie Algebra is actually generated by successive commutations, starting with $\boldsymbol{A}$ and $\boldsymbol{B}$. Hence we conclude that failure to generate the entire $U(4)$ algebra is nongeneric, and find that almost all two-qubit gates are universal.

Other adequate sets of gates. One can also see that universal quantum computation can be realized with a gate set consisting of *classical* multiqubit gates and quantum single-qubit gates. For example, we can see that the XOR gate, combined with one-qubit gates, form a universal set. Consider the circuit



which applies $\boldsymbol{ABC}$ to the second qubit if $x = 0$, and $\boldsymbol{A\sigma_x B\sigma_x C}$ to the second qubit if $x = 1$. If we can find $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ such that

$$\begin{aligned} \boldsymbol{ABC} &= 1 \\ \boldsymbol{A\sigma_x B\sigma_x C} &= \boldsymbol{U}, \end{aligned} \tag{6.103}$$

then this circuit functions as a controlled-$\boldsymbol{U}$ gate. In fact unitary 2×2 $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ with this property exist for any unitary $\boldsymbol{U}$ with determinant one (as you'll show in an exercise). Therefore, the XOR plus arbitrary one-qubit transformations form a universal set. Of course, two generic (noncommuting) one-qubit transformations are sufficient to reach all. In fact, with an XOR

and a *single* generic one-qubit rotation, we can construct a second one-qubit rotation that does not commute with the first. Hence, an XOR together with just one generic single-qubit gate constitutes a universal gate set.

If we are able to perform a Toffoli gate, then even certain nongeneric one-qubit transformations suffice for universal computation. For example (another exercise) the Toffoli gate, together with $\pi/2$ rotations about the x and z axes, are a universal set.

Precision. Our discussion of universality has focused on *reachability* without any regard for *complexity*. We have only established that we can construct a quantum circuit that comes as close as we please to a desired element of $U(2^n)$, and we have not considered the size of the circuit that we need. But from the perspective of quantum complexity theory, universality is quite significant because it implies that one quantum computer can simulate another to reasonable accuracy without an unreasonable slowdown.

Actually, we have not been very precise up until now about what it means for one unitary transformation to be "close" to another; we should define a topology. One possibility is to use the sup norm as in our previous discussion of accuracy — the distance between matrices $\boldsymbol{U}$ and $\boldsymbol{W}$ is then $\| \boldsymbol{U} - \boldsymbol{W} \|_{\text{sup}}$. Another natural topology is associated with the inner product

$$\langle \boldsymbol{W} | \boldsymbol{U} \rangle \equiv \text{tr } \boldsymbol{W}^\dagger \boldsymbol{U} \tag{6.104}$$

(if $\boldsymbol{U}$ and $\boldsymbol{W}$ are $N \times N$ matrices, this is just the usual inner product on C^{N^2}, where we regard $\boldsymbol{U}_{ij}$ as a vector with N^2 components). Then we may define the distance squared between matrices as

$$\| \boldsymbol{U} - \boldsymbol{W} \|^2 \equiv \langle \boldsymbol{U} - \boldsymbol{W} | \boldsymbol{U} - \boldsymbol{W} \rangle. \tag{6.105}$$

For the purpose of analyzing complexity, just about any reasonable topology will do.

The crucial point is that given any universal gate set, we can reach within distance ε of any desired unitary transformation that acts on a fixed number of qubits, using a quantum circuit whose size is bounded above by a polynomial in ε^{-1}. Therefore, one universal quantum computer can simulate another, to accuracy ε, with a slowdown no worse than a factor that is polynomial in ε^{-1}. Now we have already seen that to have a high probability of getting the right answer when we perform a quantum circuit of size T, we should implement each quantum gate to an accuracy that scales like T^{-1}. Therefore, if you have a quantum circuit family of polynomial size that runs on your quantum computer, I can devise a polynomial size circuit family that runs on my machine, and that emulates your machine to acceptable accuracy.

Why can a poly(ε^{-1})-size circuit reach a given k-qubit $\boldsymbol{U}$ to within distance ε? We know for example that the positive integer powers of a generic k-qubit $e^{i\boldsymbol{A}}$ are dense in the 2^k-torus $\{e^{i\lambda \boldsymbol{A}}\}$. The region of the torus within distance ε of any given point has volume of order ε^{2^k}, so (asymptotically for ε sufficiently small) we can reach any $\{e^{i\lambda \boldsymbol{A}}\}$ to within distance ε with $\left(e^{i\lambda \boldsymbol{A}}\right)^n$, for some integer n of order ε^{-2^k}. We also know that we can obtain transformations $\{e^{i\boldsymbol{A}_a}\}$ where the $\boldsymbol{A}_a$'s span the full $U(2^k)$ Lie algebra, using circuits of fixed size (independent of ε). We may then approach any $\exp\left(i\sum_a \alpha_a \boldsymbol{A}_a\right)$ as in eq. (6.87), also with polynomial convergence.

In principle, we should be able to do much better, reaching a desired k-qubit unitary within distance ε using just poly$(\log(\varepsilon^{-1}))$ quantum gates. Since the number of size-T circuits that we can construct acting on k qubits is exponential in T, and the circuits fill $U(2^k)$ roughly uniformly, there should be a size-T circuit reaching within a distance of order e^{-T} of any point in $U(2^k)$. However, it might be a computationally hard problem *classically* to work out the circuit that comes exponentially close to the unitary we are trying to reach. Therefore, it would be dishonest to rely on this more efficient construction in an asymptotic analysis of quantum complexity.

6.3 Some Quantum Algorithms

While we are not yet able to show that $BPP \neq BQP$, there are three approaches that we can pursue to study the differences between the capabilities of classical and quantum computers:

(1) Nonexponential speedup. We can find quantum algorithms that are demonstrably faster than the best classical algorithm, but not *exponentially* faster. These algorithms shed no light on the conventional classification of complexity. But they do demonstrate a type of separation between tasks that classical and quantum computers can perform. Example: Grover's quantum speedup of the search of an unsorted data base.

(2) "Relativized" exponential speedup. We can consider the problem of analyzing the contents of a "quantum black box." The box performs an

a priori unknown) unitary transformation. We can prepare an input for the box, and we can measure its output; our task is to find out what the box does. It is possible to prove that quantum black boxes (computer scientists call them oracles[7]) exist with this property: By feeding quantum superpositions to the box, we can learn what is inside with an *exponential* speedup, compared to how long it would take if we were only allowed classical inputs. A computer scientist would say that $BPP \neq BQP$ "relative to the oracle." Example: Simon's exponential quantum speedup for finding the period of a 2 to 1 function.

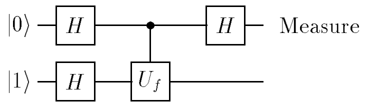
(3) Exponential speedup for "apparently" hard problems. We can exhibit a quantum algorithm that solves a problem in polynomial time, where the problem appears to be hard classically, so that it is strongly suspected (though not proved) that the problem is not in BPP. Example: Shor's factoring algorithm.

Deutsch's problem. We will discuss examples from all three approaches. But first, we'll warm up by recalling an example of a simple quantum algorithm that was previously discussed in §1.5: Deutsch's algorithm for distinguishing between constant and balanced functions $f : \{0,1\} \rightarrow \{0,1\}$. We are presented with a quantum black box that computes $f(x)$; that is, it enacts the two-qubit unitary transformation

$$U_f : |x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle, \tag{6.106}$$

which flips the second qubit iff f(first qubit) $= 1$. Our assignment is to determine whether $f(0) = f(1)$. If we are restricted to the "classical" inputs $|0\rangle$ and $|1\rangle$, we need to access the box twice ($x = 0$ and $x = 1$) to get the answer. But if we are allowed to input a coherent superposition of these "classical" states, then once is enough.

The quantum circuit that solves the problem (discussed in §1.5) is:



[7]The term "oracle" signifies that the box responds to a query *immediately*; that is, the time it takes the box to operate is not included in the complexity analysis.

Here H denotes the Hadamard transform

$$\boldsymbol{H} : |x\rangle \to \frac{1}{\sqrt{2}} \sum_y (-1)^{xy} |y\rangle, \tag{6.107}$$

or

$$\begin{aligned} \boldsymbol{H} : |0\rangle &\to \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |1\rangle &\to \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle); \end{aligned} \tag{6.108}$$

that is, $\boldsymbol{H}$ is the 2×2 matrix

$$\boldsymbol{H} : \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}. \tag{6.109}$$

The circuit takes the input $|0\rangle|1\rangle$ to

$$\begin{aligned} |0\rangle|1\rangle \to &\frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) \\ \to &\frac{1}{2}\left((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\right)(|0\rangle - |1\rangle) \\ \to &\frac{1}{2}\Bigg[\left((-1)^{f(0)} + (-1)^{f(1)}\right)|0\rangle \\ &+ \left((-1)^{f(0)} - (-1)^{f(1)}\right)|1\rangle\Bigg] \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \end{aligned} \tag{6.110}$$

Then when we measure the first qubit, we find the outcome $|0\rangle$ with probability one if $f(0) = f(1)$ (constant function) and the outcome $|1\rangle$ with probability one if $f(0) \neq f(1)$ (balanced function).

A quantum computer enjoys an advantage over a classical computer because it can invoke *quantum parallelism*. Because we input a superposition of $|0\rangle$ and $|1\rangle$, the output is sensitive to both the values of $f(0)$ and $f(1)$, even though we ran the box just once.

Deutsch–Jozsa problem. Now we'll consider some generalizations of Deutsch's problem. We will continue to assume that we are to analyze a quantum black box ("quantum oracle"). But in the hope of learning something about complexity, we will imagine that we have a family of black boxes,

with variable input size. We are interested in how the time needed to find out what is inside the box scales with the size of the input (where "time" is measured by how many times we query the box).

In the *Deutsch–Jozsa problem*, we are presented with a quantum black box that computes a function taking n bits to 1,

$$f : \{0,1\}^n \to \{0,1\}, \tag{6.111}$$

and we have it on good authority that f is either constant ($f(x) = c$ for all x) or balanced ($f(x) = 0$ for exactly $\frac{1}{2}$ of the possible input values). We are to solve the decision problem: Is f constant or balanced?

In fact, we can solve this problem, too, accessing the box only once, using the same circuit as for Deutsch's problem (but with x expanded from one bit to n bits). We note that if we apply n Hadamard gates in parallel to n-qubits.

$$\boldsymbol{H}^{(n)} = \boldsymbol{H} \otimes \boldsymbol{H} \otimes \ldots \otimes \boldsymbol{H}, \tag{6.112}$$

then the n-qubit state transforms as

$$\boldsymbol{H}^{(n)} : |x\rangle \to \prod_{i=1}^{n} \left(\frac{1}{\sqrt{2}} \sum_{y_i=\{0,1\}} (-1)^{x_i y_i} |y_i\rangle \right) \equiv \frac{1}{2^{n/2}} \sum_{y=0}^{2^n-1} (-1)^{x \cdot y} |y\rangle, \tag{6.113}$$

where x, y represent n-bit strings, and $x \cdot y$ denotes the *bitwise* AND (or mod 2 scalar product)

$$x \cdot y = (x_1 \wedge y_1) \oplus (x_2 \wedge y_2) \oplus \ldots \oplus (x_n \wedge y_n). \tag{6.114}$$

Acting on the input $(|0\rangle)^n|1\rangle$, the action of the circuit is

$$\begin{aligned} (|0\rangle)^n|1\rangle &\to \left(\frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle\right) \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\ &\to \left(\frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} (-1)^{f(x)}|x\rangle\right) \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\ &\to \left(\frac{1}{2^n} \sum_{x=0}^{2^n-1} \sum_{y=0}^{2^n-1} (-1)^{f(x)}(-1)^{x \cdot y}|y\rangle\right) \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned} \tag{6.115}$$

Now let us evaluate the sum

$$\frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)}(-1)^{x \cdot y}. \tag{6.116}$$

If f is a constant function, the sum is

$$(-1)^{f(x)}\left(\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{x\cdot y}\right)=(-1)^{f(x)}\delta_{y,0}; \tag{6.117}$$

it vanishes unless $y=0$. Hence, when we measure the n-bit register, we obtain the result $|y=0\rangle \equiv (|0\rangle)^n$ with probability one. But if the function is balanced, then for $y=0$, the sum becomes

$$\frac{1}{2^n}\sum_{x=0}^{2^{n-1}}(-1)^{f(x)}=0, \tag{6.118}$$

(because half of the terms are $(+1)$ and half are (-1)). Therefore, the probability of obtaining the measurement outcome $|y=0\rangle$ is zero.

We conclude that one query of the quantum oracle suffices to distinguish constant and balanced function with 100% confidence. The measurement result $y=0$ means constant, any other result means balanced.

So quantum computation solves this problem neatly, but is the problem really hard classically? If we are restricted to classical input states $|x\rangle$, we can query the oracle repeatedly, choosing the input x at random (without replacement) each time. Once we obtain distinct outputs for two different queries, we have determined that the function is balanced (not constant). But if the function is in fact constant, we will not be *certain* it is constant until we have submitted $2^{n-1}+1$ queries and have obtained the same response every time. In contrast, the quantum computation gives a definite response in only one go. So in this sense (if we demand absolute certainty) the classical calculation requires a number of queries exponential in n, while the quantum computation does not, and we might therefore claim an exponential quantum speedup.

But perhaps it is not reasonable to demand absolute certainty of the classical computation (particularly since any real quantum computer will be susceptible to errors, so that the quantum computer will also be unable to attain absolute certainty.) Suppose we are satisfied to guess balanced or constant, with a probability of success

$$P(\text{success})>1-\varepsilon. \tag{6.119}$$

If the function is actually balanced, then if we make k queries, the probability of getting the same response every time is $p=2^{-(k-1)}$. If after receiving the

same response k consecutive times we guess that the function is balanced, then a quick Bayesian analysis shows that the probability that our guess is wrong is $\frac{1}{2^{k-1}+1}$ (assuming that balanced and constant are a priori equally probable). So if we guess after k queries, the probability of a wrong guess is

$$1-P(\text{success})=\frac{1}{2^{k-1}(2^{k-1}+1)}. \tag{6.120}$$

Therefore, we can achieve success probability $1-\varepsilon$ for $\varepsilon^{-1}=2^{k-1}(2^{k-1}+1)$ or $k\sim\frac{1}{2}\log\left(\frac{1}{\varepsilon}\right)$. Since we can reach an exponentially good success probability with a polynomial number of trials, it is not really fair to say that the problem is hard.

Bernstein–Vazirani problem. Exactly the same circuit can be used to solve another variation on the Deutsch–Jozsa problem. Let's suppose that our quantum black box computes one of the functions f_a, where

$$f_a(x)=a\cdot x, \tag{6.121}$$

and a is an n-bit string. Our job is to determine a.

The quantum algorithm can solve this problem with certainty, given just one (n-qubit) quantum query. For this particular function, the quantum state in eq. (6.115) becomes

$$\frac{1}{2^n}\sum_{x=0}^{2^n-1}\sum_{y=0}^{2^n-1}(-1)^{a\cdot x}(-1)^{x\cdot y}|y\rangle. \tag{6.122}$$

But in fact

$$\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{a\cdot x}(-1)^{x\cdot y}=\delta_{a,y}, \tag{6.123}$$

so this state is $|a\rangle$. We can execute the circuit once and measure the n-qubit register, finding the n-bit string a with probability one.

If only classical queries are allowed, we acquire only one bit of information from each query, and it takes n queries to determine the value of a. Therefore, we have a clear separation between the quantum and classical difficulty of the problem. Even so, this example does not probe the relation of BPP to BQP, because the classical problem is not hard. The number of queries required classically is only linear in the input size, not exponential.

Simon's problem. Bernstein and Vazirani managed to formulate a variation on the above problem that *is* hard classically, and so establish for the first time a "relativized" separation between quantum and classical complexity. We will find it more instructive to consider a simpler example proposed somewhat later by Daniel Simon.

Once again we are presented with a quantum black box, and this time we are assured that the box computes a function

$$f : \{0,1\}^n \to \{0,1\}^n, \tag{6.124}$$

that is 2-to-1. Furthermore, the function has a "period" given by the n-bit string a; that is

$$f(x) = f(y) \qquad \text{iff} \qquad y = x \oplus a, \tag{6.125}$$

where here $\oplus$ denotes the bitwise XOR operation. (So a is the period if we regard x as taking values in $(Z_2)^n$ rather than Z_{2^n}.) This is all we know about f. Our job is to determine the value of a.

Classically this problem is *hard*. We need to query the oracle an exponentially large number of times to have any reasonable probability of finding a. We don't learn anything until we are fortunate enough to choose two queries x and y that happen to satisfy $x \oplus y = a$. Suppose, for example, that we choose $2^{n/4}$ queries. The number of pairs of queries is less than $(2^{n/4})^2$, and for each pair $\{x, y\}$, the probability that $x \oplus y = a$ is 2^{-n}. Therefore, the probability of successfully finding a is less than

$$2^{-n}(2^{n/4})^2 = 2^{-n/2}; \tag{6.126}$$

even with exponentially many queries, the success probability is exponentially small.

If we wish, we can frame the question as a decision problem: Either f is a 1-1 function, or it is 2-to-1 with some randomly chosen period a, each occurring with an a priori probability $\frac{1}{2}$. We are to determine whether the function is 1-to-1 or 2-to-1. Then, after $2^{n/4}$ classical queries, our probability of making a correct guess is

$$P(\text{success}) < \frac{1}{2} + \frac{1}{2^{n/2}}, \tag{6.127}$$

which does not remain bounded away from $\frac{1}{2}$ as n gets large.

But with quantum queries the problem is easy! The circuit we use is essentially the same as above, but now *both* registers are expanded to n qubits. We prepare the equally weighted superposition of all n-bit strings (by acting on $|0\rangle$ with $\boldsymbol{H}^{(n)}$), and then we query the oracle:

$$U_f : \left(\sum_{x=0}^{2^n-1} |x\rangle\right) |0\rangle \to \sum_{x=0}^{2^n-1} |x\rangle |f(x)\rangle. \tag{6.128}$$

Now we measure the second register. (This step is not actually necessary, but I include it here for the sake of pedagogical clarity.) The measurement outcome is selected at random from the 2^{n-1} possible values of $f(x)$, each occurring equiprobably. Suppose the outcome is $f(x_0)$. Then because both x_0 and $x_0 \oplus a$, and only these values, are mapped by f to $f(x_0)$, we have prepared the state

$$\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus a\rangle) \tag{6.129}$$

in the first register.

Now we want to extract some information about a. Clearly it would do us no good to measure the register (in the computational basis) at this point. We would obtain either the outcome x_0 or $x_0 \oplus a$, each occurring with probability $\frac{1}{2}$, but neither outcome would reveal anything about the value of a.

But suppose we apply the Hadamard transform $\boldsymbol{H}^{(n)}$ to the register before we measure:

$$\begin{aligned} \boldsymbol{H}^{(n)} :& \frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 + a\rangle) \\ &\to \frac{1}{2^{(n+1)/2}} \sum_{y=0}^{2^n-1} \left[(-1)^{x_0 \cdot y} + (-1)^{(x_0 \oplus a)\cdot y}\right] |y\rangle \\ &= \frac{1}{2^{(n-1)/2}} \sum_{a \cdot y = 0} (-1)^{x_0 \cdot y} |y\rangle. \end{aligned} \tag{6.130}$$

If $a \cdot y = 1$, then the terms in the coefficient of $|y\rangle$ interfere destructively. Hence only states $|y\rangle$ with $a \cdot y = 0$ survive in the sum over y. The measurement outcome, then, is selected at random from all possible values of y such that $a \cdot y = 0$, each occurring with probability $2^{-(n-1)}$.

We run this algorithm repeatedly, each time obtaining another value of y satisfying $y \cdot a = 0$. Once we have found n such linearly independent values $\{y_1, y_2, y_3 \dots y_n\}$ (that is, linearly independent over $(Z_2)^n$), we can solve the equations

$$\begin{aligned} y_1 \cdot a &= 0 \\ y_2 \cdot a &= 0 \\ &\vdots \\ y_n \cdot a &= 0, \end{aligned} \tag{6.131}$$

to determine a unique value of a, and our problem is solved. It is easy to see that with $O(n)$ repetitions, we can attain a success probability that is exponentially close to 1.

So we finally have found an example where, given a particular type of quantum oracle, we can solve a problem in polynomial time by exploiting quantum superpositions, while exponential time is required if we are limited to classical queries. As a computer scientist might put it:

> There exists an oracle relative to which $BQP \neq BPP$.

Note that whenever we compare classical and quantum complexity relative to an oracle, we are considering a quantum oracle (queries and replies are states in Hilbert space), but with a preferred orthonormal basis. If we submit a classical query (an element of the preferred basis) we always receive a classical response (another basis element). The issue is whether we can achieve a significant speedup by choosing more general quantum queries.

6.4 Quantum Database Search

The next algorithm we will study also exhibits, like Simon's algorithm, a speedup with respect to what can be achieved with a classical algorithm. But in this case the speedup is merely quadratic (the quantum time scales like the square root of the classical time), in contrast to the exponential speedup in the solution to Simon's problem. Nevertheless, the result (discovered by Lov Grover) is extremely interesting, because of the broad utility of the algorithm.

Heuristically, the problem we will address is: we are confronted by a very large unsorted database containing $N \gg 1$ items, and we are to locate one particular item, to find a needle in the haystack. Mathematically, the database is represented by a table, or a function $f(x)$, with $x \in \{0, 1, 2, \dots, N-1\}$. We have been assured that the entry a occurs in the table exactly once; that is, that $f(x) = a$ for only one value of x. The problem is, given a, to find this value of x.

If the database has been properly *sorted*, searching for x is easy. Perhaps someone has been kind enough to list the values of a in ascending order. Then we can find x by looking up only $\log_2 N$ entries in the table. Let's suppose $N \equiv 2^n$ is a power of 2. First we look up $f(x)$ for $x = 2^{n-1} - 1$, and check if $f(x)$ is greater than a. If so, we next look up f at $x = 2^{n-2} - 1$, *etc.* With each table lookup, we reduce the number of candidate values of x by a factor of 2, so that n lookups suffice to sift through all 2^n sorted items. You can use this algorithm to look up a number in the Los Angeles phone book, because the names are listed in lexicographic order.

But now suppose that you know someone's phone number, and you want to look up her *name*. Unless you are fortunate enough to have access to a reverse directory, this is a tedious procedure. Chances are you will need to check quite a few entries in the phone book before you come across her number.

In fact, if the N numbers are listed in a random order, you will need to look up $\frac{1}{2}N$ numbers before the probability is $P = \frac{1}{2}$ that you have found her number (and hence her name). What Grover discovered is that, if you have a quantum phone book, you can learn her name with high probability by consulting the phone book only about $\sqrt{N}$ times.

This problem, too, can be formulated as an oracle or "black box" problem. In this case, the oracle is the phone book, or lookup table. We can input a name (a value of x) and the oracle outputs either 0, if $f(x) \neq a$, or 1, if $f(x) = a$. Our task is to find, as quickly as possible, the value of x with

$$f(x) = a. \tag{6.132}$$

Why is this problem important? You may have never tried to find in the phone book the name that matches a given number, but if it weren't so hard you might try it more often! More broadly, a rapid method for searching an unsorted database could be invoked to solve any problem in NP. Our oracle could be a subroutine that interrogates every potential "witness" y that could potentially testify to certify a solution to the problem. For example, if we are confronted by a graph and need to know if it admits a Hamiltonian path, we could submit a path to the "oracle," and it could quickly answer whether the path is Hamiltonian or not. If we knew a fast way to query the oracle about all the possible paths, we would be able to find a Hamiltonian path efficiently (if one exists).

6.4.1 The oracle

So "oracle" could be shorthand for a subroutine that quickly evaluates a function to check a proposed solution to a decision problem, but let us continue to regard the oracle abstractly, as a black box. The oracle "knows" that of the 2^n possible strings of length n, one (the "marked" string or "solution" ω) is special. We submit a query x to the oracle, and it tells us whether $x=\omega$ or not. It returns, in other words, the value of a function $f_\omega(x)$, with

$$\begin{aligned} f_\omega(x) &= 0, \qquad x \neq \omega, \\ f_\omega(x) &= 1, \qquad x = \omega. \end{aligned} \tag{6.133}$$

But furthermore, it is a *quantum* oracle, so it can respond to queries that are superpositions of strings. The oracle is a quantum black box that implements the unitary transformation

$$\boldsymbol{U}_{f_\omega} : |x\rangle|y\rangle \to |x\rangle|y \oplus f_\omega(x)\rangle, \tag{6.134}$$

where $|x\rangle$ is an n-qubit state, and $|y\rangle$ is a single-qubit state.

As we have previously seen in other contexts, we may choose the state of the single-qubit register to be $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, so that the oracle acts as

$$\begin{aligned} \boldsymbol{U}_{f_\omega} &: |x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\ &\to (-1)^{f_\omega(x)}|x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \end{aligned} \tag{6.135}$$

We may now ignore the second register, and obtain

$$\boldsymbol{U}_\omega : |x\rangle \to (-1)^{f_\omega(x)}|x\rangle, \tag{6.136}$$

or

$$\boldsymbol{U}_\omega = \mathit{1} - 2|\omega\rangle\langle\omega|. \tag{6.137}$$

The oracle flips the sign of the state $|\omega\rangle$, but acts trivially on any state orthogonal to $|\omega\rangle$. This transformation has a simple geometrical interpretation. Acting on any vector in the 2^n-dimensional Hilbert space, U_ω *reflects* the vector about the hyperplane orthogonal to $|\omega\rangle$ (it preserves the component in the hyperplane, and flips the component along $|\omega\rangle$).

We know that the oracle performs this reflection for some particular computational basis state $|\omega\rangle$, but we know nothing *a priori* about the value of the string ω. Our job is to determine ω, with high probability, consulting the oracle a minimal number of times.

6.4.2 The Grover iteration

As a first step, we prepare the state

$$|s\rangle = \frac{1}{\sqrt{N}}\left(\sum_{x=0}^{N-1}|x\rangle\right), \tag{6.138}$$

The equally weighted superposition of all computational basis states – this can be done easily by applying the Hadamard transformation to each qubit of the initial state $|x=0\rangle$. Although we do not know the value of ω, we *do* know that $|\omega\rangle$ is a computational basis state, so that

$$|\langle\omega|s\rangle| = \frac{1}{\sqrt{N}}, \tag{6.139}$$

irrespective of the value of ω. Were we to measure the state $|s\rangle$ by projecting onto the computational basis, the probability that we would "find" the marked state $|\omega\rangle$ is only $\frac{1}{N}$. But following Grover, we can repeatedly iterate a transformation that enhances the probability amplitude of the unknown state $|\omega\rangle$ that we are seeking, while suppressing the amplitude of all of the undesirable states $|x \neq \omega\rangle$. We construct this Grover iteration by combining the unknown reflection $\boldsymbol{U}_\omega$ performed by the oracle with a known reflection that we can perform ourselves. This known reflection is

$$\boldsymbol{U}_s = 2|s\rangle\langle s| - \mathit{1}, \tag{6.140}$$

which preserves $|s\rangle$, but flips the sign of any vector orthogonal to $|s\rangle$. Geometrically, acting on an arbitrary vector, it preserves the component along $|s\rangle$ and flips the component in the hyperplane orthogonal to $|s\rangle$.

We'll return below to the issue of constructing a quantum circuit that implements $\boldsymbol{U}_s$; for now let's just assume that we can perform $\boldsymbol{U}_s$ efficiently.

One Grover iteration is the unitary transformation

$$\boldsymbol{R}_{\text{grov}} = \boldsymbol{U}_s \boldsymbol{U}_\omega, \tag{6.141}$$

one oracle query followed by our reflection. Let's consider how $\boldsymbol{R}_{\text{grov}}$ acts in the plane spanned by $|\omega\rangle$ and $|s\rangle$. This action is easiest to understand if we visualize it geometrically. Recall that

$$|\langle s|\omega\rangle| = \frac{1}{\sqrt{N}} \equiv \sin\theta, \tag{6.142}$$

so that $|s\rangle$ is rotated by θ from the axis $|\omega^\perp\rangle$ normal to $|\omega\rangle$ in the plane. $\boldsymbol{U}_\omega$ reflects a vector in the plane about the axis $|\omega^\perp\rangle$, and $\boldsymbol{U}_s$ reflects a vector about the axis $|s\rangle$. Together, the two reflections rotate the vector by 2θ:

The Grover iteration, then, is nothing but a rotation by 2θ in the plane determined by $|s\rangle$ and $|\omega\rangle$.

6.4.3 Finding 1 out of 4

Let's suppose, for example, that there are $N = 4$ items in the database, with one marked item. With classical queries, the marked item could be found in the 1st, 2nd, 3rd, or 4th query; on the average $2\frac{1}{2}$ queries will be needed before we are successful and four are needed in the worst case.[8] But since $\sin\theta = \frac{1}{\sqrt{N}} = \frac{1}{2}$, we have $\theta = 30^o$ and $2\theta = 60^o$. After one Grover iteration, then, we rotate $|s\rangle$ to a 90^o angle with $|\omega^\perp\rangle$; that is, it lines up with $|\omega\rangle$. When we measure by projecting onto the computational basis, we obtain the result $|\omega\rangle$ *with certainty*. Just one quantum query suffices to find the marked state, a notable improvement over the classical case.

[8] Of course, if we know there is one marked state, the 4th query is actually superfluous, so it might be more accurate to say that at most three queries are needed, and $2\frac{1}{4}$ queries are required on the average.

There is an alternative way to visualize the Grover iteration that is sometimes useful, as an "inversion about the average." If we expand a state $|\psi\rangle$ in the computational basis

$$|\psi\rangle = \sum_x a_x|x\rangle, \tag{6.143}$$

then its inner product with $|s\rangle = \frac{1}{\sqrt{N}}\sum_x |x\rangle$ is

$$\langle s|\psi\rangle = \frac{1}{\sqrt{N}}\sum_x a_x = \sqrt{N}\langle a\rangle, \tag{6.144}$$

where

$$\langle a\rangle = \frac{1}{N}\sum_x a_x, \tag{6.145}$$

is the mean of the amplitude. Then if we apply $\boldsymbol{U}_s = 2|s\rangle\langle s| - \mathit{1}$ to $|\psi\rangle$, we obtain

$$\boldsymbol{U}_s|\psi\rangle = \sum_x (2\langle a\rangle - a_x)|x\rangle; \tag{6.146}$$

the amplitudes are transformed as

$$\boldsymbol{U}_s : a_x - \langle a\rangle \to \langle a\rangle - a_x, \tag{6.147}$$

that is the coefficient of $|x\rangle$ is inverted about the mean value of the amplitude.

If we consider again the case $N = 4$, then in the state $|s\rangle$ each amplitude is $\frac{1}{2}$. One query of the oracle flips the sign of the amplitude of marked state, and so reduces the mean amplitude to $\frac{1}{4}$. Inverting about the mean then brings the amplitudes of all unmarked states from $\frac{1}{2}$ to zero, and raises the amplitude of the marked state from $-\frac{1}{2}$ to 1. So we recover our conclusion that one query suffices to find the marked state with certainty.

We can also easily see that one query is sufficient to find a marked state if there are N entries in the database, and exactly $\frac{1}{4}$ of them are marked. Then, as above, one query reduces the mean amplitude from $\frac{1}{\sqrt{N}}$ to $\frac{1}{2\sqrt{N}}$, and inversion about the mean then reduces the amplitude of each unmarked state to zero.

(When we make this comparison between the number of times we need to consult the oracle if the queries can be quantum rather than classical, it may be a bit unfair to say that only one query is needed in the quantum case. If the oracle is running a routine that computes a function, then some scratch space will be filled with garbage during the computation. We will need to erase the garbage by running the computation backwards in order to maintain quantum coherence. If the classical computation is irreversible there is no need to run the oracle backwards. In this sense, one query of the quantum oracle may be roughly equivalent, in terms of complexity, to two queries of a classical oracle.)

6.4.4 Finding 1 out of N

Let's return now to the case in which the database contains N items, and exactly one item is marked. Each Grover iteration rotates the quantum state in the plane determined by $|s\rangle$ and $|\omega\rangle$; after T iterations, the state is rotated by $\theta + 2T\theta$ from the $|\omega^\perp\rangle$ axis. To optimize the probability of finding the marked state when we finally perform the measurement, we will iterate until this angle is close to 90^o, or

$$(2T+1)\theta \simeq \frac{\pi}{2} \Rightarrow 2T+1 \simeq \frac{\pi}{2\theta}, \tag{6.148}$$

we recall that $\sin\theta = \frac{1}{\sqrt{N}}$, or

$$\theta \simeq \frac{1}{\sqrt{N}}, \tag{6.149}$$

for N large; if we choose

$$T = \frac{\pi}{4}\sqrt{N}(1 + O(N^{-1/2})), \tag{6.150}$$

then the probability of obtaining the measurement result $|\omega\rangle$ will be

$$\text{Prob}(\omega) = \sin^2\left((2T+1)\theta\right) = 1 - O\left(\frac{1}{N}\right). \tag{6.151}$$

We conclude that only about $\frac{\pi}{4}\sqrt{N}$ queries are needed to determine ω with high probability, a quadratic speedup relative to the classical result.

6.4.5 Multiple solutions

If there are $r > 1$ marked states, and r is known, we can modify the number of iterations so that the probability of finding one of the marked states is still very close to 1. The analysis is just as above, except that the oracle induces a reflection in the hyperplane orthogonal to the vector

$$|\tilde{\omega}\rangle = \frac{1}{\sqrt{r}}\left(\sum_{i=1}^{r}|\omega_i\rangle\right), \tag{6.152}$$

the equally weighted superposition of the marked computational basis states $|\omega_i\rangle$. Now

$$\langle s|\tilde{\omega}\rangle = \sqrt{\frac{r}{N}} \equiv \sin\theta, \tag{6.153}$$

and a Grover iteration rotates a vector by 2θ in the plane spanned by $|s\rangle$ and $|\tilde{\omega}\rangle$; we again conclude that the state is close to $|\tilde{\omega}\rangle$ after a number of iterations

$$T \simeq \frac{\pi}{4\theta} = \frac{\pi}{4}\sqrt{\frac{N}{r}}. \tag{6.154}$$

If we then measure by projecting onto the computational basis, we will find one of the marked states (each occurring equiprobably) with probability close to one. (As the number of solutions increases, the time needed to find one of them declines like $r^{-1/2}$, as opposed to r^{-1} in the classical case.)

Note that if we continue to perform further Grover iterations, the vector continues to rotate, and so the probability of finding a marked state (when we finally measure) begins to decline. The Grover algorithm is like baking a soufflé – if we leave it in the oven for too long, it starts to fall. Therefore, if we don't know anything about the number of marked states, we might fail to find one of them. For example, $T \sim \frac{\pi}{4}\sqrt{N}$ iterations is optimal for $r = 1$, but for $r = 4$, the probability of finding a marked state after this many iterations is quite close to zero.

But even if we don't know r *a priori*, we can still find a solution with a quadratic speed up over classical algorithms (for $r \ll N$). For example, we might choose the number of iterations to be random in the range 0 to $\frac{\pi}{4}\sqrt{N}$. Then the expected probability of finding a marked state is close to 1/2 for each r, so we are unlikely to fail to find a marked state after several repetitions. And each time we measure, we can submit the state we find to the oracle as a classical query to confirm whether that state is really marked.

In particular, if we don't find a solution after several attempts, there probably is no solution. Hence with high probability we can correctly answer the yes/no question, "Is there a marked state?" Therefore, we can adopt the Grover algorithm to solve any NP problem, where the oracle checks a proposed solution, with a quadratic speedup over a classical exhaustive search.

6.4.6 Implementing the reflection

To perform a Grover iteration, we need (aside from the oracle query) a unitary transformation

$$\boldsymbol{U}_s = 2|s\rangle\langle s| - \mathit{1}, \tag{6.155}$$

that reflects a vector about the axis defined by the vector $|s\rangle$. How do we build this transformation efficiently from quantum gates? Since $|s\rangle = \boldsymbol{H}^{(n)}|0\rangle$, where $\boldsymbol{H}^{(n)}$ is the bitwise Hadamard transformation, we may write

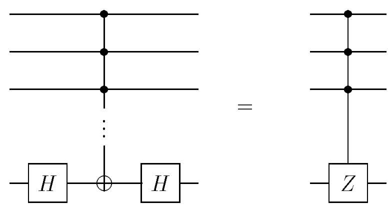
$$\boldsymbol{U}_s = \boldsymbol{H}^{(n)}(2|0\rangle\langle 0| - \mathit{1})\boldsymbol{H}^{(n)}, \tag{6.156}$$

so it will suffice to construct a reflection about the axis $|0\rangle$. We can easily build this reflection from an n-bit Toffoli gate $\theta^{(n)}$.

Recall that

$$\boldsymbol{H}\boldsymbol{\sigma}_x\boldsymbol{H} = \boldsymbol{\sigma}_z; \tag{6.157}$$

a bit flip in the Hadamard rotated basis is equivalent to a flip of the relative phase of $|0\rangle$ and $|1\rangle$. Therefore:



after conjugating the last bit by $\boldsymbol{H}$, $\theta^{(n)}$ becomes controlled$^{(n-1)}$-$\boldsymbol{\sigma}_z$, which flips the phase of $|11\ldots|1\rangle$ and acts trivially on all other computational basis states. Conjugating by NOT$^{(n)}$, we obtain $\boldsymbol{U}_s$, aside from an irrelevant overall minus sign.

You will show in an exercise that the n-bit Toffoli gate $\theta^{(n)}$ can be constructed from $2n-5$ 3-bit Toffoli gates $\theta^{(3)}$ (if sufficient scratch space is available). Therefore, the circuit that constructs $\boldsymbol{U}_s$ has a size *linear* in $n = \log N$. Grover's database search (assuming the oracle answers a query instantaneously) takes a time of order $\sqrt{N}\log N$. If we regard the oracle as a subroutine that performs a function evaluation in polylog time, then the search takes time of order $\sqrt{N}\mathrm{poly}(\log N)$.

6.5 The Grover Algorithm Is Optimal

Grover's quadratic quantum speedup of the database search is already interesting and potentially important, but surely with more cleverness we can do better, can't we? No, it turns out that we can't. Grover's algorithm provides the fastest possible quantum search of an unsorted database, if "time" is measured according to the number of queries of the oracle.

Considering the case of a single marked state $|\omega\rangle$, let $\boldsymbol{U}(\omega, T)$ denote a quantum circuit that calls the oracle T times. We place *no* restriction on the circuit aside from specifying the number of queries; in particular, we place no limit on the number of quantum gates. This circuit is applied to an initial state $|\psi(0)\rangle$, producing a final state

$$|\psi_\omega(t)\rangle = \boldsymbol{U}(\omega, T)|\psi(0)\rangle. \tag{6.158}$$

Now we are to perform a measurement designed to distinguish among the N possible values of ω. If we are to be able to perfectly distinguish among the possible values, the states $|\psi_\omega(t)\rangle$ must all be mutually orthogonal, and if we are to distinguish correctly with high probability, they must be nearly orthogonal.

Now, if the states $\{|\psi_\omega\rangle$ are an orthonormal basis, then, for any fixed normalized vector $|\varphi\rangle$,

$$\sum_{\omega=0}^{N-1} \parallel |\psi_\omega\rangle - |\varphi\rangle \parallel^2 \geq 2N - 2\sqrt{N}. \tag{6.159}$$

(The sum is minimized if $|\varphi\rangle$ is the equally weighted superposition of all the basis elements, $|\varphi\rangle = \frac{1}{\sqrt{N}}\sum_\omega |\psi_\omega\rangle$, as you can show by invoking a Lagrange multiplier to perform the constrained extremization.) Our strategy will be to choose the state $|\varphi\rangle$ suitably so that we can use this inequality to learn something about the number T of oracle calls.

Our circuit with T queries builds a unitary transformation

$$\boldsymbol{U}(\omega, T) = \boldsymbol{U}_\omega \boldsymbol{U}_T \boldsymbol{U}_\omega \boldsymbol{U}_{T-1} \ldots \boldsymbol{U}_\omega \boldsymbol{U}_1, \tag{6.160}$$

where $\boldsymbol{U}_\omega$ is the oracle transformation, and the $\boldsymbol{U}_t$'s are arbitrary non-oracle transformations. For our state $|\varphi(T)\rangle$ we will choose the result of applying $\boldsymbol{U}(\omega, T)$ to $|\psi(0)\rangle$, except with each $\boldsymbol{U}_\omega$ replaced by $\mathit{1}$; that is, the same circuit, but with all queries submitted to the "empty oracle." Hence,

$$|\varphi(T)\rangle = \boldsymbol{U}_T \boldsymbol{U}_{T-1} \ldots \boldsymbol{U}_2 \boldsymbol{U}_1 |\psi(0)\rangle, \tag{6.161}$$

while

$$|\psi_\omega(T)\rangle = \boldsymbol{U}_\omega \boldsymbol{U}_T \boldsymbol{U}_\omega \boldsymbol{U}_{T-1} \ldots \boldsymbol{U}_\omega \boldsymbol{U}_1 |\psi(0)\rangle. \tag{6.162}$$

To compare $|\varphi(T)\rangle$ and $|\psi_\omega(T)\rangle$, we appeal to our previous analysis of the effect of errors on the accuracy of a circuit, regarding the ω oracle as an "erroneous" implementation of the empty oracle. The error vector in the t-th step (cf. eq. (6.63)) is

$$\begin{aligned} \parallel |E(\omega, t)\rangle \parallel &= \parallel (\boldsymbol{U}_\omega - \mathit{1})|\varphi(t)\rangle \parallel \\ &= 2|\langle \omega|\varphi(t)\rangle|, \end{aligned} \tag{6.163}$$

since $\boldsymbol{U}_\omega = \mathit{1} - 2|\omega\rangle\langle\omega|$. After T queries we have (cf. eq. (6.66))

$$\parallel |\psi_\omega(T)\rangle - |\varphi(T)\rangle \parallel \leq 2\sum_{t=1}^{T} |\langle \omega|\varphi(t)\rangle|. \tag{6.164}$$

From the identity

$$\begin{aligned} &\left(\sum_{t=1}^{T} c_t\right)^2 + \frac{1}{2}\sum_{s,t=1}^{T}(c_s - c_t)^2 \\ &= \sum_{s,t=1}^{T}\left(c_t c_s + \frac{1}{2}c_s^2 - c_t c_s + \frac{1}{2}c_s^2\right) = T\sum_{t=1}^{T} c_t^2, \end{aligned} \tag{6.165}$$

we obtain the inequality

$$\left(\sum_{t=1}^{T} c_t\right)^2 \leq T\sum_{t=1}^{T} c_t^2, \tag{6.166}$$

which applied to eq. (6.164) yields

$$\parallel |\psi_\omega(T)\rangle - |\varphi(T)\rangle \parallel^2 \leq 4T\left(\sum_{t=1}^{T} |\langle\omega|\varphi(t)\rangle|^2\right). \tag{6.167}$$

Summing over ω we find

$$\sum_\omega \parallel |\psi_\omega(T)\rangle - |\varphi(T)\rangle \parallel^2 \leq 4T\sum_{t=1}^{T}\langle\varphi(t)|\varphi(t)\rangle \ = \ 4T^2. \tag{6.168}$$

Invoking eq. (6.159) we conclude that

$$4T^2 \geq 2N - 2\sqrt{N}, \tag{6.169}$$

if the states $|\psi_\omega(T)\rangle$ are mutually orthogonal. We have, therefore, found that any quantum algorithm that can distinguish all the possible values of the marked state must query the oracle T times where

$$T \geq \sqrt{\frac{N}{2}}, \tag{6.170}$$

(ignoring the small correction as $N \to \infty$). Grover's algorithm finds ω in $\frac{\pi}{4}\sqrt{N}$ queries, which exceeds this bound by only about 11%. In fact, it is possible to refine the argument to improve the bound to $T \geq \frac{\pi}{4}\sqrt{N}(1-\varepsilon)$, which is asymptotically saturated by the Grover algorithm.[9] Furthermore, we can show that Grover's circuit attains the optimal success probability in $T \leq \frac{\pi}{4}\sqrt{N}$ queries.

One feels a twinge of disappointment (as well as a surge of admiration for Grover) at the realization that the database search algorithm cannot be improved. What are the implications for quantum complexity?

For many optimization problems in the NP class, there is no better method known than exhaustive search of all the possible solutions. By exploiting quantum parallelism, we can achieve a quadratic speedup of exhaustive search. Now we have learned that the quadratic speedup is the best possible if we rely on the power of sheer quantum parallelism, if we don't design our quantum algorithm to exploit the specific structure of the problem we wish to solve. Still, we might do better if we are sufficiently clever.

The optimality of the Grover algorithm might be construed as evidence that $BQP \not\supseteq NP$. At least, if it turns out that $NP \subseteq BQP$ and $P \neq NP$, then the NP problems must share a deeply hidden structure (for which there is currently no evidence) that is well-matched to the peculiar capabilities of quantum circuits.

Even the quadratic speedup may prove useful for a variety of NP-complete optimization problems. But a quadratic speedup, unlike an exponential one, does not really move the frontier between solvability and intractability. Quantum computers may someday outperform classical computers in performing exhaustive search, but only if the clock speed of quantum devices does not lag too far behind that of their classical counterparts.

6.6 Generalized Search and Structured Search

In the Grover iteration, we perform the transformation $\boldsymbol{U}_s = 2|s\rangle\langle s| - \mathit{1}$, the reflection in the axis defined by $|s\rangle = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle$. Why this axis? The advantage of the state $|s\rangle$ is that it has the same overlap with each and every computational basis state. Therefore, the overlap of any marked state $|\omega\rangle$ with $|s\rangle$ is guaranteed to be $|\langle\omega|s\rangle| = 1/\sqrt{N}$. Hence, if we know the number of marked states, we can determine how many iterations are required to find a marked state with high probability – the number of iterations needed does

[9]C. Zalka, "Grover's Quantum Searching Algorithm is Optimal," quant-ph/9711070.

not depend on which states are marked.

But of course, we could choose to reflect about a different axis. If we can build the unitary $\boldsymbol{U}$ (with reasonable efficiency) then we can construct

$$\boldsymbol{U}(2|0\rangle\langle 0| - \mathit{1})\boldsymbol{U}^\dagger = 2\boldsymbol{U}|0\rangle\langle 0|\boldsymbol{U}^\dagger - \mathit{1}, \tag{6.171}$$

which reflects in the axis $\boldsymbol{U}|0\rangle$.

Suppose that

$$|\langle\omega|\boldsymbol{U}|0\rangle| = \sin\theta, \tag{6.172}$$

where $|\omega\rangle$ is the marked state. Then if we replace $\boldsymbol{U}_s$ in the Grover iteration by the reflection eq. (6.171), one iteration performs a rotation by 2θ in the plane determined by $|\omega\rangle$ and $\boldsymbol{U}|0\rangle$ (by the same argument we used for $\boldsymbol{U}_s$). Thus, after T iterations, with $(2T+I)\theta \cong \pi/2$, a measurement in the computational basis will find $|\omega\rangle$ with high probability. Therefore, we can still search a database if we replace $\boldsymbol{H}^{(n)}$ by $\boldsymbol{U}$ in Grover's quantum circuit, as long as $\boldsymbol{U}|0\rangle$ is not orthogonal to the marked state.[10] But if we have no *a priori* information about which state is marked, then $\boldsymbol{H}^{(n)}$ is the best choice, not only because $|s\rangle$ has a known overlap with each marked state, but also because it has the largest *average* overlap with all the possible marked states.

But sometimes when we are searching a database, we *do* have some information about where to look, and in that case, the generalized search strategy described above may prove useful.[11]

As an example of a problem with some auxiliary structure, suppose that $f(x,y)$ is a one-bit-valued function of the two n-bit strings x and y, and we are to find the unique solution to $f(x,y) = 1$. With Grover's algorithm, we can search through the N^2 possible values ($N = 2^n$) of (x,y) and find the solution (x_0, y_0) with high probability after $\frac{\pi}{4}N$ iterations, a quadratic speedup with respect to classical search.

But further suppose that $g(x)$ is a function of x only, and that it is known that $g(x) = 1$ for exactly M values of x, where $1 \ll M \ll N$. And furthermore, it is known that $g(x_0) = 1$. Therefore, we can use g to help us find the solution (x_0, y_0).

[10]L.K. Grover "Quantum Computers Can Search Rapidly By Using Almost Any Transformation," quant-ph/9712011.

[11]E. Farhi and S. Gutmann, "Quantum-Mechanical Square Root Speedup in a Structured Search Problem," quant-ph/9711035; L.K. Grover, "Quantum Search On Structured Problems," quant-ph/9802035.

Now we have two oracles to consult, one that returns the value of $f(x,y)$, and the other returning the value of $g(x)$. Our task is to find (x_0, y_0) with a minimal number of queries.

Classically, we need of order NM queries to find the solution with reasonable probability. We first evaluate $g(x)$ for each x; then we restrict our search for a solution to $f(x,y) = 1$ to only those M values of x such that $g(x) = 1$. It is natural to wonder whether there is a way to perform a quantum search in a time of order the square root of the classical time. Exhaustive search that queries only the f oracle requires time $N \gg \sqrt{NM}$, and so does not do the job. We need to revise our method of quantum search to take advantage of the structure provided by g.

A better method is to first apply Grover's algorithm to $g(x)$. In about $\frac{\pi}{4}\sqrt{\frac{N}{M}}$ iterations, we prepare a state that is close to the equally weighted superposition of the M solutions to $g(x) = 1$. In particular, the state $|x_0\rangle$ appears with amplitude $\frac{1}{\sqrt{M}}$. Then we apply Grover's algorithm to $f(x,y)$ with x fixed. In about $\frac{\pi}{4}\sqrt{N}$ iterations, the state $|x_0\rangle|s\rangle$ evolves to a state quite close to $|x_0\rangle|y_0\rangle$. Therefore $|x_0, y_0\rangle$ appears with amplitude $\frac{1}{\sqrt{M}}$.

The unitary transformation we have constructed so far, in about $\frac{\pi}{4}\sqrt{N}$ queries, can be regarded as the transformation $\boldsymbol{U}$ that defines a generalized search. Furthermore, we know that

$$\langle x_0, y_0|U|0,0\rangle \cong \frac{1}{\sqrt{M}}. \tag{6.173}$$

Therefore, if we iterate the generalized search about $\frac{\pi}{4}\sqrt{M}$ times, we will have prepared a state that is quite close to $|x_0, y_0\rangle$. With altogether about

$$\left(\frac{\pi}{4}\right)^2 \sqrt{NM}, \tag{6.174}$$

queries, then, we can find the solution with high probability. This is indeed a quadratic speedup with respect to the classical search.

6.7 Some Problems Admit No Speedup

The example of structured search illustrates that quadratic quantum speedups over classical algorithms can be attained for a variety of problems, not just for an exhaustive search of a structureless database. One might even dare to hope that quantum parallelism enables us to significantly speedup any classical algorithm. This hope will now be dashed – for many problems, no quantum speedup is possible.

We continue to consider problems with a quantum black box, an oracle, that computes a function f taking n bits to 1. But we will modify our notation a little. The function f can be represented as a string of $N = 2^n$ bits

$$X = X_{N-1}X_{N-2}\ldots X_1X_0, \tag{6.175}$$

where X_i denotes $f(i)$. Our problem is to evaluate some one-bit-valued function of X, that is, to answer a yes/no question about the properties of the oracle. What we will show is that for some functions of X, we can't evaluate the function with low error probability using a quantum algorithm, unless the algorithm queries the oracle as many times (or nearly as many times) as required with a classical algorithm.[12]

The key idea is that any Boolean function of the X_i's can be represented as a polynomial in the X_i's. Furthermore, the probability distribution for a quantum measurement can be expressed as a polynomial in X, where the degree of the polynomial is $2T$, if the measurement follows T queries of the oracle. The issue, then, is whether a polynomial of degree $2T$ can provide a reasonable approximation to the Boolean function of interest.

The action of the oracle can be represented as

$$\boldsymbol{U}_O : |i, y; z\rangle \to |i, y \oplus X_i; z\rangle, \tag{6.176}$$

where i takes values in $\{0, 1, \ldots, N-1\}, y \in \{0,1\}$, and z denotes the state of auxiliary qubits not acted upon by the oracle. Therefore, in each 2×2 block spanned by $|i, 0, z\rangle$ and $|i, 1, z\rangle, \boldsymbol{U}_O$ is the 2×2 matrix

$$\begin{pmatrix} 1 - X_i & X_i \\ X_i & 1 - X_i \end{pmatrix}. \tag{6.177}$$

Quantum gates other than oracle queries have no dependence on X. Therefore after a circuit with T queries acts on any initial state, the resulting state $|\psi\rangle$ has amplitudes that are (at most) Tth-degree polynomials in X. If we perform a POVM on $|\psi\rangle$, then the probability $\langle\psi|\boldsymbol{F}|\psi\rangle$ of the outcome associated with the positive operator $\boldsymbol{F}$ can be expressed as a polynomial in X of degree at most $2T$.

[12] E. Farhi, *et al.*, quant-ph/9802045; R. Beals, *et al.*, quant-ph/9802049.

Now any Boolean function of the X_i's can be expressed (uniquely) as a polynomial of degree $\leq N$ in the X_i's. For example, consider the OR function of the N X_i's; it is

$$\mathrm{OR}(X) = 1 - (1 - X_0)(1 - X_1)\cdots(1 - X_{N-1}), \tag{6.178}$$

a polynomial of degree N.

Suppose that we would like our quantum circuit to evaluate the OR function *with certainty*. Then we must be able to perform a measurement with two outcomes, 0 and 1, where

$$\begin{aligned} \mathrm{Prob}(0) &= 1 - \mathrm{OR}(X), \\ \mathrm{Prob}(1) &= \mathrm{OR}(X). \end{aligned} \tag{6.179}$$

But these expressions are polynomials of degree N, which can arise only if the circuit queries the oracle at least T times, where

$$T \geq \frac{N}{2}. \tag{6.180}$$

We conclude that no quantum circuit with fewer than $N/2$ oracle calls can compute OR exactly. In fact, for this function (or any function that takes the value 0 for just one of its N possible arguments), there is a stronger conclusion (exercise): we require $T \geq N$ to evaluate OR with certainty.

On the other hand, evaluating the OR function (answering the yes/no question, "Is there a marked state?") is just what the Grover algorithm can achieve in a number of queries of order $\sqrt{N}$. Thus, while the conclusion is correct that N queries are needed to evaluate OR *with certainty*, this result is a bit misleading. We can evaluate OR *probabilistically* with far fewer queries. Apparently, the Grover algorithm can construct a polynomial in X that, though only of degree $O(\sqrt{N})$, provides a very adequate approximation to the N-th degree polynomial $\mathrm{OR}(X)$.

But OR, which takes the value 1 for every value of X except $X = \vec{0}$, is a very simple Boolean function. We should consider other functions that might pose a more serious challenge for the quantum computer.

One that comes to mind is the PARITY function: $\mathrm{PARITY}(X)$ takes the value 0 if the string X contains an even number of 1's, and the value 1 if the string contains an odd number of 1's. Obviously, a classical algorithm must query the oracle N times to determine the parity. How much better

can we do by submitting quantum queries? In fact, we can't do much better at all – at least $N/2$ quantum queries are needed to find the correct value of $\mathrm{PARITY}(X)$, with probability of success greater than $\frac{1}{2} + \delta$.

In discussing PARITY it is convenient to use new variables

$$\tilde{X}_i = 1 - 2X_i, \tag{6.181}$$

that take values ± 1, so that

$$\mathrm{PARITY}(\tilde{X}) = \prod_{i=0}^{N-1} \tilde{X}_i, \tag{6.182}$$

also takes values ± 1. Now, after we execute a quantum circuit with altogether T queries of the oracle, we are to perform a POVM with two possible outcomes $\boldsymbol{F}_{\mathrm{even}}$ and $\boldsymbol{F}_{\mathrm{odd}}$; the outcome will be our estimate of $\mathrm{PARITY}(\tilde{X})$. As we have already noted, the probability of obtaining the outcome even (say) can be expressed as a polynomial $P^{(2T)}_{\mathrm{even}}$ of degree (at most) $2T$ in $\tilde{X}$,

$$\langle \boldsymbol{F}_{\mathrm{even}} \rangle = P^{(2T)}_{\mathrm{even}}(\tilde{X}). \tag{6.183}$$

How often is our guess correct? Consider the sum

$$\begin{aligned} &\sum_{\{\tilde{X}\}} P^{(2T)}_{\mathrm{even}}(\tilde{X}) \cdot \mathrm{PARITY}(\tilde{X}) \\ &= \sum_{\{\tilde{X}\}} P^{(2T)}_{\mathrm{even}}(\tilde{X}) \prod_{i=0}^{N-1} \tilde{X}_i. \end{aligned} \tag{6.184}$$

Since each term in the polynomial $P^{(2T)}_{\mathrm{even}}(\tilde{X})$ contains at most $2T$ of the $\tilde{X}_i$'s, we can invoke the identity

$$\sum_{\tilde{X}_i \in \{0,1\}} \tilde{X}_i = 0, \tag{6.185}$$

to see that the sum in eq. (6.184) must vanish if $N > 2T$. We conclude that

$$\sum_{\mathrm{par}(\tilde{X})=1} P^{(2T)}_{\mathrm{even}}(\tilde{X}) = \sum_{\mathrm{par}(\tilde{X})=-1} P^{(2T)}_{\mathrm{even}}(\tilde{X}); \tag{6.186}$$

hence, for $T < N/2$, we are just as likely to guess "even" when the actual $\mathrm{PARITY}(\tilde{X})$ is odd as when it is even (on average). Our quantum algorithm fails to tell us anything about the value of PARITY($\tilde{X}$); that is, averaged over the (a priori equally likely) possible values of X_i, we are just as likely to be right as wrong.

We can also show, by exhibiting an explicit algorithm (exercise), that $N/2$ queries (assuming N even) are *sufficient* to determine PARITY (either probabilistically or deterministically.) In a sense, then, we can achieve a factor of 2 speedup compared to classical queries. But that is the best we can do.

6.8 Distributed database search

We will find it instructive to view the quantum database search algorithm from a fresh perspective. We imagine that two parties, Alice and Bob, need to arrange to meet on a mutually agreeable day. Alice has a calendar that lists $N = 2^n$ days, with each day marked by either a 0, if she is unavailable that day, or a 1, if she is available. Bob has a similar calendar. Their task is to find a day when they will both be available.

Alice and Bob both have quantum computers, but they are very far apart from one another. (Alice is on earth, and Bob has traveled to the Andromeda galaxy). Therefore, it is very expensive for them to communicate. They urgently need to arrange their date, but they must economize on the amount of information that they send back and forth.

Even if there exists a day when both are available, it might not be easy to find it. If Alice and Bob communicate by sending classical bits back and forth, then in the worst case they will need to exchange of order $N = 2^n$ calendar entries to have a reasonable chance of successfully arranging their date.. We will ask: can they do better by exchanging qubits instead?[13] (The quantum

[13]In an earlier version of these notes, I proposed a different scenario, in which Alice and Bob had nearly identical tables, but with a single mismatched entry; their task was to find the location of the mismatched bit. However, that example was poorly chosen, because the task can be accomplished with only $\log N$ bits of classical communication. (Thanks to Richard Cleve for pointing out this blunder.) We want Alice to learn the address (a binary string of length n) of the one entry where her table differs from Bob's. So Bob computes the parity of the $N/2$ entries in his table with a label that takes the value 0 in its least significant bit, and he sends that one parity bit to Alice. Alice compares to the parity of the same entries in her table, and she infers one bit (the least significant bit) of the address of the mismatched entry. Then they do the same for each of the remaining $n-1$ bits, until Alice knows the complete address of the "error". Altogether just n bits

information highway from earth to Andromeda was carefully designed and constructed, so it does not cost much more to send qubits instead of bits.)

To someone familiar with the basics of quantum information theory, this sounds like a foolish question. Holevo's theorem told us once and for all that a single qubit can convey no more than one bit of classical information. On further reflection, though, we see that Holevo's theorem does not really settle the issue. While it bounds the mutual information of a state preparation with a measurement outcome, it does not assure us (at least not directly) that Alice and Bob need to exchange as many qubits as bits to compare their calendars. Even so, it comes as a refreshing surprise[14] to learn that Alice and Bob can do the job by exchanging $O(\sqrt{N}\log N)$ qubits, as compared to $O(N)$ classical bits.

To achieve this Alice and Bob must work in concert, implementing a distributed version of the database search. Alice has access to an oracle (her calendar) that computes a function $f_A(x)$, and Bob has an oracle (his calendar) that computes $f_B(x)$. Together, they can query the oracle

$$f_{AB}(x) = f_A(x) \wedge f_B(x)\ . \tag{6.187}$$

Either one of them can implement the reflection $\boldsymbol{U}_s$, so they can perform a complete Grover iteration, and can carry out exhaustive search for a suitable day x such that $f_{AB}(x) = 1$ (when Alice and Bob are both available). If a mutually agreeable day really exists, they will succeed in finding it after of order $\sqrt{N}$ queries.

How do Alice and Bob query f_{AB}? We'll describe how they do it acting on any one of the computational basis states $|x\rangle$. First Alice performs

$$|x\rangle|0\rangle \to |x\rangle|f_A(x)\rangle, \tag{6.188}$$

and then she sends the $n+1$ qubits to Bob. Bob performs

$$|x\rangle|f_A(x)\rangle \to (-1)^{f_A(x)\wedge f_B(x)}|x\rangle|f_A(x)\rangle. \tag{6.189}$$

This transformation is evidently unitary, and you can easily verify that Bob can implement it by querying his oracle. Now the phase multiplying $|x\rangle$ is $(-1)^{f_{AB}(x)}$ as desired, but $|f_A(x)\rangle$ remains stored in the other register, which

are sent (and all from Bob to Alice).

[14]H. Burhman, *et al.*, "Quantum vs. Classical Communication and Computation," quant-ph/9802040.

would spoil the coherence of a superposition of x values. Bob cannot erase that register, but Alice can. So Bob sends the $n+1$ qubits back to Alice, and she consults her oracle once more to perform

$$(-1)^{f_A(x)\wedge f_B(x)}|x\rangle|f_A(x)\rangle \to (-1)^{f_A(x)\wedge f_B(x)}|x\rangle|0\rangle. \tag{6.190}$$

By exchanging $2(n+1)$ qubits, the have accomplished one query of the f_{AB} oracle, and so can execute one Grover iteration.

Suppose, for example, that Alice and Bob know that there is only one mutually agreeable date, but they have no *a priori* information about which date it is. After about $\frac{\pi}{4}\sqrt{N}$ iterations, requiring altogether

$$Q \cong \frac{\pi}{4}\sqrt{N}\cdot 2(\log N+1), \tag{6.191}$$

qubit exchanges, Alice measures, obtaining the good date with probability quite close to 1.

Thus, at least in this special context, exchanging $O(\sqrt{N}\log N)$ qubits is as good as exchanging $O(N)$ classical bits. Apparently, we have to be cautious in interpreting the Holevo bound, which ostensibly tells us that a qubit has no more information-carrying capacity than a bit!

If Alice and Bob don't know in advance how many good dates there are, they can still perform the Grover search (as we noted in §6.4.5), and will find a solution with reasonable probability. With $2\cdot\log N$ bits of classical communication, they can verify whether the date that they found is really mutually agreeable.

6.8.1 Quantum communication complexity

More generally, we may imagine that several parties each possess an n-bit input, and they are to evaluate a function of all the inputs, with one party eventually learning the value of the function. What is the minimum amount of communication needed to compute the function (either deterministically or probabilistically)? The well-studied branch of classical complexity theory that addresses this question is called *communication complexity*. What we established above is a quadratic separation between quantum and classical communication complexity, for a particular class of two-party functions.

Aside from replacing the exchange of classical bits by the exchange of qubits, there are other interesting ways to generalize classical communication complexity. One is to suppose that the parties share some preexisting entangled state (either Bell pairs or multipartite entanglement), and that they may exploit that entanglement along with classical communication to perform the function evaluation. Again, it is not immediately clear that the shared entanglement will make things any easier, since entanglement alone doesn't permit the parties to exchange classical messages. But it turns out that the entanglement *does* help, at least a little bit.[15]

The analysis of communication complexity is a popular past time among complexity theorists, but this discipline does not yet seem to have assumed a prominent position in practical communications engineering. Perhaps this is surprising, considering the importance of efficiently distributing the computational load in parallelized computing, which has become commonplace. Furthermore, it seems that nearly all communication in real life can be regarded as a form of remote computation. I don't really need to receive all the bits that reach me over the telephone line, especially since I will probably retain only a few bits of information pertaining to the call tomorrow (the movie we decided to go to). As a less prosaic example, we on earth may need to communicate with a robot in deep space, to instruct it whether to enter and orbit around a distant star system. Since bandwidth is extremely limited, we would like it to compute the correct answer to the Yes/No question "Enter orbit?" with minimal exchange of information between earth and robot.

Perhaps a future civilization will exploit the known quadratic separation between classical and quantum communication complexity, by exchanging qubits rather than bits with its flotilla of spacecraft. And perhaps an exponential separation will be found, at least in certain contexts, which would significantly boost the incentive to develop the required quantum communications technology.

6.9 Periodicity

So far, the one case for which we have found an exponential separation between the speed of a quantum algorithm and the speed of the corresponding

[15] R. Cleve, et al., "Quantum Entanglement and the Communication Complexity of the Inner Product Function," quant-ph/9708019; W. van Dam, et al., "Multiparty Quantum Communication Complexity," quant-ph/9710054.

classical algorithm is the case of Simon's problem. Simon's algorithm exploits quantum parallelism to speed up the search for the period of a function. Its success encourages us to seek other quantum algorithms designed for other kinds of period finding.

Simon studied periodic functions taking values in $(Z_2)^n$. For that purpose the n-bit Hadamard transform $\boldsymbol{H}^{(n)}$ was a powerful tool. If we wish instead to study periodic functions taking values in Z_{2^n}, the (discrete) Fourier transform will be a tool of comparable power.

The moral of Simon's problem is that, while finding needles in a haystack may be difficult, finding *periodically* spaced needles in a haystack can be far easier. For example, if we scatter a photon off of a periodic array of needles, the photon is likely to be scattered in one of a set of preferred directions, where the Bragg scattering condition is satisfied. These preferred directions depend on the spacing between the needles, so by scattering just one photon, we can already collect some useful information about the spacing. We should further explore the implications of this metaphor for the construction of efficient quantum algorithms.

So imagine a quantum oracle that computes a function

$$f : \{0,1\}^n \to \{0,1\}^m, \tag{6.192}$$

that has an unknown period r, where r is a positive integer satisfying

$$1 \ll r \ll 2^n. \tag{6.193}$$

That is,

$$f(x) = f(x + mr), \tag{6.194}$$

where m is any integer such that x and $x + mr$ lie in $\{0, 1, 2, \ldots, 2^n - 1\}$. We are to find the period r. Classically, this problem is *hard*. If r is, say, of order $2^{n/2}$, we will need to query the oracle of order $2^{n/4}$ times before we are likely to find two values of x that are mapped to the same value of $f(x)$, and hence learn something about r. But we will see that there is a quantum algorithm that finds r in time poly (n).

Even if we know how to compute efficiently the function $f(x)$, it may be a hard problem to determine its period. Our quantum algorithm can be applied to finding, in poly(n) time, the period of any function that we can compute in poly(n) time. Efficient period finding allows us to efficiently

solve a variety of (apparently) hard problems, such as factoring an integer, or evaluating a discrete logarithm.

The key idea underlying quantum period finding is that the Fourier transform can be evaluated by an efficient quantum circuit (as discovered by Peter Shor). The quantum Fourier transform (QFT) exploits the power of quantum parallelism to achieve an exponential speedup of the well-known (classical) fast Fourier transform (FFT). Since the FFT has such a wide variety of applications, perhaps the QFT will also come into widespread use someday.

6.9.1 Finding the period

The QFT is the unitary transformation that acts on the computational basis according to

$$QFT : |x\rangle \to \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x y/N} |y\rangle, \tag{6.195}$$

where $N = 2^n$. For now let's suppose that we can perform the QFT efficiently, and see how it enables us to extract the period of $f(x)$.

Emulating Simon's algorithm, we first query the oracle with the input $\frac{1}{\sqrt{N}} \sum_x |x\rangle$ (easily prepared by applying $\boldsymbol{H}^{(n)}$ to $|0\rangle$), and so prepare the state

$$\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle |f(x)\rangle. \tag{6.196}$$

Then we measure the output register, obtaining the result $|f(x_0)\rangle$ for some $0 \le x_0 < r$. This measurement prepares in the input register the coherent superposition of the A values of x that are mapped to $f(x_0)$:

$$\frac{1}{\sqrt{A}} \sum_{j=0}^{A-1} |x_0 + jr\rangle, \tag{6.197}$$

where

$$N - r \le x_0 + (A-1)r < N, \tag{6.198}$$

or

$$A - 1 < \frac{N}{r} < A + 1. \tag{6.199}$$

Actually, the measurement of the output register is unnecessary. If it is omitted, the state of the input register is an incoherent superposition (summed over $x_0 \in \{0, 1, \ldots r-1\}$) of states of the form eq. (6.197). The rest of the algorithm works just as well acting on this initial state.

Now our task is to extract the value of r from the state eq. (6.197) that we have prepared. Were we to measure the input register by projecting onto the computational basis at this point, we would learn nothing about r. Instead (cf. Simon's algorithm), we should Fourier transform first and *then* measure.

By applying the QFT to the state eq. (6.197) we obtain

$$\frac{1}{\sqrt{NA}} \sum_{y=0}^{N-1} e^{2\pi i x_0 y} \sum_{j=0}^{A-1} e^{2\pi i j r y/N} |y\rangle. \tag{6.200}$$

If we now measure in the computational basis, the probability of obtaining the outcome y is

$$\text{Prob}(y) = \frac{A}{N} \left| \frac{1}{A} \sum_{j=0}^{A-1} e^{2\pi i j r y/N} \right|^2. \tag{6.201}$$

This distribution strongly favors values of y such that yr/N is close to an integer. For example, if N/r happened to be an integer (and therefore equal to A), we would have

$$\text{Prob}(y) = \frac{1}{r} \left| \frac{1}{A} \sum_{j=0}^{A-1} e^{2\pi i j y/A} \right| = \begin{cases} \frac{1}{r} & y = A \cdot (\text{integer}) \\ 0 & \text{otherwise.} \end{cases} \tag{6.202}$$

More generally, we may sum the geometric series

$$\sum_{j=0}^{A-1} e^{i\theta j} = \frac{e^{iA\theta} - 1}{e^{i\theta} - 1}, \tag{6.203}$$

where

$$\theta_y = \frac{2\pi y r (\text{mod } N)}{N}. \tag{6.204}$$

There are precisely r values of y in $\{0, 1, \ldots, N-1\}$ that satisfy

$$-\frac{r}{2} \leq yr (\text{mod } N) \leq \frac{r}{2}. \tag{6.205}$$

(To see this, imagine marking the multiples of r and N on a number line ranging from 0 to $rN-1$. For each multiple of N, there is a multiple of r no more than distance $r/2$ away.) For each of these values, the corresponding θ_y satisfies.

$$-\pi \frac{r}{N} \leq \theta_y \leq \pi \frac{r}{N}. \tag{6.206}$$

Now, since $A - 1 < \frac{N}{r}$, for these values of θ_y all of the terms in the sum over j in eq. (6.203) lie in the same half-plane, so that the terms interfere constructively and the sum is substantial.

We know that

$$|1 - e^{i\theta}| \leq |\theta|, \tag{6.207}$$

because the straight-line distance from the origin is less than the arc length along the circle, and for $A|\theta| \leq \pi$, we know that

$$|1 - e^{iA\theta}| \geq \frac{2A|\theta|}{\pi}, \tag{6.208}$$

because we can see (either graphically or by evaluating its derivative) that this distance is a convex function. We actually have $A < \frac{N}{r} + 1$, and hence $A\theta_y < \pi\left(1 + \frac{r}{N}\right)$, but by applying the above bound to

$$\left| \frac{e^{i(A-1)\theta} - 1}{e^{i\theta} - 1} + e^{i(A-1)\theta} \right| \geq \left| \frac{e^{i(A-1)\theta} - 1}{e^{i\theta} - 1} \right| - 1, \tag{6.209}$$

we can still conclude that

$$\left| \frac{e^{iA\theta} - 1}{e^{i\theta} - 1} \right| \geq \frac{2(A-1)|\theta|}{\pi|\theta|} - 1 = \frac{2}{\pi} A - \left(1 + \frac{2}{\pi}\right). \tag{6.210}$$

Ignoring a possible correction of order $2/A$, then, we find

$$\text{Prob}(y) \geq \left(\frac{4}{\pi^2}\right) \frac{1}{r}, \tag{6.211}$$

for each of the r values of y that satisfy eq. (6.205). Therefore, with a probability of at least $4/\pi^2$, the measured value of y will satisfy

$$k\frac{N}{r} - \frac{1}{2} \leq y \leq k\frac{N}{r} + \frac{1}{2}, \tag{6.212}$$

or

$$\frac{k}{r} - \frac{1}{2N} \leq \frac{y}{N} \leq \frac{k}{r} + \frac{1}{2N}, \tag{6.213}$$

where k is an integer chosen from $\{0, 1, \ldots, r-1\}$. The output of the computation is reasonable likely to be within distance $1/2$ of an integer multiple of N/r.

Suppose that we know that $r < M \ll N$. Thus N/r is a rational number with a denominator less than M. Two distinct rational numbers, each with denominator less than M, can be no closer together than $1/M^2$, since $\frac{a}{b} - \frac{c}{d} = \frac{ad-bc}{bd}$. If the measurement outcome y satisfies eq. (6.212), then there is a *unique* value of k/r (with $r < M$) determined by y/N, provided that $N \geq M^2$. This value of k/r can be efficiently extracted from the measured y/N, by the continued fraction method.

Now, with probability exceeding $4/\pi^2$, we have found a value of k/r where k is selected (roughly equiprobably) from $\{0, 1, 2, \ldots, r-1\}$. It is reasonably likely that k and r are relatively prime (have no common factor), so that we have succeeded in finding r. With a query of the oracle, we may check whether $f(x) = f(x+r)$. But if $\mathrm{GCD}(k, r) \neq 1$, we have found only a factor (r_1) of r.

If we did not succeed, we could test some nearby values of y (the measured value might have been close to the range $-r/2 \leq yr(\mathrm{mod}\ N) \leq r/2$ without actually lying inside), or we could try a few multiples of r (the value of $\mathrm{GCD}(k, r)$, if not 1, is probably not large). That failing, we resort to a repetition of the quantum circuit, this time (with probability at least $4/\pi^2$) obtaining a value k'/r. Now k', too, may have a common factor with r, in which case our procedure again determines a factor (r_2) of r. But it is reasonably likely that $\mathrm{GCD}(k, k') = 1$, in which case $r = \mathrm{LCM}, (r_1, r_2)$. Indeed, we can estimate the probability that randomly selected k and k' are relatively prime as follows: Since a prime number p divides a fraction $1/p$ of all numbers, the probability that p divides both k and k' is $1/p^2$. And k and k' are coprime if and only if there is no prime p that divides both. Therefore,

$$\mathrm{Prob}(k, k' \text{ coprime}) = \prod_{\text{prime } p} \left(1 - \frac{1}{p^2}\right) = \frac{1}{\zeta(2)} = \frac{6}{\pi^2} \simeq .607 \tag{6.214}$$

(where $\zeta(z)$ denotes the Riemann zeta function). Therefore, we are likely to succeed in finding the period r after some constant number (independent of N) of repetitions of the algorithm.

6.9.2 From FFT to QFT

Now let's consider the implementation of the quantum Fourier transform. The Fourier transform

$$\sum_x f(x)|x\rangle \rightarrow \sum_y \left(\frac{1}{\sqrt{N}} \sum_x e^{2\pi i xy/N} f(x)\right) |y\rangle, \tag{6.215}$$

is multiplication by an $N \times N$ unitary matrix, where the (x, y) matrix element is $(e^{2\pi i/N})^{xy}$. Naively, this transform requires $O(N^2)$ elementary operations. But there is a well-known and very useful (classical) procedure that reduces the number of operations to $O(N \log N)$. Assuming $N = 2^n$, we express x and y as binary expansions

$$\begin{aligned} x &= x_{n-1} \cdot 2^{n-1} + x_{n-2} \cdot 2^{n-2} + \ldots + x_1 \cdot 2 + x_0 \\ y &= y_{n-1} \cdot 2^{n-1} + y_{n-2} \cdot 2^{n-2} + \ldots + y_1 \cdot 2 + y_0. \end{aligned} \tag{6.216}$$

In the product of x and y, we may discard any terms containing n or more powers of 2, as these make no contribution to $e^{2\pi i xy}/2^n$. Hence

$$\begin{aligned} \frac{xy}{2^n} \equiv\ & y_{n-1}(.x_0) + y_{n-2}(.x_1x_0) + y_{n-3}(.x_2x_1x_0) + \ldots \\ & + y_1(.x_{n-2}x_{n-3}\ldots x_0) + y_0(.x_{n-1}x_{n-2}\ldots x_0), \end{aligned} \tag{6.217}$$

where the factors in parentheses are binary expansions; *e.g.*,

$$.x_2x_1x_0 = \frac{x_2}{2} + \frac{x_1}{2^2} + \frac{x_0}{2^3}. \tag{6.218}$$

We can now evaluate

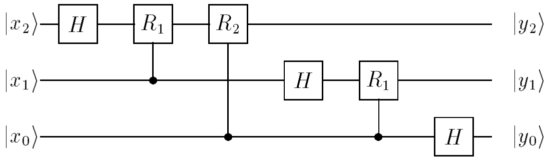
$$\tilde{f}(x) = \frac{1}{\sqrt{N}} \sum_y e^{2\pi i xy/N} f(y), \tag{6.219}$$

for each of the N values of x. But the sum over y factors into n sums over $y_k = 0, 1$, which can be done sequentially in a time of order n.

With quantum parallelism, we can do far better. From eq. (6.217) we obtain

$$\begin{aligned} QFT : |x\rangle &\rightarrow \frac{1}{\sqrt{N}} \sum_y e^{2\pi i xy/N} |y\rangle \\ &= \frac{1}{\sqrt{2^n}} \left(|0\rangle + e^{2\pi i(.x_0)}|1\rangle\right) \left(|0\rangle + e^{2\pi i(.x_1x_0)}|1\rangle\right) \\ &\ldots \left(|0\rangle + e^{2\pi i(.x_{n-1}x_{n-2}\ldots x_0)}|1\rangle\right). \end{aligned} \tag{6.220}$$

The QFT takes each computational basis state to an *unentangled* state of n qubits; thus we anticipate that it can be efficiently implemented. Indeed, let's consider the case $n = 3$. We can readily see that the circuit



does the job (but note that the order of the bits has been reversed in the output). Each Hadamard gate acts as

$$\boldsymbol{H} : |x_k\rangle \to \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i(.x_k)}|1\rangle\right). \tag{6.221}$$

The other contributions to the relative phase of $|0\rangle$ and $|1\rangle$ in the kth qubit are provided by the two-qubit conditional rotations, where

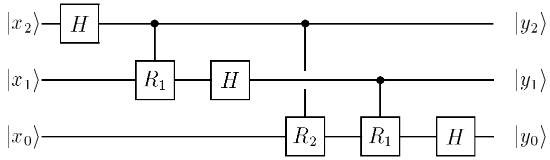
$$\boldsymbol{R}_d = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2^d} \end{pmatrix}, \tag{6.222}$$

and $d = (k - j)$ is the "distance" between the qubits.

In the case $n = 3$, the QFT is constructed from three $\boldsymbol{H}$ gates and three controlled-$\boldsymbol{R}$ gates. For general n, the obvious generalization of this circuit requires n $\boldsymbol{H}$ gates and $\binom{n}{2} = \frac{1}{2}n(n-1)$ controlled R's. A two qubit gate is applied to each pair of qubits, again with controlled relative phase $\pi/2^d$, where d is the "distance" between the qubits. Thus the circuit family that implements QFT has a size of order $(\log N)^2$.

We can reduce the circuit complexity to linear in $\log N$ if we are willing to settle for an implementation of fixed accuracy, because the two-qubit gates acting on distantly separated qubits contribute only exponentially small phases. If we drop the gates acting on pairs with distance greater than m, than each term in eq. (6.217) is replaced by an approximation to m bits of accuracy; the total error in $xy/2^n$ is certainly no worse than $n2^{-m}$, so we can achieve accuracy ε in $xy/2^n$ with $m \geq \log n/\varepsilon$. If we retain only the gates acting on qubit pairs with distance m or less, then the circuit size is $mn \sim n \log n/\varepsilon$.

In fact, if we are going to measure in the computational basis immediately after implementing the QFT (or its inverse), a further simplification is possible – no two-qubit gates are needed at all! We first remark that the controlled – $\boldsymbol{R}_d$ gate acts symmetrically on the two qubits – it acts trivially on $|00\rangle, |01\rangle$, and $|10\rangle$, and modifies the phase of $|11\rangle$ by $e^{i\theta_d}$. Thus, we can interchange the "control" and "target" bits without modifying the gate. With this change, our circuit for the 3-qubit QFT can be redrawn as:



Once we have measured $|y_0\rangle$, we *know* the value of the control bit in the controlled-$\boldsymbol{R}_1$ gate that acted on the first two qubits. Therefore, we will obtain the same probability distribution of measurement outcomes if, instead of applying controlled-$\boldsymbol{R}_1$ and then measuring, we instead measure y_0 first, and then apply $(\boldsymbol{R}_1)^{y_0}$ to the next qubit, conditioned on the outcome of the measurement of the first qubit. Similarly, we can replace the controlled-$\boldsymbol{R}_1$ and controlled-$\boldsymbol{R}_2$ gates acting on the third qubit by the single qubit rotation

$$(\boldsymbol{R}_2)^{y_0}(\boldsymbol{R}_1)^{y_1}, \tag{6.223}$$

(that is, a rotation with relative phase $\pi(.y_1y_0)$) *after* the values of y_1 and y_0 have been measured.

Altogether then, if we are going to measure after performing the QFT, only n Hadamard gates and $n-1$ single-qubit rotations are needed to implement it. The QFT is remarkably simple!

6.10 Factoring

6.10.1 Factoring as period finding

What does the factoring problem (finding the prime factors of a large composite positive integer) have to do with periodicity? There is a well-known (randomized) reduction of factoring to determining the period of a function. Although this reduction is not directly related to quantum computing, we will discuss it here for completeness, and because the prospect of using a quantum computer as a factoring engine has generated so much excitement.

Suppose we want to find a factor of the n-bit number N. Select pseudo-randomly $a < N$, and compute the greatest common divisor GCD(a, N), which can be done efficiently (in a time of order $(\log N)^3$) using the Euclidean algorithm. If GCD$(a, N) \neq 1$ then the GCD is a nontrivial factor of N, and we are done. So suppose GCD$(a, N) = 1$.

[**Aside: The Euclidean algorithm.** To compute GCD(N_1, N_2) (for $N_1 > N_2$) first divide N_1 by N_2 obtaining remainder R_1. Then divide N_2 by R_1, obtaining remainder R_2. Divide R_1 by R_2, *etc.* until the remainder is 0. The last nonzero remainder is $R =$ GCD(N_1, N_2). To see that the algorithm works, just note that (1) R divides all previous remainders and hence also N_1 and N_2, and (2) *any* number that divides N_1 and N_2 will also divide all remainders, including R. A number that divides both N_1 and N_2, and also is divided by any number that divides both N_1 and N_2 must be GCD(N_1, N_2). To see how long the Euclidean algorithm takes, note that

$$R_j = qR_{j+1} + R_{j+2}, \tag{6.224}$$

where $q \geq 1$ and $R_{j+2} < R_{j+1}$; therefore $R_{j+2} < \frac{1}{2}R_j$. Two divisions reduce the remainder by at least a factor of 2, so no more than 2 $\log N_1$ divisions are required, with each division using $O((\log N)^2)$ elementary operations; the total number of operations is $O((\log N)^3)$.]

The numbers $a < N$ coprime to N (having no common factor with N) form a finite group under multiplication mod N. [Why? We need to establish that each element a has an inverse. But for given $a < N$ coprime to N, each $ab \pmod N$ is distinct, as b ranges over all $b < N$ coprime to N.[16] Therefore, for some b, we must have $ab \equiv 1 \pmod N$; hence the inverse of a exists.] Each element a of this finite group has a finite *order* r, the smallest positive integer such that

$$a^r \equiv 1 \pmod N. \tag{6.225}$$

[16]If N divides $ab - ab'$, it must divide $b - b'$.

The order of a mod N is the period of the function

$$f_{N,a}(x) = a^x \pmod N. \tag{6.226}$$

We know there is an efficient quantum algorithm that can find the period of a function; therefore, if we can compute $f_{N,a}$ efficiently, we can find the order of a efficiently.

Computing $f_{N,a}$ may look difficult at first, since the exponent x can be very large. But if $x < 2^m$ and we express x as a binary expansion

$$x = x_{m-1} \cdot 2^{m-1} + x_{m-2} \cdot 2^{m-2} + \ldots + x_0, \tag{6.227}$$

we have

$$a^x (\text{mod } N) = (a^{2^{m-1}})^{x_{m-1}} (a^{2^{m-2}})^{x_{m-2}} \ldots (a)^{x_0} \pmod N. \tag{6.228}$$

Each a^{2^j} has a large exponent, but can be computed efficiently by a *classical* computer, using repeated squaring

$$a^{2^j} \pmod N = (a^{2^{j-1}})^2 \pmod N. \tag{6.229}$$

So only $m - 1$ (classical) mod N multiplications are needed to assemble a table of all a^{2^j}'s.

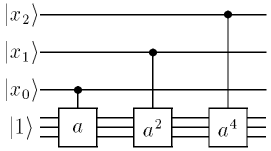
The computation of $a^x (\text{mod } N)$ is carried out by executing a routine:

INPUT 1

For $j = 0$ to $m - 1$, if $x_j = 1$, MULTIPLY a^{2^j}.

This routine requires at most m mod N multiplications, each requiring of order $(\log N)^2$ elementary operations.[17] Since $r < N$, we will have a reasonable chance of success at extracting the period if we choose $m \sim 2 \log N$. Hence, the computation of $f_{N,a}$ can be carried out by a circuit family of size $O((\log N)^3)$. Schematically, the circuit has the structure:

[17]Using tricks for performing efficient multiplication of very large numbers, the number of elementary operations can be reduced to $O(\log N \log\log N \log\log\log N)$; thus, asymptotically for large N, a circuit family with size $O(\log^2 N \log\log N \log\log\log N)$ can compute $f_{N,a}$.



Multiplication by a^{2^j} is performed if the control qubit x_j has the value 1.

Suppose we have found the period r of a mod N. Then *if* r is even, we have

$$N \text{ divides } \left(a^{\frac{r}{2}}+1\right)\left(a^{\frac{r}{2}}-1\right). \tag{6.230}$$

We know that N does not divide $a^{r/2}-1$; if it did, the order of a would be $\leq r/2$. Thus, *if* it is also the case that N does not divide $a^{r/2}+1$, or

$$a^{r/2} \neq -1 \ (\mathrm{mod}\ N), \tag{6.231}$$

then N must have a nontrivial common factor with each of $a^{r/2}\pm 1$. Therefore, $\mathrm{GCD}(N, a^{r/2}+1) \neq 1$ is a factor (that we can find efficiently by a classical computation), and we are done.

We see that, once we have found r, we succeed in factoring N *unless* either (1) r is odd or (2) r is even and $a^{r/2} \equiv -1$ (mod N). How likely is success?

Let's suppose that N is a product of two prime factors $p_1 \neq p_2$,

$$N = p_1 p_2 \tag{6.232}$$

(this is actually the least favorable case). For each $a < p_1p_2$, there exist unique $a_1 < p_1$ and $a_2 < p_2$ such that

$$\begin{aligned} a &\equiv a_1 \quad (\mathrm{mod}\ p_1) \\ a &\equiv a_2 \quad (\mathrm{mod}\ p_2). \end{aligned} \tag{6.233}$$

Choosing a random $a < N$ is, therefore, equivalent to choosing random $a, < p_1$ and $a_2 < p_2$.

[**Aside**: We're using the **Chinese Remainder Theorem.** The a solving eq. (6.233) is unique because if a and b are both solutions, then both p_1 and p_2 must divide $a-b$. The solution exists because every $a < p_1p_2$ solves eq. (6.233) for *some* a_1 and a_2. Since there are exactly p_1p_2 ways to choose a_1 and a_2, and exactly p_1p_2 ways to choose a, uniqueness implies that there is an a corresponding to each pair a_1, a_2.]

Now let r_1 denote the order of a_1 mod p_1 and r_2 denote the order of a_2 mod p_2. The Chinese remainder theorem tells us that $a^r \equiv 1$ (mod p_1p_2) is equivalent to

$$\begin{aligned} a_1^r &\equiv 1 \ (\mathrm{mod}\ p_1) \\ a_2^r &\equiv 1 \ (\mathrm{mod}\ p_2). \end{aligned} \tag{6.234}$$

Therefore $r = \mathrm{LCM}(r_1, r_2)$. If r_1 and r_2 are both odd, then so is r, and we lose.

But if *either* r_1 or r_2 is even, then so is r, and we are still in the game. If

$$\begin{aligned} a^{r/2} &\equiv -1 \quad (\mathrm{mod}\ p_1) \\ a^{r/2} &\equiv -1 \quad (\mathrm{mod}\ p_2). \end{aligned} \tag{6.235}$$

Then we have $a^{r/2} \equiv -1$ (mod p_1p_2) and we still lose. But if either

$$\begin{aligned} a^{r/2} &\equiv -1 \quad (\mathrm{mod}\ p_1) \\ a^{r/2} &\equiv 1 \quad (\mathrm{mod}\ p_2), \end{aligned} \tag{6.236}$$

or

$$\begin{aligned} a^{r/2} &\equiv 1 \quad (\mathrm{mod}\ p_1) \\ a^{r/2} &\equiv -1 \quad (\mathrm{mod}\ p_2), \end{aligned} \tag{6.237}$$

then $a^{r/2} \not\equiv -1$(mod p_1p_2) and we win. (Of course, $a^{r/2} \equiv 1$ (mod p_1) and $a^{r/2} \equiv 1$ (mod p_2) is not possible, for that would imply $a^{r/2} \equiv 1$ (mod p_1p_2), and r could not be the order of a.)

Suppose that

$$\begin{aligned} r_1 &= 2^{c_1} \cdot \mathrm{odd} \\ r_2 &= 2^{c_2} \cdot \mathrm{odd}, \end{aligned} \tag{6.238}$$

where $c_1 > c_2$. Then $r = \mathrm{LCM}(r_1, r_2) = 2r_2\cdot$ integer, so that $a^{r/2} \equiv 1$ (mod p_2) and eq. (6.236) is satisfied – we win! Similarly $c_2 > c_1$ implies eq. (6.237) – again we win. But for $c_1 = c_2$, $r = r_1 \cdot (\mathrm{odd}) = r_2 \cdot (\mathrm{odd}')$ so that eq. (6.235) is satisfied – in that case we lose.

Okay – it comes down to: for $c_1 = c_2$ we lose, for $c_1 \neq c_2$ we win. How likely is $c_1 \neq c_2$?

It helps to know that the multiplicative group mod p is cyclic – it contains a primitive element of order $p-1$, so that all elements are powers of the primitive element. [Why? The integers mod p are a finite *field*. If the group were not cyclic, the maximum order of the elements would be $q < p-1$, so that $x^q \equiv 1 \pmod p$ would have $p-1$ solutions. But that can't be: in a finite field there are no more than q qth roots of unity.]

Suppose that $p - 1 = 2^k \cdot s$, where s is odd, and consider the orders of all the elements of the cyclic group of order $p-1$. For brevity, we'll discuss only the case $k = 1$, which is the least favorable case for us. Then if b is a primitive (order $2s$) element, the even powers of b have odd order, and the odd powers of b have order $2\cdot$ (odd). In this case, then, $r = 2^c\cdot$ (odd) where $c \in \{0, 1\}$, each occuring equiprobably. Therefore, if p_1 and p_2 are both of this (unfavorable) type, and a_1, a_2 are chosen randomly, the probability that $c_1 \neq c_2$ is $\frac{1}{2}$. Hence, once we have found r, our probability of successfully finding a factor is at least $\frac{1}{2}$, if N is a product of two distinct primes. If N has more than two distinct prime factors, our odds are even better. The method fails if N is a prime power, $N = p^\alpha$, but prime powers can be efficiently factored by other methods.

6.10.2 RSA

Does anyone care whether factoring is easy or hard? Well, yes, some people do.

The presumed difficulty of factoring is the basis of the security of the widely used RSA[18] scheme for public key cryptography, which you may have used yourself if you have ever sent your credit card number over the internet.

The idea behind public key cryptography is to avoid the need to exchange a secret key (which might be intercepted and copied) between the parties that want to communicate. The enciphering key is public knowledge. But using the enciphering key to infer the deciphering key involves a prohibitively difficult computation. Therefore, Bob can send the enciphering key to Alice and everyone else, but only Bob will be able to decode the message that Alice (or anyone else) encodes using the key. Encoding is a "one-way function" that is easy to compute but very hard to invert.

[18] For Rivest, Shamir, and Adleman

(Of course, Alice and Bob could have avoided the need to exchange the public key if they had decided on a private key in their previous clandestine meeting. For example, they could have agreed to use a long random string as a one-time pad for encoding and decoding. But perhaps Alice and Bob never anticipated that they would someday need to communicate privately. Or perhaps they did agree in advance to use a one-time pad, but they have now used up their private key, and they are loath to reuse it for fear that an eavesdropper might then be able to break their code. Now they are two far apart to safely exchange a new private key; public key cryptography appears to be their most secure option.)

To construct the public key Bob chooses two large prime numbers p and q. But he does not publicly reveal their values. Instead he computes the product

$$N = pq. \tag{6.239}$$

Since Bob knows the prime factorization of N, he also knows the value of the Euler function $\varphi(N)$ – the number of number less than N that are coprime with N. In the case of a product of two primes it is

$$\varphi(N) = N - p - q + 1 = (p-1)(q-1), \tag{6.240}$$

(only multiples of p and q share a factor with N). It is easy to find $\varphi(N)$ if you know the prime factorization of N, but it is hard if you know only N.

Bob then pseudo-randomly selects $e < \varphi(N)$ that is coprime with $\varphi(N)$. He reveals to Alice (and anyone else who is listening) the value of N and e, but nothing else.

Alice converts her message to ASCII, a number $a < N$. She encodes the message by computing

$$b = f(a) = a^e (\text{mod } N), \tag{6.241}$$

which she can do quickly by repeated squaring. How does Bob decode the message?

Suppose that a is coprime to N (which is overwhelmingly likely if p and q are very large – anyway Alice can check before she encodes). Then

$$a^{\varphi(N)} \equiv 1 \pmod N \tag{6.242}$$

(Euler's theorem). This is so because the numbers less than N and coprime to N form a group (of order $\varphi(N)$) under mod N multiplication. The order of any group element must divide the order of the group (the powers of a form a subgroup). Since GCD$(e, \varphi(N) = 1$, we know that e has a multiplicative inverse $d = e^{-1} \bmod \varphi(N)$:

$$ed \equiv 1 \pmod{\varphi(N)}. \tag{6.243}$$

The value of d is Bob's closely guarded secret; he uses it to decode by computing:

$$\begin{aligned} f^{-1}(b) &= b^d \pmod N \\ &= a^{ed} \pmod N \\ &= a \cdot (a^{\varphi(N)})^{\text{integer}} \pmod N \\ &= a \pmod N). \end{aligned} \tag{6.244}$$

[**Aside:** How does Bob compute $d = e^{-1}$? The multiplicative inverse is a byproduct of carrying out the Euclidean algorithm to compute GCD$(e, \varphi(N)) = 1$. Tracing the chain of remainders from the bottom up, starting with $R_n = 1$:

$$\begin{aligned} 1 = R_n &= R_{n-2} - q_{n-1}R_{n-1} \\ R_{n-1} &= R_{n-3} - q_{n-2}R_{n-2} \\ R_{n-2} &= R_{n-4} - q_{n-3}R_{n-3} \\ &etc.\ldots \end{aligned} \tag{6.245}$$

(where the q_j's are the quotients), so that

$$\begin{aligned} 1 &= (1 + q_{n-1}q_{n-2})R_{n-2} - q_{n-1}R_{n-3} \\ 1 &= (-q_{n-1} - q_{n-3}(1 + q_{n-1}q_{n-2}))R_{n-3} \\ &\quad + (1 + q_{n-1}q_{n-2})R_{n-4}, \\ &etc.\ldots . \end{aligned} \tag{6.246}$$

Continuing, we can express 1 as a linear combination of any two successive remainders; eventually we work our way up to

$$1 = d \cdot e + q \cdot \varphi(N), \tag{6.247}$$

and identify d as $e^{-1} \pmod{\varphi(N)}$.]

Of course, if Eve has a superfast factoring engine, the RSA scheme is insecure. She factors N, finds $\varphi(N)$, and quickly computes d. In fact, she does not really need to factor N; it is sufficient to compute the order modulo N of the encoded message $a^e \pmod N$. Since e is coprime with $\varphi(N)$, the order of $a^e \pmod N$ is the same as the order of a (both elements generate the same *orbit*, or cyclic subgroup). Once the order Ord(a) is known, Eve computes $\tilde{d}$ such that

$$\tilde{d}e \equiv 1 \pmod{\mathrm{Ord}(a)} \tag{6.248}$$

so that

$$(a^e)^{\tilde{d}} \equiv a \cdot (a^{\mathrm{Ord}(a)})^{\text{integer}} \pmod N \equiv a \pmod N, \tag{6.249}$$

and Eve can decipher the message. If our only concern is to defeat RSA, we run the Shor algorithm to find $r = \mathrm{Ord}(a^e)$, and we needn't worry about whether we can use r to extract a factor of N or not.

How important are such prospective cryptographic applications of quantum computing? When fast quantum computers are readily available, concerned parties can stop using RSA, or can use longer keys to stay a step ahead of contemporary technology. However, people with secrets sometimes want their messages to remain confidential for a while (30 years?). They may not be satisfied by longer keys if they are not confident about the pace of future technological advances.

And if they shun RSA, what will they use instead? Not so many suitable one-way functions are known, and others besides RSA are (or may be) vulnerable to a quantum attack. So there really is a lot at stake. If fast large scale quantum computers become available, the cryptographic implications may be far reaching.

But while quantum theory taketh away, quantum theory also giveth; quantum computers may compromise public key schemes, but also offer an alternative: secure quantum key distribution, as discussed in Chapter 4.

6.11 Phase Estimation

There is an alternative way to view the factoring algorithm (due to Kitaev) that deepens our insight into how it works: we can factor because we can measure efficiently and accurately the eigenvalue of a certain unitary operator.

Consider $a < N$ coprime to N, let x take values in $\{0, 1, 2, \ldots, N-1\}$, and let $\boldsymbol{U}_a$ denote the unitary operator

$$\boldsymbol{U}_a : |x\rangle \to |ax \ (\mathrm{mod}\ N)\rangle. \tag{6.250}$$

This operator is unitary (a permutation of the computational basis) because multiplication by a mod N is invertible.

If the order of a mod N is r, then

$$\boldsymbol{U}_a^r = \mathit{1}. \tag{6.251}$$

It follows that all eigenvalues of $\boldsymbol{U}_a$ are rth roots of unity:

$$\lambda_k = e^{2\pi i k/r}, \quad k \in \{0, 1, 2, \ldots, r-1\}. \tag{6.252}$$

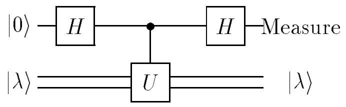
The corresponding eigenstates are

$$|\lambda_k\rangle = \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} e^{-2\pi i k j/r} |a^j x_0 (\mathrm{mod}\ N)\rangle; \tag{6.253}$$

associated with each orbit of length r generated by multiplication by a, there are r mutually orthogonal eigenstates.

$\boldsymbol{U}_a$ is not hermitian, but its *phase* (the Hermitian operator that generates $\boldsymbol{U}_a$) is an observable quantity. Suppose that we can perform a measurement that projects onto the basis of $\boldsymbol{U}_a$ eigenstates, and determines a value λ_k selected equiprobably from the possible eigenvalues. Hence the measurement determines a value of k/r, as does Shor's procedure, and we can proceed to factor N with a reasonably high success probability. But how do we measure the eigenvalues of a unitary operator?

Suppose that we can execute the unitary $\boldsymbol{U}$ conditioned on a control bit, and consider the circuit:



Here $|\lambda\rangle$ denotes an eigenstate of $\boldsymbol{U}$ with eigenvalue λ ($\boldsymbol{U}|\lambda\rangle = \lambda|\lambda\rangle$). Then the action of the circuit on the control bit is

$$\begin{aligned} |0\rangle &\to \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \to \frac{1}{\sqrt{2}}(|0\rangle + \lambda|1\rangle) \\ &\to \frac{1}{2}(1+\lambda)|0\rangle + \frac{1}{2}(1-\lambda)|1\rangle. \end{aligned} \tag{6.254}$$

Then the outcome of the measurement of the control qubit has probability distribution

$$\begin{aligned} \mathrm{Prob}(0) &= \left|\frac{1}{2}(1+\lambda)\right|^2 = \cos^2(\pi\phi) \\ \mathrm{Prob}(1) &= \left|\frac{1}{2}(1-\lambda)\right)|^2 = \sin^2(\pi\phi), \end{aligned} \tag{6.255}$$

where $\lambda = e^{2\pi i \phi}$.

As we have discussed previously (for example in connection with Deutsch's problem), this procedure distinguishes with certainty between the eigenvalues $\lambda = 1$ ($\phi = 0$) and $\lambda = -1$ ($\phi = 1/2$). But other possible values of λ can also be distinguished, albeit with less statistical confidence. For example, suppose the state on which $\boldsymbol{U}$ acts is a superposition of $\boldsymbol{U}$ eigenstates

$$\alpha_1|\lambda_1\rangle + \alpha_2|\lambda_2\rangle. \tag{6.256}$$

And suppose we execute the above circuit n times, with n distinct control bits. We thus prepare the state

$$\begin{aligned} &\alpha_1|\lambda_1\rangle \left(\frac{1+\lambda_1}{2}|0\rangle + \frac{1-\lambda_1}{2}|1\rangle\right)^{\otimes n} \\ +&\alpha_2|\lambda_2\rangle \left(\frac{1+\lambda_2}{2}|0\rangle + \frac{1-\lambda_2}{2}|1\rangle\right)^{\otimes n}. \end{aligned} \tag{6.257}$$

If $\lambda_1 \neq \lambda_2$, the overlap between the two states of the n control bits is exponentially small for large n; by measuring the control bits, we can perform the orthogonal projection onto the $\{|\lambda_1\rangle, |\lambda_2\rangle\}$ basis, at least to an excellent approximation.

If we use enough control bits, we have a large enough sample to measure Prob $(0) = \frac{1}{2}(1 + \cos 2\pi\phi)$ with reasonable statistical confidence. By executing a controlled-($i\boldsymbol{U}$), we can also measure $\frac{1}{2}(1 + \sin 2\pi\phi)$ which suffices to determine ϕ modulo an integer.

However, in the factoring algorithm, we need to measure the phase of $e^{2\pi ik/r}$ to exponential accuracy, which seems to require an exponential number of trials. Suppose, though, that we can efficiently compute high powers of $\boldsymbol{U}$ (as is the case for $\boldsymbol{U}_a$) such as

$$\boldsymbol{U}^{2^j}. \tag{6.258}$$

By applying the above procedure to measurement of $\boldsymbol{U}^{2^j}$, we determine

$$\exp(2\pi i 2^j \phi), \tag{6.259}$$

where $e^{2\pi i\phi}$ is an eigenvalue of $\boldsymbol{U}$. Hence, measuring $\boldsymbol{U}^{2^j}$ to one bit of accuracy is equivalent to measuring the jth bit of the eigenvalue of $\boldsymbol{U}$.

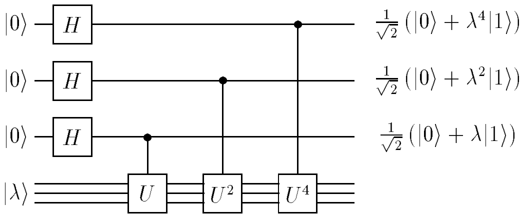
We can use this phase estimation procedure for order finding, and hence factorization. We invert eq. (6.253) to obtain

$$|x_0\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\lambda_k\rangle; \tag{6.260}$$

each computational basis state (for $x_0 \neq 0$) is an equally weighted superposition of r eigenstates of $\boldsymbol{U}_a$.

Measuring the eigenvalue, we obtain $\lambda_k = e^{2\pi ik/r}$, with k selected from $\{0, 1 \ldots, r-1\}$ equiprobably. If $r < 2^n$, we measure to $2n$ bits of precision to determine k/r. In principle, we can carry out this procedure in a computer that stores fewer qubits than we would need to evaluate the QFT, because we can attack just one bit of k/r at a time.

But it is instructive to imagine that we incorporate the QFT into this phase estimation procedure. Suppose the circuit



acts on the eigenstate $|\lambda\rangle$ of the unitary transformation $\boldsymbol{U}$. The conditional $\boldsymbol{U}$ prepares $\frac{1}{\sqrt{2}}(|0\rangle + \lambda|1\rangle)$, the conditional $\boldsymbol{U}^2$ prepares $\frac{1}{\sqrt{2}}(|0\rangle + \lambda^2|1\rangle)$, the conditional $\boldsymbol{U}^4$ prepares $\frac{1}{\sqrt{2}}(|0\rangle + \lambda^4|1\rangle)$, and so on. We could perform a Hadamard and measure each of these qubits to sample the probability distribution governed by the jth bit of ϕ, where $\lambda = e^{2\pi i\phi}$. But a more efficient method is to note that the state prepared by the circuit is

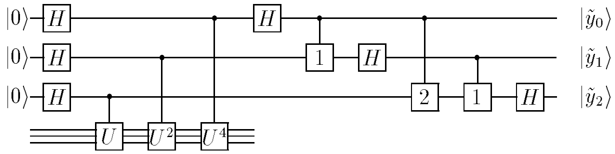
$$\frac{1}{\sqrt{2^m}} \sum_{y=0}^{2^m-1} e^{2\pi i\phi y}|y\rangle. \tag{6.261}$$

A better way to learn the value of ϕ is to perform the QFT$^{(m)}$, not the Hadamard $\boldsymbol{H}^{(m)}$, before we measure.

Considering the case $m = 3$ for clarity, the circuit that prepares and then Fourier analyzes the state

$$\frac{1}{\sqrt{8}} \sum_{y=0}^{7} e^{2\pi i\phi y}|y\rangle \tag{6.262}$$

is



This circuit very nearly carries out our strategy for phase estimation outlined above, but with a significant modification. Before we execute the final Hadamard transformation and measurement of $\tilde{y}_1$ and $\tilde{y}_2$, some conditional phase rotations are performed. It is those phase rotations that distinguish the QFT$^{(3)}$ from Hadamard transform $\boldsymbol{H}^{(3)}$, and they strongly enhance the reliability with which we can extract the value of ϕ.

We can understand better what the conditional rotations are doing if we suppose that $\phi = k/8$, for $k \in \{0, 1, 2 \ldots, 7\}$; in that case, we know that the Fourier transform will generate the output $\tilde{y} = k$ with probability one. We may express k as the binary expansion

$$k = k_2 k_1 k_0 \equiv k_2 \cdot 4 + k_1 \cdot 2 + k_0. \tag{6.263}$$

In fact, the circuit for the least significant bit $\tilde{y}_0$ of the Fourier transform is precisely Kitaev's measurement circuit applied to the unitary $\boldsymbol{U}^4$, whose eigenvalue is

$$(e^{2\pi i\phi})^4 = e^{i\pi k} = e^{i\pi k_0} = \pm 1. \tag{6.264}$$

The measurement circuit distinguishes eigenvalues ± 1 perfectly, so that $\tilde{y}_0 = k_0$.

The circuit for the next bit $\tilde{y}_1$ is almost the measurement circuit for $\boldsymbol{U}^2$, with eigenvalue

$$(e^{2\pi i\phi})^2 = e^{i\pi k/2} = e^{i\pi(k_1 \cdot k_0)}. \tag{6.265}$$

Except that the conditional phase rotation has been inserted, which multiplies the phase by $\exp[i\pi(\cdot k_0)]$, resulting in $e^{i\pi k_1}$. Again, applying a Hadamard followed by measurement, we obtain the outcome $\tilde{y}_1 = k_1$ with certainty. Similarly, the circuit for $\tilde{y}_2$ measures the eigenvalue

$$e^{2\pi i\phi} = e^{i\pi k/4} = e^{i\pi(k_2 \cdot k_1 k_0)}, \tag{6.266}$$

except that the conditional rotation removes $e^{i\pi(\cdot k_1 k_0)}$, so that the outcome is $\tilde{y}_2 = k_2$ with certainty.

Thus, the QFT implements the phase estimation routine with maximal cleverness. We measure the less significant bits of ϕ first, and we exploit the information gained in the measurements to improve the reliability of our estimate of the more significant bits. Keeping this interpretation in mind, you will find it easy to remember the circuit for the $\mathrm{QFT}^{(n)}$!

6.12 Discrete Log

Sorry, I didn't have time for this.

6.13 Simulation of Quantum Systems

Ditto.

6.14 Summary

Classical circuits. The complexity of a problem can be characterized by the size of a uniform family of logic circuits that solve the problem: The problem is hard if the size of the circuit is a superpolynomial function of the size of the input. One classical universal computer can simulate another efficiently, so the classification of complexity is machine independent. The 3-bit Toffoli gate is universal for classical reversible computation. A reversible computer can simulate an irreversible computer without a significant slowdown and without unreasonable memory resources.

Quantum Circuits. Although there is no proof, it seems likely that polynomial-size quantum circuits cannot be simulated by polynomial-size probabilistic classical circuits ($BQP \neq BPP$); however, polynomial space is sufficient ($BQP \subseteq PSPACE$). A noisy quantum circuit can simulate an ideal quantum circuit of size T to acceptable accuracy if each quantum gate has an accuracy of order $1/T$. One universal quantum computer can simulate another efficiently, so that the complexity class BQP is machine independent. A generic two-qubit quantum gate, if it can act on any two qubits in a device, is adequate for universal quantum computation. A controlled-NOT gate plus a generic one-qubit gate is also adequate.

Fast Quantum Searching. Exhaustive search for a marked item in an unsorted database of N items can be carried out by a quantum computer in a time of order $\sqrt{N}$, but no faster. Quadratic quantum speedups can be achieved for some structured search problems, too, but some oracle problems admit no significant quantum speedup. Two parties, each in possession of a table with N entries, can locate a "collision" between their tables by exchanging $O(\sqrt{N})$ qubits, in apparent violation of the spirit (but not the letter) of the Holevo bound.

Period Finding. Exploiting quantum parallelism, the Quantum Fourier Transform in an N-dimensional space can be computed in time of order $(\log N)^2$ (compared to time $N \log N$ for the classical fast Fourier transform); if we are to measure immediately afterward, one qubit gates are sufficient to compute the QFT. Thus quantum computers can efficiently solve certain problems with a periodic structure, such as factoring and the discrete log problem.

6.15 Exercises

6.1 Linear simulation of Toffoli gate.

In class we constructed the n-bit Toffoli gate $(\theta^{(n)})$ from 3-bit Toffoli gates $(\theta^{(3)}$'s). The circuit required only one bit of scratch space, but the number of gates was exponential in n. With more scratch, we can substantially reduce the number of gates.

a) Find a circuit family with $2n-5$ $\theta^{(3)}$'s that evaluates $\theta^{(n)}$. (Here $n-3$ scratch bits are used, which are set to 0 at the beginning of the computation and return to the value 0 at the end.)

b) Find a circuit family with $4n-12$ $\theta^{(3)}$'s that evaluates $\theta^{(n)}$, which works irrespective of the initial values of the scratch bits. (Again the $n-3$ scratch bits return to their initial values, but they don't need to be set to zero at the beginning.)

6.2 A universal quantum gate set.

The purpose of this exercise is to complete the demonstration that the controlled-NOT and arbitrary one-qubit gates constitute a universal set.

a) If $\boldsymbol{U}$ is any unitary 2×2 matrix with determinant one, find unitary $\boldsymbol{A}, \boldsymbol{B}$, and $\boldsymbol{C}$ such that

$$\boldsymbol{ABC} = \mathit{1} \tag{6.267}$$

$$\boldsymbol{A}\boldsymbol{\sigma}_x\boldsymbol{B}\boldsymbol{\sigma}_x\boldsymbol{C} = \boldsymbol{U}. \tag{6.268}$$

Hint: From the Euler angle construction, we know that

$$\boldsymbol{U} = \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\phi), \tag{6.269}$$

where, *e.g.*, $\boldsymbol{R}_z(\phi)$ denotes a rotation about the z-axis by the angle ϕ. We also know that, *e.g.*,

$$\boldsymbol{\sigma}_x\boldsymbol{R}_z(\phi)\boldsymbol{\sigma}_x = \boldsymbol{R}_z(-\phi). \tag{6.270}$$

b) Consider a two-qubit *controlled phase gate*: it applies $\boldsymbol{U} = e^{i\alpha}\mathit{1}$ to the second qubit if the first qubit has value $|1\rangle$, and acts trivially otherwise. Show that it is actually a one-qubit gate.

c) Draw a circuit using controlled-NOT gates and single-qubit gates that implements controlled-$\boldsymbol{U}$, where $\boldsymbol{U}$ is an arbitrary 2×2 unitary transformation.

6.3 Precision.

The purpose of this exercise is to connect the accuracy of a quantum state with the accuracy of the corresponding probability distribution.

a) Let $\parallel \boldsymbol{A} \parallel_{\sup}$ denote the sup norm of the operator $\boldsymbol{A}$, and let

$$\parallel \boldsymbol{A} \parallel_{\mathrm{tr}} = \mathrm{tr}\left[(\boldsymbol{A}^\dagger\boldsymbol{A})^{1/2}\right], \tag{6.271}$$

denote its *trace norm*. Show that

$$\parallel \boldsymbol{AB} \parallel_{\mathrm{tr}} \le \parallel \boldsymbol{B} \parallel_{\sup} \cdot \parallel \boldsymbol{A} \parallel_{\mathrm{tr}} \ \text{ and } \ |\ \mathrm{tr}\, \boldsymbol{A}\ | \ \le \parallel \boldsymbol{A} \parallel_{\mathrm{tr}} \ . \tag{6.272}$$

b) Suppose $\boldsymbol{\rho}$ and $\tilde{\boldsymbol{\rho}}$ are two density matrices, and $\{|a\rangle\}$ is a complete orthonormal basis, so that

$$P_a = \langle a|\boldsymbol{\rho}|a\rangle,$$

$$\tilde{P}_a = \langle a|\tilde{\boldsymbol{\rho}}|a\rangle, \tag{6.273}$$

are the corresponding probability distributions. Use (a) to show that

$$\sum_a |P_a - \tilde{P}_a| \ \le \ \parallel \boldsymbol{\rho} - \tilde{\boldsymbol{\rho}} \parallel_{\mathrm{tr}} . \tag{6.274}$$

c) Suppose that $\boldsymbol{\rho} = |\psi\rangle\langle\psi|$ and $\tilde{\boldsymbol{\rho}} = |\tilde{\psi}\rangle\langle\tilde{\psi}|$ are pure states. Use (b) to show that

$$\sum_a |P_a - \tilde{P}_a| \ \le \ 2 \parallel |\psi\rangle - |\tilde{\psi}\rangle \parallel . \tag{6.275}$$

6.4 Continuous-time database search

A quantum system with an n-qubit Hilbert space has the Hamiltonian

$$\boldsymbol{H}_\omega = E|\omega\rangle\langle\omega|, \tag{6.276}$$

where $|\omega\rangle$ is an unknown computational-basis state. You are to find the value of ω by the following procedure. Turn on a time-independent perturbation $\boldsymbol{H}'$ of the Hamiltonian, so that the total Hamiltonian becomes

$$\boldsymbol{H} = \boldsymbol{H}_\omega + \boldsymbol{H}'. \tag{6.277}$$

Prepare an initial state $|\psi_0\rangle$, and allow the state to evolve, as governed by $\boldsymbol{H}$, for a time T. Then measure the state. From the measurement result you are to infer ω.

a) Suppose the initial state is chosen to be

$$|s\rangle = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle, \tag{6.278}$$

and the perturbation is

$$\boldsymbol{H}' = E|s\rangle\langle s|. \tag{6.279}$$

Solve the time-independent Schrödinger equation

$$i\frac{d}{dt}|\psi\rangle = \boldsymbol{H}|\psi\rangle \tag{6.280}$$

to find the state at time T. How should T be chosen to optimize the likelihood of successfully determining ω?

b) Now suppose that we may choose $|\psi_0\rangle$ and $\boldsymbol{H}'$ however we please, but we demand that the state of the system after time T is $|\omega\rangle$, so that the measurement determines ω with success probability one. Derive a lower bound that T must satisfy, and compare to your result in (**a**). (**Hint:** As in our analysis in class, compare evolution governed by $\boldsymbol{H}$ with evolution governed by $\boldsymbol{H}'$ (the case of the "empty oracle"), and use the Schrödinger equation to bound how rapidly the state evolving according to $\boldsymbol{H}$ deviates from the state evolving according to $\boldsymbol{H}'$.)