Computing and Quantum Computing

Malik Magdon-Ismail

June 17, 2025

Contents

1 Outline and Introduction to Classical Computing

The first 6 chapters are covered in detail in Magdon-Ismail (2020, Chapters 22-29).

1.1 Outline

1. Theory of classical computing. Why study it? Quantum computing is not a sprint, it's a marathon. At least you will have something golden to put into the bank even if QC does not materialize in your lifetime. Also, to study QC and its potential benefits, we need a baseline.

2. Why study QC if we don't have Q-computers? Historicaly, algorithms precede the machines that implement them. The Babylonians knew how to multiply, but we only had the calculator 4,000 years later. Turing developed the theory of classical computing and the Universal Turing Machine (UTM) in the 1930s, but we only had programmable computers 30 years laters.

   Let us study QC-algorithms now and hope we do not have to wait 4,000 years for the first viable QC. There are many challenges, both algorithmically in terms of quantum algorithm design, and practically in terms of stable physical components to implement these algorithms.

3. What is QC?

4. QC Formalism.

5. QC algorithms and ~~building~~ simulating our first QC.

6. What does it take to get a viable QC. The fundamental premise of classical computing is that you can write a bit to a register and it is "permanent." An algorithm can remember and refer to things as needed. Well, the classical computing bit is not quite permanent. It can get flipped, for example by ambient gamma and other EM rays with probability about 10^{-12} per hour. We have error correcting codes (ECC) to fight this. For example, one can store the bit 0 as 000. Now two bits need to get flipped before it is not recognizable. Stability at the expense of more memory. Hence, we pay big bucks for ECC memory.

   A QC must be able to store the equivalent of a bit "permanently". That is already a challenge. On top of that, we need good quantum-ECC, another challenge.

7. If we do get Q-computers, what will become a sprint is to develop good quantum programming languages and quantum algorithms. A programming language is just an interface between your mind, where you build algorithms, and the hardware. A compiler is the interpreter.

   What does it take to have a quantum-programming language. For starters, we need a good formalism within which to design quantum algorithms, which must also allow mixing of classical and quantum algorithms.

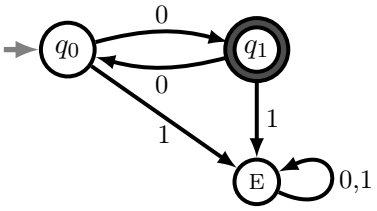
1.2 What is classical computing?

- What is computing?
- Computing uses algorithms to solve computing problems.
- What is an algorithm? What is a computing problem?

A computing problem is a language. Consider deciding if a number is odd. Define the language

$$\mathcal{L} = \{0, 000, 00000, 0000000\} = \{0^{\bullet 2k-1} \mid k \in \mathbb{N}\}. \tag{1}$$

The alphabet we will use is binary. Any string can be converted to binary. Testing if a string w is in $\mathcal{L}$ amounts to "computing oddness." A DFA can solve this problem.



states	
$\rightarrow q_0$	NO
q_1	YES
E	NO

transitions			
1 :	q_0	0	q_1
2 :	q_0	1	E
3 :	q_1	0	q_0
4 :	q_1	1	E
5 :	E	0	E
6 :	E	1	E

(2)

The notion of an algorithm is embedded implicitly in the DFA which is used to solve the problem. Every DFA is an "algorithm." The DFA is easy to build using standard physical components. For example a vending machine.

Formulating a computing problem as a language has withstood the test of time. Here are some example problems. Prime factorization, defined by the language

$$\mathcal{L} = \{(N, K) \mid N \text{ has a prime factor } K\}. \tag{3}$$

Finding the shortest path between two vertices in a graph (formulate this as a language). Determining equality, defined by the language.

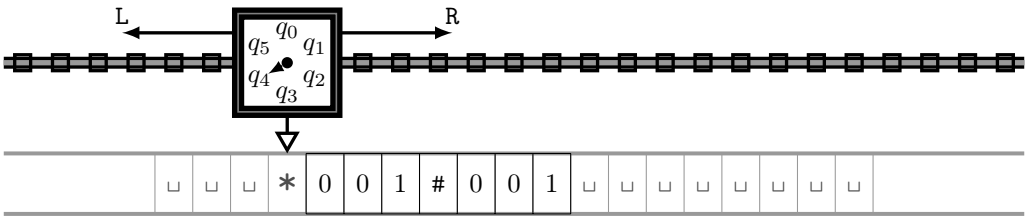
$$\mathcal{L} = \{0^{\bullet n} 1^{\bullet n} \mid n \geq 0\} \tag{4}$$

There is no DFA to solve the equality problem.

Proof. By contradiction. Assume some DFA with k states can solve this problem. Show there are two strings which end in the same state, but one is in $\mathcal{L}$ and one is not. This gives the desired contradiction, because that state cannot be both accepting and rejecting. ∎

This is a strong theorem. It is not about DFA-programming skills. It is not just you and I that cannot find the DFA for equality. No one can. Such a DFA provably does not exist. So what now? Since we would like to solve equality, we need a more powerful machine.

The fundamental problem with the DFA is that it can't remember how many 0s have passed. It only has a finite number of internal states which can function as a primitive but bounded counter. We need an external scratch paper. So we add an unbounded RAM (tape) and allow the DFA to move around this RAM, read from it and write to it. This computer is a Turing Machine.



(5)

This machine is easy to implement modulo the infinite tape. But in practice we can implement it and run it on any specific input string. If the machine comes to the end of the tape and needs more tape, we just fetch it more tape (memory on demand). The Turing Machine is the definition of an algorithm. This is the Church Turing thesis.

This notion of algorithms which need to dynamically create more memory because they do not know how much memory they will need for their specific input instance are common in computing. For example dynamic hashing for efficient search. We want to hash with no collisions, so the table size depends on the number of items hashed. If ahead of time you don't know how many items will be hashed, you increase the table size (add memory) on an as needed basis. This type of algorithm is very important to Google.

When you have a theory of computing based on some computing machine, it is very important to make sure you can build it. Otherwise what use is the computer if it can't be built?

Let's say you have designed a fancy TM, but the engineer who will build it lives on the other side of the country. What do you do? Well you write down the description of your TM and email it to them. What does this description look like?

$$\text{states, transition/move/read/write instructions, halting states, etc..} \tag{6}$$

The engineer gets this description, reads it and builds the machine. Your description is just some big string of characters (for example in ASCII) which we can convert to binary. Let M be your machine and $\langle M \rangle$ this binary string which describes M.

$$M \to (\langle M \rangle, w) \stackrel{\text{email}}{\longrightarrow} \boxed{\text{engineer} \stackrel{\text{build}}{\longrightarrow} M \to \text{run on input } w} \to \text{YES/NO}. \tag{7}$$

The boxed process performed by the engineer is quite complex. Do you think we could build a Turing machine to implement it instead of using an engineer? It would have to work no matter what its input $(\langle M \rangle, w)$ was. This was one of the stunning results from Turing's seminal work. It is possible to build a single Turing machine which does the engineer's job. Well not quite. This TM does not have the ability to build and run M. It simulates the building and running of M on its tape. This grand TM is called a *Universal Turing Machine* (UTM). It is what we call a stored program computer today. The description $\langle M \rangle$ is the program fed into the computer and the string w is the input for the program M. The input to the UTM is $(\langle M \rangle, w)$.

1.3 Unsolvable Computing Problems

The requirement that we are able to build the computing machine that implements an algorithm which solves a computing problem is necessary. Otherwise computing is useless. This requirement is also very limiting. Let us see a remarkable consequence.

If it is buildable, you have to be able to describe it to your engineer. In which case you have to be able to write down a description, which is essentially some massive binary string. So every computing machine has a binary description. This means every computing problem that can be solved by a computing machine has an associated binary description. Further, two different computing problems cannot be solved by the same computing machine.

$$\textsf{solvable computing problem} \leftrightarrow \textsf{computing machine } M \textsf{ for problem} \leftrightarrow \textsf{description } \langle M \rangle \textsf{ of } M. \tag{8}$$

Mathematically, there is an injection from solvable problems to finite binary strings (the description of the TM that solves the problem).

$$|\{\textsf{solvable computing problems}\}| \leq |\{\textsf{all finite binary strings}\}| = \text{countable.} \tag{9}$$

Here is a lexicographic ordering of all finite binary strings,

$$B = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, \ldots\} \tag{10}$$

Consider any infinite binary string, for example,

$$10011101\cdots \tag{11}$$

Define a computing problem $\mathcal{L}$ corresponding to this infinite binary string as follows. Align each bit with the corresponding position in B with 1 meaning the string in B is in $\mathcal{L}$ and 0 meaning the string in B is not in $\mathcal{L}$. Our infinite binary string defines the language

$$\mathcal{L} = \{\varepsilon, 00, 01, 10, 000, \ldots\} \tag{12}$$

Every infinite binary string defines a different problem. Hence,

$$|\{\textsf{all computing problems}\}| = |\{\textsf{all infinite binary strings}\}| = \text{uncountable.} \tag{13}$$

The conclusion is that there are uncountably many computing problems that do not have a computing machine that solves them. The only requirement here is that a computing machine should be buildable, that is describable.

Theorem 1.1. There are countably many computing machines and there are uncountably many computing problems that cannot be solved by a computing machine.

There are many unsolvable computing problems. To catch and put some of them on display, we need to delve deeper into Turing Machines. Before we do, let's see how QC will change things.

1.4 What Changes When We Move to Quantum Computing

The problems we want to solve don't change. So a computing problem is still a language, and there are uncountably many of those.

We still want to build quantum computers to solve problems, so a quantum computer must be describable. But then Theorem 1.1 applies and there are uncountably unsolvable computing problems, even on a quantum computer. Indeed, a quantum computer can be simulated (very slowly) using a classical computer. So any problem that cannot be solved by a classical computer also cannot be solved by a quantum computer.

So, what do we gain? What changes? Efficiency. We hope that problems that can't be solved efficiently on classical computers can be solved efficiently on quantum computers.

2 Turing Machines and Unsolvable Problems

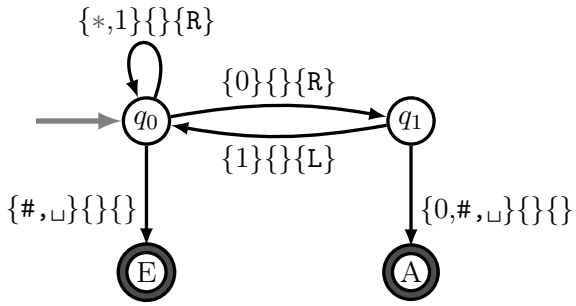
Let us summarize. Any theory of computing machines must involve computers that are buildable. It does not matter whether we are talking about DFA, classical computers, biological computers, molecular computers or quantum computers. Hence a computer M must be describable, that is there is a binary encoding (description) $\langle M \rangle$ that can be used by an engineer to build M. This means the computing machines are countable and can be listed,

$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle M_5 \rangle, \langle M_6 \rangle, \ldots \tag{14}$$

A computing problem is specified by an infinite binary string. There are uncountably many computing problems. This means there are uncoutably many problems which are not solvable by our conputing machines. For this conclusion, we only require that a computing machine be describable.

2.1 Turing Machines

Consider the following TM run on input 101. Each edge (transition instruction) is labeled with three entries, what the TM reads on the tape, what it writes and how it moves. It is instructive to run the TM on its tape.



This TM infinite loops on input 101 by jiggling left and right. A TM's ability to move is the source of its power to solve complex problems. Ironically it is this same ability to move at will that leads to this possibility of infinite loops. A TM M run on input w can do one of three things.

$$M(w) = \begin{cases} \text{HALT with YES;} \\ \text{HALT with NO;} \\ \text{infinite loop.} \end{cases} \tag{15}$$

If M always halts, M is a decider. If M may infinite loop, it is a recognizer. The possibility of an infinite loop is unacceptable in practice. Hence, a solvable problem (language) is one which can be solved by a decider. A problem $\mathcal{L}$ is recognizable if there exists M for which

$$\begin{aligned} w \in \mathcal{L} &\rightarrow M(w) = \text{HALT with YES} \\ w \notin \mathcal{L} &\rightarrow M(w) = \text{HALT with NO or infinite loop} \end{aligned} \tag{16}$$

That is, there is a recognizer which "solves" the problem. However the possiblity of an infinite loop is not a practical solution. A problem $\mathcal{L}$ is decidable if there exists M for which

$$\begin{aligned} w \in \mathcal{L} &\rightarrow M(w) = \text{HALT with YES} \\ w \notin \mathcal{L} &\rightarrow M(w) = \text{HALT with NO} \end{aligned} \tag{17}$$

That is, there is a decider which solves the problem. Note, every decider is a recognizer, so

$$\{\text{deciders}\} \subset \{\text{recognizers}\}. \tag{18}$$

Also, every decidable language is also recognizable. Since both deciders and recognizers are describable, they are both countable. Hence there are uncountably many undecidable problems and uncountably many unrecognizable problems.

In practice, we do not build a new TM for every problem we want to solve. We just build the universal TM U_{TM} which can simulate the operation of *any* M on *any* input w,

$$U_{\text{TM}}(\langle M\rangle \# w) = \begin{cases} \text{HALT with YES}, & \text{if } M(w) = \text{HALT with YES}; \\ \text{HALT with NO}, & \text{if } M(w) = \text{HALT with NO}; \\ \text{infinite loop}, & \text{if } M(w) = \text{infinite loop}. \end{cases} \tag{19}$$

2.2 Program Testing is Unsolvable

It is not possible to write a program which can test if another program works correctly on a particular input. Define the set of programs with their accepting inputs $\mathcal{L}_{\text{TM}}$,

$$\mathcal{L}_{\text{TM}} = \{\langle M\rangle \# w \mid M \text{ is a TM and } M(w) = \text{HALT and YES}\}. \tag{20}$$

$\mathcal{L}_{\text{TM}}$ consists of programs and their inputs which terminate succesfully. Is there a decider for $\mathcal{L}_{\text{TM}}$, that is, can we solve the problem $\mathcal{L}_{\text{TM}}$. That means, given an input program $\langle M\rangle$ and its input w, you must determine if M halts on w with YES or not. Isn't this problem trivially solvable:

$$\text{Build } M \text{ and run it on } w, \text{ outputing the result.} \tag{21}$$

Actually, we don't build and run M, we just simulate running M on w using U_{TM}. Not so fast. The the catch is that M may infinite loop. In this case our solution of just running M will never end and we will never output NO. What next? Since we can just build and run M, we need to do something more sophisticated, that looks more deeply at the inner workings of M.

Can we build another program A_{TM} that takes as input $\langle M\rangle \# w$ and halts with YES if M halts with YES on w, and halts with NO otherwise,

$$A_{\text{TM}}(\langle M\rangle \# w) = \begin{cases} \text{halt with YES}, & \text{if } M(w) = \text{halt with YES}; \\ \text{halt with NO}, & \text{otherwise}. \end{cases} \tag{22}$$

A_{TM} is a decider for $\mathcal{L}_{\text{TM}}$. Unfortunately, we can't build A_{TM} because A_{TM} doesn't exist. There is no program which can test another program's correctness. It just does not exist. Let's say it again, A_{TM} *does not exist*!

Proof. (Contradiction) Assume the program A_{TM} exists. We can use A_{TM} anyway we want. In particular, we can use it as a subroutine in any other program. Define the program D as follows. D takes as input just the encoding $\langle M\rangle$ of a TM.

$D(\langle M\rangle)$:

1: Run $A_{\text{TM}}(\langle M\rangle \# \langle M\rangle) \to$ YES/NO.
2: If YES, output NO. If NO, output YES.

The input to A_{TM} is two strings, it can be any two strings. Hence, the first step is a valid input to A_{TM}, and since A_{TM} is a decider, D is also a decider. This is because the first step must halt, and the second step is trivial. Recall that Turing Machine encodings are finite binary strings, hence they can be listed.

$$\text{List of Turing Machines:} \qquad \langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \ldots, \langle D\rangle, \ldots$$

Notice that our diabolical diagonal decider D is on this list. When A_{TM} runs on $\langle M_i\rangle\#\langle M_j\rangle$, the result is a table like:

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\cdots$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$	ACCEPT	ACCEPT	REJECT	$\cdots$	ACCEPT	$\cdots$
$\langle M_2\rangle$	REJECT	REJECT	REJECT	$\cdots$	ACCEPT	$\cdots$
$\langle M_3\rangle$	ACCEPT	ACCEPT	REJECT	$\cdots$	ACCEPT	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\cdots$
$\langle D\rangle$	REJECT	ACCEPT	ACCEPT	$\cdots$	ACCEPT / REJECT ?	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\ddots$

For example, A_{TM} rejects $\langle M_2\rangle\#\langle M_3\rangle$. This means M_2 rejects or loops forever on the input $\langle M_3\rangle$. The diagonal entries are what happens when M_i runs on its own description $\langle M_i\rangle$. The row for $\langle D\rangle$ is the opposite of the underlined diagonal entries in the table, because D rejects $\langle M_i\rangle$ if and only if M_i accepts $\langle M_i\rangle$. This is reminiscent of Cantor diagonalization to show reals can't be listed. This diagonalization shows that D can't exist. The problem is the collision at the diagonal entry in the row for $\langle D\rangle$. This entry is the opposite of itself!

Formally, what does D say when its input is $\langle D\rangle$ (it must halt and say something). If it says YES, then $A_{\text{TM}}(\langle D\rangle\#\langle D\rangle) = \text{YES}$ by definition of A_{TM}, which means D, by definition, must say the opposite, which is NO, a contradiction. If it says NO, then $A_{\text{TM}}(\langle D\rangle\#\langle D\rangle) = \text{NO}$ by definition of A_{TM}, which means D, by definition, must say the opposite, which is YES, a contradiction. Either possibility leads to a contradiction. Hence A_{TM} cannot exist. ∎

2.3 The Halting Problem is Unsolvable

Now the floodgate is open. We can use the non-existence of A_{TM} to prove non-existence of other programs using the general methodology of reduction. Consider a program $\langle M\rangle$ to decide a language $\mathcal{L}$. If M can be used as a subroutine to build A_{TM}, then since A_{TM} does not exist, M cannot exist. This would prove that $\mathcal{L}$ is undecidable. Let us see how this method of reduction works in a concrete case. Consider the halting problem,

$$\mathcal{L}_{\text{HALT}} = \{\langle M\rangle\#w \mid M \text{ is a TM and } M \text{ halts on } w\}. \tag{23}$$

Note, the difference between $\mathcal{L}_{\text{TM}}$ and $\mathcal{L}_{\text{HALT}}$. To belong in $\mathcal{L}_{\text{HALT}}$, M just needs to halt on w, it does not need to say yes. A program that tests if another program halts is super powerful. For example, it could be used to resolve many conjectures in mathematics like Goldbach's conjecture or the twin-primes conjecture. Just write a program to verify if some property holds for all integers

by testing the property one by one. If the property fails for any integer the program should halt and report the counterexample. Now testing if this program halts or not will resolve the conjecture that the property holds for all integers.

If we have a program H_{TM} that solves $\mathcal{L}_{\text{HALT}}$, then we can build A_{TM} as follows.

$A_{\text{TM}}(\langle M \rangle \# w)$:

1: Run $H_{\text{TM}}(\langle M \rangle \# w)$. If NO, halt with NO. If YES, run step 2.
2: Build and run M on w, outputing the result.

The A_{TM} we constructed always halts and always outputs the correct answer. The build and run method works here because we only use it when we know that M halts. The conclusion is that H_{TM} cannot exist and so $\mathcal{L}_{\text{HALT}}$ is undecidable. Many problems are unsolvable. Uncountably many. The domino puzzle, Post's Correspondence Problem (PCP), starts with a list of dominos. An instance of the domino puzzle (PCP) is given by these three dominos.

$$\begin{array}{ccc} d_1 & d_2 & d_3 \\ \begin{array}{|c|}\hline 0 \\ \hline 100 \\ \hline \end{array} & \begin{array}{|c|}\hline 01 \\ \hline 00 \\ \hline \end{array} & \begin{array}{|c|}\hline 110 \\ \hline 11 \\ \hline \end{array} \end{array} \tag{24}$$

A sequence of dominoes produces a combined domino in which the top string is the concatenation of all the top strings in order, and similarly for the bottom string. For example,

$$d_3 d_1 d_3 \;=\; \begin{array}{|c|c|c|}\hline 110 & 0 & 110 \\ \hline 11 & 100 & 11 \\ \hline \end{array}, \quad \text{which gives the combined domino} \quad \begin{array}{|c|}\hline 1100110 \\ \hline 1110011 \\ \hline \end{array}.$$

Is there a sequence of these dominos (repetition allowed) with the concatenation of the to strings matching the concatenation of the bottom strings. In this case, you can verify that $d_3 d_2 d_3 d_1$ solves the puzzle. The domino puzzle (PCP), treated as a computing problem where the input is an arbitrary set of dominos is unsolvable.

Halting and programing testing are very important in computer science. Anytime someone comes up with a program to solve a problem, it is important to show that the program always halts and always gives the correct answer. We have just showed that A_{TM} and H_{TM} do not exist. So what is going on in industry. Are programmers routinely putting out code that is incorrect and untested and/or does not always terminate?

The undecidability of $\mathcal{L}_{\text{TM}}$ and $\mathcal{L}_{\text{HALT}}$ are in the general sense. There is no *general* program that can test *any* other program for halting or correctness. However, through ingenuity and proof methods, it may be possible to test a *specific* program for correctness.

Also note, a general autograder for CS1-assignments is not possible. Yet, CS1 assignments are autograded. In what sense are CS1 assignments autogradeable? General antivirus programs are not possible. So what exactly are you paying for in antivirus software?

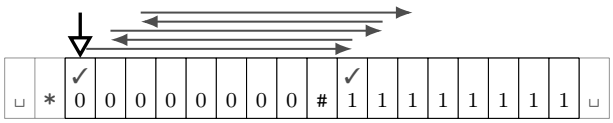
3 Efficiency: The Class P

To summarize the Church-Turing thesis, an algorithm is a Turing-Machine. A problem is solvable by an algorithm if there is a TM-decider for the corresponding language.

We talk about efficiency in the context of a specific problem.

$$\mathcal{L} = \{0^{\bullet n}\#1^{\bullet n}, n \geq 0\}. \tag{25}$$

The simplest TM for this problem zig-zags checking off each 0 with its corresponding 1,



M = Turing Machine that solves $\{0^{\bullet k}\#1^{\bullet k}\}$

INPUT: Binary string w.

1: Check that the input has the correct format and return to $*$.
2: Match each 0 (left of #) to a 1 (right of #):
  Move right and mark the first unmarked 0 (if none, GOTO step 3).
  Move right and mark the first unmarked 1 (if none, REJECT).
  Move left until you come to a marked 0.
3: If there are any unmarked 1's, REJECT. Otherwise ACCEPT.

What is the runtime of this TM. This question is ambiguous. On what input? In computer science we identify the "size" of the input by a parameter n, and we want the runtime for the worst input of size n. Further, we are only interested in $n \to \infty$, that is asymptotic analysis. This algorithm makes makes n scans, and each scan takes about n steps, so the TM has $\Theta(n^2)$ runtime. This is the time-complexity of the algorithm/TM.

What if we are interested in the time-complexity of the problem itself, that is the time-complexity of $\mathcal{L}$? This quantifies how hard a problem is $\mathcal{L}$, given it is solvable. For our language $\mathcal{L}$ above, there is another TM based on halving the number of unmarked bits in each scan. The number of scans required is $O(\log n)$, and each scan is n operations, so this is a TM with $O(n \log n)$ runtime. If we are talking about the time-complexity of a problem, we must consider the time-complexity of the best TM for the problem. In this case, you cannot beat $n \log n$.

Let us think outside the box and envision a TM with two tapes and two independent read-write heads. The input is on one tape. In one scan you can copy the 0s to the second tape and then mark-off the 1s on the first tape with the 0s on the second tape. So now we have a $O(n)$ runtime, but with a different TM-architecture. This new architecture is a parallel architecture. It requires us to be able to operate two TM's independently in parallel. So, by moving to a new architecture, we can improve the runtime even more, by a logarithmic factor. So, it looks like the time-complexity of a problem depends on the TM architecture.

Things are now getting complicated. Should we use the best possible architecture? Should we fix the architecture to 1-tape. But in practice, we do use parallelism. The solution we will take is to define a time-complexity by considering broad classes. This has withstood both the test of time and the test of practice.

A TM is fast if when you double the input size, the runtime increases by at most some constant factor λ. Let $f(n)$ be an upper bound on the worst case runtime on an input of size n. We say $f(n)$

is fast if

$$f(2n) \leq \lambda f(n). \tag{26}$$

Which of these are fast:

$$\log n \qquad \sqrt{n} \qquad n \qquad n^2 \log n \qquad (\log n)^{\log n} \qquad n^{\log n} \qquad 2^{\sqrt{n}} \qquad 2^n? \tag{27}$$

Theorem 3.1. $f(n)$ is fast if and only if $f(n)$ is polynomial.

We care about fast algorithms, that is polynomial algorithms. In the end, practice does care about the difference between $O(n^2)$ versus $O(n \log n)$ versus $O(n)$. But there is a sharp theoretical divide between polynomial runtimes and non-polynomial, i.e. exponential. In practice polynomial TMs are tolerable, exponential ones are intolerable. In theory also, polynomial is a property of a problem, not a particular TM-architecture. This is the extended Church-Turing thesis: A problem which is polynomial on a single tape deterministic TM is polynomial on any reasonable TM-architecture.

Let us justify polynomial as a well-defined architecture independent notion of time-complexity that is to be sought by algorithms/TMs. To do this, we prove the simulation theorem. We show how to simulate our two-tape TM for $\mathcal{L}$ using one tape. The two tapes are interleaved to form one tape. The heads of the two tapes are marked on the single tape. When the two-tape TM performs some operation using a particular head, the one-tape simulator first finds the mark for the head, and only then can it perform the operation on the relevant tape. If the two tape TM runs in time T, the search for a head's tape takes time $O(T)$. This means the runtime for the one tape simulator is $O(T^2)$, because we need to find one of the heads at most T times. We therefore have the simulation theorem,

Theorem 3.2. A K-tape TM with runtime $t(n)$ can be simulated by a 1-tape TM with runtime $O(t(n)^2)$.

So the speedup in going from sequential to parallel is at most square-root. This means if you are polynomial in a parallel architecture, you are at most polynomial-squared on a sequential architecture. If a problem is polynomial on one architecture, it is polynomial on any architecture, ..., except we are not sure about non-deterministic TM-architectures and Quantum-architectures.

Parallel architectures cannot give you exponential speed up, but could these other two architectures do that? We don't know. Nevertheless, we will see some polynomial speedups for some surprising problems by going to quantum architectures.

So in our theoretical classification, a problem is either polynomial, in the class P, or exponential. We know many problems that provably do not have polynomial solutions. Testing if a program stops in fewer than 2^n steps on an input of size n is one such problem. Other problems are solving chess, and in general games of strategy. Many useful problems are polynomial.

There are some super-useful problems in practice: clique, 3-SAT, TSP, ... We don't know if these problems are polynomial or exponential. They are on the boundary. These problems are the keys to defining the beautiful theory of NP-completeness. We discuss that next.

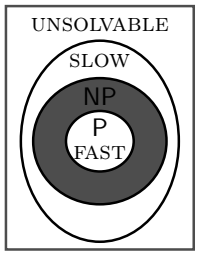
4 Solvable versus Verifiable

In class, we talked about the frequent basket problem that your manager might ask you to solve (see also Magdon-Ismail (2020)). Here is some customer-data (1 indicates a purchase).

		beer	chips	milk	diapers	cheese
Alice	(A)	1	1	0	1	0
Bob	(B)	1	1	0	1	1
Charles	(C)	0	0	1	1	0
David	(D)	1	1	1	1	1

Three people {A, B, D} purchased the basket {beer, chips, diapers}. Your boss asks for a basket with 10 or more items purchased by at least $5,000$ customers. For a big input, finding the solution won't be easy. Instead of telling your manager you couldn't solve it, you can tell your manager that no one can solve it efficiently today. The theory of NP-completeness allows you to make this claim.

Every solvable problem can either be solved by a fast TM or not. If not, every TM for the problem runs in super-polynomial time, i.e. exponential time. So every problem is either fast or slow, where the speed of a problem refers to the speed with which it can be solved. We do not know if the frequent basket problem is fast or slow. It lies in a hazy boundary between fast and slow consisting of all the problems which we cannot definitively place into one or other category.



Here is another example. Consider two sets

$$S_1 = \{3, 5, 3, 11, 6, 2\} \qquad S_2 = \{3, 6, 2, 11, 6, 2\} \tag{28}$$

and the following questions.

LARGE-SUM Is there a small subset (at most size K) whose sum is at least half the total?
PARTITION Is there any subset whose sum is exactly half the total? (29)

There are many efficient algorithms to solve the first question. In the general case where the input set has n elements, an $n \log n$ solution first sorts the elements and checks the sum of the largest K. An $n \log K$ solution uses a minheap to keep track of the top-K elements and checks their sum. A student even suggested an $O(n)$ algorithm using the linear-time K-selection algorithm, and then processing the top K elements larger than that element.

The second problem looks very similar. Instead of asking for a small subset with a large sum, it asks for any subset with an exact sum. Can you partition the set into two equal halves? Well, this small change makes the problem very hard, and this second problem is very similar to the frequent baskets problem. We can't solve it fast and we don't know if it can be solved fast. This problem is also in the hazy region between fast and slow. We are going to build a theory for exactly those problems, because they occur so often in practice and they are all related. We won't cover techniques to solve those problems (take the course in approximation algorithms for that).

Let's focus on partition. Check that PARTITION(S_1) = (NO) and PARTITION(S_2) = (YES). So imagine your manager gave you S_1 and S_2 and asked you to solve partition. You would answer (NO) and (YES) respectively. But your manager wasn't born yesterday. You are asked to convince them. They want a proof that your answer is correct. They want a proof that is easy for them to check.

How would you prove that your (NO)-answer for S_1 is correct? There is no smart way other than a brute-force check that every subset fails.

How would you prove that your (YES)-answer for S_2 is correct? That's easy, just present the subset $\{3, 6, 6\}$ or the string 110010 which identifies that subset with its 1s. Given this subset, it is easy for your manager to verify the yes answer is correct. This subset 110010 is called the evidence or *certificate*. Without this evidence, it is hard for your manager to verify the (YES) answer without actually solving the problem. With the evidence, it is trivial to verify the yes answer.

4.1 Polynomialy Verifiable and NP

The class of problems NP stands for nondeterministic polynomial. It is also equal to the class of problems whose (YES) answer is polynomialy veriviable. PARTITION is a problem whose (YES)-answer is polynomialy verifiable. Let us formally define this.

Consider a problem $\mathcal{L}$. The problem $\mathcal{L}$ is polynomialy verifiable if there exists a polynomial certifier C for $\mathcal{L}$. The certifier C is a TM that verifies if a (YES) answer is correct. Specifically, consider any $w \in \mathcal{L}$. Then there must exist some evidence E so that $C(w\#\text{E}) =$ (YES). The runtime of the certifier must be bounded by a polynomial in the length of w.

In our partition example, $w = S_2$. The evidence is E = 110010. And the certifier is a TM whose input is w and E, summarised in the algorithm:

1: Compute sum(w).
2: Compute the sum of w masked by E, that is $\sum_i w[i] \times \text{E}[i]$.
3: **if** $2 \times$ masked-sum = sum **then**
4: return (YES)
5: **else**
6: return (NO)

It is clear that this verifier works for a general instance of PARTITION.

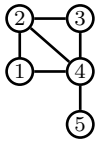
Definition 4.1 (Polynomialy verifiable)**.** A language $\mathcal{L}$ is polynomialy verifiable if there exists a TM C such that:

(i) For every $w \in \mathcal{L}$, there exists an evidence E with $|\text{E}| \leq \text{poly}(|w|)$ such that $C(w\#\text{E})$ has runtime at most a polynomial in $|w|$ and returns (YES).

(ii) For every $w \notin \mathcal{L}$ and every E with $|\text{E}| \leq \text{poly}(|w|)$, $C(w\#\text{E})$ has runtime at most a polynomial in $|w|$ and returns (NO).

Notice the asymmetry between (YES) inputs and (NO) inputs. Also notice that the evidence is given, and we do not have to worry about how the evidence was obtained. That is, the runtime/effort needed to find the correct evidence is not part of the runtime of the certifier. Let us look at another example, the clique problem.

CLIQUE: Given a graph G and $k > 1$, is there a clique of size at least k?



(30)

A graph can be represented as a binary string using the edge sequence

$$e_{1,2}e_{1,3}e_{1,4}e_{1,5}e_{2,3}e_{2,4}e_{2,5}e_{3,4}e_{3,5}e_{4,5}. \tag{31}$$

For our graph, the edge sequence is 1010110101. Note, the number of vertices can be inferred from the number of edges. The CLIQUE problem asks if the graph has a subset of vertices of a particular size such that every pair of vertices in the subset are neighbors. For example size 3. In this case the answer is (YES). What evidence would allow us to quickly verify this answer. The clique itself, which could be represented by the binary string 11010, where the 1s in the string identify the vertices in the clique.

In our case the input is 1010110101#111. The second string, 111 is unary for 3, the size of the desired clique. The evidence is 11010. The certifier C(1010110101#111#11010) is summarized in the algorithm:

1: Compute sum(E) and verify that it is at least K, the desired clique size.
2: Identify the clique from E.
3: **for** each pair of vertices in the clique **do**
4: check that the edge for that pair in the edge-sequence is 1.

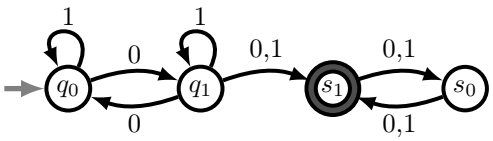
This algorithm clearly generalizes and runs in polynomial time in the input size. One can get a linear algorithm for the verifier.

Many popular problems are polynomialy verifiable: TSP, Clique, 3-SAT, Knapsack, Independent Set, Vertex Cover, Dominating Set, ... It means we can quickly verify a (YES) instance given the right evidence. Yet, we don't have polynomial algorithms for any of these problems. Every problem in P that is polynomialy solvable is also polynomialy verifiable.

4.2 Nondeterministic Polynomial (NP) and Polynomialy Verifiable

When we study DFA, we encounter nondeterministic finite automata (NFA) as a useful tool to address concatenation and Kleen star.

(32)



When you run the NFA on 100101, the computation branches and we accept if any one of the branches accepts. Using subset states, we can implement an NFA by a DFA with a possible exponential increase in the number of states. So nondeterminism does not add any additional computing capability to DFA. It is just a useful tool.

We can extend nondeterminism to Turing Machines.

(33)

A nondeterministic TM, at each step, can try out different possibilities. Again this results in a branching of the computation, and the TM accepts if any branch accepts. The runtime is the runtime

of the longest branch. We imagine running all branches simultaneously, and so a nondeterministic TM provides some kind of unbounded parallelism. A nondeterministic TM can be implemented using a deterministic TM, but with an exponential increase in runtime.

The class NP are the problems that can be solved in polynomial time on a nondeterministic TM. It turns out that a computing problem that is polynomialy verifiable can be solved in polynomial time on a nondeterministic TM. Here is the intuition in the context of our clique-problem. We give a nondeterministic TM M which uses our certifier as a subroutine in a nondeterministic way. The input graph plus K is 1010110101#111.

$$10110110101\#111 \to M \begin{cases} \text{try } \text{E} = 00000 & \to \text{run } C(10110110101\#111\#00000) \to \boxed{\text{YES}} \text{ or } \boxed{\text{NO}} \\ \text{try } \text{E} = 00001 & \to \text{run } C(10110110101\#111\#00001) \to \boxed{\text{YES}} \text{ or } \boxed{\text{NO}} \\ \text{try } \text{E} = 00010 & \to \text{run } C(10110110101\#111\#00010) \to \boxed{\text{YES}} \text{ or } \boxed{\text{NO}} \\ \text{try } \text{E} = 00011 & \to \text{run } C(10110110101\#111\#00011) \to \boxed{\text{YES}} \text{ or } \boxed{\text{NO}} \\ \text{try } \text{E} = 00100 & \to \text{run } C(10110110101\#111\#00100) \to \boxed{\text{YES}} \text{ or } \boxed{\text{NO}} \\ \vdots & \\ \text{try } \text{E} = 11111 & \to \text{run } C(10110110101\#111\#11111) \to \boxed{\text{YES}} \text{ or } \boxed{\text{NO}} \end{cases} \tag{34}$$

The nondeterminism is only in trying out different choices for E which causes the TM to branch (exponential number of branches). Each branch runs in polynomial time because the verifier C is polynomial. Hence this is a polynomial TM. If any branch accepts, it means there is a 3-clique. This is guaranteed by the properties of the certifier - if there is no 3-clique then the certifier says no for all possible evidence.

It is a little trickier to show that if a problem is in NP then the problem is polynomialy verifiable. The evidence is the sequence of choices made by the nondeterministic TM on any YES-branch. The certifier simply runs the nondeterministic TM using only these choices to verify the YES. Since the nondeterministic TM is polynomial, the certifier is also polynomial.

As you can see, the nondeterministic TM for CLIQUE above has unbounded parallelism. To make this computation naively sequential would require running each branch in sequence and since there are exponentially many branches, this is an exponential slowdown. Could there be some other way to make this computation sequential that does not result in an exponential slowdown? That is THE BURNING QUESTION in computer science. Does P = NP?

We are asking if polynomialy verifiable problems are also polynomialy solvable. The smart money says no. Intuitively how could you possibly condense exponentially many branches to a polynomial runtime. Alternatively, polynomial verifiability is easy given the evidence, which is the solution itself. So just because one can quickly verify a solution, it seems like a big leap to think that one can quickly find the solution. Nevertheless, we have no proof either way.

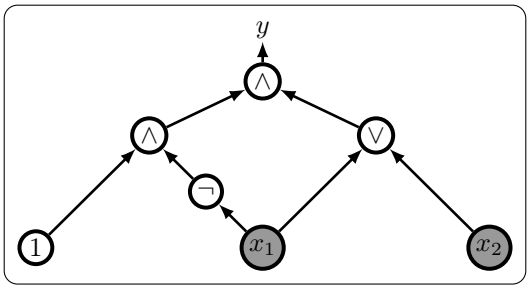
Our next task is to unravel some of the beautiful structure surrounding the polynomialy verifiable problems. In particular to identify a hardest such problem, whatever that means.

5 A Hardest Problem in NP: CIRCUIT-SAT

The last lecture discussed polynomialy solvable (P) versus polynomialy verifiable (NP). There is a hardest problem in NP, so hard that if you can solve it in polynomial time, you can use the solver as a subroutine to solve any NP-problem in polynomial time. Such hardest problems NP-complete.

5.1 Boolean Circuits

A Boolean circuit has vertices which can be logic gates ($\wedge, \neg, \vee$) or inputs.



(35)

Some inputs are hard-coded, and others user specified. The output of this circuit is

$$y = (1 \wedge \overline{x_1}) \wedge (x_1 \vee x_2). \tag{36}$$

$x_1x_2 = 01$ is a satisfying assignment of this circuit, which means $y = 1$. In general, a circuit is a DAG where the vertices with in-degree 0 are the inputs. The output vertex has out-degree 0. All other vertices are logic gates which have inputs from other vertices and outputs to other vertices.

CIRCUIT-SAT: Given a circuit (DAG) is there a satisfying assignment for inputs $x_1, \ldots, x_n$?

CIRCUIT-SAT is in NP since a satisfying assignment can be verified by just running the circuit (linear time traversal). The evidence is just the satisfying assignment.

An exponential algorithm to solve CIRCUIT-SAT is to try all 2^n assignments. No polynomial algorithm is known. It is conjectured that there is no algorithm substantially faster than 2^n.

5.2 Cook-Levin Theorem

CIRCUIT-SAT is harder than all NP-problems. This means that every other problem in NP is *polynomialy reducible* to CIRCUIT-SAT. Specifically, if an alien gave you a black box that solves CIRCUIT-SAT in polynomial time, you can literally use this black-box to solve any NP-problem in polynomial time. This is a deep result, and proving it has two main challenges. (i) The proof must work for any NP problem, *without knowing any specific details of the problem.* (ii) We don't know any details of how the black box works, yet we must show that this specific black box can be used to solve any NP problem. The proof will rely on a link between fast TMs and small circuits.

5.2.1 Fast TMs and Small Circuits

Consider any fast TM. It can operate on any w, returning TM(w) in poly($|w|$)-time. The nice thing about a TM is that it can take any-length input w. In contrast, a circuit has a fixed hard-coded

input length. So, let us consider a TM with a fixed input w of length n and assume it runs in time $t(n) \leq \text{poly}(n)$. Each step of the TM is something like

$$\text{if in state } q \text{ at slot } i \text{ reading 0: write 1, move L and transition to state } s \tag{37}$$

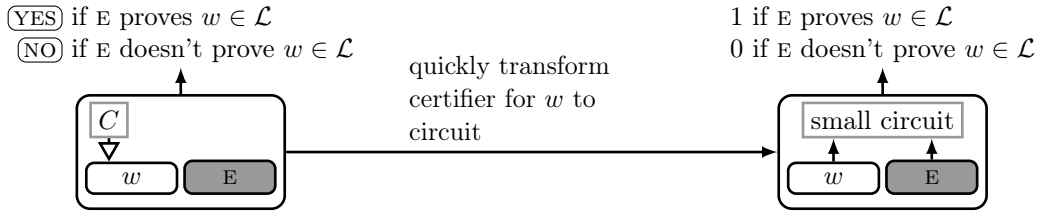
After this instruction, the TM head is at slot $i-1$. The configuration of the TM, a binary string, encodes where the head is, the state of the system and what is on the tape. The instruction takes the system from one configuration to another, that is it takes the binary string for the initial configuration to a binary string for the next configuration. This transformation of the binary string can be implemented by a small circuit (see also Magdon-Ismail (2020, Problems 29.25, 29.26)). Since the TM is fast, only a polynomial number of tape-slots are examined, so there are only polynomially many choices for i. Thus, a polynomial number of circuits suffice to implement one step of the TM computation. All these small circuits operate in parallel in a "layer" of the circuits. Since the TM is fast, to represent all steps of the computation, we require at most a polynomial number of layers. Hence the entire circuit that implements the TM on w has a polynomial number of gates. This final mega-circuit is constructed in time $\text{poly}(t(n))$ and has size $\text{poly}(t(n))$, where $t(n) \leq \text{poly}(|w|)$.

Conclusion. A fast TM on input w can be quickly converted to a small, $\text{poly}(|w|)$-sized, circuit.

We did not prove this theorem. We only gave the hand-wavy idea for the proof. The interested reader can fill in the details. But, the conclusion is intuitively plausible. A fast TM on an input of size n has an equivalent small, $\text{poly}(n)$, circuit which can be constructed quickly, in $\text{poly}(n)$-time.

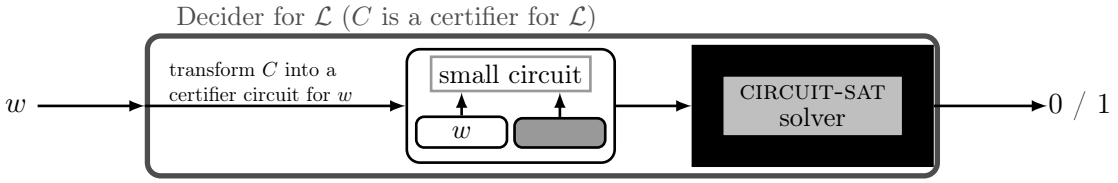
5.2.2 Proof of the Cook-Levin Theorem

Consider any NP-problem $\mathcal{L}$. It has a fast certifier C such that $C(w, \text{E})$ is (YES) if E verifies $w \in \mathcal{L}$ and (NO) otherwise. Since C is fast, it can be quickly converted into a small circuit with input w, E.



(38)

Treating w as hard-coded and E as the "input", we can now build a fast decider for $\mathcal{L}$.



(39)

1: Given input w, transform C into a certifier circuit for w, with E now an unknown input.
2: Use the black-box CIRCUIT-SAT solver to decide if there is a satisfying assignment for E.
3: Output what the black-box says.

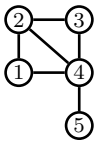
It takes poly($t(n)$)-time to convert C into its circuit of size poly($t(n)$). The black box is polynomial in the size of its input, its runtime is poly(poly($t(n)$)). So the total runtime is poly($t(n)$) + poly(poly($t(n)$)) which is poly($t(n)$). Since $t(n) \leq \text{poly}(|w|)$, the runtime of our decider is poly($|w|$) as required. We have used the CIRCUIT-SAT black box to solve $\mathcal{L}$ in polynomial time.

Theorem 5.1. CIRCUIT-SAT is NP-complete: any NP-problem polynomialy reduces to CIRCUIT-SAT.

In a nutshell, C is a TM-certifier for $\mathcal{L}$. Given w, we would like to know if there is an evidence E for w that satisfies C. By definition, $w \in \mathcal{L}$ if and only if the answer is yes. We quickly convert C into a small circuit-certifier with w hard-coded. We want to ask the same question, is there an evidence E that satisfies the circuit-certifier with w hard-coded. The black-box for CIRCUIT-SAT answers this question in polynomial time. Hence using this black-box, we have a polynomial solver for $\mathcal{L}$.

5.3 Example: Solving CLIQUE using CIRCUIT-SAT

Consider the CLIQUE problem with clique size 3 on this graph,



(40)

The binary sequence encoding of this graph is

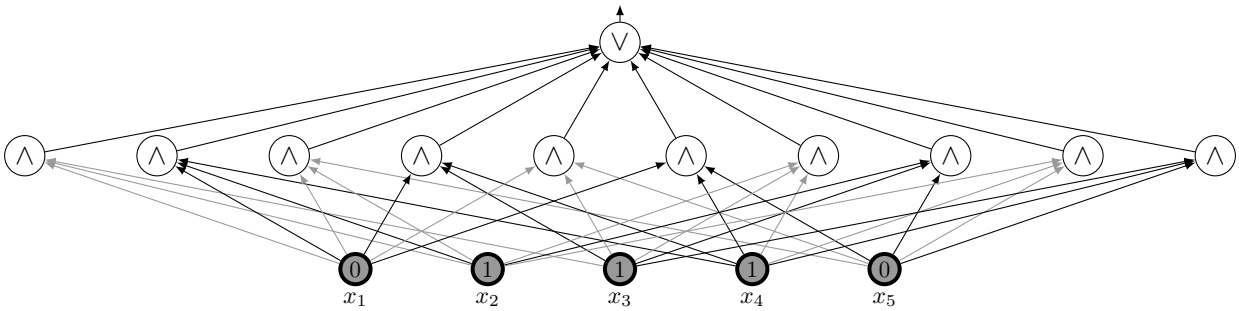
$$\langle G \rangle = e_{12}e_{13}e_{14}e_{15}e_{23}e_{24}e_{25}e_{34}e_{35}e_{45} = 1010110101 \tag{41}$$

The answer is yes, and the evidence is the vertex subset encoded in

$$\text{E} = x_1x_2x_3x_4x_5 = 01110. \tag{42}$$

Let us build a circuit verifier for this problem. The circuit verifier will work for any graph of 5 vertices and any evidence of length 5. The circuit will be small and we will build it "quickly" which is evidenced by us building it manually. (If we can do it manually, it must be quick .)

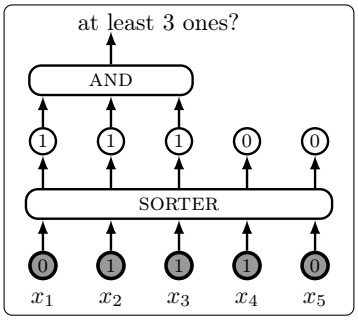
First we need a circuit whose output verifies if the evidence is a vertex-subset of size at least 3. We could check each 3-subset of the evidence (3-bit AND) plus an OR of all these ANDs,



In general, this circuit is too large, requiring $k\binom{n}{k}$ AND gates for n vertices and clique-size k. It is more efficient to first sort the string and take the AND of the first K its. This verifies if there are at

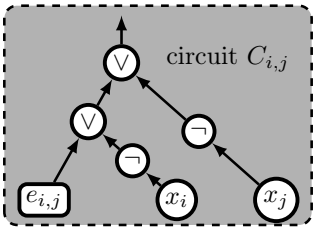
least K ones. A circuit to sort n-bits can be built with $O(n \log n)$ gates. This is illustrated below.

(43)

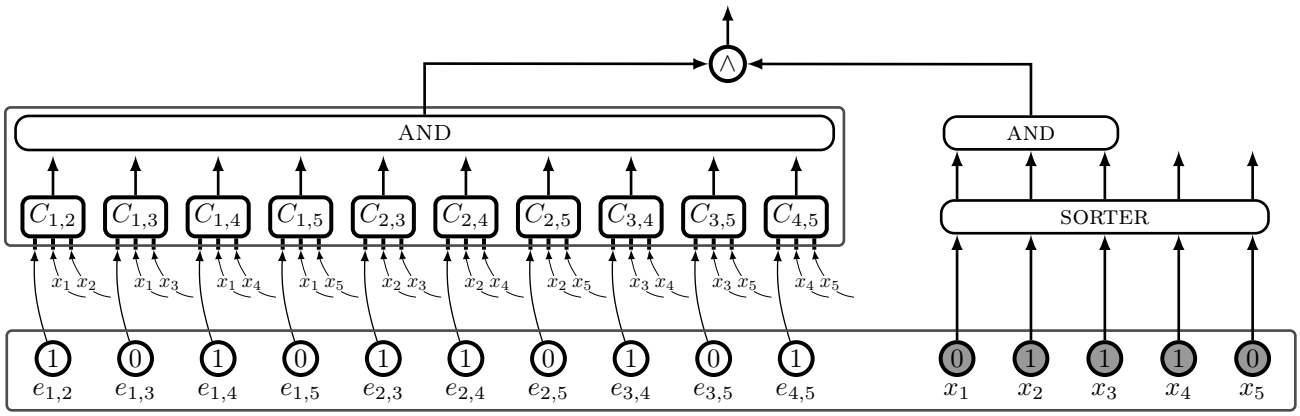


The AND needs $O(K)$ gates, so ultimately the circuit has $O(n \log n)$ gates. We now need a circuit that processes every edge $e_{i,j}$. If $e_{i,j} = 1$, the circuit outputs 1. If $e_{i,j} = 0$, the output is 1 if either $x_i = 0$ or $x_j = 0$. The output for $e_{i,j}$ is $e_{i,j} \vee \overline{x_i} \vee \overline{x_j}$. This is a small circuit $C_{i,j}$ with 5 gates,

(44)



For each edge, we need one such gadget to check that it satisfies the clique-constraint, that is $\binom{n}{2}$ gadgets, each with 5 gates. To ensure all clique constraints, take an AND of all the outputs of these gadgets. The final output of the clique checker is the AND of the outputs of all the $C_{i,j}$'s. We then take the AND of the output of the clique-checker with the output of the size checker circuit.



The size-verifier (right) uses only the clique-string. The clique-verifier (left) uses both the clique-string and edge list. The inputs to each $C_{i,j}$ circuit are $e_{i,j}$, x_i and x_j, e.g. $e_{1,2}, x_1, x_2$ into $C_{1,2}$. In the final circuit treat the evidence as unknown and check if the circuit is satisfiable using CIRCUIT-SAT black box. Our construction generalizes to an arbitrary graph G and value for K.

6 NP-Completeness: Other NP-Complete Problems.

To solve any problem $\mathcal{L} \in \mathsf{NP}$, build its corresponding certifier circuit for input w and use a black box solver for CIRCUIT-SAT to solve the problem. If the black box solver is polynomial time, then you have a polynomial solution for your problem $\mathcal{L}$. Hence, CIRCUIT-SAT is harder than any NP problem and we say that any NP problem is polynomialy reducible to CIRCUIT-SAT,

$$\mathsf{NP} \leq_{\mathrm{P}} \text{CIRCUIT-SAT}. \tag{45}$$

The $\leq_{\mathrm{P}}$ means "at most as hard as." Are there other problems that are harder than any problem in NP? Yes. And showing so is considerably easier than what we did to prove this for CIRCUIT-SAT. The reason is that we can use the fact that CIRCUIT-SAT is NP-complete. Specifically consider any problem $\mathcal{L}_* \in \mathsf{NP}$ and a polynomial black box solver for $\mathcal{L}_*$. If we can show that we can use this black box solver to polynomialy solve CIRCUIT-SAT, then we would have actually built a polynomial black box solver for CIRCUIT-SAT and CIRCUIT-SAT $\leq_{\mathrm{P}} \mathcal{L}_*$. We can polynomialy solve any problem $\mathcal{L} \in \mathsf{NP}$ by first building the solver for CIRCUIT-SAT and then using it to solve $\mathcal{L}$. This means the polynomial solver for $\mathcal{L}_*$ can be used to polynomialy solve any problem in NP and

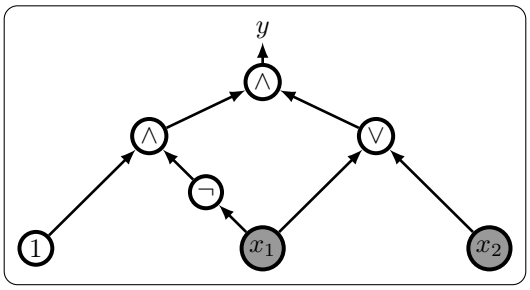
$$\mathsf{NP} \leq_{\mathrm{P}} \mathcal{L}_*. \tag{46}$$

That is, the $\leq_{\mathrm{P}}$ relation is transitive,

$$\mathsf{NP} \leq_{\mathrm{P}} \text{CIRCUIT-SAT and CIRCUIT-SAT} \leq_{\mathrm{P}} \mathcal{L}_* \quad \rightarrow \quad \mathsf{NP} \leq_{\mathrm{P}} \mathcal{L}_*. \tag{47}$$

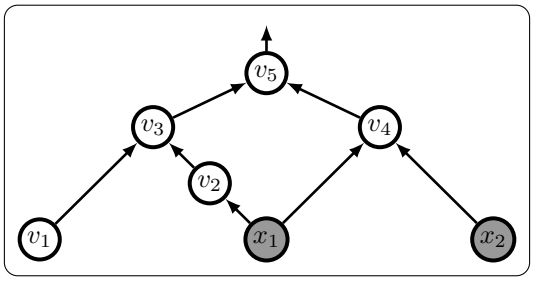
6.1 Reducing CIRCUIT-SAT to 3-SAT

Consider an instance of circuit sat. Here is the circuit which computes $(1 \wedge \overline{x_1}) \wedge (x_1 \vee x_2)$,



(48)

Any circuit computes a Boolean expression, so satisfiability of general Boolean expressions is clearly harder than CIRCUIT-SAT. Let us analyze further the circuit above and convert it into satisfiability of a very structured Boolean expression. First replace all non-input vertices by variables that indicate the output of the vertex. We have variables v_1, v_2, v_3, v_4, v_5,



(49)

The outputs of the variables corresponding to the specific gates in our circuit are given by

$$\begin{aligned} v_1 &= 1 && (50)\\ v_2 &= \overline{x_1} && (51)\\ v_3 &= v_1 \wedge v_2 && (52)\\ v_4 &= x_1 \vee x_2 && (53)\\ v_5 &= v_3 \wedge v_4. && (54) \end{aligned}$$

These are the propagation rules of the circuit. They explicitly tell us, step by step, how to compute the output of the circuit. For example, for $x_1 = x_2 = 1$:

$$v_1 = 1; \quad v_2 = 0; \quad v_3 = 0; \quad v_4 = 1; \quad v_5 = 0. \tag{55}$$

If we wish the output of the circuit to be 1, we additionally require $v_5 = 1$. Hence if the following six conditions can be simultaneously satisfied, for some choices of x_1, x_2, then the circuit is satisfiable:

$$\begin{aligned} v_1 &= 1 && (56)\\ v_2 &= \overline{x_1} && (57)\\ v_3 &= v_1 \wedge v_2 && (58)\\ v_4 &= x_1 \vee x_2 && (59)\\ v_5 &= v_3 \wedge v_4 && (60)\\ v_5 &= 1. && (61) \end{aligned}$$

Replace each condition with a Boolean expression that is true if and only if the condition is satisfied.

$$\begin{array}{lcccl} v_1 & = & 1 & \to & (v_1)\\ v_2 & = & \overline{x_1} & \to & (v_2 \vee x_1) \wedge (\overline{v_2} \vee \overline{x_1})\\ v_3 & = & v_1 \wedge v_2 & \to & (v_3 \vee \overline{v_1} \vee \overline{v_2}) \wedge (\overline{v_3} \vee v_1) \wedge (\overline{v_3} \vee v_2)\\ v_4 & = & x_1 \vee x_2 & \to & (\overline{v_4} \vee x_1 \vee x_2) \wedge (v_4 \vee \overline{x_1}) \wedge (v_4 \vee \overline{x_2})\\ v_5 & = & v_3 \wedge v_4 & \to & (v_5 \vee \overline{v_3} \vee \overline{v_4}) \wedge (\overline{v_5} \vee v_3) \wedge (\overline{v_5} \vee v_4)\\ v_5 & = & 1 & \to & (v_5). \end{array} \tag{62}$$

Verify the condition on the left is true if and only if the Boolean expression on the right is true. Also convince yourself that a general circuit with its propagation rules can be converted to a set of Boolean expressions like this. There is a Boolean expression for each non-input vertex. The circuit is satisfied if every condition is true, that is every Boolean expression is true. Each Boolean expression is the AND of clauses, where each clause is an OR of at most 3 terms. All 13 clauses must be true, that is we must satisfy a massive AND of all the clauses in all the Boolean expressions, a CNF. For n non-input vertices and d input vertices, the number of clauses is at most $3n$ and the number of variables is $n + d$. Every one of these clauses is satisfiable by appropriately setting the $n + d$ Boolean variables if and only if the circuit is satistiable. We define the NP-problem 3-SAT.

> 3-SAT: Given a set of n clauses over variables $x_1, \ldots, x_n$, where each clause is the OR of at most 3 terms, is there an assignment to $x_1, \ldots, x_n$ for which every clause is true?

That is, can all clauses be satisfied. Clearly, 3-SAT is in NP. To solve an instance of CIRCUIT-SAT with n non-input vertices and d input vertices, first transform it to the corresponding set of at most $3n$ clauses and using a black box solver for 3-SAT on these clauses. We have proved

Theorem 6.1. CIRCUIT-SAT $\leq_{\text{P}}$ 3-SAT. That is, 3-SAT is NP-complete.

6.2 Other NP-complete Problems: INDEPENDENT-SET, CLIQUE, VERTEX-COVER

CIRCUIT-SAT is a relatively complex problem to deal with, involving complex circuits. But, 3-SAT is relatively easy to handle, consisting only of a bunch of clauses with each clause being an OR of at most 3 terms. 3-SAT is a power-tool for finding other NP-complete.

In general, to show that $\mathcal{L}$ is NP-complete, polynomialy reduce any known NP-complete problem $\mathcal{L}_*$ to $\mathcal{L}$. That is, we show how we can polynomially solve $\mathcal{L}_*$ if we have a black box polynomial solver for $\mathcal{L}$. Let's see this in action by proving that INDEP-SET is NP-complete. A set of vertices in a graph is an independent set if every pair of vertices in the set is non-adjacent.

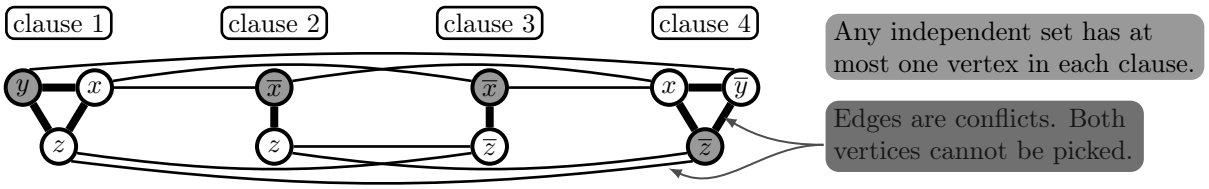
INDEP-SET: Given a graph G and $K \geq 1$, is there an independent set of size at least K.

We show that 3-SAT is polynomialy reducible to INDEP-SET. Consider the instance of 3-SAT

$$(y \vee x \vee z)(\overline{x} \vee z)(\overline{x} \vee \overline{z})(x \vee \overline{y} \vee \overline{z}). \tag{63}$$

These clauses are satisfiable if and only if one can pick exactly one term from each clause to make true without any conflicts between terms picked to be made true. A conflict is for example picking both x and $\overline{x}$ to make true. They conflict because both cannot be made true.

Build a graph as follows. Each clause is a clique with a vertex for each term. Vertices in different cliques are linked if they correspond to conflicting terms. We get a conflict graph,



If there is an independent set whose size equals the number of clauses, then it means exactly one vertex is picked from each clique and the vertices do not conflict which means they can all be made true. So, if we have a black box solver for INDEP-SET, we can solve 3-SAT. We have proved

Theorem 6.2. 3-SAT $\leq_{\text{P}}$ INDEP-SET. That is, INDEP-SET is NP-complete.

We leave it as an exercise for you to formally prove INDEP-SET is NP-complete. Also, show that CLIQUE and VERTEX-COVER are NP-complete by polynomialy reducing them to INDEP-SET. So now we have the NP-complete problems

$$\{\text{CIRCUIT-SAT}, \text{3-SAT}, \text{INDEP-SET}, \text{CLIQUE}, \text{VERTEX-COVER}\}. \tag{64}$$

Karp, in his seminal 1972 paper, gave 21 problems in diverse domains that are NP-complete. This list has since exploded to several thousands of problems occuring frequently in practice and in very diverse areas. We need to solve these problems in practice. These problems are typically phrased as optimization problems. What do we do?

1. Change the objective to an easier one for which there is a polynomial solution.
2. Change the problem.
3. Approximate the solution.

7 Linear Algebra and Complex Vector Spaces

We begin our study of quantum computing by reviewing some of the useful math. The complex numbers are perhaps one of humanity's greatest inventions, along with calculus, induction, language, written storage of knowledge, the wheel, etc. You will never encounter a complex number in the real world, yet they are essential for how we model the world, and without them there would be no quantum mechanics. They also play an important role in algorithms like the FFT.

Linear algebra is instrumental to the study of linear operators. There are two formulations of quantum mechanics, the PDE approach due to Schrödinger and the matrix mechanics approach due to Heisenberg. We will follow the latter approach which is heavily based in linear operators. We are going to try to learn about quantum computing algorithms without learning quantum mechanics. To build quantum computers, however, you will have to become one with quantum mechanics.

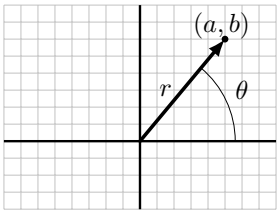
7.1 Complex Numbers

To solve $x^2 + 1 = 0$, we invented $i = \sqrt{-1}$, a placeholder for the solution of this equation. And thence, the entire field of complex analysis developed around this simple concept. It is a miracle that this "imaginary" concept has such an influence on all things real. Mathematically, the complex numbers are algebraically closed. This is the content of the fundamental theorem of algebra: every polynomial with complex coefficients (for example $x^2 + 1$) has at least one complex zero.

A complex number with real part a and imaginary part b is written

$$x = a + ib. \tag{65}$$

Complex numbers can be added, subtracted, multiplied and divided using standard algebra and $i^2 = -1$. A complex number is a point on the Cartesian plane with coordinates (a, b),



(66)

In polar coordinates, the representation is (r, θ), where $a = r\cos\theta$ and $b = r\sin\theta$, and so

$$\begin{aligned} r^2 &= a^2 + b^2 \\ \tan\theta &= b/a. \end{aligned} \tag{67}$$

The polar representation is not unique and $(r, \theta + 2k\pi)$ for $k \in \mathbb{Z}$ is the same complex number. The spectacular theorem of Euler is a power-tool,

$$e^{i\theta} = \cos\theta + i\sin\theta. \tag{68}$$

This truly remarkable relation links the complex numbers, calculus, the number e, and geometry/trigonometry. Since $(e^{i\theta})^n = e^{in\theta}$, we immediately get Demoivre's relation,

$$(\cos x + i\sin x)^n = e^{inx} = \cos nx + i\sin nx. \tag{69}$$

Euler's formula allows us to multiply and take powers of complex numbers effortlessly. Since

$$x = a + ib = re^{i\theta}, \tag{70}$$

It follows that $x^n = r^n e^{in\theta}$ and $x_1 x_2 = r_1 r_2 e^{i(\theta_1+\theta_2)}$. In the polar representation,

$$\begin{array}{rcl} (r,\theta)^n & = & (r^n, n\theta) \\ (r_1,\theta_1)\times(r_2,\theta_2) & = & (r_1 r_2, \theta_1+\theta_2) \\ (r,\theta)^{1/n} & = & (r^{1/n}, (\theta+2k\pi)/n). \end{array} \tag{71}$$

The complex conjugate of a complex number is obtained by negating the imaginary part.

$$x = a + ib \to x^* = a - ib. \tag{72}$$

In the polar representation, $x = (r,\theta) \to x^* = (r,-\theta)$.

7.2 Complex Vector Spaces

The complex vector space $\mathbb{C}^n$ of dimension n consists of all vectors with n components, where the components are complex numbers. Here is an example of a complex vector and a complex matrix.

$$x = \begin{bmatrix} 1-i \\ 2i \\ -1 \end{bmatrix} \qquad \mathrm{A} = \begin{bmatrix} 1-i & 1+i & 3 \\ 2i & -1 & 1+2i \end{bmatrix}. \tag{73}$$

A is an operator from $\mathbb{C}^3 \mapsto \mathbb{C}^2$. We compute the matrix-vector product $\mathrm{A}x$ in the usual way. We define the transpose in the usual way. We can also take the complex conjugate of a matrix by taking the complex conjugate of every entry. An important operation is to take the transpose of the complex conjugate, called the dagger,

$$x^\dagger = \begin{bmatrix} 1+i & -2i & -1 \end{bmatrix} \qquad \mathrm{A}^\dagger = \begin{bmatrix} 1+i & -2i \\ 1-i & -1 \\ 3 & 1-2i \end{bmatrix}. \tag{74}$$

Verify that

$$\begin{array}{ll} (\mathrm{A}+\mathrm{B})^{\mathrm{T}} = \mathrm{A}^{\mathrm{T}} + \mathrm{B}^{\mathrm{T}}, & (\mathrm{AB})^{\mathrm{T}} = \mathrm{B}^{\mathrm{T}}\mathrm{A}^{\mathrm{T}}; \\ (\mathrm{A}+\mathrm{B})^\dagger = \mathrm{A}^\dagger + \mathrm{B}^\dagger, & (\mathrm{AB})^\dagger = \mathrm{B}^\dagger\mathrm{A}^\dagger; \end{array} \tag{75}$$

The inner product $\langle x, y\rangle = x^{\mathrm{T}} y$ from real vector spaces won't work in complex vector spaces. One reason is that the norm $\|x\|^2 = \langle x, x\rangle$ should be non-negative. This is not true when x, y can be complex. Instead, we use the dagger and define the inner product

$$\langle x, y\rangle = x^\dagger y. \tag{76}$$

This definition implies linearity in the second argument,

$$\langle x, ay + bz\rangle = a\langle x, y\rangle + b\langle x, z\rangle. \tag{77}$$

What is $\langle ax + bz, y\rangle$? Note, one could instead insist on linearity in the first argument, in which case $\langle x, y\rangle = y^\dagger x$. Note that the inner product is not symmetric, $\langle x, y\rangle = \langle y, x\rangle^\dagger$. For operators A, B,

$$\langle \mathrm{A}x, \mathrm{B}y\rangle = (\mathrm{A}x)^\dagger \mathrm{B}y = x^\dagger \mathrm{A}^\dagger \mathrm{B}y. \tag{78}$$

An operator A (matrix) is hermitian (or self-adjoint) if $\mathrm{A}^\dagger = \mathrm{A}$. A is unitary if $\mathrm{A}^{-1} = \mathrm{A}^\dagger$, i.e.

$$\mathrm{A}\mathrm{A}^\dagger = \mathrm{A}^\dagger\mathrm{A} = \mathrm{I}. \tag{79}$$

A quantum computer is a quantum physical system. Unitary operators play an important role because they drive the time evolution of a quantum system (quantum computer), specifically $e^{it\mathrm{H}}$ where H is the Hamiltonian operator. An algorithm on a quantum computer takes the state from an initial configuration to a final configuration in which we measure a result. Thus, any algorithm is performing a time-evolution of the initial state and hence must be a unitary operator. The general form of the 2×2 unitary operator, up to a multiplicative phase is

$$\mathrm{U} = \begin{bmatrix} r & \sqrt{1-r^2}e^{i\varphi_1} \\ \sqrt{1-r^2}e^{i\varphi_2} & -re^{i(\varphi_1-\varphi_2)} \end{bmatrix} \tag{80}$$

In quantum mechanics, observables are related to Hermitian operators. The result of a quantum algorithm must be a measurable observable. Hence, quantum algorithms are related to hermitian operators. Observables are related to hermitian operators through eigenvectors and eigenvalues (eigen meaning "special"). For a matrix A, a nonzero vector v for which

$$\mathrm{A}v = \lambda v \tag{81}$$

is called an eigenvector of A and λ is the associated eigenvalue.

7.3 Basis

An orthonormal basis in an n-dimensional complex vector space is a collection of n pairwise orthogonal unit vectors. More generally a basis is a largest set of linearly independent vectors. The standard basis consists of the column vectors in

$$\mathrm{E} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n] = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ & & & & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{82}$$

The basis is represented by the square matrix E. The basis is orthonormal if $\mathrm{E}^\dagger\mathrm{E} = \mathrm{I}_n$. That is, if E is unitary. The components of a vector v in the basis E, given by $[x_1, x_2, \ldots, x_n]^{\mathrm{T}}$ represent v as a linear combination of the basis vectors,

$$v = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = x_1\mathbf{e}_1 + x_2\mathbf{e}_2 + \cdots + x_n\mathbf{e}_n. \tag{83}$$

Usually vectors will be clear from the context, but sometimes we will use bold to emphasize some quantity as a vector. The components in one orthonormal basis can be transformed into another orthonormal basis. If v has components $[z_1, z_2, \ldots, z_n]^{\mathrm{T}}$ in the orthonormal basis F, then

$$v = \mathrm{F}z \to z = \mathrm{F}^\dagger v = \mathrm{F}^\dagger \mathrm{E}x. \tag{84}$$

The matrix $\mathrm{F}^\dagger\mathrm{E}$ is a basis transformation matrix that transforms the components x in basis E to the components z in basis F.

7.4 Spectral Theorem

For any n-dimensional hermitian operator H, one can construct an orthonormal basis composed of its eigenvectors. This is the spectral theorem. That is, there is a unitary basis U for which

$$\mathrm{HU} = \mathrm{U}\Lambda, \tag{85}$$

where Λ is a diagonal matrix composed of eigenvalues of H along the diagonal. This is called the spectral theorem. Applying $\mathrm{U}^\dagger$ to both sides,

$$\mathrm{U}^\dagger\mathrm{HU} = \Lambda. \tag{86}$$

That is, H can be diagonalized by a unitary matrix. The spectral theorem is a very powerful tool.

7.5 Hadamard Matrix

A particularly important quantum gate is the Hadamard gate, which is related to the Hadamard matrix. The unnormalized Hadamard matrix Q_n is defined by

$$\mathrm{Q}_0 = [1]; \qquad \mathrm{Q}_{n+1} = \begin{bmatrix} \mathrm{Q}_n & \mathrm{Q}_n \\ \mathrm{Q}_n & -\mathrm{Q}_n \end{bmatrix}. \tag{87}$$

For example,

$$\mathrm{Q}_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \qquad \mathrm{Q}_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}. \tag{88}$$

The normalized Hadamard matrix H_n simply normalizes the matrix so each column has unit norm,

$$\mathrm{H}_n = 2^{-n/2}\mathrm{Q}_n. \tag{89}$$

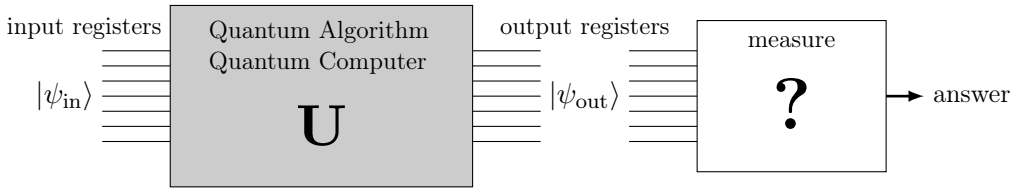
The Hadamard matrix has many useful properties. You can prove them by induction.

1. H_n is a $2^n \times 2^n$ matrix.
2. H_n is hermitian and unitary, $\mathrm{H}_n^\dagger\mathrm{H}_n = \mathrm{I}_{2^n}$.
3. Q_n has only entries ± 1.
4. The first row and column of Q_n are all 1's. The sum of every other column and row is 0.

8 Quantum Mechanics

We hope to learn about QC algorithms with minimal knowledge of quantum mechanics. However, if you want to build a quantum computer, well that will be a physical system governed by the laws of quantum mechanics. So you'll need to know some quantum mechanics. At least we will give you the postulates and how they are relevant to building a stable quantum computer.

In quantum mechanics, we use $|\psi\rangle$ to represent the state of the system you are interested in, for example the state of a quantum computer, which includes its input and output. The quantum computer is a black box unitary operator U that takes the input state to the output state. In the output state, we measure some observable which corresponds to the result,



(90)

Quantum mechanics postulates rules for:

- How the state evolves under the quantum computing algorithm as well as interactions with the environment.
- What happens when you measure the result.

We don't know *why* the rules are this way. We do know that these rules plus lots of mathematics, are remarkably accurate at describing the physical world, especially at tiny scales where classical mechanics fail. Similarly, we don't know *why* Newton's law is $F = ma$ and not $F = ma^2$, for example? However, we do know that Newton's law is remarkably accurate at describing the real world at large scales – the classical regime. How did we find the rules of quantum mechanics. Similar to how Newton found his laws. Observe, guess the laws and then test the predictions.

8.1 Postulates of Quantum Mechanics

The state of a system is specified by a vector $|\psi\rangle$ in a complex vector space with the standard inner product. The inner product of $|\psi\rangle$ and $|\phi\rangle$ is written $\langle\phi|\psi\rangle$ (bra-ket notation), where

$$\langle\phi|\psi\rangle = (|\phi\rangle)^\dagger|\psi\rangle. \tag{91}$$

The state is normalized to 1, as required by the probabilistic interpretation of measurements,

$$\langle\psi|\psi\rangle = 1. \tag{92}$$

The system's state evolves with time, so it starts at $|\psi(0)\rangle$ and evolves to $|\psi(t)\rangle$ over time t. This evolution is driven by the energy operator, or Hamiltonian operator $\hat{\text{H}}$.[1] Specifically,

$$|\psi(t)\rangle = e^{-i\hat{\text{H}}t}|\psi(0)\rangle. \tag{93}$$

[1] We sometimes use the hat notation, $\hat{\text{A}}$, to emphasize something is an operator (matrix).

(There is a constant $\hbar$, the reduced plank constant, which sets the units for energy. We set it to 1 for simplicity of presentation.) Just understanding this formula is non-trivial. What is the exponent of an operator? This is defined using the Taylor series for the exponential. It is a non-trivial exercise for the reader to show $e^{-i\hat{\mathrm{H}}t}$ is a unitary operator whenever $\hat{\mathrm{H}}$ is hermitian, which it is. This means that time-evolution preserves geometry, in particular the norm of the state stays 1,

$$\begin{aligned}
\langle\psi(t)|\psi(t)\rangle &= (e^{-i\hat{\mathrm{H}}t}|\psi(0)\rangle)^\dagger e^{-i\hat{\mathrm{H}}t}|\psi(0)\rangle && (94)\\
&= (|\psi(0)\rangle)^\dagger (e^{-i\hat{\mathrm{H}}t})^\dagger e^{-i\hat{\mathrm{H}}t}|\psi(0)\rangle && (95)\\
&= (|\psi(0)\rangle)^\dagger |\psi(0)\rangle && (96)\\
&= \langle\psi(0)|\psi(0)\rangle && (97)\\
&= 1. && (98)
\end{aligned}$$

An observable is associated to a hermitian operator. There are operators for position, momentum, energy, spin, etc. Let us consider the observable A, associated to the operator $\hat{\mathrm{A}}$. Since $\hat{\mathrm{A}}$ is hermitian it has an orthonormal eigenbasis for the state space, $|\phi_1\rangle, \ldots, |\phi_n\rangle$ (assuming the state-space is n-dimensional), and the eigenvector $|\phi_i\rangle$ has an associated eigenvalue λ_i. This means we can get the "coordinates" of $|\psi\rangle$ in this eigenbasis,

$$|\psi\rangle = \sum_{i=1}^{n} a_i |\phi_i\rangle. \tag{99}$$

The a_i can be complex. In state $|\psi\rangle$, when you measure the observable A, the result is an eigenvalue λ_i of the associated operator $\hat{\mathrm{A}}$. Which eigenvalue you see is random, with

$$\text{probability to observe } \lambda_i = \|a_i\|^2 = a_i^* a_i. \tag{100}$$

These probabilities are determined from the expansion of $|\psi\rangle$ in the eigenbasis of $\hat{\mathrm{A}}$. We can also compute the expected value of the observable,

$$\mathbb{E}[A] = \sum_i \lambda_i \|a_i\|^2 \tag{101}$$

We can get a convenient expression for this expected value as follows. Note that

$$\hat{\mathrm{A}}|\psi\rangle = \sum_{i=1}^{n} a_i \lambda_i |\phi_i\rangle. \tag{102}$$

Computing $\langle|\psi\rangle, \hat{\mathrm{A}}|\psi\rangle\rangle$ gives

$$\begin{aligned}
\langle|\psi\rangle, \hat{\mathrm{A}}|\psi\rangle\rangle &= \sum_{j=1}^{n} a_j^* |\phi_j\rangle^\dagger \sum_{i=1}^{n} a_i \lambda_i |\phi_i\rangle && (103)\\
&= \sum_{j=1}^{n}\sum_{i=1}^{n} \lambda_i a_j^* a_i |\phi_j\rangle^\dagger |\phi_i\rangle. && (104)
\end{aligned}$$

Since $|\phi_j\rangle^\dagger|\phi_i\rangle = \delta_{ij}$ because the eigenbasis is orthonormal, we have that

$$\begin{aligned}\langle|\psi\rangle, \hat{\mathrm{A}}|\psi\rangle\rangle &= \sum_{j=1}^{n}\sum_{i=1}^{n}\lambda_i a_j^* a_i \delta_{ij} &(105)\\ &= \sum_{j=1}^{n}\lambda_j a_j^* a_j = \mathbb{E}[A]. &(106)\end{aligned}$$

To avoid cumbersome notation, we write the inner product $\langle|\psi\rangle, \hat{\mathrm{A}}|\psi\rangle\rangle$ as $\langle\psi|\hat{\mathrm{A}}|\psi\rangle$.

Another wierd thing happens. The state changes upon measurement. If the measurement is λ_i, then the state collapses to the corresponding eigenvector, $|\psi\rangle \to |\phi_i\rangle$. When the state is one of these eigenvectors, we call it a pure state, otherwise it is a mixed state. State collapse is always to a pure state. This state collapse is perhaps the most counter-intuitive aspect of quantum mechanics. It is also counterintuitive that the properties of the state, when measured are random. Let us summarize.

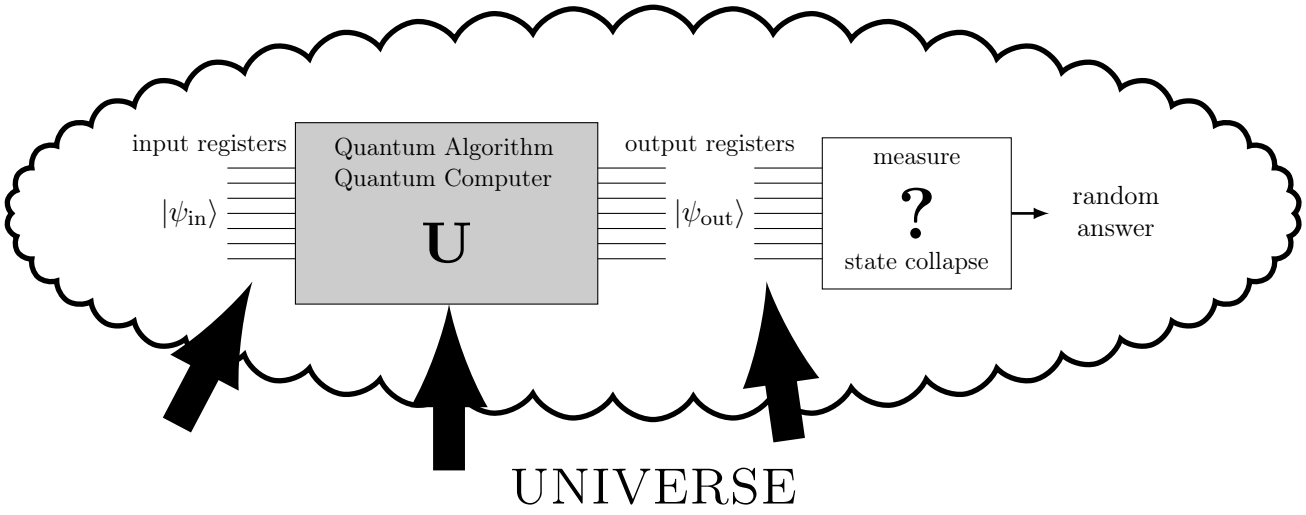
1. [State] The state $|\psi\rangle$ is a complete representation of a system. It is normalized, so $\langle\psi|\psi\rangle = 1$.
2. [Time Evolution] The state evolves according to the Hamiltonian operator $|\psi(t)\rangle = e^{-i\hat{\mathrm{H}}t}|\psi(0)\rangle$.
3. [Observables] An observable A corresponds to an operator $\hat{\mathrm{A}}$. Operators standard observables like position, momentum, energy, are specified in the postulates. We don't give them here.
4. [Measurement] When you measure an observable A with corresponding operator $\hat{\mathrm{A}}$, the possible measurements are the eigenvalues of $\hat{\mathrm{A}}$. If $\hat{\mathrm{A}}$ has orthonormal eigenbasis $|\phi_i\rangle$ with corresponding eigenvalues λ_i, then we can expand $|\psi\rangle$ in this basis,

   $$|\psi\rangle = \sum_{i=1}^{n} a_i|\phi_i\rangle. \tag{107}$$

   The eigenvalue λ_i is measured with probability $\|a_i\|^2$ and $\mathbb{E}[A] = \langle\psi|\hat{\mathrm{A}}|\psi\rangle$.
5. [State Collapse Upon Measurement] If the measured value is λ_i, then the state collapses to the corresponding eigenvector of $\hat{\mathrm{A}}$, $|\psi\rangle \to |\phi_i\rangle$.
6. There are some other postulates regarding different types of particles, e.g. the state is anti-symmetric with respect fermion-exchange. We won't go into those details.

How do these postulates of quantum mechanics affect us when we are trying to build a quantum computer. Let us go back to our idealized workflow figure from before. The initial state of the quantum computer is a quantum state. The algorithm evolves this state into a final state and we extract the result by measuring some property of the final state. We assume the result is the output of our quantum algorithm - some unitary operator that evolves the state. Unfortunately, our computer interacts with the universe. The universe interaction contaminates our algorithm by also driving some of the state evolution. We can try to isolate the quantum computer from the universe, but, because we have to ultimately make the measurement, there will always be some evolution

of the final state that is not due to the quantum algorithm. Hence the result we measure will be contaminated. We need ways to error correct. So, the actual picture looks more like



(108)

Then there is the measured result being nondeterministic. This won't do. If our quantum algorithm must tell if a number is prime, the answer must be a deterministic yes or no. Hence, we have to ensure that the final state is a pure state, so the result is definitive. This impacts the design of quantum-algorithms as well as the measurement process, which should keep the state pure. Lastly, from the practical perspective we need to store the quantum equivalent of bits, keep them stable, allow for the robust evolution of that state under an algorithm and reliable measurement of the state to obtain the final result. These are all non-trivial requirements, which largely puts quantum computation far from practice. The path to a viable quantum computer is more like a marathon, not a sprint. So patience and endurance are necessary. The first step is to develop the seeds, the theory of quantum algorithms. This is the main focus of this course. Though we don't yet have the pasture on which to plant these seeds, it is still a worthy task.

8.2 Spin

Some particles have a property called spin, e.g., spin-up or spin-down. Particles with two possible spins can encode a bit. Spin-up is 1 and spin-down is 0. Spin depends on the axis of rotation. So when we measure spin, we do so with respect to an axis. We define operators corresponding to the three common axes x, y, z. These are the three spin operators (in units with $\hbar = 1$),

$$\hat{\mathrm{S}}_x = \frac{1}{2}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad \hat{\mathrm{S}}_y = \frac{1}{2}\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \qquad \hat{\mathrm{S}}_z = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{109}$$

Consider the spin operator $\hat{\mathrm{S}}_z$. Measuring spin about the z-axis produces a result $\lambda \boldsymbol{e_3}$, where λ is the magnitude or amount of spin, and $\boldsymbol{e}_3 = [0, 0, 1]^{\mathrm{T}}$ is the basis vector in the z-direction. The $\boldsymbol{e}_3$ says it is the z-spin that you measured. According to the postulates, the possible values for λ are the eigenvalues of $\hat{\mathrm{S}}_z$. As an exercise verify that the eigenvalue, eigenvector pairs of $\hat{\mathrm{S}}_z$ are:

$$\lambda_z^+ = \frac{1}{2}, |\phi_z^+\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad \lambda_z^- = -\frac{1}{2}, |\phi_z^-\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{110}$$

The possible z-spins are $\pm\frac{1}{2}\boldsymbol{e}_3$. Similarly, the eigenvalue-eigenvector pairs for $\hat{\mathrm{S}}_x$ and $\hat{\mathrm{S}}_y$ are

$$\lambda_x^+ = \frac{1}{2}, |\phi_x^+\rangle = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \qquad \lambda_x^- = -\frac{1}{2}, |\phi_x^-\rangle = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}; \tag{111}$$

$$\lambda_y^+ = \frac{1}{2}, |\phi_y^+\rangle = \begin{bmatrix} 1/\sqrt{2} \\ i/\sqrt{2} \end{bmatrix}, \qquad \lambda_y^- = -\frac{1}{2}, |\phi_y^-\rangle = \begin{bmatrix} 1/\sqrt{2} \\ -i/\sqrt{2} \end{bmatrix}. \tag{112}$$

The possible measurements for the x-spin are $\pm\frac{1}{2}\boldsymbol{e}_1$ and for the y-spin are $\pm\frac{1}{2}\boldsymbol{e}_2$, where $\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$ are the standard basis in 3 dimensions. In practice, one can measure the spin in any direction defined by a unit vector $\boldsymbol{v} = [v_x, v_y, v_z]$. Such a vector can be represented by its two polar coordinates φ, ϑ,

$$\boldsymbol{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} \sin\vartheta\cos\varphi \\ \sin\vartheta\sin\varphi \\ \cos\vartheta \end{bmatrix}. \tag{113}$$

(We only have a limited set of notation, so please distinguish between the azimuthal angle φ and the eigenbasis-vector $|\phi\rangle$.) Abusing notation and defining the vector of operators $\hat{\mathbf{S}} = [\hat{\mathrm{S}}_x, \hat{\mathrm{S}}_y, \hat{\mathrm{S}}_z]^{\mathrm{T}}$, we define the spin-operator for the direction $\boldsymbol{v}$ by

$$\hat{\mathrm{S}}_{\boldsymbol{v}} = \boldsymbol{v} \bullet \hat{\mathbf{S}} = v_x\hat{\mathrm{S}}_x + v_y\hat{\mathrm{S}}_y + v_z\hat{\mathrm{S}}_z = \frac{1}{2}\begin{bmatrix} \cos\vartheta & e^{-i\varphi}\sin\vartheta \\ e^{i\varphi}\sin\vartheta & -\cos\vartheta \end{bmatrix} \tag{114}$$

This definition of the spin-operator in the $\boldsymbol{v}$-direction is chosen to satisfy a consistency condition. First, the possible spin measurements (eigenvalues) must be $\lambda_{\boldsymbol{v}}^{\pm} = \pm\frac{1}{2}$ as can be verified. In the pure state defined by the eigenvector $|\phi_{\boldsymbol{v}}^+\rangle$, the positive spin state about the axis $\boldsymbol{v}$, the expected x-spin should be $v_x/2$. Similarly, the expected y-spin and z-spin should be $v_y/2$ and $v_z/2$ respectively. That means the expected spin is $+\frac{1}{2}$ in the $\boldsymbol{v}$ direction. We want

$$\mathbb{E}[\hat{\mathbf{S}}] = \begin{bmatrix} \langle\phi_{\boldsymbol{v}}^+|\hat{\mathrm{S}}_x|\phi_{\boldsymbol{v}}^+\rangle \\ \langle\phi_{\boldsymbol{v}}^+|\hat{\mathrm{S}}_y|\phi_{\boldsymbol{v}}^+\rangle \\ \langle\phi_{\boldsymbol{v}}^+|\hat{\mathrm{S}}_z|\phi_{\boldsymbol{v}}^+\rangle \end{bmatrix} = \begin{bmatrix} \sin\vartheta\cos\varphi \\ \sin\vartheta\sin\varphi \\ \cos\vartheta \end{bmatrix}. \tag{115}$$

Indeed, this is the case. Prove it. First find the spin-up eigenvector,

$$|\phi_{\boldsymbol{v}}^+\rangle = \begin{bmatrix} \cos\vartheta/2 \\ e^{i\varphi}\sin\vartheta/2 \end{bmatrix}. \tag{116}$$

Show that $\langle\phi_{\boldsymbol{v}}^+|\hat{\mathrm{S}}_x|\phi_{\boldsymbol{v}}^+\rangle = \sin\vartheta\cos\varphi$. Similarly prove the other two components of (115).

Here is a useful exercise. A particle starts with x-spin up. You measure the z-spin and then measure the x-spin again. What are the probabilities for the four possible outcomes:

$$\begin{array}{cc} + & + \\ + & - \\ - & + \\ - & - \end{array} \tag{117}$$

Even though we know the particle starts with x-spin up, the measurement of the x-spin *after* you measure the z-spin could be down. This is very counterintuitive.

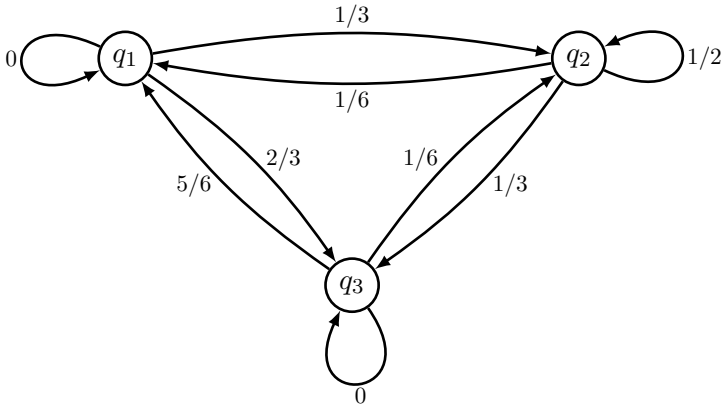
9 Dynamics

Why do we need to study dynamics? Let's look at a classical computer, the Turing machine. The TM starts off with the input written on the tape. The TM's state plus what is on the tape plus where its read-write head is can be viewed as the system's configuation. The TM-instructions (i.e., the algorithm) tell the TM what to do depending on what it reads on the tape. It can write something, transition states and move left or right, thus the system's configuration evolves to a new configuration. This dynamics is driven by the specifics of the algorithm. So, an algorithm dictates the dynamics of the TM's configuration. When the dynamics ends, we look at the final configuration to "measure" the answer delivered by the algorithm on the starting input.

None of this high-level discussion changes when we move to quantum algorithms. A quantum algorithm dicates the dynamics of the configuration of the quantum computer, starting from some initial configuration. When the dynamics ends, we measure the configuration of the quantum computer to determine our answer. Some of the details change in moving from the classical realm to the quantum realm, and these small changes make all the difference.

9.1 Classical Dynamics

A ball starting in vertex q_1 transitions according to the following graph,



(118)

An arrow indicates a possible transition from vertex q_i to q_j, and the weight on an arrow is the probability of that transition. For example, after 1 transition, the probability is 1/3 to be in q_2 and 2/3 to be in q_3. Collect these probabilities into a transition matrix T, where T_{ij} is the probability to transition from q_j to q_i. Column j in T are the probabilities to transition from q_j. We have

$$\mathrm{T} = \begin{bmatrix} 0 & 1/6 & 5/6 \\ 1/3 & 1/2 & 1/6 \\ 2/3 & 1/3 & 0 \end{bmatrix}. \tag{119}$$

Notice that every column sums to 1 because from any vertex, the ball must transition to some vertex. Such a T is called a stochastic matrix, and the ball follows a Markov chain. Our matrix T also has rows which sum to 1 (called doubly stochastic). Such a Markov chain can be run backwards in time using T^{T}, that is the process is reversible. We can represent the state of the ball by the

probabilities it is in each vertex. The start state and the state after one transition are

$$|\psi_0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad |\psi_1\rangle = \begin{bmatrix} 0 \\ 1/3 \\ 2/3 \end{bmatrix}. \tag{120}$$

You can verify that $|\psi_1\rangle = \mathrm{T}|\psi_0\rangle$. In general, you can show by induction that after k transitions,

$$|\psi_k\rangle = \mathrm{T}^k|\psi_0\rangle. \tag{121}$$

You have to show that if the vertex probabilities at step k are given by $|\psi_k\rangle$, then the vertex probabilities at step $k+1$ are given by $\mathrm{T}|\psi_k\rangle$. The system dynamics is captured by the transition matrix T and we say that T is the propagator for the system, propagating the state into the future.

9.2 Quantum Dynamics

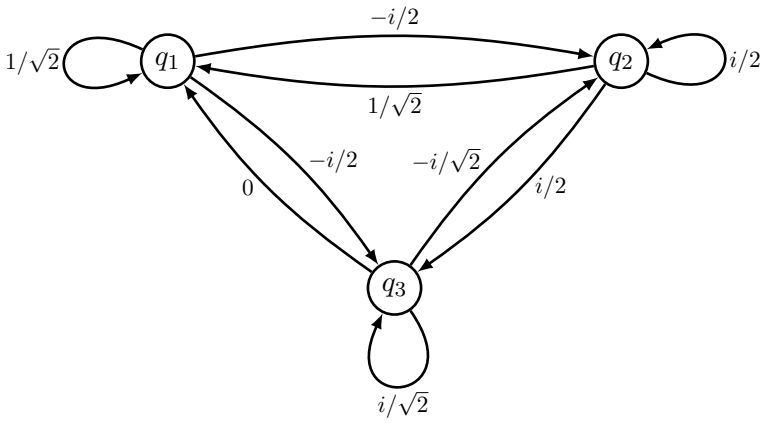
In quantum dynamics, the state $|\psi\rangle$ containing the probabilities to be in each vertex is replaced by a state $|\psi\rangle$ with *amplitudes* to be in each vertex. Here are some possible states.

$$|\psi_0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad |\psi_0\rangle = \begin{bmatrix} i \\ 0 \\ 0 \end{bmatrix}, \qquad |\psi_0\rangle = \begin{bmatrix} \sqrt{2/3} \\ 0 \\ -\sqrt{1/3} \end{bmatrix}. \tag{122}$$

Amplitudes are not probabilities. They can be negative or even complex, and they do not sum to 1. The amplitudes are converted into probabilities by computing the norm squared of the amplitude. The probabilities corresponding to the above amplitudes are

$$P_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad P_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad P_0 = \begin{bmatrix} 2/3 \\ 0 \\ 1/3 \end{bmatrix}. \tag{123}$$

These are valid non-negative real probabilities summing to 1, that is $\langle\psi_0|\psi_0\rangle = 1$. Just like in classical dynamics, the amplitudes propagate forward in time using a propagator U. Unlike in the classical case where the propagator (probabilty transition matrix) is a real stochastic matrix, the propagator in a quantum dynamics can have complex entries. Consider the following dynamics,



(124)

These quantum dynamics can be summarized in the matrix

$$\mathrm{U} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -i/2 & i/2 & -i/\sqrt{2} \\ -i/2 & i/2 & i/\sqrt{2} \end{bmatrix}. \tag{125}$$

The first column of U, for example, has the amplitudes to transition from vertex q_1 to q_i. The state after one step of propagation is

$$|\psi_1\rangle = \mathrm{U}|\psi_0\rangle. \tag{126}$$

For $|\psi_1\rangle$ to be valid amplitudes, the norm squared of its entries must be valid probabilities and sum to 1. That is $\langle\psi_1|\psi_1\rangle = 1$ for any $|\psi_0\rangle$. That is, for all $|\psi_0\rangle$

$$\langle\psi_1|\psi_1\rangle = \langle\psi_0|\mathrm{U}^\dagger\mathrm{U}|\psi_0\rangle = \langle\psi_0|\psi_0\rangle = 1. \tag{127}$$

This means U must be unitary, as can be verified for our choice of U. In quantum mechanics, the propagator in time $e^{-i\hat{\mathrm{H}}t}$ is unitary because $\hat{\mathrm{H}}$ is hermitian. The evolution of any quantum system like a quantum computer is driven by a unitary operator and hence preserves the normalization of a state. Let's see the evolution of our three example starting states under our unitary matrix U:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\mathrm{U}} \begin{bmatrix} 1/\sqrt{2} \\ -i/2 \\ -i/2 \end{bmatrix}, \qquad \begin{bmatrix} i \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\mathrm{U}} \begin{bmatrix} i/\sqrt{2} \\ 1/2 \\ 1/2 \end{bmatrix}, \qquad \begin{bmatrix} \sqrt{2/3} \\ 0 \\ -\sqrt{1/3} \end{bmatrix} \xrightarrow{\mathrm{U}} \begin{bmatrix} 1/\sqrt{3} \\ 0 \\ -i\sqrt{2/3} \end{bmatrix}. \tag{128}$$

Something interesting happens in our third example above, a result of the state containing quantum amplitudes not probabilities. The final amplitude for q_2 is 0. There are starting amplitudes for q_1 and q_3. Each of q_1 and q_3 has an amplitude to transition to q_2 *independently*. Together, these amplitudes cancel to give 0. This cannot happen in the classical setting. If q_1, q_3 have probabilities to transition to q_2 individually, then together those probabilities add, and cannot possibly give 0. This cancellation happens in the quantum system because we are adding amplitudes which are complex numbers. Non-zero amplitudes adding to give 0 is called interference. This is a long known phenomenon in wave propagation. Quantum systems have wave-like properties.

You can now square the final amplitudes to get the new probabilities to be in each vertex for each of our examples above. Notice, in the classical setting the probabilities directly evolve into probabilities under action by T. In the quantum setting it is not possible to find a linear operator to evolve probabilities to probabilities. In the background you have the quantum state which is the amplitudes. In the background these amplitudes evolve to new amplitudes under the action of the linear operator U. Once the evolution of amplitudes is done, we can get the probabilities of being each vertex. If we measure the state, we will get the probabilities to be in each vertex.

Classical We can calculate the new probabilities from previous probabilities. We can actually see the ball as it makes its transitions from one vertex to another governed by these probabilities.

Quantum We do not see the amplitudes. The amplitudes evolve in the background. We do not see where the particle is through this evolution. In the end, we measure the particle and the amplitude-norms tell us the probabilities to observe the particle in a vertex. After we take the measurement, the amplitudes collapse to a pure state.

9.3 Ensembles of Independent Particles

What happens if we have two particles? Consider two independent balls following the quantum dynamics in the previous section. Label the particles' starting amplitudes as $\boldsymbol{a}$ and $\boldsymbol{b}$,

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \overset{\mathrm{U}}{\to} \mathrm{U}\boldsymbol{a}, \qquad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \overset{\mathrm{U}}{\to} \mathrm{U}\boldsymbol{b}. \tag{129}$$

The particles each evolve independently under U. First, how do we represent the state of the two-particle system. Since each particle can be in $\{q_1, q_2, q_3\}$, there are 9 possibilities for the pair,

$$\{q_1q_1, q_1q_2, q_1q_3, q_2q_1, q_2q_2, q_2q_3, q_3q_1, q_3q_2, q_3q_3\}. \tag{130}$$

These are the so-called product states, obtained by taking the Cartesian product of the states available to $\boldsymbol{a}$ and the states available to $\boldsymbol{b}$,

$$\{q_1, q_2, q_3\} \times \{q_1, q_2, q_3\}. \tag{131}$$

The particles are independent, so

$$\mathbb{P}[\boldsymbol{a} \text{ is in } q_i \text{ and } \boldsymbol{b} \text{ is in } q_j] = \mathbb{P}[\boldsymbol{a} \text{ is in } q_i] \times \mathbb{P}[\boldsymbol{b} \text{ is in } q_j] = \|a_i\|^2 \|b_j\|^2. \tag{132}$$

But, since $\|a_i b_j\|^2 = \|a_i\|^2 \|b_j\|^2$, we obtain the correct probabilities if the amplitudes multiply. Thus, the state of the two-particle system is represented by the vector of amplitudes

$$\begin{matrix} q_1q_1 \\ q_1q_2 \\ q_1q_3 \\ q_2q_1 \\ q_2q_2 \\ q_2q_3 \\ q_3q_1 \\ q_3q_2 \\ q_3q_3 \end{matrix} \begin{bmatrix} a_1b_1 \\ a_1b_2 \\ a_1b_3 \\ a_2b_1 \\ a_2b_2 \\ a_2b_3 \\ a_3b_1 \\ a_3b_2 \\ a_3b_3 \end{bmatrix} = \begin{bmatrix} a_1\boldsymbol{b} \\ a_2\boldsymbol{b} \\ a_3\boldsymbol{b} \end{bmatrix}. \tag{133}$$

The final vector on the right is the tensor product $\boldsymbol{a} \otimes \boldsymbol{b}$ (also called the Kroneker product),

$$\boldsymbol{a} \otimes \boldsymbol{b} = \begin{bmatrix} a_1\boldsymbol{b} \\ a_2\boldsymbol{b} \\ a_3\boldsymbol{b} \end{bmatrix}. \tag{134}$$

You take $\boldsymbol{a}$ and multiply each component of $\boldsymbol{a}$ by an *entire* $\boldsymbol{b}$. Note that the possible joint states are also a tensor product, $\mathbf{b} \otimes \mathbf{b}$. The amplitudes for a system of 2 independent particles is the tensor product of the individual amplitudes,

$$|\psi_{\boldsymbol{ab}}\rangle = |\psi_{\boldsymbol{a}}\rangle \otimes |\psi_{\boldsymbol{b}}\rangle. \tag{135}$$

This is the case for *independent* particles. Not every 9-dimensional vector of amplitudes is the tensor product of two 3-dimensional amplitudes – the reader should construct an example. If the

state of the two particle system is represented by such a 9-dimensional vector of amplitudes, those two particles cannot be independent. We say that those particles are entangled, and the state is an entangled state. Generalizing to three particles, $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$, the joint state is

$$\boldsymbol{a} \otimes \boldsymbol{b} \otimes \boldsymbol{c}. \tag{136}$$

We encourage the reader to show that if the individual amplitudes $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ are normalized, then so is the tensor product $\boldsymbol{a} \otimes \boldsymbol{b} \otimes \boldsymbol{c}$.

What happens under independent evolution? The new states are $\mathrm{U}\boldsymbol{a}$ and $\mathrm{U}\boldsymbol{b}$. So, from our prior discussion, the new joint state is $\mathrm{U}\boldsymbol{a} \otimes \mathrm{U}\boldsymbol{b}$. Let us first compute $\mathrm{U}\boldsymbol{a}$,

$$\mathrm{U}\boldsymbol{a} = \begin{bmatrix} U_{11}a_1 + U_{12}a_2 + U_{13}a_3 \\ U_{21}a_1 + U_{22}a_2 + U_{23}a_3 \\ U_{31}a_1 + U_{32}a_2 + U_{33}a_3 \end{bmatrix}. \tag{137}$$

We can now compute $\mathrm{U}\boldsymbol{a} \otimes \mathrm{U}\boldsymbol{b}$ by multiplying each component of $\mathrm{U}\boldsymbol{a}$ with an entire copy of $\mathrm{U}\boldsymbol{b}$. Therefore, we have that

$$\mathrm{U}\boldsymbol{a} \otimes \mathrm{U}\boldsymbol{b} = \begin{bmatrix} (U_{11}a_1 + U_{12}a_2 + U_{13}a_3)\mathrm{U}\boldsymbol{b} \\ (U_{21}a_1 + U_{22}a_2 + U_{23}a_3)\mathrm{U}\boldsymbol{b} \\ (U_{31}a_1 + U_{32}a_2 + U_{33}a_3)\mathrm{U}\boldsymbol{b} \end{bmatrix}. \tag{138}$$

Can we write this as the evolution of the 2-particle state $\boldsymbol{a} \otimes \boldsymbol{b}$ under some operator V,

$$\mathrm{U}\boldsymbol{a} \otimes \mathrm{U}\boldsymbol{b} = \mathrm{V}(\boldsymbol{a} \otimes \boldsymbol{b}). \tag{139}$$

Indeed we can, and the question is what is V? Note that

$$\begin{aligned} (U_{11}a_1 + U_{12}a_2 + U_{13}a_3)\mathrm{U}\boldsymbol{b} &= (U_{11}\mathrm{U}a_1\boldsymbol{b} + U_{12}\mathrm{U}a_2\boldsymbol{b} + U_{13}\mathrm{U}a_3\boldsymbol{b}) && (140) \\ &= \begin{bmatrix} U_{11}\mathrm{U} & U_{12}\mathrm{U} & U_{13}\mathrm{U} \end{bmatrix} \begin{bmatrix} a_1\boldsymbol{b} \\ a_2\boldsymbol{b} \\ a_3\boldsymbol{b} \end{bmatrix} && (141) \\ &= \begin{bmatrix} U_{11}\mathrm{U} & U_{12}\mathrm{U} & U_{13}\mathrm{U} \end{bmatrix} \boldsymbol{a} \otimes \boldsymbol{b}. && (142) \end{aligned}$$

Similarly, we can write the other components in (138) to get

$$\mathrm{U}\boldsymbol{a} \otimes \mathrm{U}\boldsymbol{b} = \underbrace{\begin{bmatrix} U_{11}\mathrm{U} & U_{12}\mathrm{U} & U_{13}\mathrm{U} \\ U_{21}\mathrm{U} & U_{22}\mathrm{U} & U_{23}\mathrm{U} \\ U_{31}\mathrm{U} & U_{32}\mathrm{U} & U_{33}\mathrm{U} \end{bmatrix}}_{\mathrm{V}} \boldsymbol{a} \otimes \boldsymbol{b}. \tag{143}$$

We can now identify V. To obtain V, start with U and multiply each entry in U by an entire copy of the full matrix U. This is very similar to the tensor product of two vectors, and we can define the tensor product of two matrices $\mathrm{A} \otimes \mathrm{B}$: start with A and multiply every entry by an entire copy of B (A and B need not have the same dimensions),

$$\mathrm{A} \otimes \mathrm{B} = \begin{bmatrix} A_{11}\mathrm{B} & A_{12}\mathrm{B} & \cdots & A_{1m}\mathrm{B} \\ A_{21}\mathrm{B} & A_{22}\mathrm{B} & \cdots & A_{2m}\mathrm{B} \\ \vdots & & & \\ A_{d1}\mathrm{B} & A_{d2}\mathrm{B} & \cdots & A_{dm}\mathrm{B} \end{bmatrix}. \tag{144}$$

If A is $d \times m$ and B is $\ell \times n$ then $\mathrm{A} \otimes \mathrm{B}$ is $d\ell \times mn$. The tensor product of two vectors is a special case of the tensor product of two matrices.

Getting back to the evolution of k independent particles, if you have independent particles with amplitudes given by the states $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_k$, the joint state is the tensor product

$$\boldsymbol{a}_1 \otimes \boldsymbol{a}_2 \otimes \cdots \otimes \boldsymbol{a}_k. \tag{145}$$

If the particles independently evolve according to their own unitary matrices $\mathrm{U}_1, \ldots, \mathrm{U}_k$, then the evolution of the joint state is driven by the tensor product $\mathrm{U}_1 \otimes \cdots \otimes \mathrm{U}_k$,

$$\begin{aligned} \boldsymbol{a}_1 \otimes \boldsymbol{a}_2 \otimes \cdots \otimes \boldsymbol{a}_k &\rightarrow \mathrm{U}_1\boldsymbol{a}_1 \otimes \mathrm{U}_2\boldsymbol{a}_2 \otimes \cdots \otimes \mathrm{U}_k\boldsymbol{a}_k & (146)\\ &= (\mathrm{U}_1 \otimes \cdots \otimes \mathrm{U}_k)(\boldsymbol{a}_1 \otimes \boldsymbol{a}_2 \otimes \cdots \otimes \boldsymbol{a}_k). & (147) \end{aligned}$$

It is an exercise for the reader to show that the tensor product of unitary matrices is unitary. This implies that the normalized joint tensor product state remains normalized.[2] We also leave it to the reader to show that the tensor product is associative,

$$\mathrm{A} \otimes (\mathrm{B} \otimes \mathrm{C}) = (\mathrm{A} \otimes \mathrm{B}) \otimes \mathrm{C}. \tag{148}$$

Note, the tensor product is generally not commutative, $\mathrm{A} \otimes \mathrm{B} \neq \mathrm{B} \otimes \mathrm{A}$ (find a counter example). The two matrices $\mathrm{A} \otimes \mathrm{B}$ and $\mathrm{B} \otimes \mathrm{A}$ do however have the same entries and they can be transformed into each other by a permutation of rows and columns. We also leave it to the reader to show that the tensor product distributes over addition,

$$\begin{aligned} \mathrm{A} \otimes (\mathrm{B}_1 + \mathrm{B}_2) &= \mathrm{A} \otimes \mathrm{B}_1 + \mathrm{A} \otimes \mathrm{B}_2;\\ (\mathrm{A}_1 + \mathrm{A}_2) \otimes \mathrm{B} &= \mathrm{A}_1 \otimes \mathrm{B} + \mathrm{A}_2 \otimes \mathrm{B}. \end{aligned} \tag{149}$$

[2] It implies more, that the norm of all vectors are preserved when acted upon by $\mathrm{U}_1 \otimes \cdots \otimes \mathrm{U}_k$, not just vectors which are tensor products.

10 Classical Computing Using Linear Algebra

To move toward quantum computing, we start with classical computing and generalize. This will become seemless within a linear algebraic framework using the considerable machinary in linear algebra that we have built and since quantum dynamics has a nice linear algebraic formulation based on hermitian operators for observables and unitary operators for evolution of state.

10.1 The Bit

With an eye on the postulates of quantum mechanics, we realize that to measure a bit as 0 or 1 (an observable with two possible values), the bit-operator for this observable must have at least two eigenvectors with different eigenvalues. Hence, the minimum dimension of this operator is 2 and we may as well assume the eigenvectors are the standard basis, $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ for value 0 and $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$ for value 1.

10.2 Classical Bits

The classical bit is in one of two states, $|0\rangle$ and $|1\rangle$. In the 2-dimensional eigenvector representation,

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}, \qquad |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}. \tag{150}$$

This won't change when we move to the quantum realm. The only change will be that the classical bits are the standard basis vectors in a 2-dimensional complex vector space. What about a two-bit system, for example the first bit is 0 and the second is 1, $|01\rangle$,

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 1\begin{bmatrix}0\\1\end{bmatrix} \\ 0\begin{bmatrix}0\\1\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}. \tag{151}$$

All the possible 2-bit states correspond to all the four standard basis vectors in 4-dimensions,

$$\begin{matrix}|00\rangle & |01\rangle & |10\rangle & |11\rangle \\ \begin{bmatrix}1\\0\\0\\0\end{bmatrix} & \begin{bmatrix}0\\1\\0\\0\end{bmatrix} & \begin{bmatrix}0\\0\\1\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\1\end{bmatrix}\end{matrix}. \tag{152}$$

This generalizes to an n-bit system. The classical state has n-bits. There are 2^n of these. In the vector representation, these classical states are the standard basis vectors in 2^n-dimensions,

$$\begin{matrix}|00\cdots00\rangle & |00\cdots01\rangle & |00\cdots10\rangle & \cdots & |11\cdots10\rangle & |11\cdots11\rangle \\ \begin{bmatrix}1\\0\\0\\\vdots\\0\\0\end{bmatrix} & \begin{bmatrix}0\\1\\0\\\vdots\\0\\0\end{bmatrix} & \begin{bmatrix}0\\0\\1\\\vdots\\0\\0\end{bmatrix} & \cdots & \begin{bmatrix}0\\0\\0\\\vdots\\1\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\\vdots\\0\\1\end{bmatrix}\end{matrix}. \tag{153}$$

We can leverage the vector representation and allow any state $|\psi\rangle$,

$$|\psi\rangle = \begin{bmatrix} p_0 \\ p_1 \end{bmatrix}. \tag{154}$$

Now, p_0 is the probability the bit is $|0\rangle$ and p_1 the probability the bit is $|1\rangle$, $p_0 + p_1 = 1$. For a two probabilistic bit system with $|\psi\rangle = \left[\begin{smallmatrix} p_0 \\ p_1 \end{smallmatrix}\right]$ and $|\phi\rangle = \left[\begin{smallmatrix} t_0 \\ t_1 \end{smallmatrix}\right]$ being independent bits,

$$|\psi\phi\rangle = |\psi\rangle \otimes |\phi\rangle = \begin{bmatrix} p_0 \begin{bmatrix} t_0 \\ t_1 \end{bmatrix} \\ \\ p_1 \begin{bmatrix} t_0 \\ t_1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} p_0 t_0 \\ p_0 t_1 \\ p_1 t_0 \\ p_1 t_1 \end{bmatrix} = \begin{bmatrix} \mathbb{P}[|00\rangle] \\ \mathbb{P}[|01\rangle] \\ \mathbb{P}[|10\rangle] \\ \mathbb{P}[|11\rangle] \end{bmatrix}. \tag{155}$$

You never see a probabilistic bit. The probabilistic bit $|\psi\rangle$ just represents our uncertainty about what the bit is. When you "measure" the state, you will always see either the bit $|0\rangle$ or $|1\rangle$.

10.3 Quantum Bits

The leap from classical bits to quantum bits using the vector representation is trivial. We simply replace probabilities with amplitudes and the state of a qubit is now a complex vector,

$$|\psi\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}, \tag{156}$$

where a_0 is the amplitude for $|0\rangle$ and a_1 is the amplitude for $|1\rangle$. You never see a qubit. You only "measure" a qubit, producing a bit, either $|0\rangle$ or $|1\rangle$. The probability to get $|0\rangle$ is $\|a_0\|^2$, and the probability to get $|1\rangle$ is $\|a_1\|^2$, so $\|a_0\|^2 + \|a_1\|^2 = 1$. A general representation for a qubit is

$$|\psi\rangle = \begin{bmatrix} \cos\theta \\ e^{i\phi}\sin\theta \end{bmatrix}. \tag{157}$$

For a two qubit system $|\psi\rangle = \left[\begin{smallmatrix} a_0 \\ a_1 \end{smallmatrix}\right]$ and $|\phi\rangle = \left[\begin{smallmatrix} b_0 \\ b_1 \end{smallmatrix}\right]$, the joint state is

$$|\psi\phi\rangle = |\psi\rangle \otimes |\phi\rangle = \begin{bmatrix} a_0 \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \\ \\ a_1 \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_0 b_0 \\ a_0 b_1 \\ a_1 b_0 \\ a_1 b_1 \end{bmatrix}; \qquad \begin{bmatrix} \mathbb{P}[|00\rangle] \\ \mathbb{P}[|01\rangle] \\ \mathbb{P}[|10\rangle] \\ \mathbb{P}[|11\rangle] \end{bmatrix} = \begin{bmatrix} \|a_0 b_0\|^2 \\ \|a_0 b_1\|^2 \\ \|a_1 b_0\|^2 \\ \|a_1 b_1\|^2 \end{bmatrix}; . \tag{158}$$

Consider the joint state

$$\begin{bmatrix} a_0 b_0 \\ a_0 b_1 \\ a_1 b_0 \\ a_1 b_1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ 0 \\ 1/\sqrt{2} \end{bmatrix}. \tag{159}$$

This is a valid set of amplitudes because their squared norms sum to 1. However, this state is not the tensor product of two single qubit states. This is because $a_0 b_1 = 0 \to a_0 = 0$ or $b_1 = 0$, and both cases are not possible: If $a_0 = 0$ then $a_0 b_0 \neq 1/\sqrt{2}$; if $b_1 = 0$ then $a_1 b_1 \neq 1/\sqrt{2}$. So this joint state cannot be from two independent qubits. Such qubits are called entangled.

How do we implement a qubit in practice? The requirements are we need to create it and keep it stable.[3] We need to evolve the qubit according to a "quantum algorithm". We need to measure the final qubit without significantly perturbing the state. Some possibilities are:

- Electron orbits in small atoms. The ground-state orbit is $|0\rangle$ and the excited state is $|1\rangle$.
- Photon polarization, left-right or up-down (to store two states)
- Intrinsic spin, e.g., electron-spin is up or down in the direction of measurement.

The last word on implementing qubits is an ongoing research question.

10.4 Classical Computing Gates

Bits and qubits are both represented as vectors. Classical computing gates operate on input bits to produce output bits. We show how to represent classical gates as matrices so that their action on input bits can be implemented by multiplying the corresponding matrix with the input bits represented as a vector. Let us begin with NOT,

$$|\psi\rangle \longrightarrow \boxed{\text{NOT}} \longrightarrow |\xi\rangle$$

$$\begin{array}{cc} |0\rangle \to |1\rangle & |1\rangle \to |0\rangle \\ \begin{bmatrix}1\\0\end{bmatrix} \to \begin{bmatrix}0\\1\end{bmatrix} & \begin{bmatrix}0\\1\end{bmatrix} \to \begin{bmatrix}1\\0\end{bmatrix} \end{array} \tag{160}$$

The action of $\boxed{\text{NOT}}$ can be implemented by the matrix

$$\boxed{\text{NOT}} = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}. \tag{161}$$

This is verified by

$$\boxed{\text{NOT}} \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}. \tag{162}$$

The columns of the $\boxed{\text{NOT}}$ are constructed directly from its action on the standard basis vectors, which are the classical bits. $\boxed{\text{NOT}}$ can now be applied to any state $|\psi\rangle = \left[\begin{smallmatrix}a_0\\ a_1\end{smallmatrix}\right]$,

$$\boxed{\text{NOT}} \cdot |\psi\rangle = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \begin{bmatrix}a_0\\ a_1\end{bmatrix} = \begin{bmatrix}a_1\\ a_0\end{bmatrix}. \tag{163}$$

The action of $\boxed{\text{NOT}}$ is simply to flip amplitudes. Let us now consider AND,

$$\begin{array}{l} |\psi\rangle \\ |\phi\rangle \end{array} \longrightarrow \boxed{\text{AND}} \longrightarrow |\xi\rangle$$

$$\begin{array}{cccc} |00\rangle \to |0\rangle & |01\rangle \to |0\rangle & |10\rangle \to |0\rangle & |11\rangle \to |1\rangle \\ \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \to \begin{bmatrix}1\\0\end{bmatrix} & \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \to \begin{bmatrix}1\\0\end{bmatrix} & \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \to \begin{bmatrix}1\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \to \begin{bmatrix}0\\1\end{bmatrix} \end{array} \tag{164}$$

[3] Classical computing would be a disaster if bit were unstable and kept flipping at random.

The action of $\boxed{\text{AND}}$ can be implemented by the matrix

$$\boxed{\text{AND}} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{165}$$

This is verified by

$$\boxed{\text{AND}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{166}$$

The columns of the $\boxed{\text{AND}}$ are constructed directly from its action on the standard basis vectors (in 4-dimensions), which are the classical bits. $\boxed{\text{AND}}$ can now be applied to an arbitrary 2-bit state,

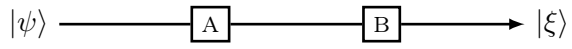
$$\boxed{\text{AND}} \cdot |\psi\rangle = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} a_0 + a_1 + a_2 \\ a_3 \end{bmatrix}. \tag{167}$$

We leave it for the reader to show

$$\boxed{\text{OR}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}, \qquad \boxed{\text{NAND}} = \boxed{\text{NOT}} \cdot \boxed{\text{AND}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}. \tag{168}$$

10.5 Circuits

A circuit combines gates sequentially (in series) or in parallel. First let us consider sequential,

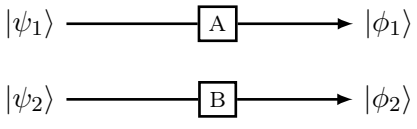


(169)

The linear operator for this circuit is obtained from the individual linear operators by multiplication,

$$|\xi\rangle = BA|\psi\rangle. \tag{170}$$

For the parallel case,

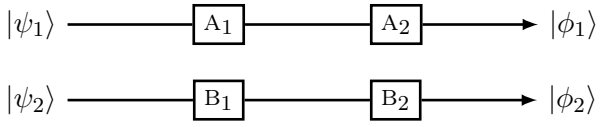


(171)

The linear operator for this circuit is obtained by using the tensor product,

$$|\psi_1\rangle \otimes |\psi_2\rangle \to A|\psi_1\rangle \otimes B|\psi_2\rangle = (A \otimes B)(|\psi_1\rangle \otimes |\psi_2\rangle). \tag{172}$$

We can combine sequential and parallel,

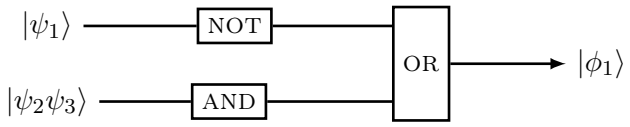


(173)

The linear operator for this circuit is a tensor product of products,

$$|\psi_1\rangle \otimes |\psi_2\rangle \to A_2A_1|\psi_1\rangle \otimes B_2B_1|\psi_2\rangle = (A_2A_1 \otimes B_2B_1)(|\psi_1\rangle \otimes |\psi_2\rangle). \tag{174}$$

Let us do an example. Consider the circuit,

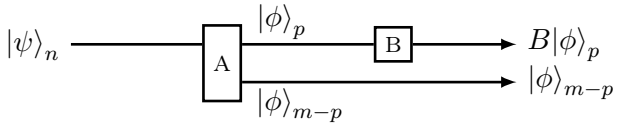


(175)

The linear operator for this circuit is $\boxed{\text{OR}} \cdot \left(\boxed{\text{NOT}} \otimes \boxed{\text{AND}}\right)$, and the reader can verify that

$$\boxed{\text{OR}} \cdot \left(\boxed{\text{NOT}} \otimes \boxed{\text{AND}}\right) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{176}$$

One last example. Often we apply a circuit, take some of the bits into one circuit and leave the others alone. Suppose the input is n-bits, and the first circuit outputs m bits. We take the first p of those m bits into another circuit and leave the others alone.



(177)

We can analyze this circuit as follows. The output of A is $|\phi\rangle_p \otimes |\phi\rangle_{m-p} = A|\psi\rangle_n$. Then we operate in parallel on $|\phi\rangle_p$ and $|\phi\rangle_{m-p}$,

$$|\phi\rangle_p \otimes |\phi\rangle_{m-p} \to B|\phi\rangle_p \otimes |\phi\rangle_{m-p} = B|\phi\rangle_p \otimes \mathrm{I}_{2^{m-p}}|\phi\rangle_{m-p} = (B \otimes \mathrm{I}_{2^{m-p}}) \underbrace{(|\phi\rangle_p \otimes |\phi\rangle_{m-p})}_{A|\psi\rangle_n} \tag{178}$$

That is, parallel to B, we are implementing the identity. We have,

$$|\phi\rangle_p \otimes |\phi\rangle_{m-p} \to (B \otimes \mathrm{I}_{2^{m-p}}) \cdot A \cdot |\psi\rangle_n. \tag{179}$$

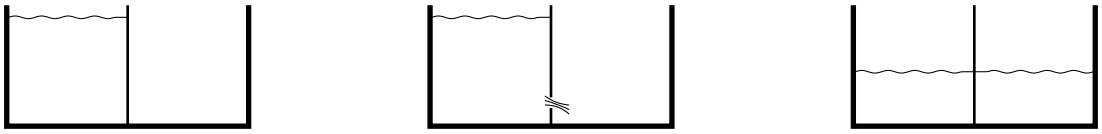
This circuit is implemented by the linear operator $(B \otimes \mathrm{I}_{2^{m-p}}) \cdot A$.

11 Reversible Gates and Quantum Gates

We saw some classical gates. We now take the bridge over to quantum gates and prove our first little result relating to quantum computing, namely **no cloning**. Let us begin with reversible gates.

11.1 Reversible Gates

NOT is reversible, from the output you can determine the input. AND is not reversible. From the output, you cannot always determine the input. If the output is $|0\rangle$, you do not know if the input was $|00\rangle, |01\rangle$ or $|10\rangle$. Information is lost. Erasing information dissipates energy in the form of heat. This is based on statistical thermodynamic considerations and the 2nd law of thermodynamics. Here is an intuition of why erasing information dissipates energy. We show a bucket with two compartments that can be used to store 1 bit of information, "left" or "right". The bit starts "left".



Open a hole in the middle barrier and water drains to the right-compartment. This water flow can power a turbine generating energy until the water levels equalize, at which point we have erased the information in the bit. We now can't tell whether the bit started "left" or "right". Erasing the information dissipated the energy stored in the bit.

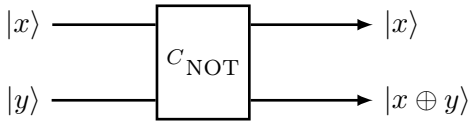
Bennet had the idea that if one could do computing with reversible processes, then energy is not lost – the ultimate in green computing. Landaurer gave a lower bound on the minimum energy dissipation in erasing a bit. The debate is still ongoing whether this lower bound can be achieved or if reversible computation can be accomplished without energy input. Independent of this debate, it is still interesting to see if we can construct reversible classical gates, and further, this is a first step toward quantum gates which requires more than reversible. Quantum gates must also be unitary.

11.1.1 Controlled-NOT

Let us consider XOR, which is not reversible,

$$\text{XOR} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}. \tag{180}$$

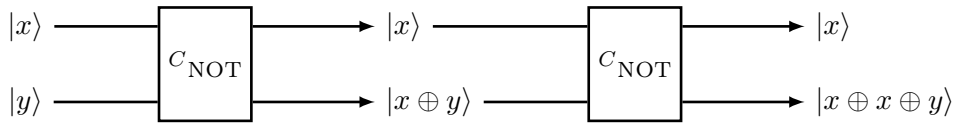
A useful trick to obtain reversible gates is to pass some of the input bits unchanged to the output so that the computation is reversible. Here is the idea in action with XOR,



(181)

We are computing the XOR, but passing through $|x\rangle$ for the sole purpose of being able to reconstruct the input given the output. This gate is called controlled-NOT, even though we are computing XOR

(we will see why soon). The name is not important. You should convince yourself that if you ran the output through another ^{C}NOT gate, you can recover the input,



(182)

The reason we have recovered the input is because $x \oplus x \oplus y = 0 \oplus y = y$. This gate is called the controlled not, because when $|x\rangle = |1\rangle$, the y-output gets negated, otherwise the y-output is unchanged. Hence, whether y is negated is *controlled* by the value of x. This is the linear algebraic equivalent of the "if... else" statement,

$$\text{If } |x\rangle = |1\rangle \text{ then flip } |y\rangle, \text{ else keep } |y\rangle.$$

What is the linear operator for ^{C}NOT. The action of ^{C}NOT on the standard basis vectors is

$$|00\rangle \to |00\rangle \qquad |01\rangle \to |01\rangle \qquad |10\rangle \to |11\rangle \qquad |11\rangle \to |10\rangle, \tag{183}$$

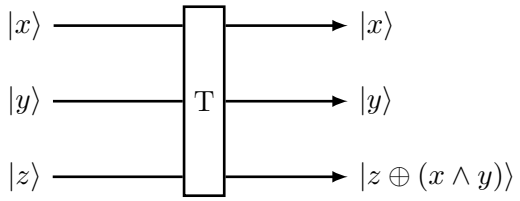
from which we get the operator,

$$^{C}\text{NOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{184}$$

This matrix is its own inverse, $^{C}\text{NOT}^2 = \text{I}_4$. Controlling bits will be crucial in quantum algorithms.

11.1.2 Toffoli Gate

Another interesting reversible gate is the Toffoli gate, with two controlling bits to compute the XOR,

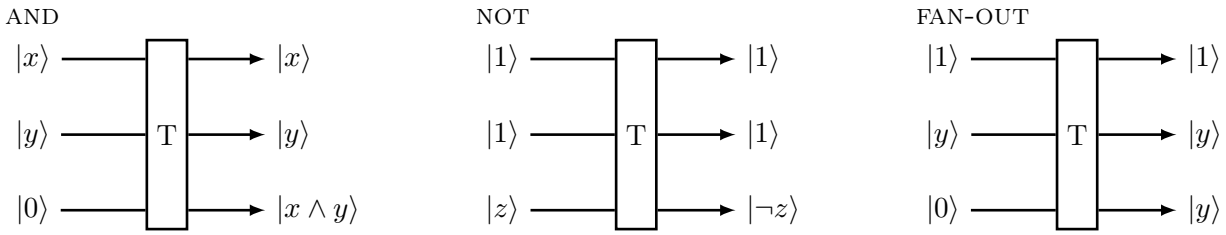


(185)

Using the action of T on the standard basis, you can construct the linear operator for T,

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{186}$$

Setting the inputs appropriately implements AND, NOT and FAN-OUT, as you can verify:



(187)

FAN-OUT is essentially a copy operation. Given these three operations, we can implement any Boolean circuit, since {AND, NOT, FAN-OUT} are a universal set of gates. In practice, AND and NOT alone are universal because we can implement FAN-OUT by just splitting the wire (assuming bits are implemented using voltage and curents). Not only is the Toffoli gate universal, the Toffoli gate is its own inverse, $T^2 = \mathrm{I}_8$, and further T is unitary. There are other universal gates with these properties, for example the Fredkin gate.

It is a useful exercise for the reader to construct the NAND operation using one Toffoli gate, and the OR operation using two Toffoli gates.

11.2 Quantum Gates

Quantum gates are unitary operators that operate on qubits. From the theoretical perspective, that's all there is to it. However, from the practical perspective one has to consider what unitary operators can be implemented efficiently by natural physical processes. So in classical computing, we settled on NAND because we can implement this gate very efficiently (space and time) using semiconductor transistors. If we had not invented semiconductor transistors, the face of computing in terms of what gates we use might look very different today.

So too with quantum gates. We will consider several quantum gates, and there are different sets of universal gates – unitary operators. We can build algorithms by combining these universal gates. But which algorithms will stand the test of time will depend on which quantum gates become efficiently implementable, if any. Quantum computing is a marathon, not a sprint. The general form of a unitary operator for one qubit, up to an overall phase, can be written

$$U = \begin{bmatrix} r & \sqrt{1-r^2}e^{i\phi_1} \\ \sqrt{1-r^2}e^{i\phi_2} & -re^{i(\phi_1+\phi_2)} \end{bmatrix} \tag{188}$$

Here are some example unitary operators for one qubit. Each is a valid quantum gate. Whether these gates can be implemented on a massive scale using natural physical processes will not be our concern. Among the operators we have already seen, the Hadamard, NOT, ^{C}NOT and Toffoli are quantum gates (unitary). Just as a reminder, the Hadamard gate is

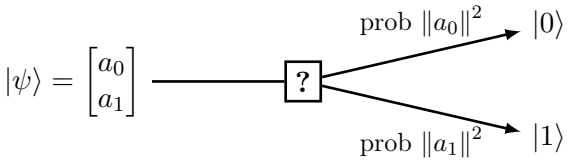
$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{189}$$

Some other valid quantum gates are:

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad R(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix} \tag{190}$$

The operator $R(\theta)$ shifts the phase of the amplitude for $|1\rangle$. The phase gate $S = R(\pi/2)$ is a useful special case because it plays a core role in the Gottesman-Knill theorem. This phase shift-operator is useful for building larger gates from smaller ones.

An important difference between the quantum computing and classical computing worlds is measurement. Measuring a qubit produces a classical bit $|0\rangle$ or $|1\rangle$. In such a case the qubit collapses to the classical bit measured. We will denote this measurement operation $\boxed{?}$,



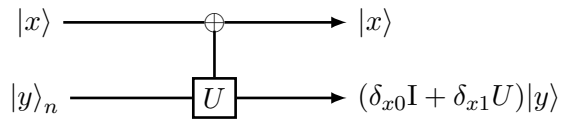
(191)

The measurement operation is non-linear, non-unitary and non-reversible. It returns a classical bit and also collapses the state to that classical bit.

11.2.1 Building Larger Gates

Any unitary operation on an n-qubit state is a valid quantum gate. But, we are not going to go off and build a new massive quantum gate for every quantum algorithm. Instead, we would build a small set of gates and combine those in various ways to get larger gates (unitary operators). This is similar to what we do with classical circuits. We have the core universal gates, for example $\{\text{NAND}\}$, and from those we can build circuits to implement arbitrary Boolean functions. In classical circuit theory the interesting question is what Boolean functions can be implemented with polynomialy many basic gates. A similar issue faces quantum computing. What unitary operators can be implemented using a small set of core quantum gates, and can these unitary operators be classically simulated efficiently. Relevant to these issues are the Solovay-Kitaev theorem and the Gottesman-Knill theorem which we give at the end for informational completeness.

Let us begin with the operations for combining quantum gates (unitary operators). We can combine serially, which is the product. We can combine in parallel, which is the tensor product. There is one other important operation. We saw it before, namely controlling an operator with a bit. If U is a quantum gate, we define the controlled-U, or ${}^{C}U$ as the quantum gate which does nothing if the controlling bit is $|0\rangle$ and implements U if the controlling bit is $|1\rangle$. The controlled gate is the quantum equivalent of the classical IF ... THEN ... controlling mechanism which we are all familiar with from classical programming. The circuit diagram below represents this operation,



(192)

If $U = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$, then by considering the action of ${}^{C}U$ on the standard basis $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ we find

$$\begin{array}{lcl} |0\rangle \otimes |0\rangle & \rightarrow & |0\rangle \otimes |0\rangle \\ |0\rangle \otimes |1\rangle & \rightarrow & |0\rangle \otimes |1\rangle \\ |1\rangle \otimes |0\rangle & \rightarrow & |1\rangle \otimes U|0\rangle = |1\rangle \otimes (a|0\rangle + c|1\rangle) \\ |1\rangle \otimes |1\rangle & \rightarrow & |1\rangle \otimes U|1\rangle = |1\rangle \otimes (b|0\rangle + d|1\rangle). \end{array} \tag{193}$$

The reader can now verify that

$$ {}^{C}U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix} = \begin{bmatrix} \mathrm{I} & 0 \\ 0 & U \end{bmatrix}. \tag{194} $$

In general, when $|y\rangle$ is an n-qubit state, adding the controlling bit doubles the size of the state and the dimensions of the operator, producing the controlled operator

$$ {}^{C}U = \begin{bmatrix} \mathrm{I}_{2^n} & 0_{2^n \times 2^n} \\ 0_{2^n \times 2^n} & U \end{bmatrix}. \tag{195} $$

Notice, that since U is unitary, so is ${}^{C}U$,

$$ ({}^{C}U)^{\dagger} {}^{C}U = \begin{bmatrix} \mathrm{I}_{2^n} & 0_{2^n \times 2^n} \\ 0_{2^n \times 2^n} & U^{\dagger} \end{bmatrix} \begin{bmatrix} \mathrm{I}_{2^n} & 0_{2^n \times 2^n} \\ 0_{2^n \times 2^n} & U \end{bmatrix} = \begin{bmatrix} \mathrm{I}_{2^n} & 0_{2^n \times 2^n} \\ 0_{2^n \times 2^n} & U^{\dagger}U \end{bmatrix} = \mathrm{I}_{2^{n+1}}. \tag{196} $$

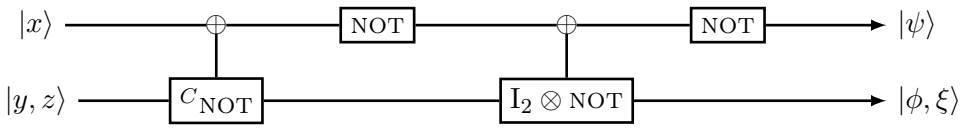
Here are some of the basic one and two qubit quantum gates

$$ \mathrm{I}_2, \quad \textsc{not}, \quad H_1, \quad R(\cos^{-1}(3/5)); \qquad\qquad \mathrm{I}_4, \quad {}^{C}\textsc{not}, \quad H_2. \tag{197} $$

It turns out that these gates are universal, in that they can be combined to approximate any unitary operator within accuracy ϵ. The number of gates needed is $O(4^n \log(1/\epsilon))$, which is essentially the content of the Solovay-Kitaev theorem. The Gottesman-Knill theorem says that if you combine only ${}^{C}\textsc{not}, H, R(\pi/2)$, which are generators of the Clifford algebra, then this quantum gate can be polynomialy simulated on a classical computer.

11.2.2 Practice

To get some practice with quantum gates and building unitary operators from basic quantum gates, see if you can use combine the gates in (197) to implement the unitary operator in (216) on page 55. Remember that a controlling bit can be used to operate on the other bits in one case and not in another. A controlling bit is the quantum equivalent of the IF ... THEN .... We suggest you try to interpret and analyze the following circuit, where x, y, z are bits.



(198)

You may review the mechanics of a controlling bit in the discussion before eq. (192) on page 50.

More generally, Let A and B be two unitary operators on n bits. A and B are $2^n \times 2^n$ matrices. Show how to use a controlling bit to implemet the unitary operator

$$ \begin{bmatrix} A & \mathbf{0}_{2^n \times 2^n} \\ \mathbf{0}_{2^n \times 2^n} & B \end{bmatrix} \tag{199} $$

11.3 No Cloning Theorem

Before the fancy quantum stuff starts, let us first see one limitation of quantum gates. In classical computing you can move or copy a file (bits). In quantum computing, you can only move, not copy. Star Trek got it right. When Kirk teleports down to a plannet the copy of Kirk on the Enterprise must disappear. In quantum computing, cut-and-paste is possible but copy-and-paste is not.

Theorem 11.1 (No Cloning)**.** There is no unitary operator that can copy a quantum state exactly.

Let's prove this. In general, what would it mean to clone a qubit. Something like:

$$|x\rangle \otimes |0\rangle \xrightarrow{U} |x\rangle \otimes |x\rangle. \tag{200}$$

You have a joint state with two qubits. The first state is the qubit you would like to clone, and the second starts in some default state, say $|0\rangle$. Under the action of U, the second qubit becomes an exact copy of x. Let us consider an arbitrary state $|x\rangle = c_0|0\rangle + c_1|1\rangle$.

$$\begin{aligned} U((c_0|0\rangle + c_1|1\rangle) \otimes |0\rangle) &= (c_0|0\rangle + c_1|1\rangle) \otimes (c_0|0\rangle + c_1|1\rangle) && (201)\\ &= \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \otimes \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} c_0^2 \\ c_0 c_1 \\ c_1 c_0 \\ c_1^2 \end{bmatrix}. && (202) \end{aligned}$$

But, U is a linear operator and tensor product is also a linear operator, so

$$\begin{aligned} U((c_0|0\rangle + c_1|1\rangle) \otimes |0\rangle) &= U(c_0|0\rangle \otimes |0\rangle + c_1|1\rangle \otimes |0\rangle) && (203)\\ &= U(c_0|0\rangle \otimes |0\rangle) + U(c_1|1\rangle \otimes |0\rangle) && (204)\\ &= c_0|0\rangle \otimes c_0|0\rangle + c_1|1\rangle \otimes c_1|1\rangle && (205)\\ &= \begin{bmatrix} c_0 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} c_0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ c_1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ c_1 \end{bmatrix} = \begin{bmatrix} c_0^2 \\ 0 \\ 0 \\ c_1^2 \end{bmatrix}. && (206) \end{aligned}$$

The first step is because the tensor product is linear; the second step is because U is linear; the third step is by definition of the copy operator. We have a contradiction (two different expressions for the same quantity) unless one of c_0 or c_1 are zero, in which case we are cloning a classical bit. No linear operator can clone an arbitrary qubit, but you can clone classical bits (e.g., the Toffoli gate).

12 Unitary Operator for Classical Functions

The basic framework for a typical quantum algorithm is:

1. Initialize qubits into pure classical states.
2. Place the qubits into a superposition of states.
3. Run the quantum algorithm (unitary operator) on the superposition of states.
4. Measure the final qubits to get the answer.

In theory steps 2 and 3 can be represented by a single unitary operator. But, step 2 is often generic, whereas step 3 is the one which will depend intricately on the problem we are solving, hence we keep them separate. In a typical problem we have some classical Boolean function f and we ask some question of this function. The main speedup from quantum algorithms comes from step 2. By running the function on a superposition of classical input states, the algorithm somehow gets global information about f on all inputs. This is because of linearity. When f is applied to a superposition, you get the superposition of the outputs when f is individually applied to each classical state. So, we get information about f on all these classical states simultaneously. The task is then to unravel all this data to get the information we need to answer the question about f.

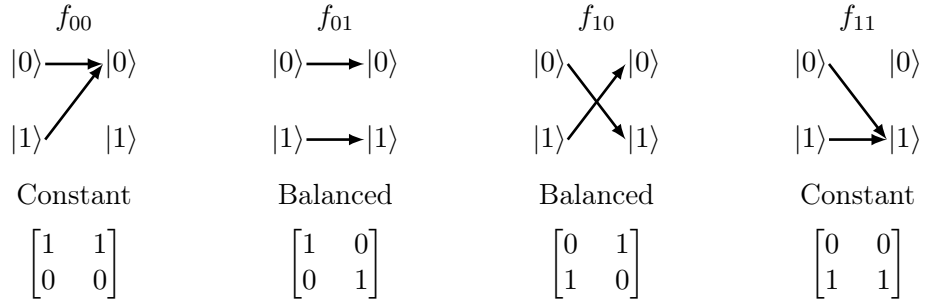
If you want a simple analogy from the physical world, go outside to take a look around. You have just one eye (okay two eyes), but play along, or just close one eye. This eye is receiving all the electromagnetic radiation bouncing off every object. All these EM rays are simultaneously hitting every retinal cell from every object. So, your eye is applying the "get-EM-radiation"-function simultaneously to every object in your environment. You then untangle all this data simultaneously arriving from every object to get specific information about specific objects, like the red bird is 2 meters away at my 2-o'clock.

We will motivate the general workflow with a simple, albeit contrived, example. First, what is the analog of a Boolean function in the quantum realm? An arbitrary Boolean function will rarely be invertible, let alone unitary. This won't do in the quantum realm, because quantum gates are unitary. Our first task will be to convert the Boolean function into a unitary operator, and then extract whatever property we need about f from this unitary operator.

12.1 The Deutsch-Jozsa Problem

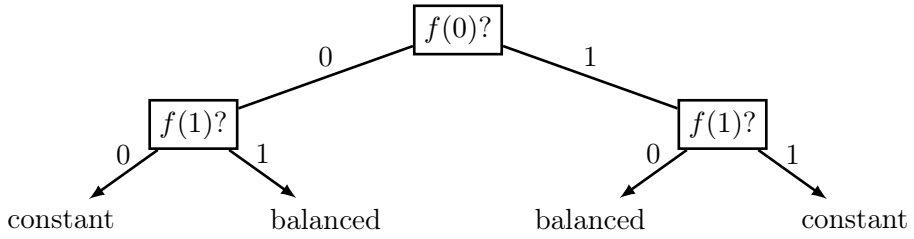
Let f be a Boolean function on n bits, $f : \{0,1\}^n \mapsto \{0,1\}$. The function is constant if $f(x) = |0\rangle$ for all $x \in \{0,1\}^n$ or $f(x) = |1\rangle$ for all $x \in \{0,1\}^n$. The function is balanced if $f(x) = |0\rangle$ for half of the $x \in \{0,1\}^n$ and $f(x) = |1\rangle$ for the other half.

Of course there are all kinds of other functions in between balanced and constant. Note that there are 2^{2^n} Boolean functions on n bits. Here are the 4 Boolean functions on 1 bit together with the linear operator for each function:



(207)

The task is to determine if the function is balanced or constant, assuming it is one of the two. The function f is given as a black box linear operator. You cannot see inside the black-box. You can, however, apply the function to any input, and using as few function evaluations as possible, you must determine if f is constant or balanced. Here is an algorithm for the one bit case, $n = 1$.

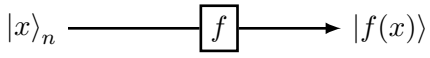


(208)

The algorithm needs to use the black box for f two times. For the general n bit case, evaluate f on any $1 + 2^{n-1}$ different n-bit inputs. If the function ever outputs two different values, it is balanced. Otherwise it is constant. This is exponentially many evaluations. A randomized algorithm is to choose k random n-bit inputs. If f is constant on these k inputs say constant, otherwise balanced. When you say balanced you will always be right. When you say constant, the chances you are wrong is $1/2^{k-1}$. The baby miracle of the day is a quantum algorithm that only needs to use the black box for f once. By applying f just once, it is able to get information about f on all inputs.

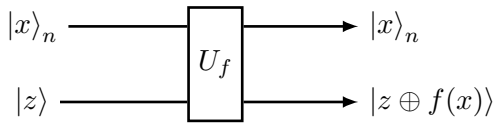
12.2 Converting Boolean Functions to Unitary Operators

The Deutsch-Josza problem is well defined, but it is not yet fit for the quantum realm because as we saw earlier, the function f may not be a unitary operator. Not to worry. One can represent any Boolean function as a unitary operator that effectively computes the function. To do this, we use a controlling bit. The general setup for a classical function taking an n-bit input is



(209)

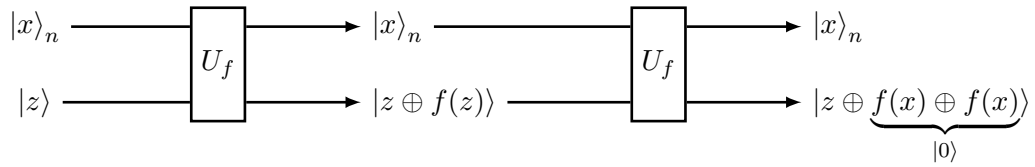
For $n > 1$, such a function takes n bits to one bit and hence cannot be invertible, let alone unitary. We define a unitary implementation of f using a controlling bit z, similar to the way we defined ^{C}NOT. This unitary implementation is U_f,



(210)

Let's examine all the ingredients in this construction. First, the number of input bits is $n + 1$ and the same for the number of output bits. For U_f to be reversible, the output must contain information about $|z\rangle$ and $|x\rangle_n$. Clearly $|x\rangle_n$ can be reconstructed, and information about z is in the $(n + 1)$th bit. The function f is also accessible in U_f by simply setting $|z\rangle = |0\rangle$. The first n qubits are referred to as the input qubits or input registers. The bottom qubit is the output register, because that is where all the information about $f(x)$ is stored. However, that is only the case when a classical pure state is fed into U_f. As we will see, something strange happens when U_f

operates on a superposition. First, lets show that U_f is invertible. Indeed, U_f is its own inverse,



(211)

The final output is $|x\rangle_n \otimes |z\rangle$ because $|z\rangle \oplus |0\rangle = |z\rangle$. Let us now show that U_f is unitary no matter what the Boolean function f is (U_f is also real and hermitian). To see this, let us work with a concrete case. In general an n-bit Boolean function is a linear operator that maps $\mathbb{C}^{2^n} \mapsto \mathbb{C}^2$, and hence is a 2×2^n matrix. Consider the two-bit Boolean function

$$|00\rangle \to |1\rangle, \qquad |01\rangle \to |1\rangle, \qquad |10\rangle \to |0\rangle, \qquad |11\rangle \to |1\rangle, \tag{212}$$

From the action of f on the standard basis, we get the linear operator

$$f = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}. \tag{213}$$

The action of U_f is given by

$$|x, z\rangle \quad \overset{U_f}{\to} \quad |x, z \oplus f(x)\rangle. \tag{214}$$

The action of U_f on the standard basis vectors is given by

$$\underset{\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}}{|000\rangle}\to\underset{\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}}{|001\rangle} \quad \underset{\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}}{|001\rangle}\to\underset{\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}}{|000\rangle} \quad \underset{\begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}}{|010\rangle}\to\underset{\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}}{|011\rangle} \quad \underset{\begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix}}{|011\rangle}\to\underset{\begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix}}{|010\rangle} \quad \underset{\begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}}{|100\rangle}\to\underset{\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}}{|100\rangle} \quad \underset{\begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}}{|101\rangle}\to\underset{\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}}{|101\rangle} \quad \underset{\begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}}{|110\rangle}\to\underset{\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}}{|111\rangle} \quad \underset{\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}}{|111\rangle}\to\underset{\begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}}{|110\rangle} \tag{215}$$

We can now immediately write down the matrix for U_f,

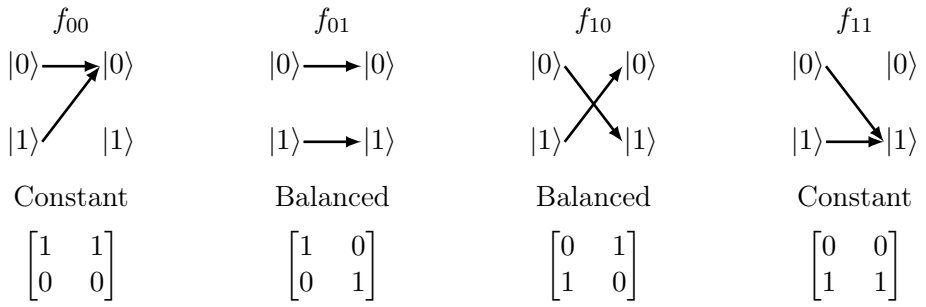
$$U_f = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} \boxed{\text{NOT}} & \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} \\ \mathbf{0}_{2\times2} & \boxed{\text{NOT}} & \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} \\ \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} & \boxed{\text{I}_2} & \mathbf{0}_{2\times2} \\ \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} & \boxed{\text{NOT}} \end{bmatrix}. \tag{216}$$

Observe that U_f has a very nice structure. It is block-diagonal, composed of (2×2)-blocks along the diagonal, 2^n of them. Each block is either NOT or I_2. If the first n bits of the corresponding state map to 1, the block is NOT, otherwise the block is I_2. You should convince yourself that this block-diagonal structure is true in general for any Boolean function on n-bits. From this block-diagonal structure, you should be able to prove that U_f is real, hermitian and unitary.

Summary. In the classical world, f is given as a black-box linear operator. In the quantum world we construct a unitary operator U_f from f using a controlling bit z. U_f is a black box unitary operator implemented by some collection of quantum gates (physical processes). I cannot tell you what those physical processes are – that is, what will quantum hardware look like, but who cares. Lets build quantum algorithms based off unitary operators and hope that some day we can implement these algorithms on quantum hardware. If the quantum hardware (gates) are very different from what we assume here, you will we have the necessary machinary to adapt your algorithms to whatever the quantum hardware be. Just as you learned to program in Pascal. It's no biggie to program those same algorithms in C^{++} or Python.

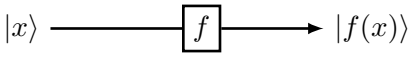
13 Testing Balance of 1-bit Functions

Recall the Deutsch-Jozsa problem for 1-bit Boolean functions. Here are the four possible functions,



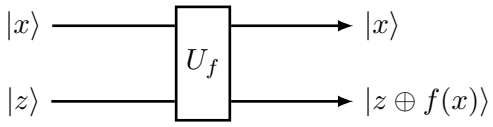
(217)

A function f is given as a black box,



(218)

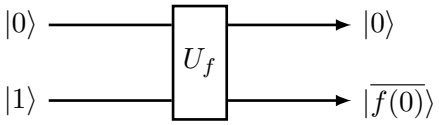
The task is to determine if f is balanced or constant. A classical algorithm evaluates f twice, on the two possible inputs $|0\rangle$ and $|1\rangle$. In the quantum realm, the function is given as a quantum black box unitary operator U_f,



(219)

The unitary operator U_f is a 4×4 matrix. The idea behind a quantum algorithm is to evaluate U_f on a superposition of states. This gives global information about f on all possible inputs. The task is to then untangle this information and see if we can figure out whether f is balanced or constant. Take as an analogy the eye which gathers, in one shot, the electromagnetic signals from your entire environment, from all the objects. Then your brain unravels this superposition of EM-signals to conclude specific information about specific objects in specific locations.

Let's warmup with a simple computation,

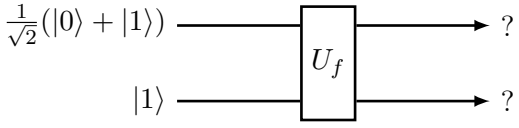


(220)

Clearly, this computation gives only information about $f(0)$, if we measure the bottom bit.

13.1 Applying U_f to Superpositions

Let's now consider a superposition for the top bit,



(221)

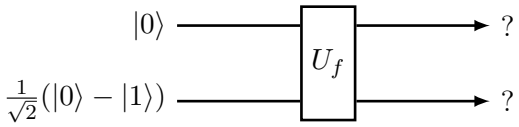
Let's work out what the output is. The input state is $(|0\rangle + |1\rangle)/\sqrt{2} \otimes |1\rangle$, and using linearity of the tensor product, the input is

$$\frac{1}{\sqrt{2}}|0\rangle \otimes |1\rangle + \frac{1}{\sqrt{2}}|1\rangle \otimes |1\rangle = \frac{1}{\sqrt{2}}|0,1\rangle + \frac{1}{\sqrt{2}}|1,1\rangle. \tag{222}$$

You should work out what this input is as a vector. Now, since U_f is a linear operator, we have

$$\begin{aligned} U_f\left(\frac{1}{\sqrt{2}}|0,1\rangle + \frac{1}{\sqrt{2}}|1,1\rangle\right) &= \frac{1}{\sqrt{2}}U_f(|0,1\rangle) + \frac{1}{\sqrt{2}}U_f(|1,1\rangle) & (223)\\ &= \frac{1}{\sqrt{2}}|0,\overline{f(0)}\rangle + \frac{1}{\sqrt{2}}|1,\overline{f(1)}\rangle & (224)\end{aligned}$$

This is an interesting state. If the function is constant, then the output state is a tensor product, but if the function is balanced, then the output is entangled. Hence, we already see qualitatively different behavior depending on whether the function is balanced or constant. Let us consider a superposition for the first bit,



(225)

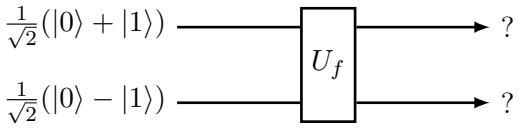
Again, we can use linearity of the tensor product to identify the input as

$$\frac{1}{\sqrt{2}}|0,0\rangle - \frac{1}{\sqrt{2}}|0,1\rangle. \tag{226}$$

Applying U_f to this and using linearity, the output is

$$\begin{aligned} \frac{1}{\sqrt{2}}U_f(|0,0\rangle) - \frac{1}{\sqrt{2}}U_f(|0,1\rangle) &= \frac{1}{\sqrt{2}}|0,f(0)\rangle - \frac{1}{\sqrt{2}}|0,\overline{f(0)}\rangle & (227)\\ &= \begin{cases} \frac{1}{\sqrt{2}}(|0,0\rangle - |0,1\rangle) & f(0)=0 \\ \frac{-1}{\sqrt{2}}(|0,0\rangle - |0,1\rangle) & f(0)=1 \end{cases} & (228)\\ &= \frac{(-1)^{f(0)}}{\sqrt{2}}(|0,0\rangle - |0,1\rangle). & (229)\\ &= \frac{(-1)^{f(0)}}{\sqrt{2}}(|0\rangle \otimes |0\rangle - |0\rangle \otimes |1\rangle). & (230)\\ &= \frac{(-1)^{f(0)}}{\sqrt{2}}|0\rangle \otimes (|0\rangle - |1\rangle). & (231)\end{aligned}$$

The reader should verify/justify every step in the derivation above. The last step uses linearity of the tensor product. Let's go all out and see what happens with a superposition on both input bits,



(232)

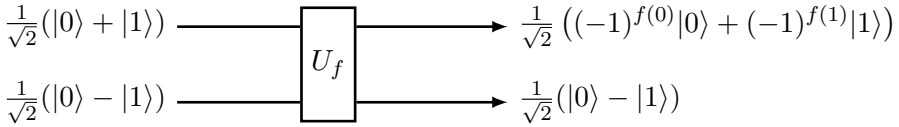
The input is $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. By linearity of the tensor product, this input is

$$\frac{1}{2}|0,0\rangle - \frac{1}{2}|0,1\rangle + \frac{1}{2}|1,0\rangle - \frac{1}{2}|1,1\rangle. \tag{233}$$

Applying U_f to this input and using linearity of U_f, we get

$$\begin{aligned}
& \tfrac{1}{2}U_f(|0,0\rangle) - \tfrac{1}{2}U_f(|0,1\rangle) + \tfrac{1}{2}U_f(|1,0\rangle) - \tfrac{1}{2}U_f(|1,1\rangle) \\
= \;& \tfrac{1}{2}|0,f(0)\rangle - \tfrac{1}{2}|0,\overline{f(0)}\rangle + \tfrac{1}{2}|1,f(1)\rangle - \tfrac{1}{2}|1,\overline{f(1)}\rangle && \text{[definition of } U_f\text{]} \\
= \;& \tfrac{(-1)^{f(0)}}{2}(|0,0\rangle - |0,1\rangle) + \tfrac{(-1)^{f(1)}}{2}(|1,0\rangle - |1,1\rangle) && \text{[Check this]} \\
= \;& \tfrac{(-1)^{f(0)}}{2}(|0\rangle \otimes |0\rangle - |0\rangle \otimes |1\rangle) + \tfrac{(-1)^{f(1)}}{2}(|1\rangle \otimes |0\rangle - |1\rangle \otimes |1\rangle) && \text{[joint state tensor product]} \\
= \;& \tfrac{(-1)^{f(0)}}{2}|0\rangle \otimes (|0\rangle - |1\rangle) + \tfrac{(-1)^{f(1)}}{2}|1\rangle \otimes (|0\rangle - |1\rangle) && \text{[tensor product linearity]} \\
= \;& \underbrace{\frac{1}{\sqrt{2}}\left((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\right)}_{\text{top qubit}} \otimes \underbrace{\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)}_{\text{bottom qubit}} && \text{[tensor product linearity]}
\end{aligned} \tag{234}$$

Again, the reader should verify every step in the derivation above. The intersting point is that the output is a tensor product, that is, we can clearly identify the top qubit output state and the bottom qubit output state. The output qubits are not entangled,



(235)

Notice that the top output qubit contains information about both $f(0)$ and $f(1)$. Interestingly, the state of the bottom qubit has been left unchanged. This might be counterintuitive. In the definition of U_f it left the top bits unchanged. That was the definition of U_f for classical bits. Here we see that when U_f operates on general qubits, it may not leave the top qubit unchanged. In the end, Q_f is just a unitary operator. Its action on classical bits just serves to define how U_f operates on a basis, which completely defines the linear operator. This linear operator is a matrix, and we can always convert the input to a vector and apply the matrix to the vector. Our analysis above gives the result. It is imperative for the reader to work through the details of the next example.

Example 13.1. Let f be f_{11} from the begining of the lecture.

1. Show that

$$U_f = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{236}$$

2. Show that the input and output to U_f in (235) are the vectors

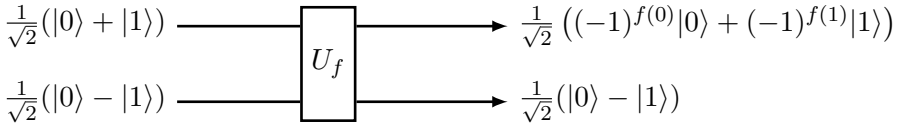
$$|\text{input}\rangle = \begin{bmatrix} 1/2 \\ -1/2 \\ 1/2 \\ -1/2 \end{bmatrix}, \qquad |\text{output}\rangle = \begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix}. \tag{237}$$

3. Verify the following computation,

$$U_f \cdot |\text{input}\rangle = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1/2 \\ -1/2 \\ 1/2 \\ -1/2 \end{bmatrix} = \begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix} = |\text{output}\rangle. \tag{238}$$

13.2 Untangling the Output

We now know how U_f acts on superpositions. Let's get back to the problem at hand, to determine if f is balanced. The recap of where we are is:



(239)

We can now relate to the human eye which receives all the EM-rays from the environment. We need to untangle the signal. In our case, the signal is the top qubit which contains information about f on all inputs. So, let's measure the top qubit. The result is $|0\rangle$ with probability 1/2 and $|1\rangle$ with probability 1/2. That's no help, because the result of the measurement is independent of f, even though the top bit is dependent on f. Let's take a closer look at the top qubit in the output,

$$\frac{1}{\sqrt{2}}\left((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\right) = \begin{bmatrix} (-1)^{f(0)}/\sqrt{2} \\ (-1)^{f(1)}/\sqrt{2} \end{bmatrix}. \tag{240}$$

Recall the Hadamard matrix,

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{241}$$

What happens if we hit this top qubit with the Hadamard matrix,

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} (-1)^{f(0)}/\sqrt{2} \\ (-1)^{f(1)}/\sqrt{2} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} (-1)^{f(0)} + (-1)^{f(1)} \\ (-1)^{f(0)} - (-1)^{f(1)} \end{bmatrix} \tag{242}$$

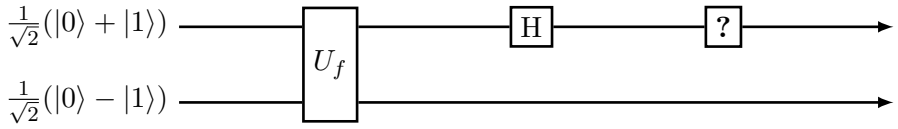
$$= \begin{cases} \begin{bmatrix} \pm 1 \\ 0 \end{bmatrix} & f \text{ is constant} \\ \\ \begin{bmatrix} 0 \\ \pm 1 \end{bmatrix} & f \text{ is balanced} \end{cases} \tag{243}$$

We are at the punchline. The Hadamard matrix is the great untangler of superpositions. By applying the Hadamard matrix to the top qubit, it has untangled the superposition of information

to always give a pure state, a classical bit. If you measure the top qubit after applying the great untangler, you will *always* measure $|0\rangle$ if f is constant, and you will *always* measure $|1\rangle$ if f is balanced. Done! We have a way to test if f is balanced by evaluating f, i.e. applying U_f, once.

13.3 Quantum Circuit for 1-bit Deutsch-Jozsa

The final quantum algorithm that achieves the miracle of testing if a function is balanced using just one evaluation of the function is represented in the following circuit.



(244)

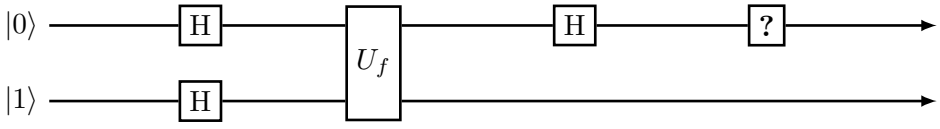
Recall that ? is the measurement operation. So, the circuit above indicates that we apply H to the top qubit and then measure the top qubit. We would be done, except for a practical consideration. We evaluate U_f on a joint state which is a superposition. However, we cannot in general create arbitrary qubit states, in particular the superpositions that are input into U_f. It is easy to create pure states, i.e. classical bits. For example if electron spin is how we are encoding the bit, then the bit can be set to $|\uparrow\rangle$ by appropriately setting a magnetic field, and similarly for $|\downarrow\rangle$.

How do we get these superposition states to feed into U_f, given that we can only start with pure classical bits. We need to apply some unitary operator to the classical states, and also hope that unitary operator can be implemented as a quantum gate by some physical process. Luckily we do not need to look too far for this unitary operator. The Hadamard will do, because note that,

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix} = \begin{bmatrix}1/\sqrt{2}\\ 1/\sqrt{2}\end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{245}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0\\ 1\end{bmatrix} = \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \tag{246}$$

These are exactly what we need. So the mighty Hadamard is not only the great untangler of superposed states, but it is also the great creator of superposed states. The final quantum circuit to solve the 1-bit Deutsch Jozsa problem is



(247)

We can now write down the unitary operator that corresponds to the circuit above,

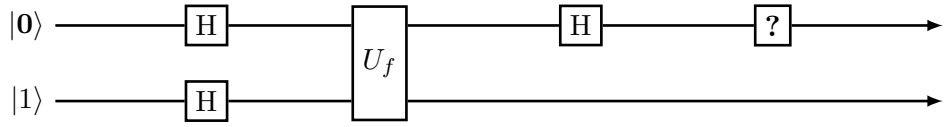
$$(\mathrm{H}\otimes\mathrm{I})\cdot U_f \cdot (\mathrm{H}\otimes\mathrm{H}). \tag{248}$$

Note that $\mathrm{H}\otimes\mathrm{H}$ is a 4×4 Hadamard matrix. We are applying this unitary operator to $|0\rangle\otimes|1\rangle$, but this operator can be applied to any input $|\psi\rangle$. The next exercise is imperative.

Example 13.2. Compute the operator in (248) for f_{11} in Example 13.1. Apply this operator to the input $|0\rangle\otimes|1\rangle = [0,1,0,0]^{\mathrm{T}}$. Explain how the output vector agrees with the top bit being $|0\rangle$.

14 Quantum Circuits

We just saw our first quantum circuit that tests balance of 1-bit function,



(249)

Let us briefly discuss building and analyzing quantum circuits. We must guess what gates will be stantard when quantum hardware is widely available. Presumably these gates will be stable and implemented in some physical quantum substrate, much like today we implement Boolean logic gates like NAND in semiconductors. With no other guide, let us focus on some useful gates,

$$\mathrm{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \qquad \text{NOT} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \qquad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \qquad R(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}. \tag{250}$$

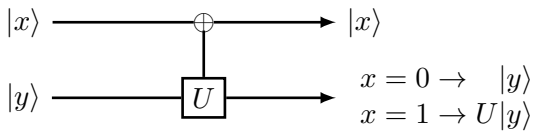
Additionally, we one can multiply by a phase $e^{i\phi}$,[4] use a control bit to control the output of a gate, and also take the controlled XOR of output qubits (all these operations are unitary). We already saw the use of controlling bits, and we used the controlled XOR when we converted a Boolean function f to its unitary equivalent U_f. Using these standard gates plus the additional operations, we can pretty much implement any unitary operator we need. We address two tasks:

1. Given a quantum circuit that uses the standard gates and controlling inputs, what does it do. That is, what is the unitary operator it implements.
2. Given a task, i.e., a unitary operator, find a circuit that implements the operator.

As we discuss several examples, we will also see conventions for graphically defining the circuits.

14.1 Finding the Operator for a Circuit

Let U be the unitary operator $\left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$. The operator U operates on 1 qubit. We start with a simple example, the quantum equivalent of IF. . . ELSE. . . , the controlled U in (192) on page 50,



(251)

The bottom qubit outputs $|y\rangle$ if $x = 0$. The output is $U|y\rangle$ if $x = 1$. U only operates when $x = 1$. To analyze this circuit, we apply it to the standard basis $|00\rangle, |01\rangle, |10\rangle, |11\rangle$,

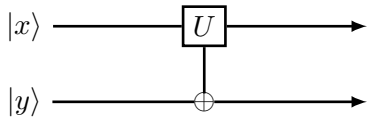
$$|00\rangle \to |00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad |01\rangle \to |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad |10\rangle \to |1\rangle \otimes \begin{bmatrix} a \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ a \\ c \end{bmatrix} \qquad |11\rangle \to |1\rangle \otimes \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ b \\ d \end{bmatrix}. \tag{252}$$

We can now write down the operator for controlled-U,

$${}^{C}U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix} = \begin{bmatrix} \mathrm{I} & \mathbf{0} \\ \mathbf{0} & U \end{bmatrix}. \tag{253}$$

[4] This is just unitary evolution according to a constant Hamiltonian.

Let us use this method for several similar examples. What if we flip the controlling bit and U,



(254)

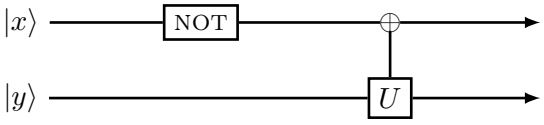
Verify that the standard basis transforms as

$$|00\rangle \to |00\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \qquad |01\rangle \to \begin{bmatrix}a\\c\end{bmatrix} \otimes |1\rangle = \begin{bmatrix}0\\a\\0\\c\end{bmatrix} \qquad |10\rangle \to |10\rangle = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \qquad |11\rangle \to \begin{bmatrix}b\\d\end{bmatrix} \otimes |1\rangle = \begin{bmatrix}0\\b\\0\\d\end{bmatrix}. \tag{255}$$

We can now write down the operator,

$$\begin{bmatrix}1 & 0 & 0 & 0\\0 & a & 0 & b\\0 & 0 & 1 & 0\\0 & c & 0 & d\end{bmatrix}. \tag{256}$$

What if we negate the controlling bit first,



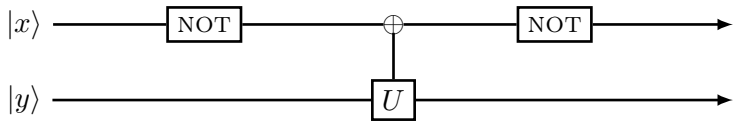
(257)

$$|00\rangle \to |1\rangle \otimes \begin{bmatrix}a\\c\end{bmatrix} = \begin{bmatrix}0\\0\\a\\c\end{bmatrix} \qquad |01\rangle \to |1\rangle \otimes \begin{bmatrix}b\\d\end{bmatrix} = \begin{bmatrix}0\\0\\b\\d\end{bmatrix} \qquad |10\rangle \to |00\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \qquad |11\rangle \to |01\rangle = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}. \tag{258}$$

We can now write down the operator,

$$\begin{bmatrix}0 & 0 & 1 & 0\\0 & 0 & 0 & 1\\a & b & 0 & 0\\c & d & 0 & 0\end{bmatrix} = \begin{bmatrix}\mathbf{0} & \mathrm{I}\\U & \mathbf{0}\end{bmatrix}. \tag{259}$$

This is not quite the equivalent of IF NOT $x\ldots$ because x is negated at the end. To get a true negatively controlled U, we can negate x back to its original bit after it has done its controlling,



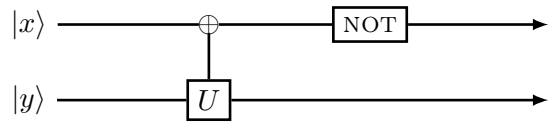
(260)

$$|00\rangle \to |0\rangle \otimes \begin{bmatrix}a\\c\end{bmatrix} = \begin{bmatrix}a\\c\\0\\0\end{bmatrix} \qquad |01\rangle \to |0\rangle \otimes \begin{bmatrix}b\\d\end{bmatrix} = \begin{bmatrix}b\\d\\0\\0\end{bmatrix} \qquad |10\rangle \to |10\rangle = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \qquad |11\rangle \to |11\rangle = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}. \tag{261}$$

We can now write down the operator,

$$\begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} U & \mathbf{0} \\ \mathbf{0} & \mathrm{I} \end{bmatrix}. \tag{262}$$

One final example which we leave to the reader. If we negate only after controlling,

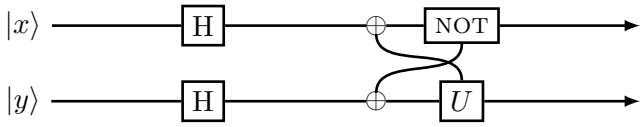


(263)

show that the resulting operator is $\left[\begin{smallmatrix} \mathbf{0} & U \\ \mathrm{I} & \mathbf{0} \end{smallmatrix}\right]$. So, given any operator U, we see that we can construct the following four operators using either controlling bits or negated controlling bits,

$$\begin{bmatrix} \mathrm{I} & \mathbf{0} \\ \mathbf{0} & U \end{bmatrix} \qquad \begin{bmatrix} \mathbf{0} & \mathrm{I} \\ U & \mathbf{0} \end{bmatrix} \qquad \begin{bmatrix} U & \mathbf{0} \\ \mathbf{0} & \mathrm{I} \end{bmatrix} \qquad \begin{bmatrix} \mathbf{0} & U \\ \mathrm{I} & \mathbf{0} \end{bmatrix}. \tag{264}$$

Let us find the operator for this complicated circuit,



(265)

Note the use of two controlling bits simultaneously. That is perfectly fine. Let us try to figure out what happens when we hit the basis vector $|00\rangle$ with this circuit. The first "level" of the circuit produces $\mathrm{H}|0\rangle \otimes \mathrm{H}|0\rangle$. That was easy enough. But now when we feed this to the second "level", it is not clear what happens.

We saw that the circuit has two levels in sequence. The action plan is to figure out the operator for each level, and then we get the operator for the whole circuit by multiplying. The operator for the first level is easy, it is just $\mathrm{H} \otimes \mathrm{H}$.

Let is see what happens if the standard basis is fed directly into the second level,

$$|00\rangle \to |00\rangle \qquad |01\rangle \to \textsc{not}|0\rangle \otimes |1\rangle \qquad |10\rangle \to |1\rangle \otimes U|0\rangle \qquad |11\rangle \to \textsc{not}|1\rangle \otimes U|1\rangle. \tag{266}$$

We can now write down the operator for the second level,

$$\begin{bmatrix} 1 & 0 & 0 & b \\ 0 & 0 & 0 & d \\ 0 & 0 & a & 0 \\ 0 & 1 & c & 0 \end{bmatrix}. \tag{267}$$

Note that this is not a unitary operator unless U is diagonal, so it could not be implemented by a quantum mechanical circuit. Nevertheless, it is a useful exercise to flex our muscles. The (generally non-unitary) operator for the circuit is then

$$\begin{bmatrix} 1 & 0 & 0 & b \\ 0 & 0 & 0 & d \\ 0 & 0 & a & 0 \\ 0 & 1 & c & 0 \end{bmatrix} \cdot (H \otimes H). \tag{268}$$

14.2 Building a Circuit for an Operator

Finding the circuit for an operator is like computer programming. Writing a program to solve a problem requires creativity. There is no prescription. And so it is with quantum computer programming. Given a unitary operator that solves a task, one has to use some creativity to find a circuit to implement the operator. There are some useful tricks to know, starting with (264). Here are a few more tricks. Given n-qubit unitary operators A, B, we can get

$$\begin{bmatrix} A & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix} \tag{269}$$

using the trick

$$\begin{bmatrix} A & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix} = \begin{bmatrix} A & \mathbf{0} \\ \mathbf{0} & \mathrm{I} \end{bmatrix} \begin{bmatrix} \mathrm{I} & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix} = \begin{bmatrix} \mathrm{I} & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix} \begin{bmatrix} A & \mathbf{0} \\ \mathbf{0} & \mathrm{I} \end{bmatrix}. \tag{270}$$

We know circuits for both operators on the right (see (264)) which we run in sequence. Consider

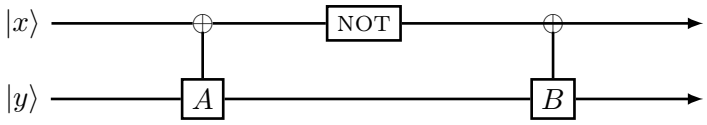
$$\begin{bmatrix} \mathbf{0} & A \\ B & \mathbf{0} \end{bmatrix} \tag{271}$$

(It is a useful exercise to show that this operator is indeed unitary.) We use the identity

$$\begin{bmatrix} \mathbf{0} & A \\ B & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathrm{I} \\ \mathrm{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} B & \mathbf{0} \\ \mathbf{0} & A \end{bmatrix}. \tag{272}$$

The first operator on the RHS is $\text{NOT} \otimes \mathrm{I}$. Applying these in sequence gives this circuit for $\left[\begin{smallmatrix} \mathbf{0} & A \\ B & \mathbf{0} \end{smallmatrix}\right]$,

$\begin{bmatrix} \mathbf{0} & A \\ B & \mathbf{0} \end{bmatrix}$:



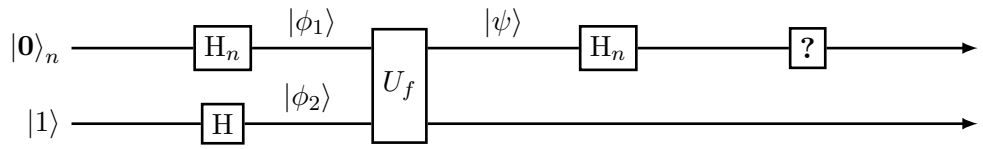
(273)

Verify the circuit above. Practice makes perfect. Try to construct a circuit for this operator,

$$U = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{274}$$

15 Testing Balance of n-bit Functions

Assume the n-qubit function f is either balanced or constant. Recall that U_f is the $2^{n+1} \times 2^{n+1}$ unitary operator that is equivalent to f. A simple extension of the circuit that tested balance of a 1-bit function is



(275)

It is going to be a heavy linear algebra lift to figure out what this circuit is doing. The only way to learn this stuff is to work the algebra yourself.

15.1 Deutsch-Jozsa Algorithm

Let's analyze this circuit in the two cases, f is constant and f is balanced. First note that the input to the first H_n is $|0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle = |00\cdots 0\rangle$ which is the first standard basis vector. Therefore $\mathrm{H}_n|\mathbf{0}\rangle_n$ is just the first column of H_n. We know that the first column of H_n is just a 2^n-vector of 1's, normalized. Hence,

$$|\phi_1\rangle = \mathrm{H}_n|\mathbf{0}\rangle_n = \frac{1}{2^{n/2}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix} = \frac{1}{2^{n/2}} \sum_{i=1}^{2^n} \boldsymbol{e}_i = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle. \tag{276}$$

The RHS is simply a statement that H_n is the great entangler, constructing a uniform superposition of every possible n-bit pure state starting from $|\mathbf{0}\rangle$. A much simpler computation gives

$$|\phi_2\rangle = H|1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \tag{277}$$

Therefore, the input to U_f is

$$|\phi_1\rangle \otimes |\phi_2\rangle = \left(\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \right) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{278}$$

$$= \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{279}$$

In the last step we used linearity of the tensor product. We now apply U_f to this sum of tensor products to get $|\psi\rangle$. Using linearity of U_f, we get

$$\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} U_f \left(|x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \right). \tag{280}$$

Recall that $|x\rangle$ is a classical pure state, a standard basis vector. We have

$$\begin{aligned} U_f\left(|x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right) &= \frac{1}{\sqrt{2}} U_f(|x\rangle \otimes |0\rangle - |x\rangle \otimes |1\rangle) && (281)\\ &= \frac{1}{\sqrt{2}} U_f(|x\rangle \otimes |0\rangle) - \frac{1}{\sqrt{2}} U_f(|x\rangle \otimes |1\rangle) && (282)\\ &= \frac{1}{\sqrt{2}} |x\rangle \otimes |f(x)\rangle - \frac{1}{\sqrt{2}} |x\rangle \otimes |\overline{f(x)}\rangle && (283)\\ &= (-1)^{f(x)} |x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} && (284) \end{aligned}$$

The first step uses linearity of tensor product; the second step uses linearity of U_f; the third step uses the definition of U_f, because $|x\rangle$ is a classical pure state; the last step is for the reader to verify, depending on whether $f(x) = 0$ or $f(x) = 1$. Plugging this back into the expression for the output from U_f, we get

$$\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{285}$$

$$= \left(\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle \right) \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}, \tag{286}$$

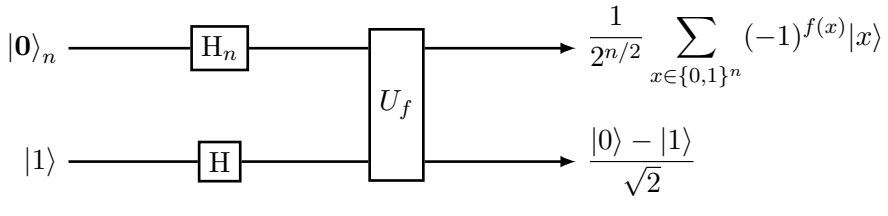
where the last step uses linearity of the tensor product. The state on the top n output qubits is

$$|\psi\rangle = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle. \tag{287}$$

and the state on the bottom output is

$$\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \tag{288}$$

In terms of the circuit diagram, here is where we are,



(289)

Something strange. All of the information about f is in the input registers, even though U_f was designed with the information about f in the output bit. This is one of the strange things that can happen when an operator hits a superposition of states. For a pure classical state as input, the information about f would be in the output qubit. This is not so when the input is a superposition.

To get the final output of the top n qubits, we hit $|\psi\rangle$ with H_n, now acting as the great untangler. Let's first consider the case where f is constant. Then

$$|\psi\rangle = \pm \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle = \pm \mathrm{H}_n |0\rangle, \tag{290}$$

where the $\pm$ depends on whether $f(x) = 0$ or $f(x) = 1$. In this case, $\mathrm{H}_n|\psi\rangle$ is given by

$$\mathrm{H}_n|\psi\rangle = \pm \mathrm{H}_n^2|0\rangle = \pm|0\rangle, \tag{291}$$

because $\mathrm{H}_n^2 = \mathrm{I}$. So, when we measure the top n qubits, they will all be zero, because no other classical state has non-zero amplitude. Let's see what happens if f is balanced. In this case $\mathrm{H}_n|\psi\rangle$ is complicated. Let us write

$$|\psi\rangle = \frac{1}{2^{n/2}} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_{2^n} \end{bmatrix} \tag{292}$$

Each ψ_i is ± 1 because $|\psi\rangle$ is a linear combination of all the basis vectors with each amplitude being $(-1)^{f(x_i)}/2^{n/2}$. Since f is balanced, half of the ψ_i are $+1$ and the other half are -1. This means $\mathbf{1}^{\mathrm{T}}|\psi\rangle = 0$. This is interesting, because when we consider $\mathrm{H}_n|\psi\rangle$, we get

$$\mathrm{H}_n|\psi\rangle = \frac{1}{2^{n/2}} \begin{bmatrix} \boldsymbol{h}_1^{\mathrm{T}} \text{———} \\ \boldsymbol{h}_2^{\mathrm{T}} \text{———} \\ \boldsymbol{h}_3^{\mathrm{T}} \text{———} \\ \vdots \\ \boldsymbol{h}_{2^n}^{\mathrm{T}} \text{———} \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_{2^n} \end{bmatrix}. \tag{293}$$

The $\boldsymbol{h}_i^{\mathrm{T}}$ are the rows of H_n. The first entry in $\mathrm{H}_n|\psi\rangle$ is the amplitude for the first basis vector $|000\cdots0\rangle$. So, the first entry in $\mathrm{H}_n|\psi\rangle$ is the amplitude for the pure state $|000\cdots0\rangle$. The first row of H_n is all ones, $\boldsymbol{h}_1^{\mathrm{T}} = \mathbf{1}^{\mathrm{T}}$. Because $\mathbf{1}^{\mathrm{T}}|\psi\rangle = 0$, $\mathrm{H}_n|\psi\rangle$ has zero in its first entry,

$$\mathrm{H}_n|\psi\rangle = \begin{bmatrix} 0 \\ ? \\ ? \\ \vdots \\ ? \end{bmatrix}. \tag{294}$$

While we do not know the other entries in $\mathrm{H}_n|\psi\rangle$, the fact that the first entry is 0 is huge. It means the amplitude to measure all bits 0 is zero when f is balanced. If f is balanced, at least one of the measured bits will be 1. If f is constant, all measured bits will be 0. We have our algorithm:

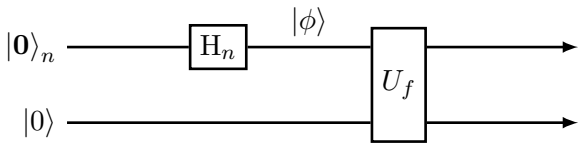
1: Run the circuit.
2: Measure the top n qubits.
3: **if** all qubits are 0 **then**
4: $\quad f$ is constant.
5: **else**
6: $\quad f$ is balanced.

Our analysis proves the algorithm works. Yes, the problem is contrived, requiring the function to be either constant or perfectly balanced. It is interesting to analyze other cases. We have our first quantum algorithm. With one function evaluation it tests balance, where the classical algorithm requires $2^{n-1} + 1$ function evaluations. That is an exponential gain.

However, is it really. If we are to implement this circuit by explicitly encoding U_f then all efficiency gains are lost since U_f is exponential in size. The information theoretic gain is still there, though, that one function "evaluation" suffices. The quantum gains come when the function is specified directly as U_f using a small circuit, implemented using basic quantum gates. By one application of this circuit, we test if f is balanced or constant. There is no information paradox here. We traded 1-bit of information of f on some input with 1-bit of information regarding the relationships between f on several inputs. In both cases, we have received just 1-bit of information.

16 Philosophy of Quantum Algorithms

Let's take stock. A quantum algorithm somehow miraculously gets global information about a Boolean function f by simultaneously evaluating it on all possible inputs at once. To accomplish this, the typical first step is to place the starting pure state into an appropriate superposition of all the pure states. Here is an example,



(295)

The input to U_f is $|\phi\rangle \otimes |0\rangle$, where

$$|\phi\rangle = \frac{1}{2^{n/2}} \sum_{z \in \{0,1\}^n} |z\rangle. \tag{296}$$

The sum is over all n-bit pure states. The definition of U_f for pure states is

$$U_f(|z\rangle \otimes |0\rangle) = |z\rangle \otimes |f(z)\rangle. \tag{297}$$

Using linearity of U_f and tensor product, we find that the output is

$$|\text{output}\rangle = \frac{1}{2^{n/2}} \sum_{z \in \{0,1\}^n} |z\rangle \otimes |f(z)\rangle. \tag{298}$$

A lot of the magic in a quantum algorithm happens here. We have miraculously applied f to every possible input z, and all this information is contained in the output state. The rest of the magic in the quantum algorithm is to see if we can untangle all this information to get at what we want. One way to untangle is to measure the output. Now the state collapses by randomly picking one of the pure states, $|z_0\rangle \otimes |f(z_0)\rangle$. So, in the end we get only the single bit of information $f(z_0)$. We are only entitled to one bit of information about $f(x)$ because we only made one evaluation of U_f.

So what is going on with testing balance of f. It seems like we are getting information about f globally, a lot more than one bit of information. No. It is still just one bit of information, a binary outcome on whether f is balanced or constant. It is just that the nature of the information is different. Rather than the 1-bit of information giving the value of $f(z_0)$, it gives information on the relationship between values of f. But, it is still one bit of information.

Can we get a free-lunch by continuing to measure the output? First we get $|z_0\rangle \otimes |f(z_0)\rangle$. The state collapses to this pure state and from then we always get $|z_0\rangle \otimes |f(z_0)\rangle$. So we don't get additional information about f by continuing to measure. Quantum state collapse prevents this. Here is a new idea. Before measuring, make several identical copies of $|\text{output}\rangle$. Each copy will collapse independently to possibly different pure states, giving f for those pure states. Thus, we extract multiple bits from one evaluation of U_f. This too is prevented. Why? Because you cannot clone quantum states, by the no-cloning theorem we proved earlier. Everything neatly fits together.

16.1 Directly Building a Circuit for U_f

You may be worried about where we got U_f. All the information by f is stored in U_f (twice infact), so if we explicitly construct the $2^{n+1} \times 2^{n+1}$ matrix U_f, then we are basically examining f on every input. And certainly, if the quantum circuit is doing no more than explicitly computing the matrix product $U_f(|\phi\rangle \otimes |0\rangle)$, then the runtime will be exponential. For this to be practical in any way, it is necessary for us to be given U_f as a black-box quantum circuit, presumably one that only uses poly(n) quantum gates. Let's see an example of this.

Fix an n-bit parameter $a = [a_1, \ldots, a_n]$ and define a function on n bits $x = [x_1, \ldots, x_n]$,

$$f(x) = a_1 x_1 \oplus a_2 x_2 \oplus a_3 x_3 \oplus \cdots \oplus a_n x_n. \tag{299}$$

The addition is modulo 2, equivalent to XOR. $f(x)$ is like an inner product for binary vectors. f takes the inner product of x with $\boldsymbol{a}$, a fixed parameter. For $\boldsymbol{a} = [1, 0, 1]$, we have

x_1	x_2	x_3	$f(x)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

(300)

We can immediately write down U_f as a block diagonal with 2×2 blocks,

$$\begin{bmatrix}
\boxed{\mathrm{I}_2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} \\
0_{2\times2} & \boxed{\text{NOT}} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} \\
0_{2\times2} & 0_{2\times2} & \boxed{\mathrm{I}_2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} \\
0_{2\times2} & 0_{2\times2} & 0_{2\times2} & \boxed{\text{NOT}} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} \\
0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & \boxed{\text{NOT}} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} \\
0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & \boxed{\mathrm{I}_2} & 0_{2\times2} & 0_{2\times2} \\
0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & \boxed{\text{NOT}} & 0_{2\times2} \\
0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times2} & \boxed{\mathrm{I}_2}
\end{bmatrix} \tag{301}$$

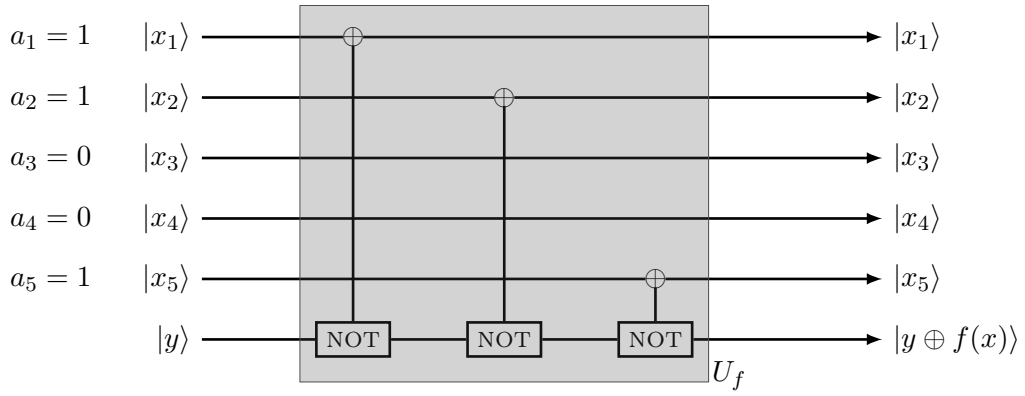
The columns can be partitioned into consecutive pairs. Each pair corresponds to a pure state on the first n-bit input vector $|x\rangle$ with the control bit $|0\rangle$ or $|1\rangle$. The pair of columns corresponds to $|x, 0\rangle$ and $|x, 1\rangle$. The block is I when $f(x) = 0$ and NOT otherwise. Try to implement this unitary operator using the tricks that we saw earlier. A quicker approach is to look at the form for $f(x)$ and build the circuit directly. Recall that the output of U_f on a pure state is

$$U_f(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle. \tag{302}$$

The bottom (output) qubit is just

$$|y \oplus a_1x_1 \oplus a_2x_2 \oplus a_3x_3 \oplus \cdots \oplus a_nx_n\rangle. \tag{303}$$

Starting with the y-bit, process each x_i. If the corresponding a_i is 0, do nothing. If the corresponding a_i is 1, flip the y-bit provided the x_i-bit is 1. That is, x_i is controlling a NOT on y. It is only such a control bit when a_i is 1. Here is the circuit that implements this logic,

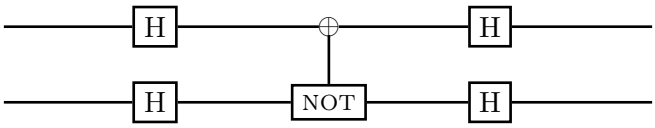


(304)

We went directly from the definition of an important function to its circuit. This circuit generalizes to n-bits and any a, and is compact (linear in size). The matrix U_f grows exponentially, which does not matter if we are given U_f as a compact black-box circuit using standard quantum gates.

16.2 Circuit Uniqueness

The circuit for a given operator is not unique. Consider this circuit,

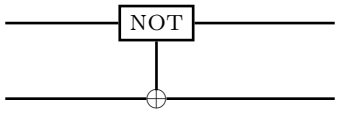


(305)

The reader may verify that the operator for this circuit is

$$(H \otimes H) \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot (H \otimes H) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{306}$$

Now consider the circuit where we remove the Hadamards and reverse the roles of the controlling bit and the controlled bit,



(307)

Again, please verify that the operator for this circuit is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{308}$$

These two circuits implement the same operator. So circuits for a given operator are not unique. It is a very hard task to find the minimum sized circuit for a given operator. Looking at the first circuit, it appears that the top qubit is unaffected by the circuit, as it is hit twice with a Hadamaard (which is the identity) and it is used as a controlling bit (which shouldn't change the bit). However, looking at the second circuit it is clear that the bottom qubit is unchanged, and it is the bottom bit that is actually controlling the top qubit. It is also interesting that sandwiching a controlled not between Hadamards is equivalent to simply switching the roles of the controlling and controlled bits. This particular fact will be useful later. The next exercise is imperative.

Example 16.1. Construct the unitary operator for the circuit like the one in (304) using the function defined in (300) with $\boldsymbol{a} = [1, 0, 1]$. Verify that you get the operator in (301).

Get the operator for the circuit in (304) using two methods. First see how the circuit operates on the computational basis $[\mathbf{e}_1, \ldots, \mathbf{e}_{16}]$ to infer the operator. Second construct the operator for each c-NOT and multiply these operators together.

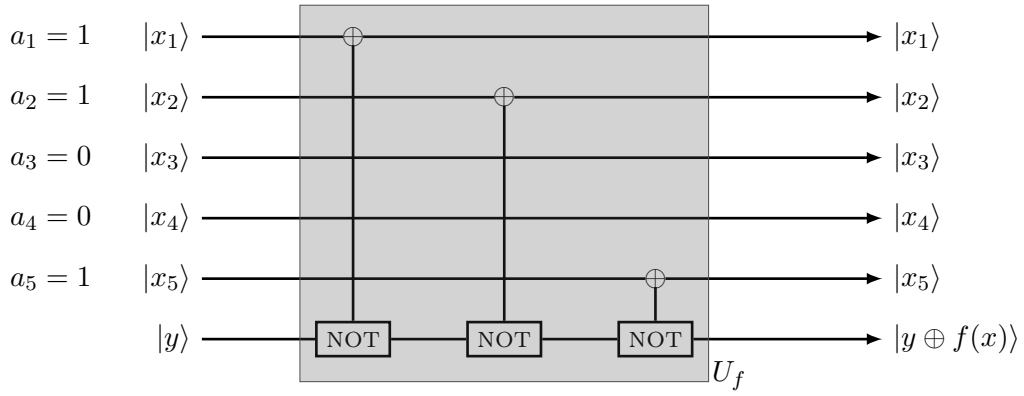
All these methods should produce the same operator in (301).

17 Learning the Weights in a Linear Function

In the last lecture we considered the Boolean version of a linear function,

$$f(x) = a_1x_1 \oplus a_2x_2 \oplus a_3x_3 \oplus \cdots \oplus a_nx_n, \tag{309}$$

where $\boldsymbol{a} = a_1, \cdots, a_n$ and $x = x_1, \cdots, x_n$ are n-bit vectors. Think of $\boldsymbol{a}$ as a hidden parameter and x is the input to the function f. This is a convenient function because we can directly construct a compact circuit for U_f. For $a = 11001$, the circuit for U_f is



(310)

Given f (or U_f) as a black box, the Bernstein-Vazirani problem is to infer the parameters $\boldsymbol{a}$.

A prominent example of this problem arose in the context of college rankings. Here is the a simplified story. U.S. News maintains a ranking of colleges. A university has a set of features

$$x = [\text{in-coming SAT}, \%\text{ small classes}, \text{faculty-to-student-ratio}, \ldots] \tag{311}$$

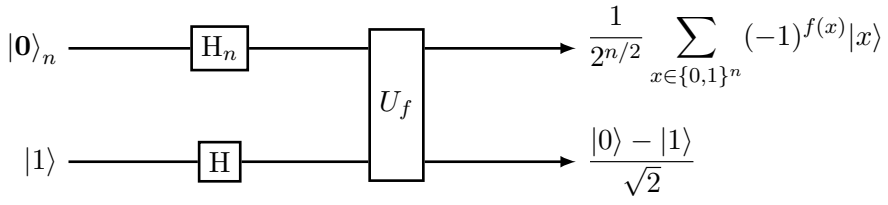
Some features in x are self-reported. U.S. News computes a score $a^{\mathrm{T}}x$ for a set of heavily guarded proprietary weights. The universities are ranked according to this score. The weights are guarded because if some university knew the weights, they could strategically improve their features for the large weights, thereby manipulating the rankings. Every year a university's features change, and they see their ranking (a proxy for the score). Hence they are repeatedly querrying a function like f with different x. At some point they have learned $\boldsymbol{a}$ and can strategically improve their rankings. This actually happened. Let's get back to f in (309). Observe that

$$\begin{array}{rcl} f(1000\cdots00) &=& a_1 \\ f(0100\cdots00) &=& a_2 \\ f(0010\cdots00) &=& a_3 \\ f(0001\cdots00) &=& a_4 \\ \vdots & \vdots & \vdots \\ f(0000\cdots10) &=& a_{n-1} \\ f(0000\cdots01) &=& a_n \end{array} \tag{312}$$

We can learn $\boldsymbol{a}$ using n function evaluations. If n is large, the weights are safe because many queries are needed to learn $\boldsymbol{a}$. Each querry gives us 1 bit of information. We get n bits from n queries, namely the bits $a_1, \cdots, a_n$. We cannot hope to do better using 1-bit queries. Here is an information

theoretic proof. There are 2^m possible outcomes using m one-bit queries. Since there are 2^n possible choices for $\boldsymbol{a}$, if $2^m < 2^n$, by pigeonhole, any mapping from the set of $\{a\}$ to the query outcomes assigns at least two $\boldsymbol{a}$'s to the same query outcome. Those two $\boldsymbol{a}$'s aren't disambiguated.

In the classical world, the weights are safe. In the quantum world this is not the case. A single evaluation of U_f suffices to reveal $\boldsymbol{a}$. We begin our quantum algorithm by applying U_f to a superposition of states. This is usually the first bit of magic in any quantum algorithm. We get information on f for all inputs using just one evaluation of U_f. The second bit of magic comes when it is time to untangle this information to get at what we need. Here is the first step,



(313)

We have already derived this first step many times, using linearity of tensor product and linearity of U_f (see, for example, the discussion accompanying the Deutsch-Jozsa algorithm on page 66). The reader should derive it one more time themselves and commit this first step to memory. This is typically the start of any quantum algorithm.

How do we untangle the information in the first n *qubits*. Since H is also the great untangler, lets try H and see what happens. At this point it is good to tinker with an example. Let $a = 01$.

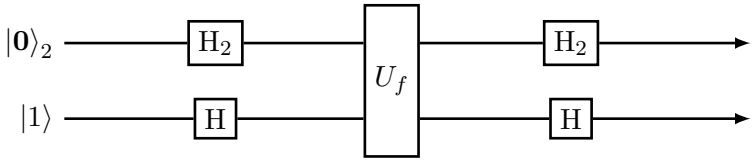
x_1	x_2	$f(x)$
0	0	0
0	1	1
1	0	0
1	1	1

(314)

We can immediately write down U_f,

$$U_f = \begin{bmatrix} \boxed{\mathrm{I}_2} & 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} \\ 0_{2\times 2} & \boxed{\mathrm{NOT}} & 0_{2\times 2} & 0_{2\times 2} \\ 0_{2\times 2} & 0_{2\times 2} & \boxed{\mathrm{I}_2} & 0_{2\times 2} \\ 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} & \boxed{\mathrm{NOT}} \end{bmatrix} \tag{315}$$

The circuit we are proposing to extract $\boldsymbol{a}$ is



(316)

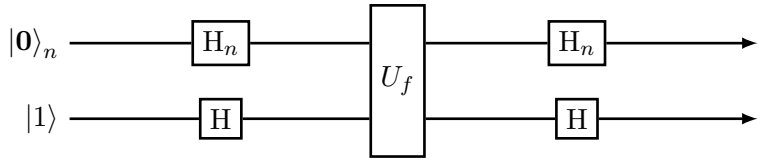
The operator for this circuit is $(\mathrm{H}_2 \otimes \mathrm{H}) \cdot U_f \cdot (\mathrm{H}_2 \otimes H)$. The input to the circuit is $|001\rangle = [0,1,0,0,0,0,0,0]^{\mathrm{T}}$. It is now just an exercise in careful linear algebra to verify

$$(\mathrm{H}_2 \otimes \mathrm{H}) \cdot U_f \cdot (\mathrm{H}_2 \otimes H) \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} |\mathbf{00}0\rangle \\ |\mathbf{00}1\rangle \\ |\mathbf{01}0\rangle \\ |\mathbf{01}1\rangle \\ |\mathbf{10}0\rangle \\ |\mathbf{10}1\rangle \\ |\mathbf{11}0\rangle \\ |\mathbf{11}1\rangle \end{matrix}.$$

On the right we show the pure state corresponding to each amplitude. We also highlighted the first two bits of the pure state. Note that the amplitude is only non-zero when the first two bits are 01, which is exactly $\boldsymbol{a}$. If we measure the top two qubits, we get $\boldsymbol{a}$. Is this always the case. Yes! Note, since we only care about the top two qubits in the measurement, we can save one gate by removing the last H on the lower register. This means that the lower register will be in the state $(|0\rangle - |1\rangle)/\sqrt{2}$ and the measurement will yield either $|010\rangle$ or $|011\rangle$ with equal probabilities. In either case the first two qubits give $\boldsymbol{a}$.

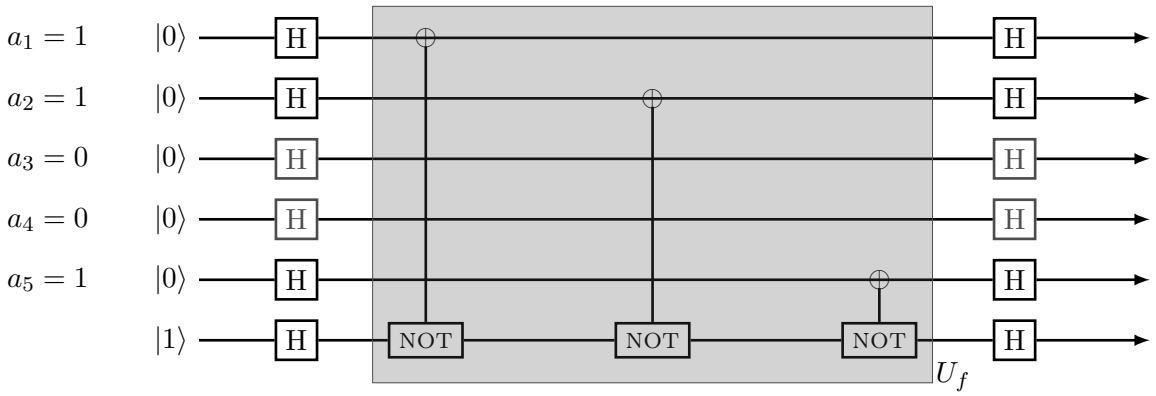
17.1 Circuit for Bernstein-Vazirani

We claim, in general, that the circuit



(317)

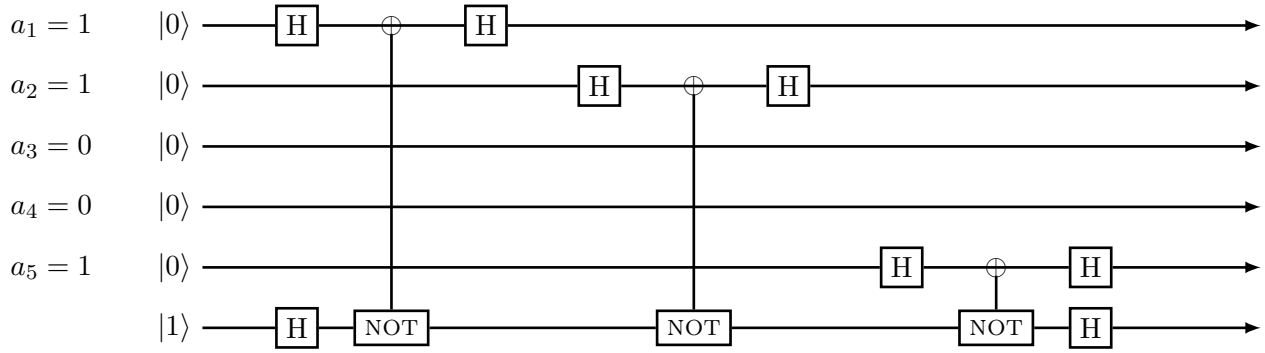
Here is a proof by picture on an example with $a = 11001$. We directly examine the circuit above using the circuit for U_f. The circuit to learn $\boldsymbol{a}$ is



(318)

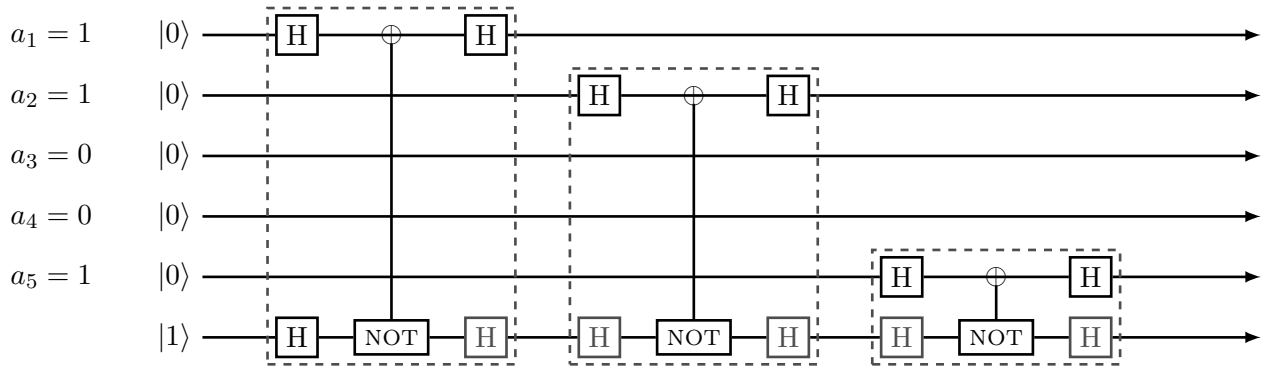
Note that on the third and fourth qubits where $a_3 = a_4 = 0$, the input bit is hit by $\mathrm{H}^2 = \mathrm{I}$ in sequence. The circuit can be simplified by removing pairs of Hadamards in sequence (the Hadamards in blue). We can also slide an operator along a qubit wire up to any other operator or controlling

bit. For example, the Hadamard on the right in the top qubit wire can slide all the way left up to the controlling bit. We cannot slide it further, because that would interfere with the controlling operator. Similarly, we can slide the left Hadamard on the 2nd qubit wire to the right, and the right Hadamard on this same wire to the left. After removing Hadamards in sequence and sliding the left Hadamards right and the right Hadamards left, the simplified circuit is



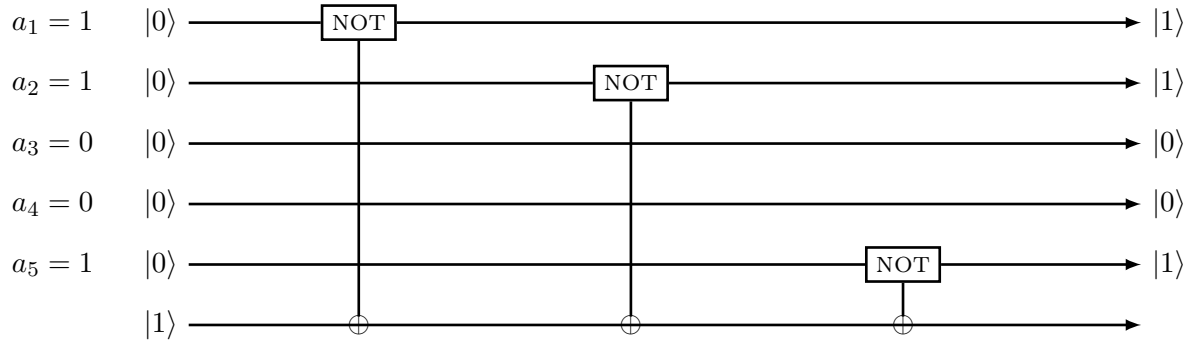
(319)

Now a trick. Between each pair of NOTs, insert a pair of Hadamards in sequence, which is allowed because $\mathrm{H}^2 = \mathrm{I}$. We get,



(320)

We highlighted three 2-qubit circuits in dashed boxes. Each is a ${}^{C}\mathrm{NOT}$ sandwiched between Hadamards. We saw in Section 16.2 on page 72 that such a circuit is equivalent to a ${}^{C}\mathrm{NOT}$, where the roles of the controlling bit and negated bit are interchanged. Hence, we get the equivalent circuit



(321)

For this simple circuit, one can now find the outputs of the top 5 qubits, because the controlling bit is $|1\rangle$ and so all the NOTs are activated on input qubits {1,2,5} and the other input qubits are unaltered. This circuit produces $\boldsymbol{a}$ at the output. The construction generalizes to any n and $\boldsymbol{a}$.

17.2 Algebraic Proof

It is instructive to work through the algebraic proof that the circuit in the previous section produces $\boldsymbol{a}$ on the outputs of the top n qubits. We need to analyze

$$\begin{aligned} & \mathrm{H}_n \left(\frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle \right) \qquad (322) \\ = \; & \frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} (-1)^{\boldsymbol{a}\cdot x}\mathrm{H}_n|x\rangle \\ = \; & \frac{1}{2^{n/2}} \sum_{x_1,\ldots,x_n\in\{0,1\}^n} (-1)^{a_1x_1\oplus a_2x_2\oplus\cdots\oplus a_nx_n} H|x_1\rangle\otimes H|x_2\rangle\otimes\cdots\otimes H|x_n\rangle \end{aligned}$$

Let us consider the case $a_1 = 0$. We get

$$\begin{aligned} & \frac{1}{2^{n/2}} \sum_{x_1,\ldots,x_n\in\{0,1\}^n} (-1)^{a_2x_2\oplus\cdots\oplus a_nx_n} H|x_1\rangle\otimes H|x_2\rangle\otimes\cdots\otimes H|x_n\rangle \qquad (323) \\ = \; & \frac{1}{2^{n/2}} \sum_{x_1,\in\{0,1\}} H|x_1\rangle \otimes \sum_{x_2,\ldots,x_n\in\{0,1\}^n} (-1)^{a_2x_2\oplus\cdots\oplus a_nx_n} H|x_2\rangle\otimes\cdots\otimes H|x_n\rangle \\ = \; & \frac{1}{2^{n/2}} H(|0\rangle+|1\rangle)\otimes \sum_{x_2,\ldots,x_n\in\{0,1\}^n} (-1)^{a_2x_2\oplus\cdots\oplus a_nx_n} H|x_2\rangle\otimes\cdots\otimes H|x_n\rangle \\ = \; & |0\rangle\otimes\frac{1}{2^{(n-1)/2}} \sum_{x_2,\ldots,x_n\in\{0,1\}^n} (-1)^{a_2x_2\oplus\cdots\oplus a_nx_n} H|x_2\rangle\otimes\cdots\otimes H|x_n\rangle. \end{aligned}$$

The last step follows because $\mathrm{H}^2 = \mathrm{I}$, so $= H(|0\rangle+|1\rangle) = \sqrt{2}H^2|0\rangle = \sqrt{2}|0\rangle$. So the first bit is in the pure state $|0\rangle$. Now suppose $a_1 = 1$. Summing over x_1, we get

$$\begin{aligned} & \frac{1}{2^{n/2}} \sum_{x_2,\ldots,x_n\in\{0,1\}^n} (-1)^{a_2x_2\oplus\cdots\oplus a_nx_n} H|0\rangle\otimes H|x_2\rangle\otimes\cdots\otimes H|x_n\rangle + \qquad (324) \\ & \frac{1}{2^{n/2}} \sum_{x_2,\ldots,x_n\in\{0,1\}^n} (-1)^{1\oplus a_2x_2\oplus\cdots\oplus a_nx_n} H|1\rangle\otimes H|x_2\rangle\otimes\cdots\otimes H|x_n\rangle \\ = \; & \frac{1}{2^{n/2}} H(|0\rangle-|1\rangle)\otimes \sum_{x_2,\ldots,x_n\in\{0,1\}^n} (-1)^{a_2x_2\oplus\cdots\oplus a_nx_n} H|x_2\rangle\otimes\cdots\otimes H|x_n\rangle \\ = \; & |1\rangle\otimes\frac{1}{2^{(n-1)/2}} \sum_{x_2,\ldots,x_n\in\{0,1\}^n} (-1)^{a_2x_2\oplus\cdots\oplus a_nx_n} H|x_2\rangle\otimes\cdots\otimes H|x_n\rangle. \qquad (325) \end{aligned}$$

The last step is because $= H(|0\rangle-|1\rangle) = \sqrt{2}H^2|1\rangle = \sqrt{2}|1\rangle$. This time, the first bit is in the pure state $|1\rangle$. We have shown that the first bit is in the pure state $|a_1\rangle$. The remaining sum is exactly of the same form as the full sum, but over one fewer bit. It follows by induction that every output qubit j is in the pure state $|a_j\rangle$. Measuring the n output qubits gives $|a_1\rangle\otimes|a_2\rangle\otimes\cdots\otimes|a_n\rangle$.

18 The Search Problem

You have a container of objects, for example a list of 2^n elements $L = \{z_1, z_2, \ldots, z_{2^n}\}$ and some target element z_*. The task is to find the element z_* in L, or equivalently to return the index of the element z_* in L. We can formulate this task using Boolean functions. Let x index the elements, x has n bits. Define the function

$$f(x) = \begin{cases} 1 & \text{if } z_x = z_*; \\ 0 & \text{if } z_x \neq z_*. \end{cases} \tag{326}$$

The goal is to find an x for which $f(x) = 1$. So, the general search task can be formulated as finding the 1's of a Boolean function: given a (quantum) black-box for the function $f(x)$, find an x for which $f(x) = 1$. Here is an example. Given a graph as an edge list, $G = e_{1,2}e_{1,3}\cdots, e_{n-1,n}$ define the function $f(x|G, K)$ with parameters G and $K \in \mathbb{N}$. The function takes an n bit input $x = x_1 \cdots x_n$ that identifies a vertex set. The function outputs 1 if and only if x is a clique in G of size at least K. We saw such "certifiers" before when we talked about NP-completeness. Finding an x for which $f(x|G, K) = 1$ is equivalent to finding a clique in G of size at least K.

18.1 Seaching for a Unique Element

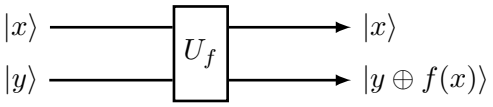
We first consider a simple version of the problem. There is exactly one x_* on which evaluates to 1. We have access to a (quantum) black-box function $f(x)$

$$f(x) = \begin{cases} 1 & \text{if } x = x_*; \\ 0 & \text{if } x \neq x_*. \end{cases} \tag{327}$$

The classical algorithm evaluates $f(x)$ on each of the 2^n possible x to find x_*. The worst case runtime is 2^n. A randomized algorithm which samples M of the x_i has a success probability $M/2^n$. A success probability of $1/2$ can be achieved with $2^n/2$ evaluations of f.

18.2 Quantum Circuit for f

For a quantum search algorithm, we are given a quantum black-box circuit U_f that implements $f(x)$.

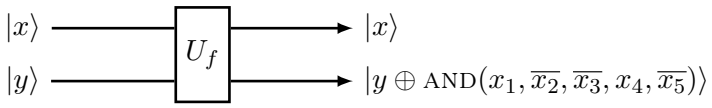


(328)

One can always specify the function using a small quantum circuit. This does not mean the function itself is given as a small circuit, only that it is possible to do so. To specify f, it suffices to specify x_*. Let us take an example with $x_* = 10010$, that is,

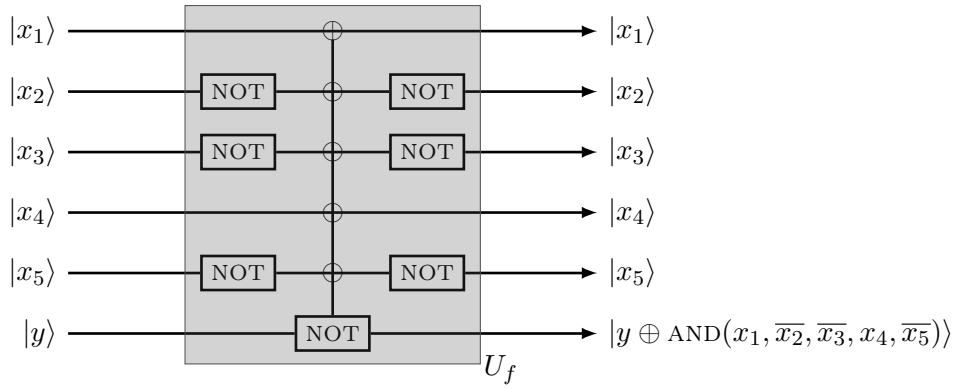
$$f(x) = \text{AND}(x_1, \overline{x_2}, \overline{x_3}, x_4, \overline{x_5}). \tag{329}$$

We need a circuit for U_f that implements



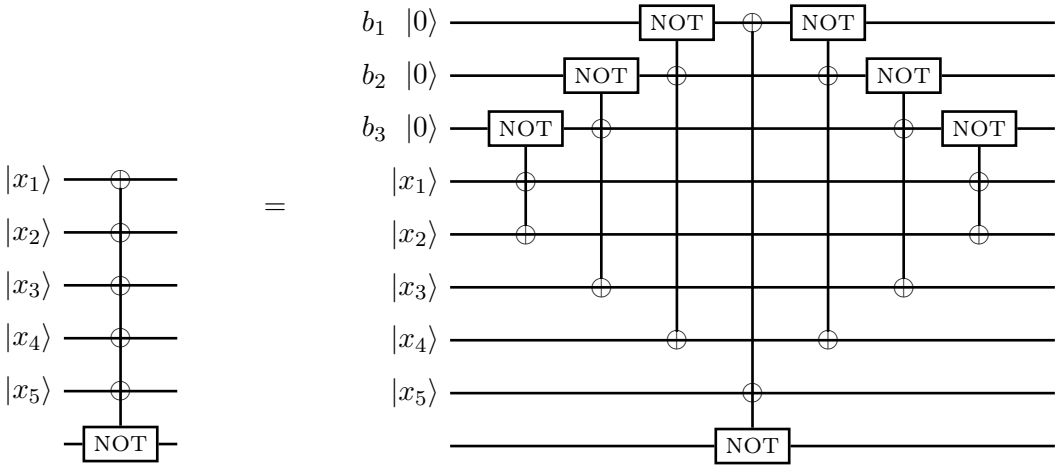
(330)

So, U_f needs to implement a controlled-NOT of $|y\rangle$, controlled by $\text{AND}(x_1, \overline{x_2}, \overline{x_3}, x_4, \overline{x_5})$,



(331)

The circuit above is nice and compact, but it does not use standard gates. You cannot ask your hardware team to manufacture quantum gates on demand. We need a circuit that uses only standard gates, for example c-NOT, cc-NOT (Toffoli), Hadamard, etc. We need to implement a massive AND of $|x_1\rangle, \ldots, |x_5\rangle$ and use this to control the NOT on $|y\rangle$. If we have auxillary qubits, we can do this using singly and doubly controlled gates. These auxillary qubits are sometimes called ancilliary qubits. Here is a simple way to do this.



(332)

Let's analyze the circuit equivalence above to make sure it works. Note that the ccNOT gates are just Toffoli gates. For classical inputs, the first NOT on ancillary qubit b_3 emits $|1\rangle$ if and only if $|x_1\rangle = |x_2\rangle = |1\rangle$. Then, b_2's NOT emits $|1\rangle$ if and only if b_3 and $|x_3\rangle$ are both $|1\rangle$ which is if and only if $|x_1\rangle = |x_2\rangle = |x_3\rangle = |1\rangle$. Then, b_1's NOT emits $|1\rangle$ if and only if b_2 and $|x_4\rangle$ are both $|1\rangle$ which is if and only if $|x_1\rangle = |x_2\rangle = |x_3\rangle = |x_4\rangle = |1\rangle$. Finally, the bottom not is controlled if and only if b_1 and $|x_4\rangle$ are both $|1\rangle$ which is if and only if $|x_1\rangle = |x_2\rangle = |x_3\rangle = |x_4\rangle = |x_5\rangle = |1\rangle$, as desired. Note, the remaining three Toffoli gates after the bottom NOT simply undo the actions of the first three Tofolli gates as the Toffoli gate is its own inverse.

The circuit for U_f can be compact. That does not mean the circuit given as a black-box will be implemented in the most compact way. Note also that ancilliary qubits are costly to initialize and maintain in a stable state. The construction above needs $n-2$ ancillary qubits. As a challenge, try

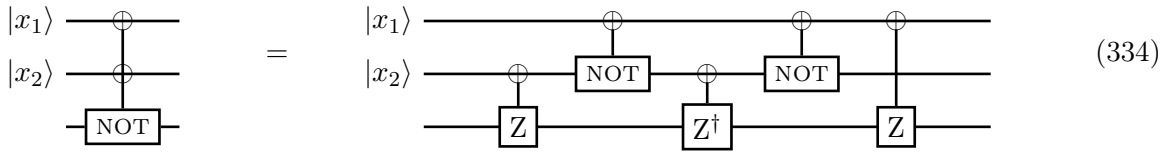
to implement this mulitiply controlled NOT using no ancillary qubits. We will give you some hints and you can fill in the details. Let Z be the 1-qubit circuit/gate defined by

$$\mathrm{Z} = \frac{1}{2}\begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}. \tag{333}$$

Note that Z can be implemented using basic gates as $\mathrm{H}R(\pi)\mathrm{H}$, where $R(\theta)$ is the phase gate defined in (190) on page 49. The important properties that Z satisfies are

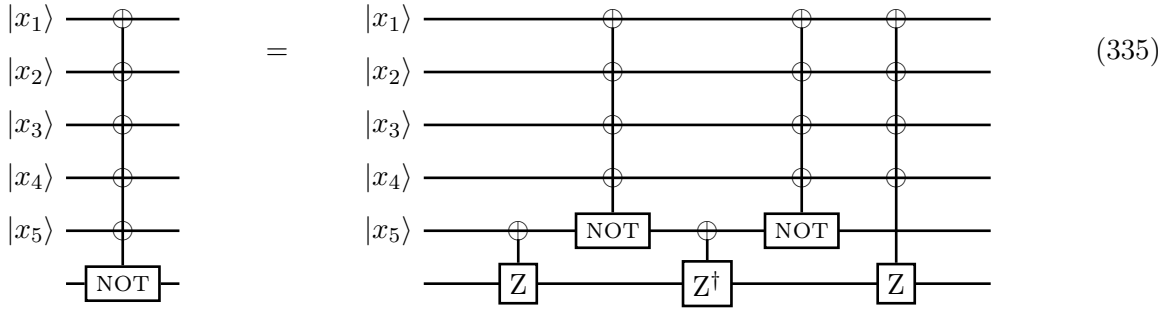
1. $\mathrm{Z}^2 = \text{NOT}$, and
2. Z is unitary.

Verify the following circuit equivalence, by checking the action on the standard basis or otherwise.



(334)

More generally, show the equivalence



(335)

Show that the NOT on the LHS can be replaced by an arbitrary unitary gate U with an appropriately defined Z that is unitary and satisfies $\mathrm{Z}^2 = \mathrm{U}$.

You can now use this equivalence to recursively construct a circuit for the 5-multiply controlled NOT using standard 1-qubit gates and controlled one-qubit gates. How many gates did you need for this construction. Note that our earlier construction used a linear number of Toffoli gates but required three additional work qubits. This current construction will produce an exponential sized circuit that does not need auxillary work-qubits. This is a common space-complexity tradeoff in computing where at the expense of extra workspace (one type of resource) the task can be solved with less complexity (another type of resource). It is an interesting question whether fewer gates can be used for the multiply controlled NOT when no auxilliary work-qubits are allowed.

18.3 Quantum Circuit for 3-SAT

As we saw in Section 6, 3-SAT is NP-complete. If we have an efficient quantum circuit for solving 3-SAT problems, we can use that to build an efficient circuit for any NP problem. Let's build a circuit for an instance of 3-SAT to illustrate the general idea. The basic idea is to represent an instance of 3-SAT as a Boolean function whose solutions are satisfying assignments. We can then use Grover's algorithm a circuit for the corresponding unitary operator U_f.

18.3.1 An Instance of 3-SAT

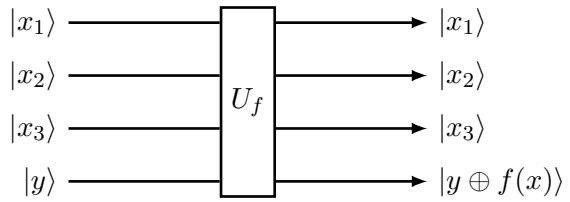
Consider this instance of 3-SAT with three clauses,

$$(x_1 \vee x_2 \vee \overline{x_3})(\overline{x_1} \vee \overline{x_2} \vee x_3)(\overline{x_1} \vee x_2 \vee x_3) \tag{336}$$

Recall the goal is to decide if there is a satisfying assignment to the variables so that every clause is TRUE. In this example there are many satisfying assignments. This example has $n = 3$ variables and $m = 3$ clauses. Define the Boolean function

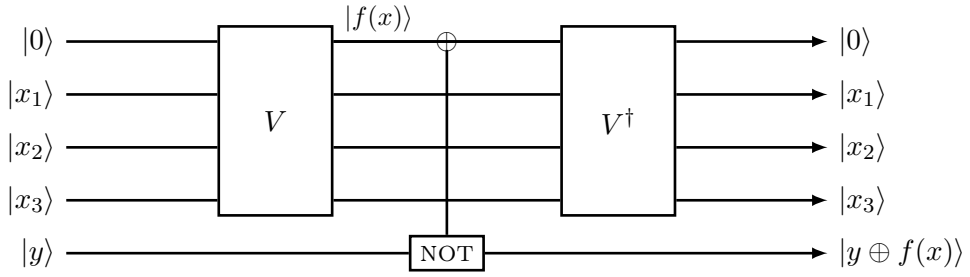
$$f(x_1, x_2, x_3) = \begin{cases} 1 & \text{if } (x_1 \vee x_2 \vee \overline{x_3})(\overline{x_1} \vee \overline{x_2} \vee x_3)(\overline{x_1} \vee x_2 \vee x_3) \text{ is TRUE;} \\ 0 & \text{otherwise.} \end{cases} \tag{337}$$

The function f is just evaluating an AND of three Boolean expressions. The experienced reader would recognize this as a disjunctive normal form (DNF) representation of the function f. recall that the equivalent unitary operator is defined by



(338)

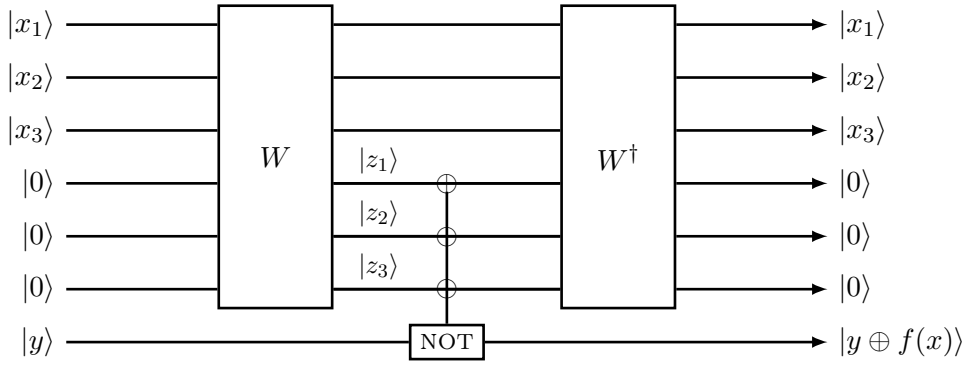
View the role of f as flipping the $|y\rangle$-bit whenever f is 1. That is, f is a controlling bit for a NOT on $|y\rangle$. We can equivalently get U_f from a circuit like



(339)

The operator V produces $|f(x)\rangle$ on the top ancillary bit, which is used to control the not on $|y\rangle$. Our function f is the AND of three clauses. Label the three clauses $|z_1\rangle, |z_2\rangle, |z_3\rangle$. Controlling the NOT on $|y\rangle$ with $|f\rangle$ is equivalent to a multiply controlled NOT on $|y\rangle$, where the controlling bits are $|z_1\rangle, |z_2\rangle, |z_3\rangle$. We discussed one circuit for controlling an operator with multiple bits in (332)

on page 80. Here is a circuit implementing this idea that uses additional ancillary bits.

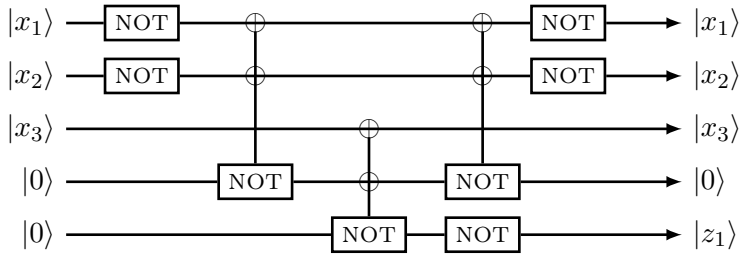


(340)

Let us find a circuit for $|z_1\rangle$. We use

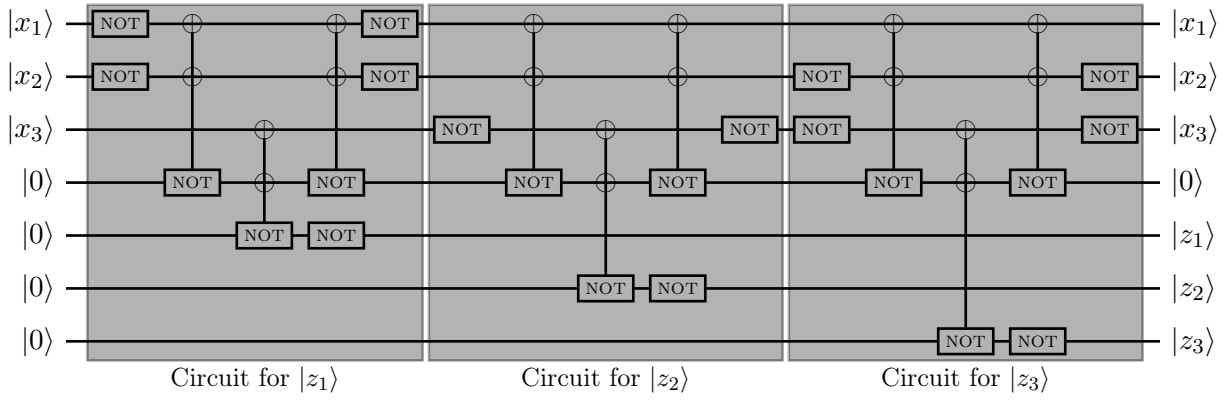
$$z_1 = x_1 \vee x_2 \vee \overline{x_3} \stackrel{\text{eqv}}{\equiv} \text{NOT}(\overline{x_1} \wedge \overline{x_2} \wedge x_3). \tag{341}$$

Since a multiple AND is just a multiply controlled NOT on a $|0\rangle$-bit, the reader should verify in detail that the following quantum circuit computes $|z_1\rangle$,



(342)

The NOTs and Toffolis on qubits 1,2,3,4 are undone by NOTs and Toffolis in reverse order, which is important so that the states $|x_1\rangle, |x_2\rangle, |x_3\rangle, |0\rangle$ are reproduced on the top four qubits. This means that the ancillary $|0\rangle$-qubit that was restored to $|0\rangle$ can be reused, together with $|x_1\rangle, |x_2\rangle, |x_3\rangle$ to produce $|z_2\rangle, |z_3\rangle$. A similar construction works to get $|z_2\rangle, |z_3\rangle$. We leave it to the reader to go through the details and derive the full circuit to compute $|z_1\rangle, |z_2\rangle, |z_3\rangle$ shown below,



(343)

The circuit in (343) can be plugged into (340) to implement U_f. Note that in a repeated use of U_f it is important to undo the effects of W by applying $W^\dagger$ so that the three ancillary qubits that produced $|z_1\rangle, |z_2\rangle, |z_3\rangle$ can be reused. Since the circuit only uses NOT and Toffoli, and both of these gates are their own inverse, one can simply apply the gates that have been used to get $|z_1\rangle, |z_2\rangle, |z_3\rangle$ in reverse order to obtain $W^\dagger$. This inverse has to be performed after using the clauses $|z_1\rangle, |z_2\rangle, |z_3\rangle$ to control the NOT on $|y\rangle$. In our example, we need three Toffolis and three NOTs to reset $|z_1\rangle, |z_2\rangle, |z_3\rangle$ to $|0\rangle, |0\rangle, |0\rangle$.

18.3.2 General Case

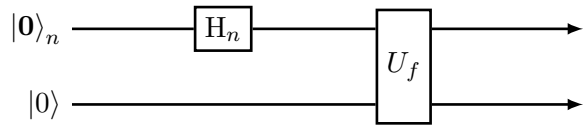
Consider an instance of 3-SAT with n variables $x_1, \ldots, x_n$ and m clauses $z_1, \ldots, z_m$. We need three Toffolis to implement each clause and at most seven additional NOTs. To reset each clause $|z_i\rangle$ to $|0\rangle$ we need one more Toffoli and NOT, so we need $4m$ Toffolis and at most $8m$ NOTs.

In terms of qubits, which are costly, we need n qubits for the variables, $m+1$ qubits initialized to $|0\rangle$ to compute the clauses. We also need to implement the multiply controlled NOT on $|y\rangle$ where the m clauses are the controlling bits. This requires $m-2$ qubits initialized to $|0\rangle$ using the construction in (332) on page 80 and $m-1$ Toffolis. We can reuse the qubit that is reset to $|0\rangle$ in (343), so we need an additional $m-1$ qubits. A more advanced construction does not even need these additional $m-1$ qubits, and can make do with the single $|0\rangle$ from (343). In summary, the resources we need are at most:

Toffoli	$5m-1$
NOT	$8m$
qubits	$n+2m-2$

18.4 Quantum Search – Warm Up

Let's start with the following setup.



$$|\phi\rangle = \frac{1}{2^{n/2}} \sum_{|x\rangle \in \{0,1\}^n} |x\rangle \otimes |f(x)\rangle \tag{344}$$

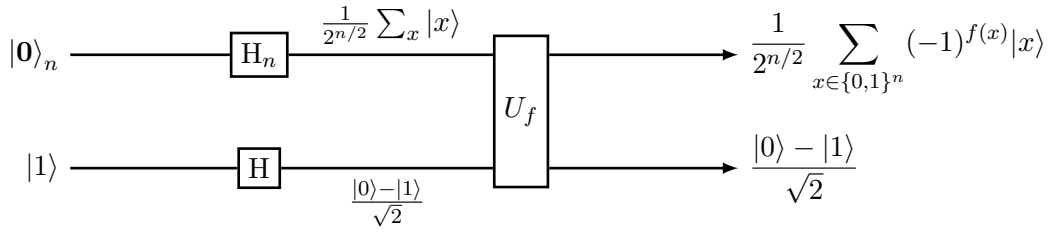
By now, we are expert in the analysis that leads to the expression for $|\phi\rangle$ above,

$$\begin{aligned}
|\phi\rangle &= U_f(\mathrm{H}_n|\mathbf{0}\rangle_n \otimes |0\rangle) && (345)\\
&= U_f\left(\frac{1}{2^{n/2}} \sum_x |x\rangle \otimes |0\rangle\right) && (346)\\
&= \frac{1}{2^{n/2}} \sum_x U_f(|x\rangle \otimes |0\rangle) && (347)\\
&= \frac{1}{2^{n/2}} \sum_x |x\rangle \otimes |f(x)\rangle). && (348)
\end{aligned}$$

This is great, in that $|\phi\rangle$ contains information about $f(x)$ on all x, but if we measure, the state will collapse uniformly randomly to some pure state $|x_0\rangle \otimes |f(x_0)\rangle$. The amplitudes of every pure

state are exactly the same, $1/2^{n/2}$. This is just random guess and check. The probability to get the correct x_* is $1/2^n$, exactly the same as classical guess and check.

Here is a second try. We leave you to derive the output state for this familiar setup.



(349)

Now, all the information about f is contained in the top n qubits. The amplitude for each pure state is $(-1)^{f(x)}/2^{n/2}$. The amplitude for $|x_*\rangle$ is $-1/2^{n/2}$. The amplitude for all the other $|x\rangle$ is $1/2^{n/2}$. This is good. There is an asymmetry between x_* and all the other x. However, if we measure the top n bits, we still have the same problem as guess and check. The probabilities are $1/2^n$ for all x, so we get a random $|x_0\rangle$. It's worse, because we don't even recover the value $f(x_0)$. So far, we haven't been able to do any better than random guess and check. Our short term goal is to find some way to do better than random guess and check with one function evaluation. Anything better, no matter how small the improvement, will be a breakthrough.

Here is an example. Suppose $x_* = 10$, so

$$f(00) = 0 \qquad f(01) = 0 \qquad f(10) = 1 \qquad f(11) = 0. \tag{350}$$

and

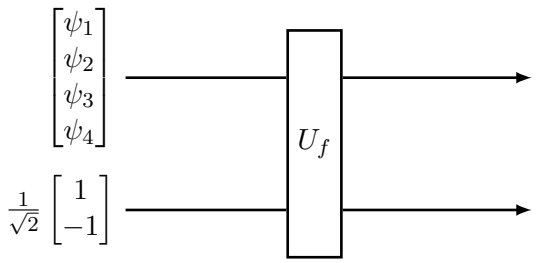
$$U_f = \begin{bmatrix} \boxed{\mathrm{I}_2} & 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} \\ 0_{2\times 2} & \boxed{\mathrm{I}_2} & 0_{2\times 2} & 0_{2\times 2} \\ 0_{2\times 2} & 0_{2\times 2} & \boxed{\mathrm{NOT}} & 0_{2\times 2} \\ 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} & \boxed{\mathrm{I}_2} \end{bmatrix} \tag{351}$$

The output state for this U_f is

$$|\phi\rangle = \begin{bmatrix} 1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix} \otimes \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}. \tag{352}$$

As we already said, measuring the top two qubits give a uniform distribution over all pure states. But all is not lost, because we have introduced an asymmetry for the state $|10\rangle$. Suppose we could somehow subtract $1/2$ from each amplitude for the top qubits. Then we would get $[0, 0, -1, 0]^{\mathrm{T}}$. Now if we measure, we are golden. The result will be $|10\rangle$ with probability 1. It turns out we will be able to do something like that, which is what Grover's iteration accomplishes. But for that you have to stay tuned to the next lecture.

A crucial ingredient in Grover's iteration is the following operator. Let's analyze it.



(353)

This operator is just U_f, but with its bottom bit set to $(|0\rangle - |1\rangle)/\sqrt{2}$. So, this is a linear operator on the top two bits. This discussion will generalize to n-bit functions, even if the function evaluates to 1 on more than one input. Let us compute the output. We want $U_f(|\psi\rangle \otimes \left[\begin{smallmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{smallmatrix}\right])$,

$$\begin{bmatrix} \boxed{\mathrm{I}_2} & 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} \\ 0_{2\times 2} & \boxed{\mathrm{I}_2} & 0_{2\times 2} & 0_{2\times 2} \\ 0_{2\times 2} & 0_{2\times 2} & \boxed{\text{NOT}} & 0_{2\times 2} \\ 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} & \boxed{\mathrm{I}_2} \end{bmatrix} \begin{bmatrix} \psi_1 \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix} \\ \psi_2 \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix} \\ \psi_3 \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix} \\ \psi_4 \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix} \end{bmatrix} \tag{354}$$

The highlighted red NOT only operates on the ψ_3 component. Its effect is to flip the top and bottom component, which is equivalent to multiplying by -1. The result is

$$U_f\left(|\psi\rangle \otimes \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix}\right) = \begin{bmatrix} \psi_1 \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix} \\ \psi_2 \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix} \\ \psi_3 \begin{bmatrix}-1/\sqrt{2}\\ 1/\sqrt{2}\end{bmatrix} \\ \psi_4 \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix} \end{bmatrix} = \begin{bmatrix} \psi_1 \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix} \\ \psi_2 \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix} \\ -\psi_3 \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix} \\ \psi_4 \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix} \end{bmatrix} = \begin{bmatrix}\psi_1\\ \psi_2\\ -\psi_3\\ \psi_4\end{bmatrix} \otimes \begin{bmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{bmatrix}.$$

The end effect of this operator is quite simple. It leaves the bottom qubit unaltered, and flips the sign of the amplitude for the component corresponding to x_*, the pure state on which f evaluates to 1. This is a general fact about this operator. If f had m solutions to $f(x) = 1$, then this operator, treated as an operator on the state of the top n qubits, flips the sign only of those components corresponding to the inputs x_* for which $f(x_*) = 1$, leaving the bottom qubit unaltered. This is a useful operator that can be invoked anytime we wish to flip the sign of a special few components, the ones corresponding to the pure states for which $f = 1$.

It is a useful exercise to consider the following f,

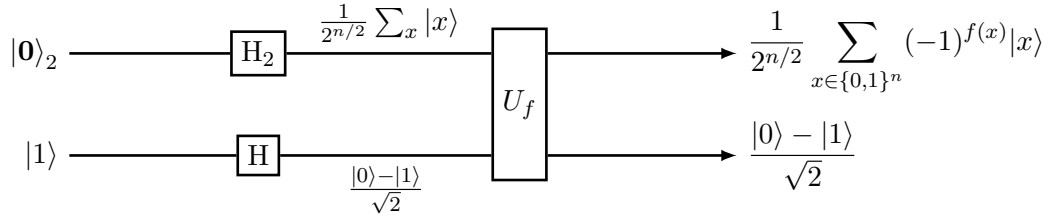
$$f(00) = 1 \qquad f(01) = 0 \qquad f(10) = 1 \qquad f(11) = 0. \tag{355}$$

Construct U_f and show that the operator in (353) flips the sign of two components of $|\psi\rangle$.

19 Grover's Iteration

We are considering this setup for a 2-bit function f,

$$f(00) = 0 \qquad f(01) = 0 \qquad f(10) = 1 \qquad f(11) = 0, \tag{356}$$

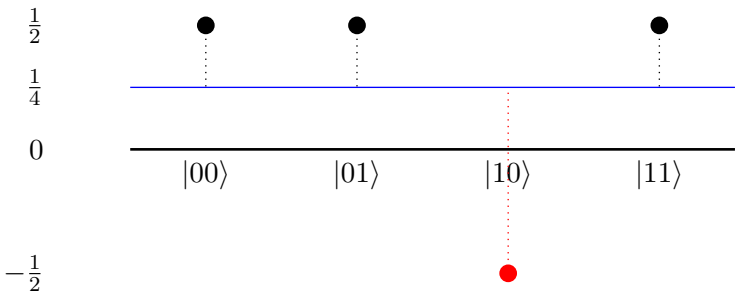


(357)

The state of the top qubits is

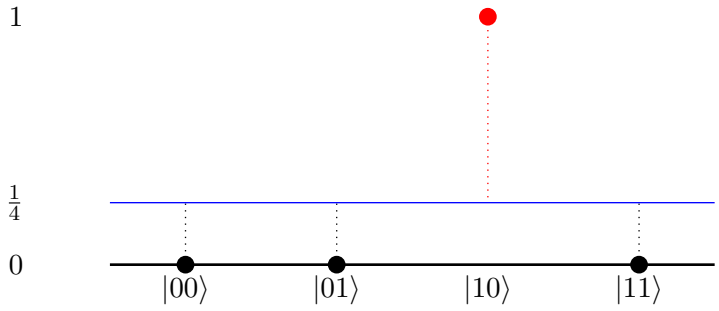
$$\begin{bmatrix} 1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix}. \tag{358}$$

Measuring the top qubits gives a random pure state. We successfully find $x_* = 10$ with probability 1/4. Our goal is to amplify the amplitude for $|10\rangle$ so that the measurement will yield x_* with a probability greater than 1/4. No matter how small the improvement above 1/4, it is a breakthrough. Here is Grover's great insight, illustrated with a pictorial view of these four amplitudes.



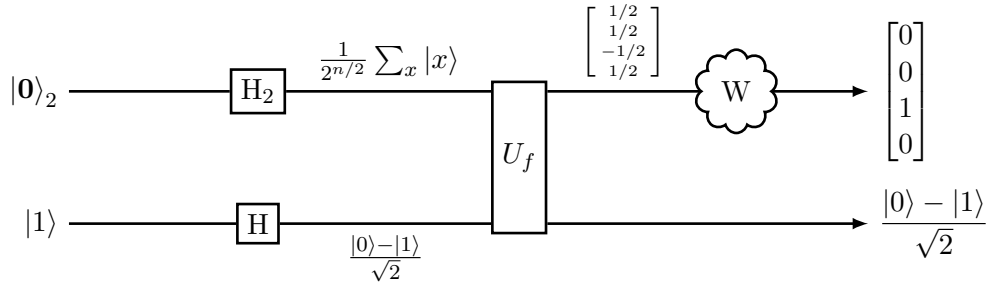
(359)

The blue line is the average amplitude. Since most amplitudes are 1/2, the average (blue line) is positive and the distance from the positive amplitudes to the average is smaller than from the negative amplitude for x_* to the average. Suppose we reflect all the amplitudes around the average.



(360)

A miracle has occurred. The amplitude for $|10\rangle$ is 1, and all other amplitudes are 0. A measurement yields $|x_*\rangle$ with probability 1. Let's call this wierd operation of reflecting the amplitudes around the average W. The situation after applying applying W is summarized in the picture below.

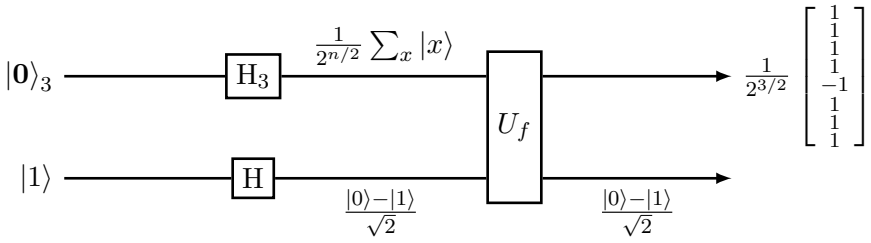


(361)

The situation is similar if $n > 2$. Let's consider $n = 3$ with $x_* = 100$, so

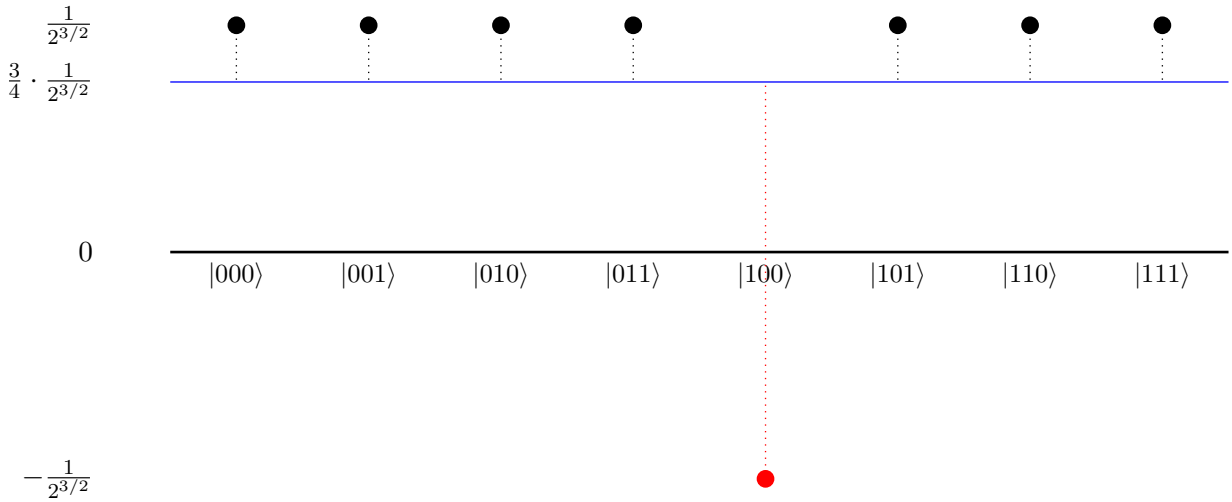
$$f(000) = 0, \quad f(001) = 0, \quad f(010) = 0, \quad f(011) = 0, \quad f(100) = 1, \quad f(101) = 0, \quad f(110) = 0, \quad f(111) = 0. \tag{362}$$

Here is the situation before we apply our wierd operation W,



(363)

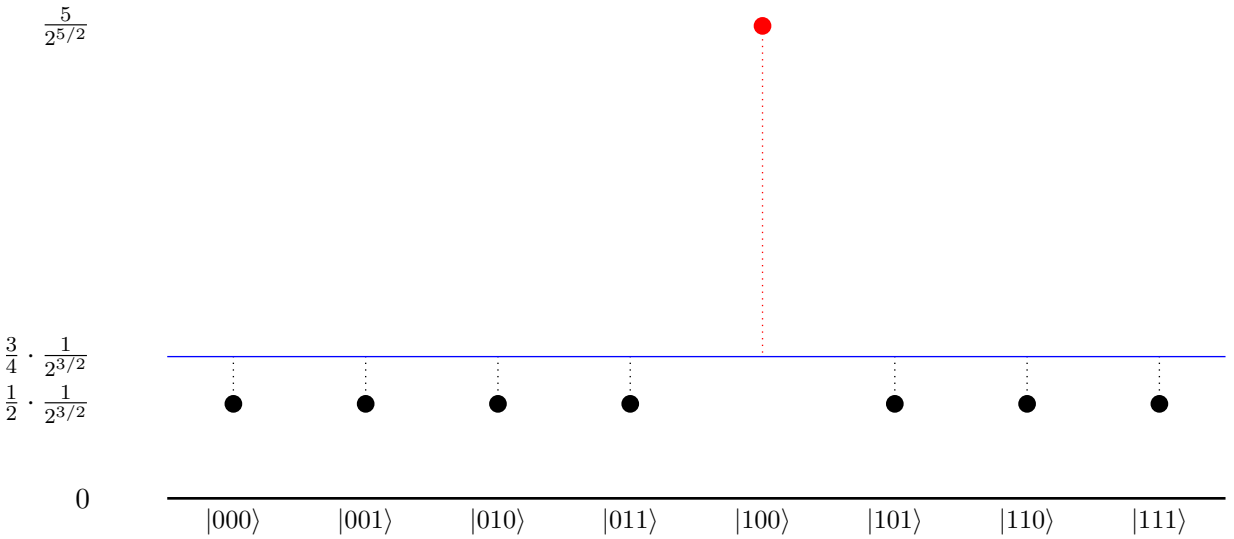
Pictorially, the amplitudes before applying W are:



(364)

The black amplitudes are at $1/2^{3/2}$, and the red one is at $-1/2^{3/2}$. The average (blue line) is at $3/2^{7/2}$, which is much closer to the black amplitudes (relatively speaking) than for the case $n = 2$.

To apply W, we reflect all the amplitudes about the average blue line. The results is



(365)

The probability to measure $|100\rangle$ after one use of U_f is

$$\mathbb{P}[\text{success}] = \left(\frac{5}{2^{5/2}}\right)^2 = \frac{25}{32} = 0.78125 \gg \frac{1}{8}. \tag{366}$$

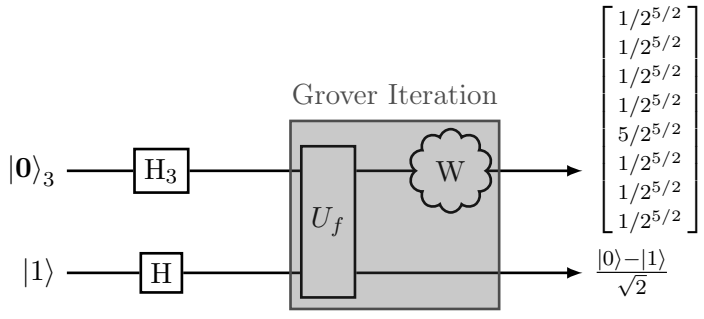
It is not as big of a miracle as with the $n = 2$ case, where the probability of success was 1. For the general n-bit function, the situation is similar. After applying U_f, all amplititudes are $1/2^{n/2}$, except for the solution $|x_*\rangle$ whose amplitude is $-1/2^{n/2}$. The average is

$$\frac{1}{2^n} \times \frac{2^n - 2}{2^{n/2}}. \tag{367}$$

Reflecting the amplitude of x_* about this average gives

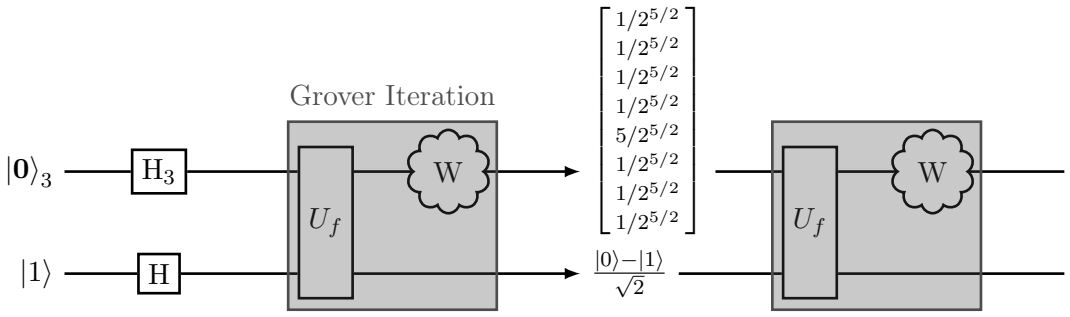
$$\mathbb{P}[\text{success}] = \frac{1}{2^n}\left(3 - \frac{4}{2^n}\right)^2. \tag{368}$$

When n gets large, the probability of success is approximately $9/2^n$, which is miniscule. You might think this is nothing to write home about, but it is a breakthrough, because it is 9 times bigger than random guess and check! Applying U_f followed by W is one Grover iteration,



(369)

What if we apply a second Grover Iteration,



(370)

We leave it to the reader to show that the state for the top 3 qubits is

$$\begin{bmatrix} -1/2^{7/2} \\ -1/2^{7/2} \\ -1/2^{7/2} \\ -1/2^{7/2} \\ 11/2^{7/2} \\ -1/2^{7/2} \\ -1/2^{7/2} \\ -1/2^{7/2} \end{bmatrix}. \tag{371}$$

The probability of success is now

$$\mathbb{P}[\text{success}] = \left(\frac{11}{2^{7/2}}\right)^2 = \frac{121}{128} \approx 0.95, \tag{372}$$

and we are in business. We can go wild and continue doing Grover iterations. The obvious cost is more evaluations of U_f, but something worse happens. We leave it to you to show that with another Grover iteration, the probability of success goes down. There is an optimal number of Grover Iterations. In this case you can see why. The non-solution components have become negative. So those amplitudes will get larger in absolute value after reflecting about the average. We will analyze this in the next lecture. Before that, let us examine more carefully this wierd operation W. It turns out to not be so wierd.

19.1 Operator for Reflecting About the Average

The operation W which reflects about the average is an important technique for amplifying the amplitude of specific components of the state. Starting from state $|\psi\rangle$, reflecting about the average transforms each component ψ_i as follows,

$$\psi_i \to \overline{\psi} - (\psi_i - \overline{\psi}) = 2\overline{\psi} - \psi_i. \tag{373}$$

In vector form,

$$\mathrm{W}|\psi\rangle = 2\mathbf{1}\overline{\psi} - |\psi\rangle. \tag{374}$$

Averaging is a linear operation, so W is a linear operation. Indeed, $\overline{\psi} = \mathbf{1}^{\mathrm{T}}|\psi\rangle/2^n$, and we have

$$\mathrm{W}|\psi\rangle = \frac{2}{2^n}\mathbf{1}\mathbf{1}^{\mathrm{T}}|\psi\rangle - |\psi\rangle, \tag{375}$$

from which we identify the operator W as

$$\mathrm{W} = \frac{2}{2^n}\mathbf{1}\mathbf{1}^{\mathrm{T}} - \mathrm{I}. \tag{376}$$

W is hermitian, and even more, it is unitary,

$$\begin{aligned}
\mathrm{W}^{\dagger}\mathrm{W} &= \left(\frac{2}{2^n}\mathbf{1}\mathbf{1}^{\mathrm{T}} - \mathrm{I}\right)\left(\frac{2}{2^n}\mathbf{1}\mathbf{1}^{\mathrm{T}} - \mathrm{I}\right) && (377)\\
&= \frac{4}{2^{2n}}\mathbf{1}\mathbf{1}^{\mathrm{T}}\mathbf{1}\mathbf{1}^{\mathrm{T}} - \frac{4}{2^n}\mathbf{1}\mathbf{1}^{\mathrm{T}} + \mathrm{I} && (378)\\
&= \frac{4}{2^{2n}}\mathbf{1}\underbrace{(\mathbf{1}^{\mathrm{T}}\mathbf{1})}_{2^n}\mathbf{1}^{\mathrm{T}} - \frac{4}{2^n}\mathbf{1}\mathbf{1}^{\mathrm{T}} + \mathrm{I} && (379)\\
&= \cancel{\frac{4\cdot 2^n}{2^{2n}}\mathbf{1}\mathbf{1}^{\mathrm{T}}} - \cancel{\frac{4}{2^n}\mathbf{1}\mathbf{1}^{\mathrm{T}}} + \mathrm{I} && (380)\\
&= \mathrm{I}. && (381)
\end{aligned}$$

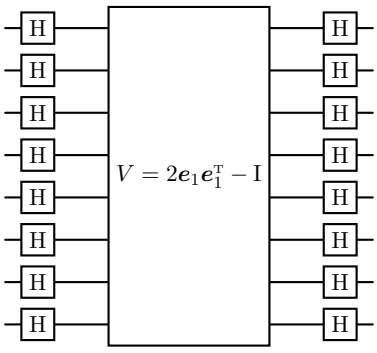
This means W can be implemented by a quantum circuit. In fact, W can be implemented by a compact circuit using the standard one and two bit gates. Let's see how. We need to first massage W into a more suitable form. Recall that $\mathrm{H}_n|\mathbf{0}\rangle_n = \mathbf{1}/2^{n/2}$. Since $|\mathbf{0}\rangle_n$ is the first standard basis vector $\boldsymbol{e}_1$,

$$\mathrm{W} = 2\mathrm{H}_n\boldsymbol{e}_1\boldsymbol{e}_1^{\mathrm{T}}\mathrm{H}_n - \mathrm{I}. \tag{382}$$

Using the fact that $\mathrm{H}_n^2 = \mathrm{I}$,

$$\mathrm{W} = \mathrm{H}_n\underbrace{(2\boldsymbol{e}_1\boldsymbol{e}_1^{\mathrm{T}} - \mathrm{I})}_{\mathrm{V}}\mathrm{H}_n. \tag{383}$$

Since $\mathrm{H}_n = H^{\otimes n}$, we have that W is the circuit



(384)

We now need a circuit for V to be done. It suffices to get a circuit for $-\mathrm{V}$ because this has the effect of just flipping the sign of all the amplitudes in $|\psi\rangle$, which is a benign operation to the state in quantum mechanics (it does not affect the measurement probabilities). We need a circuit for

$$-V = \mathrm{I} - 2\boldsymbol{e}_1\boldsymbol{e}_1^{\mathrm{T}}. \tag{385}$$

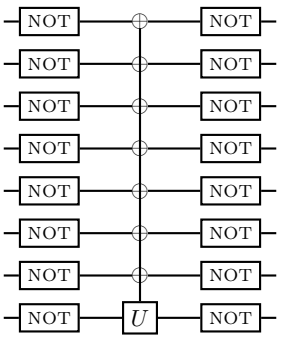
Let us first see what the operator $-\mathrm{V}$ looks like, as a matrix. We need to examine its action on the pure basis states $\boldsymbol{e}_i$. Using $\boldsymbol{e}_i^{\mathrm{T}}\boldsymbol{e}_j = \delta_{ij}$,

$$\begin{cases} -\mathrm{V}\boldsymbol{e}_1 = -\boldsymbol{e}_1; & \\ -\mathrm{V}\boldsymbol{e}_i = \boldsymbol{e}_i & i \neq 1. \end{cases} \tag{386}$$

We can now write down $-\mathrm{V}$ as a matrix,

$$-\mathrm{V} = \begin{bmatrix} -\boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{e}_3 & \cdots & \boldsymbol{e}_{2^n} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}. \tag{387}$$

Recall that a matrix of the form $\left[\begin{smallmatrix} \mathrm{I} & \mathbf{0} \\ \mathbf{0} & U \end{smallmatrix}\right]$ is a multiply controlled-U. Our $-\mathrm{V}$ is of the form $\left[\begin{smallmatrix} U & \mathbf{0} \\ \mathbf{0} & \mathrm{I} \end{smallmatrix}\right]$, also a multiply controlled-U with some minor role modifications, such as negations. The reader may wish to review Lecture 11 and compute the operator for the following circuit with $U = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$,



(388)

We need to analyze the action of the circuit on the pure basis states. The only non-trivial cases are when U is activated, because otherwise this circuit acts as the identity. We only need to consider the case where all the top bits are 0, $|000\cdots00\rangle$ and $|000\cdots01\rangle$.

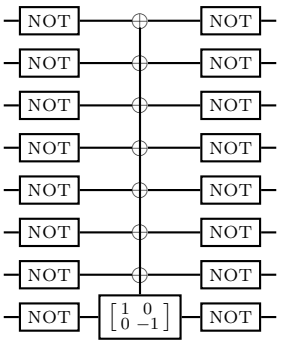
$$|0\rangle \otimes |0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle \otimes |0\rangle \quad \to \quad |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle \otimes \textsc{not} \cdot U|1\rangle \tag{389}$$

$$|0\rangle \otimes |0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle \otimes |0\rangle \quad \to \quad |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle \otimes \textsc{not} \cdot U|0\rangle. \tag{390}$$

Since $\textsc{not} \cdot U|1\rangle = \left[\begin{smallmatrix} d \\ b \end{smallmatrix}\right]$ and $\textsc{not} \cdot U|0\rangle = \left[\begin{smallmatrix} c \\ a \end{smallmatrix}\right]$, we have that the operator for this circuit is

$$\begin{bmatrix} d & c & 0 & 0 & \cdots & 0 & 0 \\ b & a & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}. \tag{391}$$

By inspection, we can write down the circuit for −V,



(392)

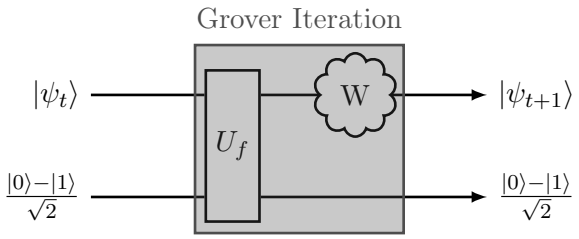
The operator being controlled is the Pauli spin-z operator σ_z, and is considered a basic 1-qubit gate. Alternatively, you can implement it as $\mathrm{H} \cdot \textsc{not} \cdot H$,

$$\sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = H \cdot \textsc{not} \cdot H. \tag{393}$$

We can now use a construction such as in (332) on page 80 to implement the multiply controlled circuit above using standard gates. Plugging −V into (383), we get the circuit for W.

20 Analysis of Grover's Search Algorithm

It should not surprise you that computer scientists have had a healthy dose of mathematics. You need to take your medicine because a computer is essentially a mathematical device. If you want to analyze any algorithm on any architecture, you need to do so mathematically. We are going to analyze Grover's iteration, in the general case where $f(x) = 1$ has m solutions.



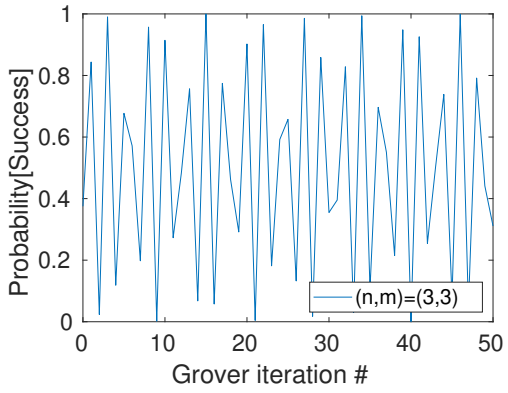
(394)

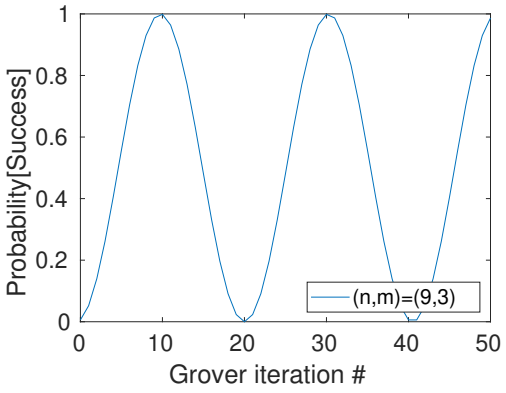
In this setup, U_f flips the sign of all the components corresponding to the pure states which are solutions to $f(x) = 1$. Then, W reflects all the amplitides about the average. The Grover iteration begins with $|\psi_0\rangle = \mathbf{1}/2^{n/2}$. Here is an example with $n = 3$ and $m = 3$. The components corresponding to solutions of $f(x) = 1$ are in bold (amplitudes are rounded).

	$\|\psi_0\rangle$		$\|\psi_1\rangle$		$\|\psi_2\rangle$		$\|\psi_3\rangle$		$\|\psi_4\rangle$		$\|\psi_5\rangle$
$\|000\rangle$	0.354		−0.177		−0.442		−0.044		0.420		0.254
$\mathbf{\|001\rangle}$	**0.354**		**0.530**		**−0.088**		**−0.575**		**−0.199**		**0.475**
$\mathbf{\|010\rangle}$	**0.354**		**0.530**		**−0.088**		**−0.575**		**−0.199**		**0.475**
$\|011\rangle$	0.354		−0.177		−0.442		−0.044		0.420		0.254
$\|100\rangle$	0.354	$\rightarrow$	−0.177	$\rightarrow$	−0.442	$\rightarrow$	−0.044	$\rightarrow$	0.420	$\rightarrow$	0.254
$\|101\rangle$	0.354		−0.177		−0.442		−0.044		0.420		0.254
$\mathbf{\|110\rangle}$	**0.354**		**0.530**		**−0.088**		**−0.575**		**−0.199**		**0.475**
$\|111\rangle$	0.354		−0.177		−0.442		−0.044		0.420		0.254
$\mathbb{P}$[success]	0.375		0.844		0.023		0.990		0.119		0.677

(395)

Success occurs if a bold pure state is measured, so the success probability is the sum of squares of the bold entries in the state. We plot the probability of success versus Grover iteration # below,





There are two takeaways from the example above. The first takeaway is repeated application of the Grover iteration does not monotonically increase the probability of success. The success probability oscillates. On the left, we show the the example with $n = 3$ and $m = 3$. The behavior looks rather erratic. On the right, we show the case with $n = 9, m = 3$. As you can see things get much smoother. We wish to perform the fewest iterations (evaluations of U_f), so we need to decide when to stop iterating. We should stop when the success probability stops increasing. In this case, after 1 iteration. This does not give the maximum probability, but one that is high enough that it can be boosted to as large a probability that we want by repeating the whole process. For example, if we run for 1 iteration and repeat three times, we get a success probability of 0.996.

The second takeaway is the bold entries corresponding to solutions stay equal. This is because they start that way and the operators U_f and W treat them symmetrically. The same is true of the non-bold entries corresponding to non-solutions. Let x_t be the value of a bold entry at iteration t and y_t the value of a non-bold entry. The initial condition is

$$x_0 = y_0 = \frac{1}{2^{n/2}}. \tag{396}$$

The probability of success after t iterations is

$$\mathbb{P}[\text{success}] = m x_t^2. \tag{397}$$

Our first task is to find x_t, y_t after t iterations.

20.1 Grover's Coupled Recurrence

A Grover iteration updates the vector $\left[\begin{smallmatrix} x_t \\ y_t \end{smallmatrix}\right]$ to $\left[\begin{smallmatrix} x_{t+1} \\ y_{t+1} \end{smallmatrix}\right]$. In the first step U_f is applied,

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} \to \begin{bmatrix} -x_t \\ y_t \end{bmatrix} \tag{398}$$

In the second step, we apply W. The average amplitude is $((2^n - m)y_t - m x_t)/2^n$ and each amplitude $\psi_i \to 2\overline{\psi} - \psi_i$, so

$$x_{t+1} = 2 \times \frac{(2^n - m)y_t - m x_t}{2^n} + x_t \tag{399}$$

$$y_{t+1} = 2 \times \frac{(2^n - m)y_t - m x_t}{2^n} - y_t. \tag{400}$$

Simplifying and defining $q = m/2^{n-1}$,

$$x_{t+1} = (1 - q)x_t + (2 - q)y_t \tag{401}$$

$$y_{t+1} = -q x_t + (1 - q)y_t. \tag{402}$$

$$\tag{403}$$

It is convenient to define the vector $\mathbf{z}_t = \left[\begin{smallmatrix} x_t \\ y_t \end{smallmatrix}\right]$. Then,

$$\mathbf{z}_{t+1} = \mathrm{A}\mathbf{z}_t, \tag{404}$$

where A is the matrix

$$\mathrm{A} = \begin{bmatrix} 1-q & 2-q \\ -q & 1-q \end{bmatrix}. \tag{405}$$

This is a recurrence for a vector, a generalization of a simple recurrence. Since A is not diagonal, this is a coupled recurrence, or discrete dynamical system. We will solve this recurrence using a general technique for solving such recurrences, a technique that is worth mastering and storing away for future use.

20.2 Solving Grover's Recurrence

The first step in solving the system is to analyze the driver of the system, namely the matrix A. In particular, we look at the spectral structure of A, its eigenvalues $\lambda_\pm$ and eigenvectors $\mathbf{v}_\pm$ defined by

$$\mathrm{A}\mathbf{v}_\pm = \lambda_\pm \mathbf{v}_\pm. \tag{406}$$

The eigenvalues are the roots of the characteristic polynomial,

$$(1-q-\lambda)^2 + q(2-q) = 0. \tag{407}$$

An exercise for the reader is to compute the roots of this quadratic,

$$\lambda_\pm = (1-q) \pm i\sqrt{q(2-q)}. \tag{408}$$

Using these computed eigenvalues, the eigenvectors are then obtained by solving (406). Another exercise for the reader is to show that the eigenvectors are

$$\mathbf{v}_\pm = \begin{bmatrix} 1 \\ \pm i\sqrt{\frac{q}{2-q}} \end{bmatrix}. \tag{409}$$

Armed with the eigenvalues and eigenvectors, we are ready to solve Grover's recurrence. We start with $\mathbf{z}_0$,

$$\mathbf{z}_0 = \begin{bmatrix} 1/2^{n/2} \\ 1/2^{n/2} \end{bmatrix}. \tag{410}$$

Since the eigenvectors $\mathbf{v}_\pm$ are independent, they are a basis Therefore $\mathbf{z}_0 = \alpha\mathbf{v}_+ + \beta\mathbf{v}_-$. Again, the reader may show that

$$\alpha = \frac{1}{2^{1+n/2}}\left(1 - i\sqrt{\frac{2-q}{q}}\right), \tag{411}$$

$$\beta = \frac{1}{2^{1+n/2}}\left(1 + i\sqrt{\frac{2-q}{q}}\right). \tag{412}$$

The matrix A behaves like a scaler when it operates on an eigenvector,

$$\mathrm{A}^t\mathbf{v}\pm = \lambda_\pm^t \mathbf{v}_\pm. \tag{413}$$

Since $\mathbf{z}_t = \mathrm{A}^t\mathbf{z}_0$, by linearity,

$$\mathbf{z}_t = \mathrm{A}^t\mathbf{z}_0 \tag{414}$$

$$= \alpha\lambda_+^t\mathbf{v}_+ + \beta\lambda_-^t\mathbf{v}_-. \tag{415}$$

We want x_t which is the first component of $\mathbf{z}_t$. Since the first component of $\mathbf{v}_\pm$ is 1, we have that

$$\begin{aligned} x_t &= \alpha\lambda_+^t + \beta\lambda_-^t && (416)\\ &= \alpha\left((1-q)+i\sqrt{q(2-q)}\right)^t + \beta\left((1-q)-i\sqrt{q(2-q)}\right)^t. && (417)\end{aligned}$$

We need to exponentiate complex numbers. That is no problem in polar form. We write

$$(1-q) \pm i\sqrt{q(2-q)} = e^{\pm i\theta}, \tag{418}$$

where

$$\tan\theta = \frac{\sqrt{q(2-q)}}{1-q}. \tag{419}$$

Then,

$$x_t = \alpha e^{it\theta} + \beta e^{-it\theta} \tag{420}$$

Using the expressions for α and β in (412), after some algebra,

$$\begin{aligned} x_t &= \frac{1}{2^{1+n/2}}(e^{it\theta}+e^{-it\theta}) - \frac{i}{2^{1+n/2}}\sqrt{\frac{2-q}{q}}(e^{it\theta}-e^{-it\theta}) && (421)\\ &= \frac{i}{2^{n/2}}\cos t\theta + \frac{i}{2^{n/2}}\sqrt{\frac{2-q}{q}}\sin t\theta. && (422)\end{aligned}$$

We now have the probability of success, mx_t^2,

$$\mathbb{P}[\text{success}] = \frac{m}{2^n}\left(\cos t\theta + \sqrt{\frac{2-q}{q}}\sin t\theta\right)^2. \tag{423}$$

We can simplify further using a trigometric identities from highschool,

$$\cos X + \gamma \sin X = \sqrt{1+\gamma^2}\sin(X+\phi), \tag{424}$$

where $\tan\phi = 1/\gamma$. In our case, $X = t\theta$ and $\gamma\sqrt{(2-q)/q}$. Since $\sqrt{1+\gamma^2} = \sqrt{2/q}$, we have

$$\mathbb{P}[\text{success}] = \frac{2m}{2^n q}\sin^2(t\theta+\phi), \tag{425}$$

where $\tan\phi = \sqrt{q/(2-q)}$. Using $q = m/2^{n-1}$ completes the calculation, giving

$$\mathbb{P}[\text{success}] = \sin^2(t\theta+\phi), \tag{426}$$

where

$$\begin{aligned} \tan\theta &= \frac{\sqrt{q(2-q)}}{1-q}; && (427)\\ \tan\phi &= \sqrt{\frac{q}{2-q}}. && (428)\end{aligned}$$

The oscillatory behavior for the success probability is now explained. Also the success probability is maximized when $t\theta + \phi = \pi/2$. We can increase t until

$$t_* = \left\lfloor \frac{\pi/2 - \phi}{\theta} \right\rfloor. \tag{429}$$

After this t_*, the success probability will start to decrease and begin its oscillatory behavior. When $m \ll 2^{n-1}$, $q \approx 0$. This means

$$\tan\theta \approx \sqrt{2q} = 2\sqrt{m/2^n} \to \theta \approx 2\sqrt{m/2^n}, \tag{430}$$

and

$$t_* \approx \frac{\pi}{2\theta} \approx \frac{\pi}{4}\sqrt{\frac{2^n}{m}}. \tag{431}$$

Since $t_* \geq (\pi/2 - \phi)/\theta - 1$,

$$\mathbb{P}[\text{success}] \geq \sin^2(\pi/2 - \theta) = \cos^2\theta \approx (1 - \theta^2/2)^2 \approx 1 - m/2^{n-2}. \tag{432}$$

So, the probability of success is exponentially approaching 1. The number of iterations is driven by $\sqrt{2^n}$. In contrast the classical approach needs 2^n, a huge gain for the quantum approach. Unfortunately, the dependence on the number of solutions is $1/\sqrt{m}$ in the quantum algorithm. In the classical algorithm, the dependence on n is $1/m$, so when m is large, the classical approach will start winning. But when m is large enough for this to be true, the search problem is easy anyway, so we only really worry about the $m \ll 2^n$ regime.

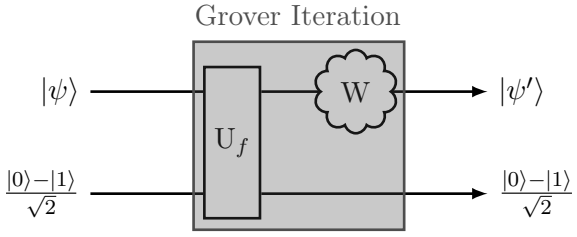
20.3 Unknown Number of Solutions

To set the number of Grover iterations, we need to know m. For the algorithm to be viable in practice, it must handle three cases, $m = 0$, $m \in o(2^{n/2})$ and $m \in \Omega(2^{n/2})$. We can use standard algorithmic techniques to build on the algorithm with known m within a guess m and check framework.

A better approach is to first estimate m and then use this estimate to run the appropriate number of Grover iterations to find a solution.

21 Quantum Counting and Phase Estimation

The optimal number of Grover iterations depends on m, the number of solutions to $f(x) = 1$, $x \in \{0,1\}^n$. Assume $m < 2^n/2$, because otherwise classical search works well. With K calls to the classical black box for f, we can find a solution with probability at least $1 - 2^{-K}$ when $m \geq 2^n/2$. So we can always start with a classical brute force random search. If this fails to find a solution in 100 tries, it is near-certain that $m < 2^n/2$. We can now switch over to quantum search, which runs the optimal number of Grover iterations. To do so we need to determine m, which means we need to analyze the Grover operator G which operates on a state $|\psi\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2}$,



(433)

$$\mathrm{G} = (\mathrm{W} \otimes \mathrm{I})\mathrm{U}_f,$$

Here, W is the unitary operator which reflects the amplitudes about the average and U_f is the black box quantum circuit for f. In the Grover search algorithm, the initial input to this operator is $|\psi\rangle = \mathbf{1}_N/\sqrt{N}$, where $N = 2^n$. Consider a function with $m = 3$ ones, for example

$$f(000) = 0, \quad f(001) = 1, \quad f(010) = 0, \quad f(011) = 0, \quad f(100) = 1, \quad f(101) = 1, \quad f(110) = 0, \quad f(111) = 0. \tag{434}$$

Define two N-dimensional orthonormal vectors $|\psi_1\rangle$ and $|\psi_2\rangle$,

$$\begin{array}{cccc} & & |\psi_1\rangle & |\psi_2\rangle \\ |000\rangle & & 1 & 0 \\ |001\rangle & & 0 & 1 \\ |010\rangle & & 1 & 0 \\ |011\rangle & \frac{1}{\sqrt{5}} & 1 & \frac{1}{\sqrt{3}} \; 0 \\ |100\rangle & & 0 & 1 \\ |101\rangle & & 0 & 1 \\ |110\rangle & & 1 & 0 \\ |111\rangle & & 1 & 0 \end{array} \tag{435}$$

$|\psi_1\rangle$ corresponds to the pure states where f evaluates to 0 and $|\psi_2\rangle$ corresponds to the pure states where f evaluates to 1. More generally,

$$|\psi_1\rangle = \frac{1}{\sqrt{N-m}} \sum_{|x\rangle : f(x)=0} |x\rangle; \tag{436}$$

$$|\psi_2\rangle = \frac{1}{\sqrt{m}} \sum_{|x\rangle : f(x)=1} |x\rangle. \tag{437}$$

Note that

$$\mathbf{1} = \sqrt{N-m} \cdot |\psi_1\rangle + \sqrt{m} \cdot |\psi_2\rangle. \tag{438}$$

Let us start with a few observations. The initial state $|\phi_0\rangle$ for the first Grover iteration is

$$|\phi_0\rangle = \frac{1}{\sqrt{N}} \cdot \mathbf{1} = \sqrt{\frac{N-m}{N}} \cdot |\psi_1\rangle + \sqrt{\frac{m}{N}} \cdot |\psi_2\rangle. \tag{439}$$

In the two dimensional subspace spanned by the two vectors $\{|\psi_1\rangle, |\psi_2\rangle\}$, the polar form for $|\phi_0\rangle$ is (r, Ω) where $r = 1$ and $\sin\Omega = \sqrt{m/N}$. Let us consider the action of the Grover operator G on any vector in the subspace spanned by $\{|\psi_1\rangle, |\psi_2\rangle\}$. A vector in this subspace is given by

$$|\psi\rangle = \alpha|\psi_1\rangle + \beta|\psi_2\rangle. \tag{440}$$

Note that

$$\begin{aligned} \mathrm{U}_f(|\psi_1\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2}) &= |\psi_1\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2}, \text{ and} & (441)\\ \mathrm{U}_f(|\psi_2\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2}) &= -|\psi_2\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2}. & (442) \end{aligned}$$

Hence,

$$\begin{aligned} \mathrm{G}\left[|\psi\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2}\right] &= (\mathrm{W} \otimes \mathrm{I})\mathrm{U}_f\left[(\alpha|\psi_1\rangle + \beta|\psi_2\rangle)|\psi\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2}\right] \\ &= (\mathrm{W} \otimes \mathrm{I})\left[(\alpha|\psi_1\rangle - \beta|\psi_2\rangle)|\psi\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2}\right] \\ &= (\alpha\mathrm{W}|\psi_1\rangle - \beta\mathrm{W}|\psi_2\rangle) \otimes (|0\rangle - |1\rangle)/\sqrt{2}. \end{aligned} \tag{443}$$

Note that the $(|0\rangle - |1\rangle)/\sqrt{2}$ always comes along for free in the bottom register and to simplify notation we will just drop it and write

$$\mathrm{G}|\psi\rangle = \alpha\mathrm{W}|\psi_1\rangle - \beta\mathrm{W}|\psi_2\rangle. \tag{444}$$

Recall that $\mathrm{W} = \frac{2}{N}\mathbf{1}\mathbf{1}^{\mathrm{T}} - \mathrm{I}$. Using $\mathbf{1}^{\mathrm{T}}|\psi_1\rangle = \sqrt{N-m}$ and $\mathbf{1}^{\mathrm{T}}|\psi_2\rangle = \sqrt{m}$ and (438), we have

$$\begin{aligned} \mathrm{W}|\psi_1\rangle &= \frac{2}{N}\mathbf{1}\mathbf{1}^{\mathrm{T}}|\psi_1\rangle - |\psi_1\rangle \\ &= \frac{2\sqrt{N-m}}{N} \cdot \mathbf{1} - |\psi_1\rangle \\ &= \frac{2\sqrt{N-m}}{N}\left(\sqrt{N-m} \cdot |\psi_1\rangle + \sqrt{m} \cdot |\psi_2\rangle\right) - |\psi_1\rangle \\ &= \left(1 - \frac{2m}{N}\right) \cdot |\psi_1\rangle + \frac{2\sqrt{m(N-m)}}{N} \cdot |\psi_2\rangle. \end{aligned} \tag{445}$$

Similarly,

$$\begin{aligned} \mathrm{W}|\psi_2\rangle &= \frac{2}{N}\mathbf{1}\mathbf{1}^{\mathrm{T}}|\psi_2\rangle - |\psi_2\rangle \\ &= \frac{2\sqrt{m}}{N} \cdot \mathbf{1} - |\psi_2\rangle \\ &= \frac{2\sqrt{m}}{N}\left(\sqrt{N-m} \cdot |\psi_1\rangle + \sqrt{m} \cdot |\psi_2\rangle\right) - |\psi_2\rangle \\ &= \frac{2\sqrt{m(N-m)}}{N} \cdot |\psi_1\rangle - \left(1 - \frac{2m}{N}\right) \cdot |\psi_2\rangle. \end{aligned} \tag{446}$$

Using (445) and (446) in (444) gives

$$\mathrm{G}|\psi\rangle = \left[\left(1-\frac{2m}{N}\right)\alpha - \frac{2\sqrt{m(N-m)}}{N}\beta\right]|\psi_1\rangle + \left[\frac{2\sqrt{m(N-m)}}{N}\alpha + \left(1-\frac{2m}{N}\right)\beta\right]|\psi_2\rangle. \quad (447)$$

Importantly, the Grover operator G takes a vector in this subspace to another vector in this subspace. The subspace spanned by $\{|\psi_1\rangle, |\psi_2\rangle\}$ is invariant under G. A vector in this subspace is represented by the two parameters α, β, that is by a vector $\left[\begin{smallmatrix}\alpha\\ \beta\end{smallmatrix}\right]$. The action of G on this vector is to produce another vector in this subspace, represented by $\left[\begin{smallmatrix}\alpha'\\ \beta'\end{smallmatrix}\right]$.

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} \quad \rightarrow \quad \begin{bmatrix}\alpha'\\ \beta'\end{bmatrix} = \begin{bmatrix} 1-\frac{2m}{N} & -\frac{2\sqrt{m(N-m)}}{N} \\ \frac{2\sqrt{m(N-m)}}{N} & 1-\frac{2m}{N}\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}. \quad (448)$$

That is, the Grover operator on this subspace can be represented by the matrix

$$\mathrm{G}_{\psi_1\psi_2} = \begin{bmatrix} 1-\frac{2m}{N} & -\frac{2\sqrt{m(N-m)}}{N} \\ \frac{2\sqrt{m(N-m)}}{N} & 1-\frac{2m}{N}\end{bmatrix}. \quad (449)$$

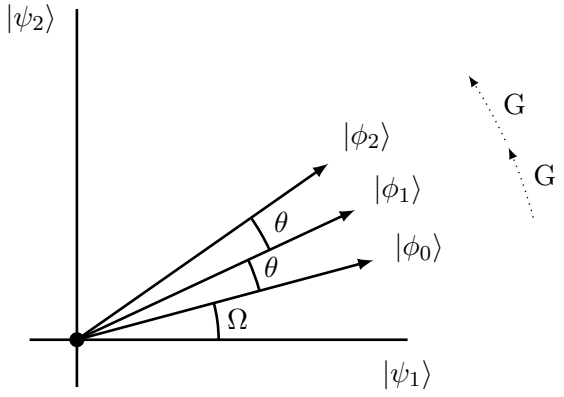
$\mathrm{G}_{\psi_1\psi_2}$ is a real orthogonal matrix, which you can see by verifying that

$$\left(1-\frac{2m}{N}\right)^2 + \left(\frac{2\sqrt{m(N-m)}}{N}\right)^2 = 1. \quad (450)$$

Hence, we can define an angle θ by $\cos\theta = 1-2m/N$ and $\sin\theta = 2\sqrt{m(N-m)}/N$. Then

$$\mathrm{G}_{\psi_1\psi_2} = \begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}. \quad (451)$$

That is, $\mathrm{G}_{\psi_1\psi_2}$ rotates by a positive angle θ. Recall, from (439), that the initial state has angle Ω, where $\sin\Omega = \sqrt{m/N}$. Here is an elegant geometric view of how G operates in this subspace,



(452)

The starting state is $|\phi_0\rangle$ with angle Ω. Each time G operates on the state, it adds θ to the angle. As you can see, this gradually aligns the state with $|\psi_2\rangle$. The angle after t Grover iterations is

$$t\theta + \Omega. \tag{453}$$

The projection of $|\phi_t\rangle = \mathrm{G}^t|\phi_0\rangle$ onto $|\psi_2\rangle$ is $\sin(t\theta + \Omega)$, and so the probability of success is

$$\sin^2(t\theta + \Omega), \tag{454}$$

where $\sin\Omega = \sqrt{m/N}$ and $\sin\theta = 2\sqrt{m(N-m)}/N$, matching the result from the eigenvalue analysis in (426). When the state $|\phi_t\rangle$ has angle $\pi/2$, it is perfectly aligned with $|\psi_2\rangle$. The only nonzero amplititudes are for pure states $|x\rangle$ with $f(x) = 1$ and measurement has success probability 1. To get near-perfect alignment, we set the number of Grover iterations to $t_* = \lfloor (\pi/2 - \Omega)/\theta \rfloor$, as in (429). When $m \ll N$, Ω is negligible compared to $\pi/2$ and $\theta \approx \sin\theta \approx 2\sqrt{m/N}$, which gives

$$t_* \approx \frac{\pi}{4}\sqrt{\frac{N}{m}}. \tag{455}$$

21.1 Eigenvalues of the Grover Operator: Phase Estimation

The Grover operator

$$\mathrm{G}_{\psi_1\psi_2} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{456}$$

is a rotation. The eigenvalues of any unitary operator have unit norm. Indeed, let $|v\rangle$ be an eigenvector with eigenvalue λ. The reader should justify steps (a), (b), (c) and (d) below,

$$1 \overset{(a)}{=} \langle v|\mathrm{G}^\dagger\mathrm{G}|v\rangle \overset{(b)}{=} (\mathrm{G}|v\rangle)^\dagger\mathrm{G}|v\rangle \overset{(c)}{=} \lambda^*\lambda\langle v|v\rangle \overset{(d)}{=} \|\lambda\|^2. \tag{457}$$

Hence, the eigenvalues are of the form $\lambda = e^{i\gamma}$, where the phase γ fully specifies the eigenvalue. The eigenvalues of $\mathrm{G}_{\psi_1\psi_2}$ are the solution to its characteristic equation

$$(\cos\theta - \lambda)^2 + \sin^2\theta = 0. \tag{458}$$

The solutions are

$$\lambda_\pm = \cos\theta \pm i\sin\theta = e^{\pm i\theta}. \tag{459}$$

The solutions to the eigenvalue equation

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = e^{\pm i\theta} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{460}$$

are $\beta = \mp i\alpha$. Hence, the (normalized) eigenvectors corresponding to eigenvalues $\lambda_\pm = e^{\pm i\theta}$ are

$$|v_\pm\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ \mp i \end{bmatrix}. \tag{461}$$

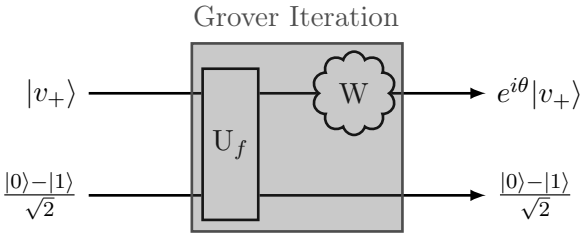
The important point is that the phase of the eigenvalues are $\pm\theta$. Specifically, $\mathrm{phase}(\lambda_+) = \theta$. Since $\cos\theta = 1 - 2m/N$, it follows that $m = N(1-\cos\theta)/2$, or

$$m = \frac{N}{2} \times \big[1 - \cos(\mathrm{phase}(\lambda_+))\big]. \tag{462}$$

If we can estimate the phase of the eigenvalue corresponding to the eigenvector $|v_+\rangle$, then we can use (462) to get an estimate of m. Estimating the phase of the eigenvalues of a unitary operator is a path to counting the 1's of a Boolean function.

Estimating the phase of the eigenvalues of a unitary operator is called *phase estimation*. One application is to quantum counting. This is important because it allows us to estimate the optimal number of Grover iterations in a quantum search. However, phase estimation has other important uses such as period finding with application to quantum factoring. Hence, phase estimation is an important tool in its own right. So, we now switch gears and discuss Quantum Phase Estimation.

There is one small detail which we need to restore. We have been analyzing the Grover operator in (433) by analyzing what it does on the top-n qubits $|\psi\rangle$. In terms of the full Grover operator on $n+1$ qubits, we have shown that



(463)

We leave it as an easy exercise for the reader to show that $|v_+\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2}$ is an eigenvector of the full Grover operator G with eigenvalue $e^{i\theta}$. That is,

$$\mathrm{G}\left(|v_+\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = e^{i\theta}\left(|v_+\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}\right). \tag{464}$$

So, if we can estimate the phase of the full Grover operator G, then we can estimate m.

Defining Phase Estimation Let us formally define the phase estimation problem as:

Given a unitary operator U as a black-box quantum circuit, and an eigenvector quantum state $\|v\rangle$, estimate the phase of the eigenvalue associated to the eigenvector quantum state $\|v\rangle$.

So, we know that

$$\mathrm{U}|v\rangle = e^{i\theta}|v\rangle, \tag{465}$$

and we are to estimate the phase θ. In the problem statement, we are given the eigenvector $|v\rangle$. For the Grover operator, if we knew $|v_+\rangle$, we would already know m and also solutions to $f(x) = 1$. So, we need to figure out how to get around this obstacle of preparing the state $|v\rangle$. As a first step, let us solve the basic phase estimation problem, where the eigenvector is given. To do phase estimation, we are going to need the Fourier Transform, so the next order of business is to develop the Quantum Fourier Transform.

22 Quantum Fourier Transform

One of humanity's great inventions is the Fourier transform with applicationis in physics, engineering, function analysis, to name a few. The discrete Fourier transform converts a set of complex numbers $a_0, a_1, \ldots, a_{N-1}$ to $f_0, f_1, \ldots, f_{N-1}$, where

$$f_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} a_j e^{(2\pi i)kj/N}. \tag{466}$$

Let us interpret this in context of a quantum state of n qubits. A quantum state has $N = 2^n$ amplitudes for each pure (classical) n-bit state. The Fourier transform gives transformed amplitudes, i.e., another quantum state. The starting quantum state $|\psi\rangle$ specified by the amplitudes $a_0, \ldots, a_{N-1}$,

$$|\psi\rangle = \sum_{j=0}^{N-1} a_j |j\rangle, \tag{467}$$

is transformed to a new quantum state

$$|\phi\rangle = \sum_{k=0}^{N-1} f_k |k\rangle. \tag{468}$$

In vector form,

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix} \rightarrow \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{bmatrix} \tag{469}$$

Note, the indices for the basis vectors are $0, 1, \ldots, N-1$. The transform in (466) is clearly linear in $\mathbf{a}$. Hence, it implements some linear operator that is represented by a matrix $\mathbf{F}$,

$$|\phi\rangle = \mathbf{F}|\psi\rangle. \tag{470}$$

Let us consider the action of $\mathbf{F}$ on a basis state $|j\rangle$. Using (466), $f_k = e^{(2\pi i)kj/N}/\sqrt{N}$ and

$$\mathbf{F}|j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{(2\pi i)kj/N} |k\rangle. \tag{471}$$

We can now directly read the entries in the matrix $\mathbf{F}$. Defining $\omega = e^{(2\pi i)/N}$

$$\mathbf{F}_{kj} = \frac{1}{\sqrt{N}} \omega^{kj}. \tag{472}$$

Let us consider a concrete case $n = 3, N = 8$. In this case, $\omega = e^{(2\pi i)/8}$. Using $\omega^8 = 1$, we get

$$\mathbf{F} = \frac{1}{\sqrt{8}} \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 \\ \omega^0 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega^1 & \omega^6 & \omega^3 \\ \omega^0 & \omega^6 & \omega^4 & \omega^2 & \omega^0 & \omega^6 & \omega^4 & \omega^2 \\ \omega^0 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{bmatrix} \tag{473}$$

We prove $\mathbf{F}$ is unitary. From (472), $\mathbf{F}_{\alpha k} = e^{(2\pi i)\alpha k/N}/\sqrt{N}$ and

$$(\mathbf{F}^\dagger)_{k\beta} = e^{-(2\pi i)\beta k/N}/\sqrt{N}. \tag{474}$$

Hence,

$$\begin{aligned}(\mathbf{F}\mathbf{F}^\dagger)_{\alpha\beta} &= \sum_{k=0}^{N-1} \mathbf{F}_{\alpha k}(\mathbf{F}^\dagger)_{k\beta} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} e^{(2\pi i)\alpha k/N - (2\pi i)\beta k/N} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} e^{(2\pi i)(\alpha-\beta)k/N}. \end{aligned} \tag{475}$$

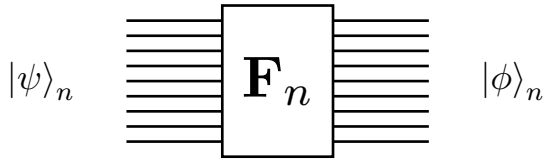
There are two cases to consider. If $\alpha = \beta$, then

$$(\mathbf{F}\mathbf{F}^\dagger)_{\alpha\beta} = \frac{1}{N}\sum_{k=0}^{N-1} 1 = 1. \tag{476}$$

If $\alpha \neq \beta$, let $z = e^{(2\pi i)(\alpha-\beta)/N} \neq 1$ because $\alpha, \beta \in [0, N-1]$. Then,

$$\begin{aligned}(\mathbf{F}\mathbf{F}^\dagger)_{\alpha\beta} &= \frac{1}{N}\sum_{k=0}^{N-1} z^k \\ &= \frac{1}{N}\cdot\frac{z^N - 1}{z-1} = 0 \end{aligned} \tag{477}$$

The last equality follows because $z^N = e^{(2\pi i)(\alpha-\beta)}$ and $(\alpha - \beta)$ is an integer. We have proved that $(\mathbf{F}\mathbf{F}^\dagger)_{\alpha\beta} = \delta_{\alpha\beta}$, that is $\mathbf{F}\mathbf{F}^\dagger = \mathrm{I}$, and $\mathbf{F}$ is unitary. This is big news. It means there is a quantum circuit that implements the discrete Fourier transform on an n-qubit quantum state. Denote this circuit by $\mathbf{F}_n$, which takes as input an n-qubit quantum state and produces an n-qubit output state whose amplitudes to be in each classical state have been transformed by a Fourier transform,



(478)

22.1 Classical DFT Algorithm

A naive computation of the Fourier transform in (466) must compute N Fourier coefficients

$$f_1, \ldots, f_{N-1}. \tag{479}$$

Each computation is a sum over N terms giving a total of $O(N^2)$ operations. One of the all time classic algorithms on any list of top-10 best algorithms is the Fast Fourier Transform (FFT) that

computes all Fourier coefficients $f_1, \ldots, f_{N-1}$ in $O(N \log N)$ operations. This is a huge speedup by a factor of almost $N = 2^n$ and is a game changer in digital signal processing.

To develop the algorithm for the FFT, we use a more convenient representation of the Fourier transform $\mathbf{F}$ defined by its action on the basis vectors in (471). The basis vector $|j\rangle$ corresponds to an n-bit state $|j_1 j_2 \cdots j_n\rangle$. The correspondence is the binary representation of the index j,

$$\begin{array}{rcl} |000\cdots00\rangle & \leftrightarrow & |0\rangle \\ |000\cdots01\rangle & \leftrightarrow & |1\rangle \\ |000\cdots10\rangle & \leftrightarrow & |2\rangle \\ |000\cdots11\rangle & \leftrightarrow & |3\rangle \\ & \vdots & \\ |111\cdots10\rangle & \leftrightarrow & |2^n-2\rangle \\ |111\cdots11\rangle & \leftrightarrow & |2^n-1\rangle. \end{array} \tag{480}$$

The classical state $|j_1 j_2 \ldots j_n\rangle \in \{0,1\}^n$ is the basis vector $|j\rangle \in \{0, 1, \ldots, 2^n - 1\}$, where

$$j = j_1 2^{n-1} + j_2 2^{n-2} + \cdots j_{n-1} 2^1 + j_n 2^0. \tag{481}$$

The sum in (471) is over all classical n-bit states $|k\rangle = |k_1 k_2 \cdots k_n\rangle$,

$$\begin{aligned} \mathbf{F}|j\rangle &= \frac{1}{\sqrt{N}} \sum_{k_1,k_2,\ldots,k_n} e^{(2\pi i) j (k_1 2^{n-1} + k_2 2^{n-2} + \cdots k_{n-1} 2^1 + k_n 2^0)/N} |k_1 k_2 \cdots k_n\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{k_1,k_2,\ldots,k_n} \exp\left[(2\pi i) j \left(\frac{k_1}{2} + \frac{k_2}{2^2} + \cdots \frac{k_{n-1}}{2^{n-1}} + \frac{k_n}{2^n}\right)\right] |k_1\rangle \otimes |k_2\rangle \otimes \cdots \otimes |k_n\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{k_1,k_2,\ldots,k_n} e^{(2\pi i) j k_1/2} |k_1\rangle \otimes e^{(2\pi i) j k_2/2^2} |k_2\rangle \otimes \cdots \otimes e^{(2\pi i) j k_n/2^2} |k_n\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{k_1} e^{(2\pi i) j k_1/2} |k_1\rangle \otimes \sum_{k_2} e^{(2\pi i) j k_2/2^2} |k_2\rangle \otimes \cdots \otimes \sum_{k_n} e^{(2\pi i) j k_n/2^2} |k_n\rangle. \end{aligned} \tag{482}$$

In the second step, we used $N = 2^n$. Each sum above has two terms,

$$\sum_{k_\ell=0}^{1} e^{(2\pi i) j k_\ell/2^\ell} |k_\ell\rangle = |0\rangle + e^{(2\pi i) j/2^\ell} |1\rangle. \tag{483}$$

Using the bit representation $j = j_1 2^{n-1} + \cdots + j_n 2^0$, for $\ell \in \{1, 2, \ldots, n\}$:

$$\begin{aligned} \exp\left((2\pi i)\frac{j}{2^\ell}\right) &= \exp\left((2\pi i)\frac{j_1 2^{n-1} + \cdots + j_n 2^0}{2^\ell}\right) \\ &= \exp\left((2\pi i)\left[j_1 2^{n-\ell-1} + \cdots + j_{n-\ell} 2^0 + \frac{j_{n+1-\ell}}{2^1} + \frac{j_{n+2-\ell}}{2^2} + \cdots + \frac{j_n}{2^\ell}\right]\right) \\ &= \exp\left((2\pi i)\left[\frac{j_{n+1-\ell}}{2^1} + \frac{j_{n+2-\ell}}{2^2} + \cdots + \frac{j_n}{2^\ell}\right]\right). \end{aligned} \tag{484}$$

The last step is because any integer multiple of $2\pi i$ in the exponent can be ignored. The sum, as a number, has a convenient binary expansion,

$$\frac{j_{n+1-\ell}}{2^1} + \frac{j_{n+2-\ell}}{2^2} + \cdots + \frac{j_n}{2^\ell} = \mathbf{0}.j_{(n+1-\ell)} j_{(n+2-\ell)} \cdots j_{(n-1)} j_n. \tag{485}$$

The binary expansion on the RHS is shorthand for the sum on the LHS. Importantly, we proved:

$$\mathbf{F}|j_1 j_2 \cdots j_n\rangle = \frac{|0\rangle + e^{(2\pi i)0.j_n}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{(2\pi i)0.j_{n-1}j_n}|1\rangle}{\sqrt{2}} \otimes \cdots \otimes \frac{|0\rangle + e^{(2\pi i)0.j_1 j_2 \cdots j_n}|1\rangle}{\sqrt{2}}. \quad (486)$$

This formula is key to the quantum circuit for $\mathbf{F}$. It also gives an $O(n2^n)$ classical DFT-algorithm. We can get another useful way to write the formula in (486). Since $e^{(2\pi i)k} = 1$ for any integer k, we get the following useful identity.

$$e^{(2\pi i)2^k \times 0.j_1 j_2 \cdots j_n} = e^{(2\pi i)0.j_{k+1} j_{k+2} \cdots j_n}. \quad (487)$$

Let $\varphi = 0.j_1 j_2 \cdots j_n \in [0, 1)$. Then, using (487) in (486) gives

$$\mathbf{F}|j_1 j_2 \cdots j_n\rangle = \frac{|0\rangle + e^{(2\pi i)\varphi \cdot 2^{n-1}}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{(2\pi i)\varphi \cdot 2^{n-2}}|1\rangle}{\sqrt{2}} \otimes \cdots \otimes \frac{|0\rangle + e^{(2\pi i)\varphi \cdot 2^0}|1\rangle}{\sqrt{2}}. \quad (488)$$

22.1.1 Fast Fourier Transform (FFT)

Given $a_0, a_1, \ldots, a_{N-1}$, we must compute all the Fourier coefficients $f_0, f_1, \ldots, f_{N-1}$. Equivalently, from the starting state $|\psi\rangle = \sum_{\boldsymbol{j} \in \{0,1\}^n} a_{\boldsymbol{j}} |\boldsymbol{j}\rangle$, we must compute $|\phi\rangle = \mathbf{F}|\psi\rangle$,

$$|\phi\rangle = \sum_{\boldsymbol{j} \in \{0,1\}^n} a_{\boldsymbol{j}} \mathbf{F}|\boldsymbol{j}\rangle. \quad (489)$$

In (486), consider the two cases $j_1 = 0$ and $j_1 = 1$:

$$\begin{aligned} \mathbf{F}|0 j_2 j_3 \cdots j_{n-1}\rangle &= \tfrac{|0\rangle + e^{(2\pi i)0.j_n}|1\rangle}{\sqrt{2}} \otimes \tfrac{|0\rangle + e^{(2\pi i)0.j_{n-1}j_n}|1\rangle}{\sqrt{2}} \otimes \cdots \otimes \tfrac{|0\rangle + e^{(2\pi i)0.0 j_2 \cdots j_n}|1\rangle}{\sqrt{2}} \\ &= \mathbf{F}|j_2 j_3 \cdots j_{n-1}\rangle \otimes \tfrac{|0\rangle + e^{(2\pi i)0.0 j_2 \cdots j_n}|1\rangle}{\sqrt{2}} \quad (490) \\ \mathbf{F}|1 j_2 j_3 \cdots j_{n-1}\rangle &= \tfrac{|0\rangle + e^{(2\pi i)0.j_n}|1\rangle}{\sqrt{2}} \otimes \tfrac{|0\rangle + e^{(2\pi i)0.j_{n-1}j_n}|1\rangle}{\sqrt{2}} \otimes \cdots \otimes \tfrac{|0\rangle - e^{(2\pi i)0.0 j_2 \cdots j_n}|1\rangle}{\sqrt{2}} \\ &= \mathbf{F}|j_2 j_3 \cdots j_{n-1}\rangle \otimes \tfrac{|0\rangle - e^{(2\pi i)0.0 j_2 \cdots j_n}|1\rangle}{\sqrt{2}}. \quad (491) \end{aligned}$$

For $j_1 = 1$, we used $e^{(2\pi i)0.1 j_2 \cdots j_n} = -e^{(2\pi i)0.0 j_2 \cdots j_n}$. The sum in (489) breaks into two terms,

$$|\phi\rangle = \sum_{j_2, \ldots, j_n} a_{0 j_2 \cdots j_n} \mathbf{F}|0 j_2 \cdots j_n \boldsymbol{j}\rangle + \sum_{j_2, \ldots, j_n} a_{1 j_2 \cdots j_n} \mathbf{F}|1 j_2 \cdots j_n \boldsymbol{j}\rangle. \quad (492)$$

Using (490) and (491) and collecting terms, we get

$$|\phi\rangle = \underbrace{\left(\sum_{j_2, \ldots, j_n} x_{j_2 \cdots j_n} \mathbf{F}|j_2 j_3 \cdots j_{n-1}\rangle \right)}_{|\phi_x\rangle} \otimes |0\rangle + \underbrace{\left(\sum_{j_2, \ldots, j_n} y_{j_2 \cdots j_n} \mathbf{F}|j_2 j_3 \cdots j_{n-1}\rangle \right)}_{|\phi_y\rangle} \otimes |1\rangle \quad (493)$$

where

$$x_{j_2 \cdots j_n} = \frac{a_{0 j_2 \cdots j_n} + a_{1 j_2 \cdots j_n}}{\sqrt{2}}; \quad (494)$$

$$y_{j_2 \cdots j_n} = \frac{e^{(2\pi i)0.0 j_2 \cdots j_n}(a_{0 j_2 \cdots j_n} - a_{1 j_2 \cdots j_n})}{\sqrt{2}}. \quad (495)$$

In (493) we have decomposed $|\phi\rangle$ into two DFT's of size $N/2$ that compute $|\phi_x\rangle$ and $|\phi_y\rangle$. The two $N/2$ sized DFT's are for the amplitudes $\mathbf{x}$ and $\mathbf{y}$ that are computed from the original amplitudes $\mathbf{a}$ as given in (494) and (495). All the amplitudes in $\mathbf{x}, \mathbf{y}$ can be computed in $O(N)$ operations. A little care is needed to compute all the $e^{(2\pi i)0.0j_2\cdots j_n}$ in $O(N)$ time, assuming complex exponentiation can be done in $O(1)$. Lastly, we compute $|\phi_x\rangle \otimes |0\rangle$, which populates the even entries $f_0, f_2, \ldots, f_{N-2}$, and $|\phi_y\rangle \otimes |1\rangle$, which populates the odd entries $f_1, f_3, \ldots, f_{N-1}$. This takes $O(N)$ operations. Therefore, we have a running time $t(N)$ which satisfise the recursion

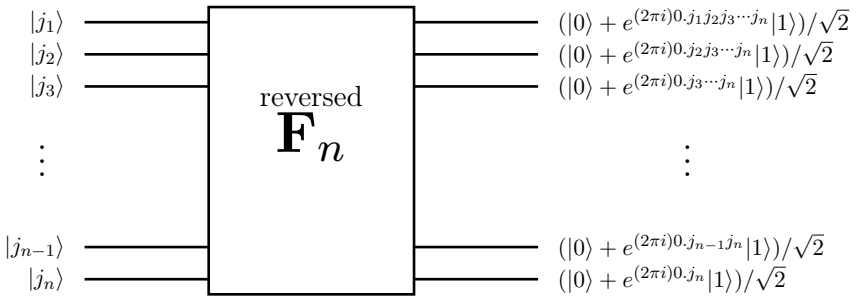
$$t(N) = 2t(N/2) + O(N). \tag{496}$$

Via the master theorem, $t(N)$ is in $O(N \log N)$. A good exercise is to exactly analyze the runtime.

22.2 Quantum Circuit for DFT

The formula in (486) has a very elegant interpretation from the perspective of a quantum circuit for DFT. If the input to the circuits is the classical state $|j_1\rangle \otimes |j_2\rangle \otimes \cdots \otimes |j_n\rangle$, then the output is also a tensor product. The top output-qubit state is $(|0\rangle + e^{(2\pi i)0.j_n}|1\rangle)/\sqrt{2}$, the next is $(|0\rangle + e^{(2\pi i)0.j_{n-1}j_n}|1\rangle)/\sqrt{2}$, etc. We give a quantum circuit to produce the outputs in reverse order, namely

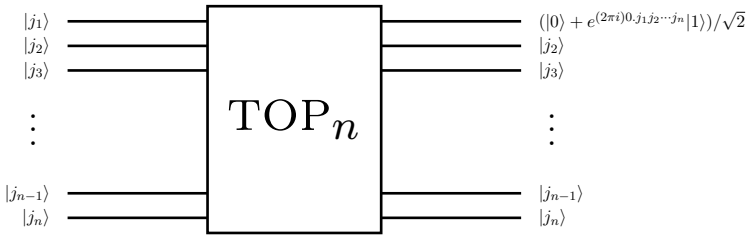
$$\frac{|0\rangle + e^{(2\pi i)0.j_1 j_2\cdots j_n}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{(2\pi i)0.j_2\cdots j_n}|1\rangle}{\sqrt{2}} \otimes \cdots \otimes \frac{|0\rangle + e^{(2\pi i)0.j_n}|1\rangle}{\sqrt{2}} \quad . \tag{497}$$



This is not a problem, because if we operate on the output with

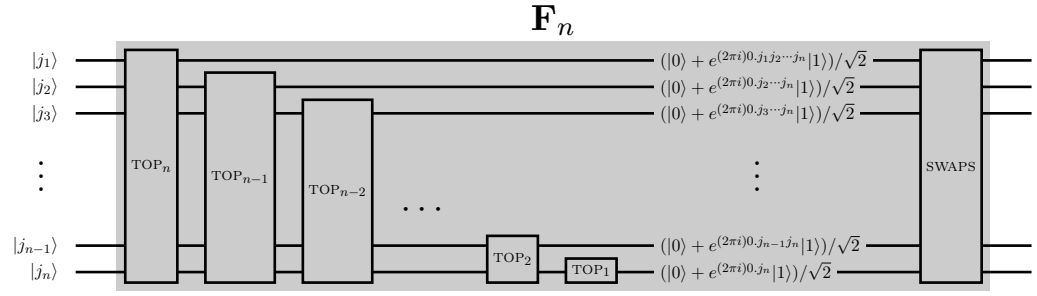
$$\text{swap}(1, n) \cdot \text{swap}(2, n-1) \cdot \text{swap}(3, n-3) \cdots , \tag{498}$$

the final circuit will have the desired output on each wire. This reversal of the outpus uses $\lfloor n/2 \rfloor$ swap gates, which is $3\lfloor n/2 \rfloor$ c-NOTs. Consider the top output-qubit in (497). It is a superposition of $|0\rangle$ with a phase times $|1\rangle$. The phase depends on all the input classical bits $j_1, \ldots, j_n$. We will construct a circuit that produces the top output-qubit. Let's call this circuit TOP_n.



(499)

From (497), the phase for output from qubit $|j_2\rangle$ is similar to the phase for the top qubit, except that it only uses bits $j_2, \ldots, j_n$. Indeed TOP_{n-1} applied to bits $j_2, \ldots, j_n$ produces exactly the desired output for qubit $|j_2\rangle$. Applying this logic recursively, we obtain the circuit for $\mathbf{F}_n$,



(500)

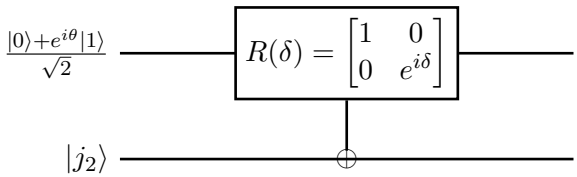
The circuit in (500) produces the correct Fourier transformed quantum state for all input pure states $|j_1 \cdots j_n\rangle$. The linear DFT-operator is fully specified by its action on the computational basis, so the circuit in (500) performs the Fourier transform correctly for any input quantum state $|\psi\rangle$ with amplitudes $a_0, \ldots, a_{N-1}$, to produce the output state $|\phi\rangle$ with amplitudes $f_0, \ldots, f_k$. To complete our construction, we need a circuit for TOP_n. To do so, we analyze two basic quantum gates, the Hadamard and the controlled phase gates. Specifically, for an input pure single qubit state $|j_1\rangle$,

$$\text{H}|j_1\rangle = \begin{cases} (|0\rangle + |1\rangle)/\sqrt{2} & j_1 = 0; \\ (|0\rangle - |1\rangle)/\sqrt{2} & j_1 = 0. \end{cases} \tag{501}$$

Using $e^{\pi i} = -1$, these two cases can be neatly summarized into a single equation

$$\text{H}|j_1\rangle = (|0\rangle + e^{(2\pi i)0.j_1}|1\rangle)/\sqrt{2}. \tag{502}$$

Thus, the Hadamard produces the $0.j_1$ part of the phase for the top output-qubit in (499). To get from here to $0.j_1 j_2$, we need to multiply the phase $e^{(2\pi i)0.j_1}$ by $e^{(2\pi i)0.0 j_2}$. That is, we need to multiply by a phase that is conditional on the classical bit j_2. The controlled phase gate does just this. Consider the circuit that uses the controlled phase gate $R(\delta)$ with inputs as shown below,



(503)

$|j_2\rangle$ is a pure single qubit state. We analyze this operator ${}^cR(\delta)$. By linearity,

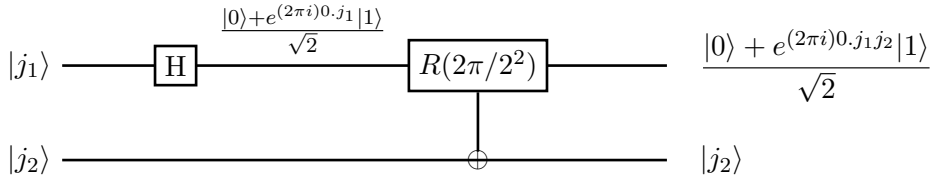
$${}^cR(\delta)\left[\frac{|0\rangle + e^{i\theta}|1\rangle}{\sqrt{2}} \otimes |j_2\rangle\right] = \frac{1}{\sqrt{2}}{}^cR(\delta)[|0\rangle \otimes |j_2\rangle] + \frac{e^{i\theta}}{\sqrt{2}}{}^cR(\delta)[|1\rangle \otimes |j_2\rangle]. \tag{504}$$

The phase operator does not affect $|0\rangle$ so the first term is $(|0\rangle \otimes |j_2\rangle)/\sqrt{2}$. In the second term, the phase gate operates on the $|1\rangle$, multiplying it by the phase $e^{i\omega}$, if and only if $j_2 = 1$. This

conditional multiplication by a phase can be neatly captured by always multiplying by the phase $e^{i\omega j_2}$. Collecting all this together and factoring out the $|j_2\rangle$ gives

$$ {}^{c}R(\delta)\left[\frac{|0\rangle + e^{i\theta}|1\rangle}{\sqrt{2}} \otimes |j_2\rangle\right] = \frac{|0\rangle + e^{i(\theta+\omega j_2)}|1\rangle}{\sqrt{2}} \otimes |j_2\rangle. \tag{505} $$

The controlled phase gate multiplies the amplitude for $|1\rangle$ by the appropriate phase and leaves the amplitude for $|0\rangle$ unaltered. Since we need to multiply by the phase $e^{(2\pi i)0.0j_2} = e^{(2\pi i)j_2/2^2}$, we set $\delta = 2\pi/2^2$. Thus, a circuit that multiplies the $|1\rangle$ in the superposition by $e^{(2\pi i)0.j_1j_2}$ is

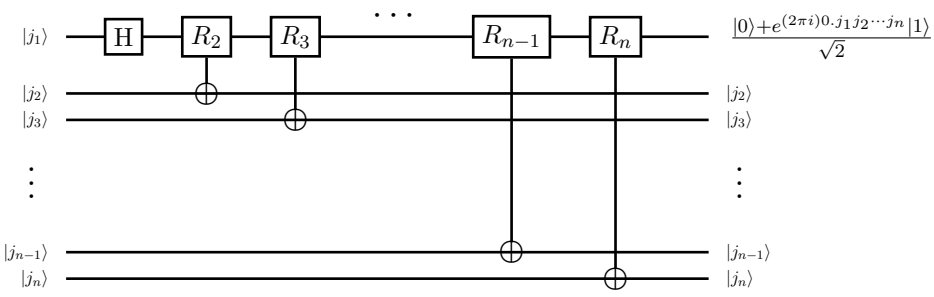


(506)

By (485), the phase $e^{(2\pi i)0.j_1\cdots j_n}$ is a product of phases,

$$ e^{(2\pi i)0.j_1j_2j_3\cdots j_n} = e^{(2\pi i)j_1/2}e^{(2\pi i)j_2/2^2}e^{(2\pi i)j_3/2^3}\cdots e^{(2\pi i)j_n/2^n}. \tag{507} $$

The phases in this product that depend on $j_2, \ldots, j_n$ are obtained by a controlled phase gate as in (506). To multiply by the phase $e^{(2\pi i)j_k/2^k}$ for $k > 1$, use the classical bit $|j_k\rangle$ to control the phase gate $R(2\pi/2^k)$ applied on the top qubit. Let $R_k = R(2\pi/2^k)$. A circuit for TOP_n is



(508)

We leave it to the reader to use the circuit for TOP_n in (508) to construct the full circuit for $\mathbf{F}_n$ using (500). The circuit for TOP_n uses one Hadamard gate and $n-1$ phase gates. $\mathbf{F}_n$ uses $\text{TOP}_n, \text{TOP}_{n-1}, \ldots, \text{TOP}_1$, which is a total of n Hadamards and $1 + 2 + \cdots + (n-1)$ phase gates. Adding in the $\lfloor n/2 \rfloor$ swaps, the gate complexity of the quantum Fourier transform is

$$ n \text{ Hadmards, } n(n-1)/2 \text{ phase gates, and } 3\lfloor n/2 \rfloor \text{ c-NOTs.} $$

The quantum Fourier transform uses $O(n^2)$ gates, which is an exponential speedup when compared to the runtime complexity of the classical FFT. There is a catch, however. The quantum Fourier transform stores $f_0, \ldots, f_{N-1}$ in the amplitudes of the output quantum state $|\phi\rangle$. There is no way to access these amplitudes individually. It is also not obvious how to prepare the initial state $|\psi\rangle$. Hence this quantum circuit, as far as we know, cannot be used to implement the classical DFT more efficiently than the FFT. We are, however, free to use this operator in other algorithms, if some final measurement can give us an answer to some other problem. One such application is to phase estimation, which we discuss next.

Example 22.1. It is an imperative exercise to construct the inverse of the quantum Fourier transform. One approach is to recall that the inverse is $\mathbf{F}^\dagger$, so construct a circuit implementing $\mathbf{F}^\dagger$. Another approach is to apply (in reverse order) the inverses of each gate in our quantum circuit for $\mathbf{F}$. The inverse of a swap is the same swap. What is the inverse of the controlled phase gate? What is the inverse of the Hadamard?

Also, for practice, it is instructive to construct the full quantum circuit for Fourier transform of 8 amplitudes, that is, the circuit for three input qubits, and also the circuit for the inverse Fourier transform on three qubits.

23 Quantum Phase Estimation

The Fourier transform is critical to the general task of phase estimation. Phase estimation in turn is a power-tool for many quantum algorithms. Our specific use case is to estimate the phase of the Grover operator. If we know this phase, then we can estimate the optimal number of Grover iterations and solve a search problem.

> Given a unitary operator U as a black-box quantum circuit, and an eigenvector quantum state $|v\rangle$, estimate the phase of the eigenvalue associated to the eigenvector quantum state $|v\rangle$.

Since U is unitary, for $\varphi \in [0,1)$ we have that

$$\mathrm{U}|v\rangle = e^{(2\pi i)\varphi}|v\rangle. \tag{509}$$

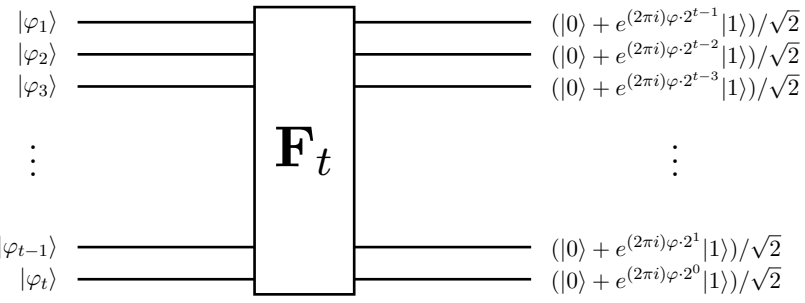
To simplify matters, let us suppose that φ has a finite binary expansion,

$$\varphi = 0.\varphi_1\varphi_2\cdots\varphi_t = \frac{\varphi_t}{2^t} + \cdots + \frac{\varphi_2}{2^2} + \frac{\varphi_1}{2^1}, \tag{510}$$

where $\varphi_k \in \{0,1\}$. Using $e^{(2\pi i)k} = 1$ for any integer k, we get the following useful identity.

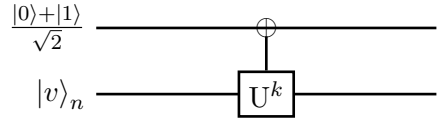
$$e^{(2\pi i)\varphi\cdot 2^k} = e^{(2\pi i)0.\varphi_{k+1}\varphi_{k+2}\cdots\varphi_t}. \tag{511}$$

Consider the pure state $|\varphi_1\varphi_2\cdots\varphi_t\rangle$ as input to the Fourier transform. Using (511) and (497) (after reversing the output-qubits), the Fourier transform produces



(512)

We show how to get the same output state produced by $\mathbf{F}_t|\varphi_1\varphi_2\cdots\varphi_t\rangle$ using U and $|v\rangle$. Consider the following circuit which uses the controlled U^k operator, ${}^c\mathrm{U}^k$,

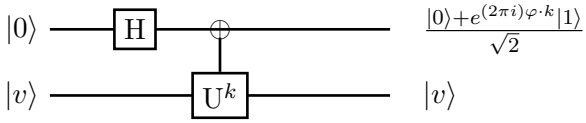


(513)

Using linearity and $\mathrm{U}^k|v\rangle = e^{(2\pi i)\varphi\cdot k}|v\rangle$, we have

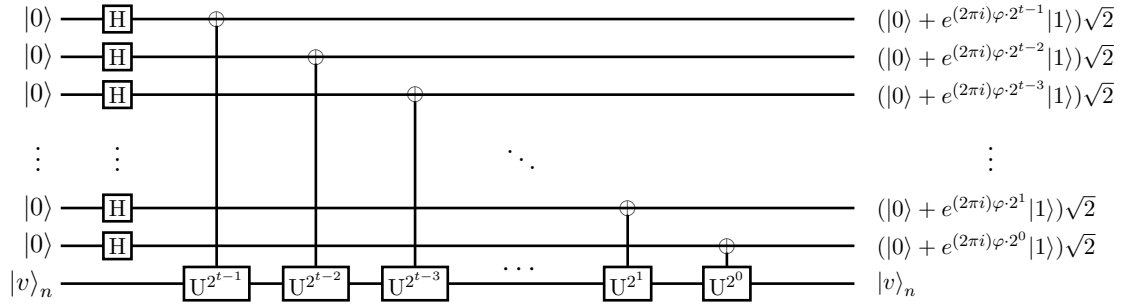
$$\begin{aligned} {}^c\mathrm{U}^k\left[\frac{|0\rangle+|1\rangle}{\sqrt{2}}\otimes|v\rangle\right] &= \frac{1}{\sqrt{2}}{}^c\mathrm{U}^k[|0\rangle\otimes|v\rangle] + \frac{1}{\sqrt{2}}{}^c\mathrm{U}^k[|1\rangle\otimes|v\rangle] \\ &= \frac{1}{\sqrt{2}}|0\rangle\otimes|v\rangle + \frac{1}{\sqrt{2}}|1\rangle\otimes e^{(2\pi i)\varphi\cdot k}|v\rangle \\ &= \frac{|0\rangle + e^{(2\pi i)\varphi\cdot k}|1\rangle}{\sqrt{2}}\otimes|v\rangle. \end{aligned} \tag{514}$$

The result in (514) is summarized below,



(515)

Setting $k = 2^{t-1}, 2^{t-2}, 2^{t-3}, \ldots, 2^1, 2^0$, we recover the outputs in (512) using the following circuit,

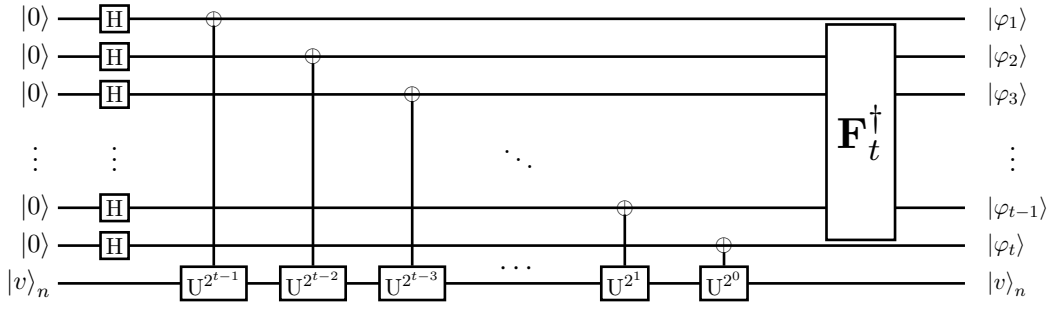


(516)

Note that this circuit operates on $t + n$ qubits. At the input, the top t qubits are set to $|0\rangle$ and the next n qubits are in the eigen quantum state of U. At the output, the top t qubits are in the quantum state $\mathbf{F}_t|\varphi_1\varphi_2\cdots\varphi_t\rangle$. If we now apply the inverse Fourier transform to the top t qubits, we recover $|\varphi_1\varphi_2\cdots\varphi_t\rangle$,

$$\mathbf{F}_t^\dagger \mathbf{F}_t|\varphi_1\varphi_2\cdots\varphi_t\rangle = |\varphi_1\varphi_2\cdots\varphi_t\rangle. \tag{517}$$

Here is the full quantum circuit for phase estimation,



(518)

Measuring the top t qubits recovers φ. There are two loose ends. How do we know t and how do we get the eigen quantum state $|v\rangle$?

To run the phase estimation circuit, one has to pick a value for t, without knowing φ. What happens if the binary expansion for φ has more than t bits. So,

$$\varphi = \frac{\varphi_1}{2^1} + \frac{\varphi_2}{2^2} + \cdots + \frac{\varphi_t}{2^t} + \frac{\varphi_{t+1}}{2^{t+1}} + \cdots \tag{519}$$

The circuit above only measures t bits. We may ask how likely is it that the circuit in (518) correctly measures the first k bits $\varphi_1, \varphi_2, \ldots \varphi_\kappa$. That is, what is the probability that the circuit in (518) produces an approximation that is within $2^{-\kappa}$ of φ? It is an interesting exercise to show that if

$$t \geq \kappa - 1 + \log_2\left(1 + \frac{4}{\pi^2\epsilon}\right). \tag{520}$$

then the circuit in (518) gives a κ-bit approximation to φ with probability at least $1-\epsilon$. For example, if $t \geq \kappa + 8$, we get a κ-bit approximation to φ with probability at least 0.999.

To run the circuit in (518), we need to prepare the eigen quantum state $|v\rangle$. What happens if the input state is not an eigen quantum state $|v\rangle$? In general the input state will be some superposition of the eigen vectors of U. In this case, what happens. We will see an example when we tackle the application to estimating the phase for the Grover operator, next.

23.1 Phase Estimation for the Grover Operator

The Grover operator G takes an $(n+1)$-qubit input for a function f defined on n bits. Recall that the $(|\psi_1\rangle, |\psi_2\rangle)$-subspace is invariant under the Grover operator,

$$\mathrm{G}\left[(\alpha|\psi_1\rangle + \beta|\psi_2\rangle) \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] = (\alpha'|\psi_1\rangle + \beta'|\psi_2\rangle) \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \tag{521}$$

Ignoring the bottom qubit input to G which stays invariant, the initial input to G in the Grover iteration is $|\phi_0\rangle = \mathbf{1}/\sqrt{N}$, which in the $(|\psi_1\rangle, |\psi_2\rangle)$-subspace is the vector

$$|\phi_0\rangle = \begin{bmatrix} \sqrt{(N-m)/N} \\ \sqrt{m/N} \end{bmatrix}. \tag{522}$$

In this subspace, the eigenvectors and eigenvalues of G are

$$|v_\pm\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ \mp i \end{bmatrix}; \qquad \lambda_\pm = e^{(2\pi i)\varphi_\pm}, \tag{523}$$

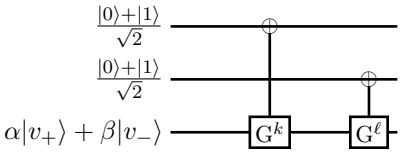
where $\cos(2\pi\varphi_+) = 1 - 2m/N$ and $\varphi_- = 1 - \varphi_+$. Hence, $\varphi_+ \in [0, 1/4)$ and $\varphi_- \in (3/4, 1]$ (see (461)). We can write the input $|\phi_0\rangle$ as a linear combination of $|v_\pm\rangle$,

$$|\phi_0\rangle = \alpha|v_+\rangle + \beta|v_-\rangle, \tag{524}$$

where

$$\alpha = \left(\sqrt{\frac{N-m}{2N}} + i\sqrt{\frac{m}{2N}}\right); \qquad \beta = \left(\sqrt{\frac{N-m}{2N}} - i\sqrt{\frac{m}{2N}}\right). \tag{525}$$

Let us see what happens if we feed in $|\phi_0\rangle$ into the phase estimation circuit, instead of $|v_+\rangle$. Let us start small, with $t = 2$, and analyze this controlled-G circuit (we suppressed the $(|0\rangle - |1\rangle)/\sqrt{2}$ bottom input which comes along for the ride).



(526)

First, the top qubit controls G^k on the bottom $n+1$ qubits. Using linearity of tensor product and ${}^c\mathrm{G}^k$, the result on the top qubit and the bottom register of $n+1$ qubits is

$${}^c\mathrm{G}^k\left[\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes (\alpha|v_+\rangle + \beta|v_-\rangle)\right] = \alpha \cdot {}^c\mathrm{G}^k\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |v_+\rangle + \beta \cdot {}^c\mathrm{G}^k\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |v_-\rangle. \tag{527}$$

Now, using the analysis in (514), evaluate each term separately to get

$$\alpha \cdot \frac{|0\rangle + e^{(2\pi i)\varphi_+ \cdot k}|1\rangle}{\sqrt{2}} \otimes |v_+\rangle + \beta \cdot \frac{|0\rangle + e^{(2\pi i)\varphi_- \cdot k}|1\rangle}{\sqrt{2}} \otimes |v_-\rangle. \tag{528}$$

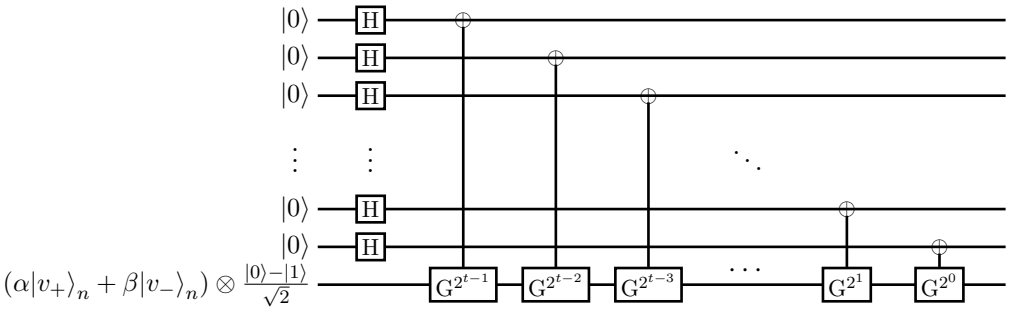
To get the full state after the first controlled-G^k we need to tensor the above state with the state of the middle qubit. By linearity of tensor product, this full state is

$$\alpha \cdot \frac{|0\rangle + e^{(2\pi i)\varphi_+ \cdot k}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |v_+\rangle + \beta \cdot \frac{|0\rangle + e^{(2\pi i)\varphi_- \cdot k}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |v_-\rangle. \tag{529}$$

We can now apply the second controlled-G^ℓ. The middle qubit is controlling. By linearity, we can apply this controlled -G^ℓ to each term and add. The result is

$$\alpha \cdot \frac{|0\rangle + e^{(2\pi i)\varphi_+ \cdot k}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{(2\pi i)\varphi_+ \cdot \ell}|1\rangle}{\sqrt{2}} \otimes |v_+\rangle + \beta \cdot \frac{|0\rangle + e^{(2\pi i)\varphi_- \cdot k}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{(2\pi i)\varphi_- \cdot \ell}|1\rangle}{\sqrt{2}} \otimes |v_-\rangle. \tag{530}$$

The pattern generalizes and we can directly write down the output for the following circuit,



(531)

Comparing to (530), the output is

$$\alpha|x\rangle \otimes |v_+\rangle_n \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} + \beta|y\rangle \otimes |v_-\rangle_n \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}, \tag{532}$$

where

$$\begin{aligned} |x\rangle &= \frac{|0\rangle + e^{(2\pi i)\varphi_+ 2^{t-1}}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{(2\pi i)\varphi_+ 2^{t-2}}|1\rangle}{\sqrt{2}} \otimes \cdots \otimes \frac{|0\rangle + e^{(2\pi i)\varphi_+ 2^{1}}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{(2\pi i)\varphi_+ 2^{0}}|1\rangle}{\sqrt{2}}; \\ |y\rangle &= \frac{|0\rangle + e^{(2\pi i)\varphi_- 2^{t-1}}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{(2\pi i)\varphi_- 2^{t-2}}|1\rangle}{\sqrt{2}} \otimes \cdots \otimes \frac{|0\rangle + e^{(2\pi i)\varphi_- 2^{1}}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{(2\pi i)\varphi_- 2^{0}}|1\rangle}{\sqrt{2}}. \end{aligned} \tag{533}$$

The next step in phase estimation applies $\mathbf{F}^\dagger$ to the top t qubits. Let $\varphi_\pm$ have binary expansions

$$\varphi_+ = 0.a_1 \cdots a_t; \qquad \varphi_- = 0.b_1 \cdots b_t. \tag{534}$$

For the Grover operator, it turns out that $b_k = 1 - a_k$. Also, $|x\rangle = \mathbf{F}|a_1 \cdots a_t\rangle$ and $|y\rangle = \mathbf{F}|b_1 \cdots b_t\rangle$. Therefore $\mathbf{F}^\dagger|x\rangle = |a_1 \cdots a_t\rangle$ and $\mathbf{F}^\dagger|y\rangle = |b_1 \cdots b_t\rangle$. Applying $\mathbf{F}^\dagger$ to (532) and using linearity, the final output of phase estimation is

$$\alpha|a_1 \cdots a_t\rangle \otimes |v_+\rangle_n \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} + \beta|b_1 \cdots b_t\rangle \otimes |v_-\rangle_n \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \tag{535}$$

We see that the final output is in a superposition of two quantum states. The top-t qubits in each of the states in the superposition are in pure classical state. So, measuring the top t qubits produces $|a_1 \cdots a_t\rangle$ with probability $\|\alpha\|^2 = 1/2$, or $|b_1 \cdots b_t\rangle$ with probability $\|\beta\|^2 = 1/2$. By looking at the top bit, we know if the measurement yielded the bits in φ_+ or the bits in $1 - \varphi_+$. In either case we have φ_+ and can compute the optimal number of Grover iterations.

Example 23.1. Here we know that the first bit of φ_+ is 0 and we know that $\varphi_- = 1 - \varphi_+$. This allows us to disambiguate the measurement and get φ_+. What if we don't these details about $\varphi_\pm$? In that case, we measure and use that measurement to compute the optimal m. If it's the right m, we succeed. If not, we may fail. In which case, we repeat the whole process. The probability of getting the right measurement is $\|\alpha\|^2 = 1/2$. We can boost this to as high a probability as needed with a small (logarithmic) number of repetitions.

23.2 Binary Expansion of φ has more than t Bits.

We now consider the case that the phase φ has more than t bits in its binary expansion,

$$\varphi = \underbrace{\frac{\varphi_1}{2^1} + \frac{\varphi_2}{2^2} + \cdots + \frac{\varphi_t}{2^t}}_{\varphi_t} + \underbrace{\frac{\varphi_{t+1}}{2^{t+1}} + \frac{\varphi_{t+2}}{2^{t+2}} + \cdots}_{\rho} \tag{536}$$

So, $\varphi = \varphi_t + \rho$ where $0 < \rho < 1/N$. Let us define $k_\varphi = 2^t \varphi_t = \left\lfloor 2^t \varphi \right\rfloor$, $0 \leq k_\varphi \leq N - 1$ and $N = 2^t$. The top t qubits from the first step in phase estimation in (516) is

$$\frac{|0\rangle + e^{(2\pi i)\varphi \cdot 2^{t-1}}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{(2\pi i)\varphi \cdot 2^{t-2}}|1\rangle}{\sqrt{2}} \otimes \cdots \otimes \frac{|0\rangle + e^{(2\pi i)\varphi \cdot 2^{1}}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{(2\pi i)\varphi \cdot 2^{0}}|1\rangle}{\sqrt{2}}. \tag{537}$$

Let $|j\rangle = |j_1 j_2 \cdots j_t\rangle$ be a pure state with t bits. Let us compute its amplitude in the superposition above. If $j_\ell = 0$ we use the $|0\rangle/\sqrt{2}$ in the j_lth term in the tensor product above. If $j_\ell = 1$ we use the $e^{(2\pi i)\varphi \cdot 2^{t-\ell}}|1\rangle/\sqrt{2}$ in the j_lth term in the tensor product above. So the contribution of j_ℓ to the amplitude of $|j_1 j_2 \cdots j_t\rangle$ is $e^{(2\pi i)\varphi j_\ell 2^{t-\ell}}$. Taking the product of these amplitudes, we get the amplitude a_j for $|j_1 j_2 \cdots j_t\rangle$,

$$a_j = \prod_{\ell=1}^{t} \frac{1}{\sqrt{2}} e^{(2\pi i)\varphi j_\ell 2^{t-\ell}} = \frac{1}{\sqrt{N}} e^{(2\pi i)\varphi \sum_{\ell=1}^{t} j_\ell 2^{t-\ell}} = \frac{1}{\sqrt{N}} e^{(2\pi i)\varphi j}. \tag{538}$$

The state in (537) can be written,

$$\sum_{j=0}^{N-1} a_j |j\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{(2\pi i)\varphi j} |j\rangle. \tag{539}$$

where $j = 0, \ldots, N-1$. The second step in phase estimation applies $\mathbf{F}^\dagger$ to this state. Using (474),

$$\mathbf{F}^\dagger |j\rangle = \sum_{k=0}^{N-1} \mathbf{F}^\dagger_{kj} |k\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{-(2\pi i)kj/N} |k\rangle. \tag{540}$$

By linearity, the output state for the top t bits after applying the phase estimation circuit is

$$\frac{1}{N}\sum_{k=0}^{N-1}\sum_{j=0}^{N-1} e^{(2\pi i)(\varphi-k/N)j}|k\rangle. \tag{541}$$

The geometric sum over j can be done in closed form to give the output state

$$\frac{1}{N}\sum_{k=0}^{N-1} b_k|k\rangle, \qquad \text{where} \qquad b_k = \frac{e^{(2\pi i)(N\varphi-k)}-1}{e^{(2\pi i)(\varphi-k/N)}-1}. \tag{542}$$

Upon measuring the top t qubits, The probability to observe $|k\rangle$ is $\|b_k\|^2$. Performing the requisite algebra, one finds

$$\mathbb{P}[|k\rangle] = \frac{1}{N^2}\times\frac{1-\cos(2\pi(N\varphi-k))}{1-\cos(2\pi(\varphi-k/N))}. \tag{543}$$

The phase corresponding to the measurement $|k\rangle$ is $\hat{\varphi} = 0.k_1\cdots k_t = k/N$. We now analyze these probabilities to determine how likely it is to measure a phase $\hat{\varphi}$ that is close to the true phase φ. Recall that $N\varphi = N(\varphi_t+\rho) = k_\varphi + N\rho$. Hence,

$$\cos(2\pi(N\varphi-k)) = \cos(2\pi(k_\varphi-k)+2\pi N\rho) = \cos(2\pi N\rho). \tag{544}$$

Let us now consider the possibilities $k = k_\varphi$ or $k = k_\varphi+1 \pmod N$ (the angle only matters modulo 2π, hence k only matters modulo N). In these cases,

$$\begin{array}{rcll} k = k_\varphi & \to & \hat{\varphi} = k_\varphi/N & \text{and} \quad \varphi-\hat{\varphi} = \rho; \\ k = k_\varphi+1 & \to & \hat{\varphi} = (k_\varphi+1)/N & \text{and} \quad \varphi-\hat{\varphi} = \rho-1/N. \end{array} \tag{545}$$

Since $1/N = 2^{-t}$, using $\cos(x) = \cos(-x)$ we have $\mathbb{P}[k = k_\varphi \text{ or } k_\varphi+1] = \mathbb{P}[|\varphi-\hat{\varphi}|\le 2^{-t}]$. Hence,

$$\mathbb{P}[|\varphi-\hat{\varphi}|\le 2^{-t}] = \frac{1-\cos(2\pi N\rho)}{N^2}\left[\frac{1}{1-\cos(2\pi\rho)}+\frac{1}{1-\cos(2\pi(\frac{1}{N}-\rho))}\right]. \tag{546}$$

This expression is minimized at $\rho = 1/2N$, so

$$\mathbb{P}[|\varphi-\hat{\varphi}|\le 2^{-t}] \ge \frac{4}{N^2(1-\cos(\pi/N))}. \tag{547}$$

Since $1/N^2(1-\cos(\pi/N))$ is decreasing in N, we can take the limit as $N\to\infty$ to get a lower bound. Since $N^2(1-\cos(\pi/N))\to\pi^2/2$, we have that

$$\mathbb{P}[|\varphi-\hat{\varphi}|\le 2^{-t}] \ge \frac{8}{\pi^2} \approx 0.81. \tag{548}$$

There is an 81% chance that phase estimation produces $\hat{\varphi}$ that is within 2^{-t} of the true phase φ, not bad. Let us generalize this discussion. Let $k = k_t-\ell$ or $k = k_t+1+\ell$. In these cases,

$$\begin{array}{rcll} k = k_\varphi-\ell & \to & \hat{\varphi} = (k_\varphi-\ell)/N & \text{and} \quad \varphi-\hat{\varphi} = \rho+\ell/N; \\ k = k_\varphi+1+\ell & \to & \hat{\varphi} = (k_\varphi+1+\ell)/N & \text{and} \quad \varphi-\hat{\varphi} = \rho-1/N-\ell/N. \end{array} \tag{549}$$

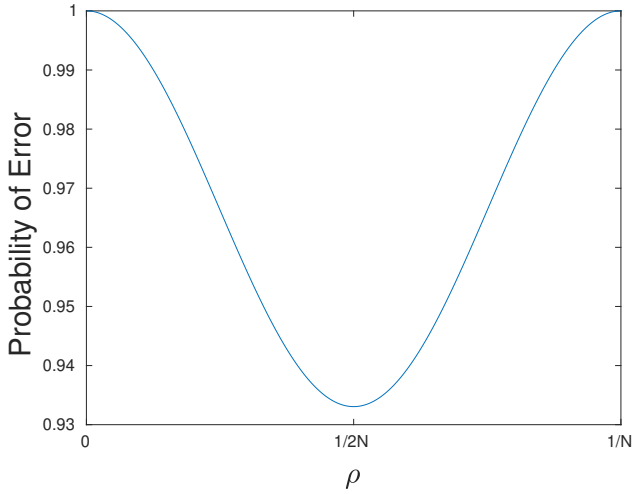
Using (543),

$$\mathbb{P}[k = k_\varphi - \ell \text{ or } k_\varphi + 1 + \ell] = \frac{1 - \cos(2\pi N\rho)}{N^2}\left[\frac{1}{1-\cos(2\pi(\rho + \frac{\ell}{N}))} + \frac{1}{1-\cos(2\pi(\frac{1}{N} - \rho + \frac{\ell}{N}))}\right]. \tag{550}$$

The terms with $\ell = 0, 1, \ldots, \ell_{\max}$ give probability that $|\hat{\varphi} - \varphi| \leq (\ell_{\max}+1)2^{-t}$. That is,

$$\mathbb{P}\left[|\hat{\varphi} - \varphi| \leq \frac{\ell_{\max}+1}{2^t}\right] = \frac{1 - \cos(2\pi N\rho)}{N^2}\sum_{\ell=0}^{\ell_{\max}} \frac{1}{1-\cos(2\pi(\rho + \frac{\ell}{N}))} + \frac{1}{1-\cos(2\pi(\frac{1}{N} - \rho + \frac{\ell}{N}))}. \tag{551}$$

The expression on the RHS is minimized when $\rho = 1/2N$. This is expected because when $\rho \to 0$ or $\rho \to 1/N$, the true phase φ has an exact t-bit expansion and by (517) one recovers the exact phase with probability 1. Hence, the lowest probability of success occurs when φ is as far away from a t-bit expansion as possible, that is when $\rho = 1/2N$. Here is an example with $t = 10$ and $\ell_{\max} = 2$.



We get a lower bound by setting $\rho = 1/2N$ in (551),

$$\mathbb{P}\left[|\hat{\varphi} - \varphi| \leq \frac{\ell_{\max}+1}{2^t}\right] \geq \frac{4}{N^2}\sum_{\ell=0}^{\ell_{\max}} \frac{1}{1-\cos(\pi(2\ell+1)/N)}. \tag{552}$$

Since $N^2(1-\cos(\pi(2\ell+1)/N))$ is decreasing in N, we can take the limit $N \to \infty$ to get the bound

$$\mathbb{P}\left[|\hat{\varphi} - \varphi| \leq \frac{\ell_{\max}+1}{2^t}\right] \geq \frac{8}{\pi^2}\sum_{\ell=0}^{\ell_{\max}} \frac{1}{(2\ell+1)^2}. \tag{553}$$

This bound is tight. We can analyze this bound using the sum $\sum_{\ell=1}^{\infty} 1/\ell^2 = \pi^2/6$ which implies $\sum_{\ell=0}^{\infty} 1/(2\ell+1)^2 = \pi^2/8$. We get

$$\begin{aligned}\mathbb{P}\left[|\hat{\varphi} - \varphi| \leq \frac{\ell_{\max}+1}{2^t}\right] &= \frac{8}{\pi^2}\left(\sum_{\ell=0}^{\infty}\frac{1}{(2\ell+1)^2} - \sum_{\ell=\ell_{\max}+1}^{\infty}\frac{1}{(2\ell+1)^2}\right) \\ &= 1 - \frac{8}{\pi^2}\sum_{\ell=\ell_{\max}+1}^{\infty}\frac{1}{(2\ell+1)^2}.\end{aligned} \tag{554}$$

We can now upper bound the last sum using an integral to get

$$\begin{aligned}\mathbb{P}\left[|\hat{\varphi}-\varphi| \leq \frac{\ell_{\max}+1}{2^t}\right] &\geq 1-\frac{8}{\pi^2}\sum_{\ell=\ell_{\max}+1}^{\infty}\frac{1}{(2\ell+1)^2} \\ &\geq \frac{8}{\pi^2}\left(\sum_{\ell=0}^{\infty}\frac{1}{(2\ell+1)^2}-\int_{\ell_{\max}}^{\infty}dx\ \frac{1}{(2x+1)^2}\right) \\ &= 1-\frac{4}{\pi^2}\cdot\frac{1}{2\ell_{\max}+1}. \end{aligned} \tag{555}$$

The reader may set $\ell_{\max}+1=2^{t-\kappa}$ to get

$$\mathbb{P}\left[|\hat{\varphi}-\varphi| \leq 2^{-\kappa}\right] \geq 1-\frac{2}{\pi^2}\cdot\frac{1}{2^{t-\kappa}-1/2}. \tag{556}$$

Setting the RHS to $1-\epsilon$ gives a κ-bit approximation with probability at least $1-\epsilon$ if

$$t \geq \kappa - 1 + \log_2\left(1+\frac{4}{\pi^2\epsilon}\right). \tag{557}$$

24 Quantum Error Correction

If the quantum state of your qubits changes during a computation, the result is corrupted. This changing of the state is called decoherence, and it is inevitable because the quantum computer interacts with everything. This is also true in classical computing, but it's more benign. The interactions of a classical computer with its environment are weak in comparison with the quantum setting. Nevertheless, many weak interactions can build up to a large perturbation. Luckily, dissipative processes, e.g. energy loss as heat to the environment, quickly dampen individual weak interactions making it hard for them to buildup. Even still, we do need to worry about bit flips in classical computers, and that is why you pay the big bucks for EC-RAM (error correcting RAM). If a bit flips, we observe it and correct it. This can only be done if there is some redundancy which tells us what the original bit was. Here is a simple example of the workflow,

$$0 \xrightarrow{\text{encode}} 000 \xrightarrow[\text{error}]{\text{observe}} 010 \xrightarrow[\text{correct}]{\text{error}} 000 \tag{558}$$

The classical bit is encoded into 3 identical bits, providing redundancy. We periodically observe the bits at a frequency higher than the rate of bit-flips due to the environment (e.g. from cosmic rays). The rate of observation must be high enough to ensure that at most one bit-flip will occur between observations. When an error is observed, it is corrected to the majority bit. Can we implement the same workflow for protecting the integrity of qubits. The first difference in the quantum setting is that a general qubit can be in a superposition,

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle. \tag{559}$$

This means there are a continuum of possible errors. Any change in α, β constitutes an error. A change from $(|0\rangle + |1\rangle)/\sqrt{2}$ to $(|0\rangle - |1\rangle)/\sqrt{2}$ could be disastrous in a quantum algorithm, even though it does not materially affect the state because the measurement probabilities are unchanged. To do quantum error-correction, we aim for something like

$$|\psi\rangle \xrightarrow{\text{encode}} |\psi\rangle \otimes |\psi\rangle \otimes |\psi\rangle \xrightarrow[\text{error}]{\text{observe}} |\psi\rangle \otimes |\psi'\rangle \otimes |\psi\rangle \xrightarrow[\text{correct}]{\text{error}} |\psi\rangle \otimes |\psi\rangle \otimes |\psi\rangle \tag{560}$$

There are two challenges here, and these are fundamental challenges because they are at the heart of quantum mechanics. The first step, encoding, is quantum state cloning. The last step also looks like state cloning, because we are copying $|\psi\rangle$ onto $|\psi'\rangle$. You cannot clone quantum states. Even worse, how do you observe the error? You won't know that it is the middle qubit that changed. Any of the qubits could have changed. So you have to "measure" all three qubits. When you measure, the state collapses to one of the 8 possible pure states $|000\rangle, \ldots, |111\rangle$. If there has been no corruption, you will observe $|000\rangle$ with probability $\|\alpha\|^2$. Suppose you do observe $|000\rangle$. Now what? You have no idea what the original state was. All information on α, β is lost. So not only does measuring the state to detect the error destroy the state. The result of the measurement cannot even tell us if there was an error. Measuring $|000\rangle$, only tell us the original state had nonzero projection on $|000\rangle$.

- There are a continuum of possible errors resulting from coupling to *anything* in the universe.
- We cannot check for error because measuring the state destroys it.
- Even if we could check for error, we restore the original state via a copy because of no-cloning.

These roadblocks spell doom for quantum error correction. Here is what we need to have any chance:

- Ability to build in redundancy without cloning.
- Ability to check for errors without actually measuring the state.
- Ability to correct any errors without cloning.

For quantum computing to become reality, we need radically new breakthrough ideas.

24.1 Quantum Redundancy

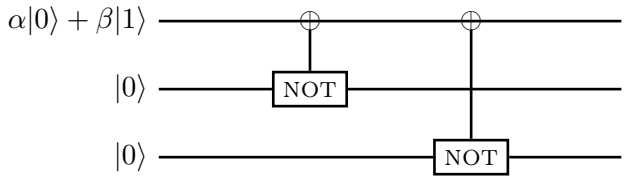
Luckily some of these breakthrough ideas have been developed. Let's begin with quantum redundancy. The key idea is that instead of copying the entire state, which we cannot do because of no-cloning, we are going to entangle the state with auxilliary qubits, to ensure that information about the state is contained in multiple qubits. Ideally, we would like, but can't have:

$$|\psi\rangle \to |\psi\rangle \otimes |\psi\rangle \otimes |\psi\rangle. \tag{561}$$

We can only clone classical bits, $|0\rangle \to |000\rangle$ and $|1\rangle \to |111\rangle$. How about a general version of this,

$$\alpha|0\rangle + \beta|1\rangle \to \alpha|000\rangle + \beta|111\rangle? \tag{562}$$

This is a 3-qubit encoding of the original state that emphasizes the superposition by enforcing it on all three qubits "simultaneously". If you measure, all three bits will be the same. So, the measured pure state is cloned. Since the output is a 3-qubit state, and quantum operators are unitary, the input must be three qubits, the other two qubits start in state $|0\rangle$. Here is a circuit for this encoding,



(563)

One way to verify that the circuit works and produces $\alpha|000\rangle + \beta|111\rangle$ is to compute the unitary operator U for this circuit by computing its action on the basis states. Apply U it to the starting state and verify that you get the desired state. We ask you to compute U and verify

$$\begin{bmatrix} & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & U & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \end{bmatrix} \begin{bmatrix} \alpha \\ 0 \\ 0 \\ 0 \\ \beta \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \beta \end{bmatrix} \tag{564}$$

We will reason directly from the circuit's gates. Label the first controlled-NOT as cNOT_{12} to indicate it is qubit 1 controlling a NOT on qubit 2. Similarly label the second controlled-NOT as cNOT_{13}. These operators are linear. By linearity of tensor product, the starting state is

$$(\alpha|0\rangle + \beta|1\rangle) \otimes |0\rangle \otimes |0\rangle = \alpha|0\rangle \otimes |0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |0\rangle \otimes |0\rangle \tag{565}$$

Applying the cNOT_{12} and using linearity,

$$\begin{aligned} & \text{cNOT}_{12}(\alpha|0\rangle \otimes |0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |0\rangle \otimes |0\rangle) && (566)\\ = \; & \alpha\text{cNOT}_{12}(|0\rangle \otimes |0\rangle \otimes |0\rangle) + \beta\text{cNOT}_{12}(|1\rangle \otimes |0\rangle \otimes |0\rangle) && (567)\\ = \; & \alpha|0\rangle \otimes |0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |1\rangle \otimes |0\rangle && (568)\end{aligned}$$

The first term is because the controlling bit is 0 and so the cNOT acts identity on the 2nd bit. The second term is because the controlling bit is 1 so the cNOT flips the 2nd bit. Now apply the cNOT_{13} and again use linearity to get

$$\begin{aligned} & \text{cNOT}_{13}(\alpha|0\rangle \otimes |0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |1\rangle \otimes |0\rangle) && (569)\\ = \; & \alpha\text{cNOT}_{13}(|0\rangle \otimes |0\rangle \otimes |0\rangle) + \beta\text{cNOT}_{13}(|1\rangle \otimes |1\rangle \otimes |0\rangle) && (570)\\ = \; & \alpha|0\rangle \otimes |0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |1\rangle \otimes |1\rangle, && (571)\end{aligned}$$

as desired. We can clone the classical bits not only when they are in isolation, but also when they are in a superposition, as is the case for a general quantum state. Since 3 qubits are used in the encoding, this is called a 3-qubit code.

24.2 Modeling the Error

There are three types of error that can occur to a qubit's state. This is because any single qubit error is caused by evolution under some 2×2 unitary matrix, E. A basis for 2×2 matrices is

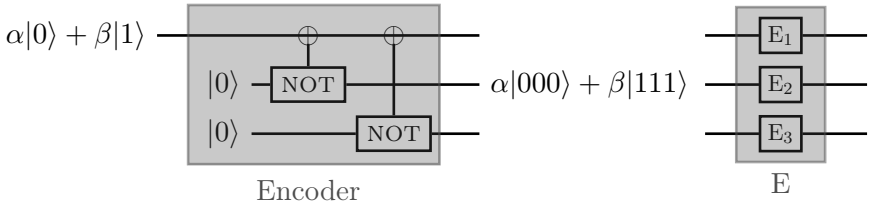
$$\text{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \qquad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{572}$$

so E can be written as a linear combination of these matrices,

$$\text{E} = (1-\epsilon)\text{I} + \epsilon_x\sigma_x + \epsilon_y\sigma_y + \epsilon_z\sigma_z \tag{573}$$

The error is caused by the $\epsilon_x\sigma_x + \epsilon_y\sigma_y + \epsilon_z\sigma_z$ term. The first of these, the σ_x term, is bit-flip (note that $\sigma_x = \text{NOT}$). The third, the σ_z term, is a phase flip. The second, the σ_y term, is a combined phase and bit flip. To keep things simple while illustrating all the main ideas, we will only focus on the bit flip error. All the main ideas extend to the more general error, and you can read about it in more complete treatments. The main change is that you need to increase redundancy by encoding a qubit using more than just 3 qubits.

To model this simple bit-flip error, we will assume that to one of the qubits in our encoded state is applied at most one NOT. The error operator for our setting is illustrated in the circuit below,



(574)

Three 1-qubit error operators $\mathrm{E}_1, \mathrm{E}_2, \mathrm{E}_3$ operate independently on each qubit of the encoded state $\alpha|000\rangle + \beta|111\rangle$. Each E_i is either I or NOT, and at most one of the E_i is not I. One of four states can result after the error. If all E_i are I, then $\mathrm{E} = \mathrm{I}$ and the resulting state is the uncorrupted state,

$$|\psi_0\rangle = \alpha|000\rangle + \beta|111\rangle. \tag{575}$$

If $E_1 = \text{NOT}$, then $\mathrm{E} = \text{NOT} \otimes \mathrm{I} \otimes \mathrm{I}$. We know E's action on pure states, it just flips the first bit. We use linearity to get E's action on the encoded state,

$$\mathrm{E}(\alpha|000\rangle + \beta|111\rangle) = \alpha\mathrm{E}(|000\rangle) + \beta\mathrm{E}(|111\rangle) = \alpha|100\rangle + \beta|011\rangle. \tag{576}$$

Similarly, if E_2 or E_3 are NOT, the corrupted states are $\alpha|010\rangle+\beta|101\rangle$ or $\alpha|001\rangle+\beta|110\rangle$ respectively. The four options for the corrupted state are

$$\begin{aligned} |\psi_0\rangle &= \alpha|000\rangle + \beta|111\rangle & \mathrm{E} &= \mathrm{I} \otimes \mathrm{I} \otimes \mathrm{I} \\ |\psi_1\rangle &= \alpha|100\rangle + \beta|011\rangle & \mathrm{E} &= \text{NOT} \otimes \mathrm{I} \otimes \mathrm{I} \\ |\psi_2\rangle &= \alpha|010\rangle + \beta|101\rangle & \mathrm{E} &= \mathrm{I} \otimes \text{NOT} \otimes \mathrm{I} \\ |\psi_3\rangle &= \alpha|001\rangle + \beta|110\rangle & \mathrm{E} &= \mathrm{I} \otimes \mathrm{I} \otimes \text{NOT} \end{aligned} \tag{577}$$

No pure state occurs in more than one of the corrupted states. This means these corrupted states are pairwise orthogonal.[5] So now, if you measure and (say) get $|100\rangle$, you know there has been an error. We have a fighting chance. You even know it was a bit flip in the first qubit of the encoded state. In contrast, if you measure the un-encoded state and get (say) $|0\rangle$, you don't know anything. We can at least detect the error. However, the state is destroyed, and all information on α, β is lost. Detecting the error in this way is not going to allow for error-correction.

Out ability to detect the error crucially depends on the possible corrupted states being orthogonal. That way, the result of the measurement uniquely identifies not just the error, but the type of error. This concept will generalize to other types of errors. This requirement that the possible corrupted states should be pairwise orthogonal can only be accomplished if the encoding uses sufficiently many qubits. We can see this as follows. We start with a superposition of two pure states, so after a possible bit-flip the possible corrupted state will be a superposition of two pure states. Orthogonality requires that no pure state be repeated in any two of the four possible corrupted states, hence we need at least 8 pure states in the encoding. This means the encoded state must use at least 3 qubits which provides $2^3 = 8$ pure states.

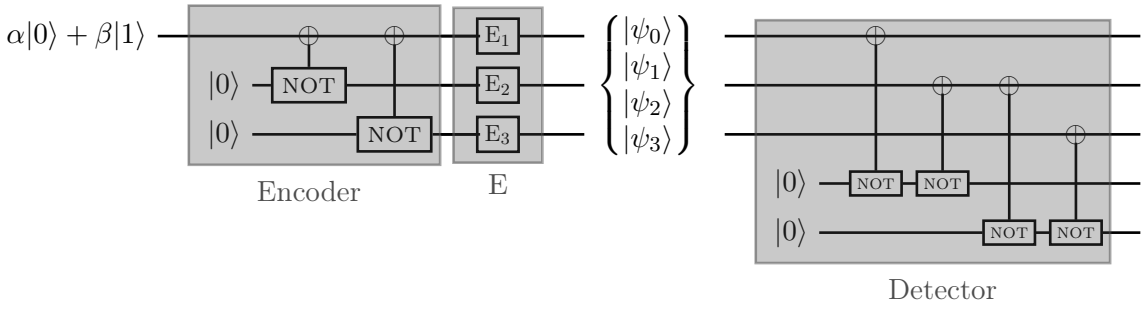
24.3 Detecting Bit-Flip Error

To detect the error, the great idea is not to measure the state itself. Rather, entangle the state with some ancillary qubits. Through entanglement, the state of the ancillary qubits will depend on which of the four corrupted states emerged after the error operator. By measuring the state of the

[5] You can also see this from the vector representations of the corrupted states,

$$|\psi_0\rangle = \begin{bmatrix} \alpha \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \beta \end{bmatrix}, \qquad |\psi_1\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \beta \\ \alpha \\ 0 \\ 0 \\ 0 \end{bmatrix}, \qquad |\psi_2\rangle = \begin{bmatrix} 0 \\ 0 \\ \alpha \\ 0 \\ 0 \\ \beta \\ 0 \\ 0 \end{bmatrix}, \qquad |\psi_3\rangle = \begin{bmatrix} 0 \\ \alpha \\ 0 \\ 0 \\ 0 \\ 0 \\ \beta \\ 0 \end{bmatrix}.$$

ancillary qubits, we can reveal the error without measuring (and destroying) the state itself. Once the great idea is revealed, it is not so hard to accomplish. To entangle qubits, use one qubit to control an operator, for example NOT, on another. We give a circuit that accomplishes this with two ancillary cubits. Why must we have at least two ancillary qubits? Because measuring the ancillary qubits produces a pure state that must distinguish between 4 possible corrupted states. Hence, there must be at least 4 pure states for the ancillae, which requires at least two ancillary qubits.



(578)

In the detector, qubits 1 and 2 of the corrupted state control NOTs on the first ancillary. Label these controlled NOTs as cNOT_{14} and cNOT_{24} to identify the controlling and controlled bit. Then, qubits 2 and 3 of the corrupted state control NOTs on the second ancillary. Label these controlled NOTs as cNOT_{25} and cNOT_{35}. We need to examine the action of the sequence of operators

$$\text{cNOT}_{14}, \text{cNOT}_{24}, \text{cNOT}_{25}, \text{cNOT}_{35} \tag{579}$$

on the four possible states

$$|\psi_0\rangle \otimes |00\rangle, \qquad |\psi_1\rangle \otimes |00\rangle, \qquad |\psi_2\rangle \otimes |00\rangle, \qquad |\psi_3\rangle \otimes |00\rangle. \tag{580}$$

By linearity of the tensor product,

$$\begin{aligned}
|\psi_0\rangle \otimes |00\rangle &= \alpha|000\rangle \otimes |00\rangle + \beta|111\rangle \otimes |00\rangle \\
|\psi_1\rangle \otimes |00\rangle &= \alpha|100\rangle \otimes |00\rangle + \beta|011\rangle \otimes |00\rangle \\
|\psi_2\rangle \otimes |00\rangle &= \alpha|010\rangle \otimes |00\rangle + \beta|101\rangle \otimes |00\rangle \\
|\psi_3\rangle \otimes |00\rangle &= \alpha|001\rangle \otimes |00\rangle + \beta|110\rangle \otimes |00\rangle
\end{aligned} \tag{581}$$

Let us now use linearity of the cNOT operators to perform the detector operations on $|\psi_0\rangle \otimes |00\rangle$. We apply the four cNOT operators to each pure state in the superposition and add the final results.

$$\begin{aligned}
|\psi_0\rangle \quad & \alpha|000\rangle \otimes |00\rangle + \beta|111\rangle \otimes |00\rangle \\
\text{cNOT}_{14} \quad & \alpha|000\rangle \otimes |\mathbf{0}0\rangle + \beta|111\rangle \otimes |\mathbf{1}0\rangle \\
\text{cNOT}_{24} \quad & \alpha|000\rangle \otimes |\mathbf{0}0\rangle + \beta|111\rangle \otimes |\mathbf{0}0\rangle \\
\text{cNOT}_{25} \quad & \alpha|000\rangle \otimes |0\mathbf{0}\rangle + \beta|111\rangle \otimes |0\mathbf{1}\rangle \\
\text{cNOT}_{35} \quad & \alpha|000\rangle \otimes |0\mathbf{0}\rangle + \beta|111\rangle \otimes |0\mathbf{0}\rangle \\
\hline
& (\alpha|000\rangle + \beta|111\rangle) \otimes |00\rangle
\end{aligned} \tag{582}$$

The last step uses linearity of the tensor product. We highlighted the bits involved in each operation, blue for controlling and red for controlled. In the first step, bit 1 controls bit 4: when bit-1 is 0,

bit-4 is unchanged; when bit-1 is 1, bit-4 flips. The other three steps are similar. The conclusion is,

$$|\psi_0\rangle \otimes |00\rangle \to |\psi_0\rangle \otimes |00\rangle \tag{583}$$

Let's repeat this calculation for $|\psi_1\rangle \otimes |00\rangle$,

$$\begin{array}{rl} |\psi_1\rangle & \alpha|100\rangle \otimes |00\rangle + \beta|011\rangle \otimes |00\rangle \\ \text{cNOT}_{14} & \alpha|100\rangle \otimes |\mathbf{1}0\rangle + \beta|011\rangle \otimes |\mathbf{0}0\rangle \\ \text{cNOT}_{24} & \alpha|100\rangle \otimes |\mathbf{1}0\rangle + \beta|011\rangle \otimes |\mathbf{1}0\rangle \\ \text{cNOT}_{25} & \alpha|100\rangle \otimes |1\mathbf{0}\rangle + \beta|011\rangle \otimes |1\mathbf{1}\rangle \\ \text{cNOT}_{35} & \alpha|100\rangle \otimes |1\mathbf{0}\rangle + \beta|011\rangle \otimes |1\mathbf{0}\rangle \\ \hline & (\alpha|100\rangle + \beta|011\rangle) \otimes |10\rangle \end{array} \tag{584}$$

The conclusion is

$$|\psi_1\rangle \otimes |00\rangle \to |\psi_1\rangle \otimes |10\rangle \tag{585}$$

It's now your turn to verify these calculations for $|\psi_2\rangle \otimes |00\rangle$ and $|\psi_3\rangle \otimes |00\rangle$.

$$\begin{array}{rl} |\psi_2\rangle & \alpha|010\rangle \otimes |00\rangle + \beta|101\rangle \otimes |00\rangle \\ \text{cNOT}_{14} & \alpha|010\rangle \otimes |\mathbf{0}0\rangle + \beta|101\rangle \otimes |\mathbf{1}0\rangle \\ \text{cNOT}_{24} & \alpha|010\rangle \otimes |\mathbf{1}0\rangle + \beta|101\rangle \otimes |\mathbf{1}0\rangle \\ \text{cNOT}_{25} & \alpha|010\rangle \otimes |\mathbf{1}1\rangle + \beta|101\rangle \otimes |\mathbf{1}0\rangle \\ \text{cNOT}_{35} & \alpha|010\rangle \otimes |\mathbf{1}1\rangle + \beta|101\rangle \otimes |\mathbf{1}1\rangle \\ \hline & (\alpha|010\rangle + \beta|101\rangle) \otimes |11\rangle \end{array} \qquad \begin{array}{rl} |\psi_3\rangle & \alpha|001\rangle \otimes |00\rangle + \beta|110\rangle \otimes |00\rangle \\ \text{cNOT}_{14} & \alpha|001\rangle \otimes |\mathbf{0}0\rangle + \beta|110\rangle \otimes |\mathbf{1}0\rangle \\ \text{cNOT}_{24} & \alpha|001\rangle \otimes |\mathbf{0}0\rangle + \beta|110\rangle \otimes |\mathbf{0}0\rangle \\ \text{cNOT}_{25} & \alpha|001\rangle \otimes |\mathbf{0}0\rangle + \beta|110\rangle \otimes |\mathbf{0}1\rangle \\ \text{cNOT}_{35} & \alpha|001\rangle \otimes |\mathbf{0}1\rangle + \beta|110\rangle \otimes |\mathbf{0}1\rangle \\ \hline & (\alpha|001\rangle + \beta|110\rangle) \otimes |01\rangle \end{array} \tag{586}$$

The four possible states after going through the detector are as follows,

$$|\psi_0\rangle \to |\psi_0\rangle \otimes |\mathbf{00}\rangle \tag{587}$$

$$|\psi_1\rangle \to |\psi_1\rangle \otimes |\mathbf{10}\rangle \tag{588}$$

$$|\psi_2\rangle \to |\psi_2\rangle \otimes |\mathbf{11}\rangle \tag{589}$$

$$|\psi_3\rangle \to |\psi_3\rangle \otimes |\mathbf{01}\rangle \tag{590}$$

The state of the ancillae is highlighted. The important fact is that the ancillae are in a *different* pure state depending on the corrupted state.

24.4 Correcting Bit-Flip Error

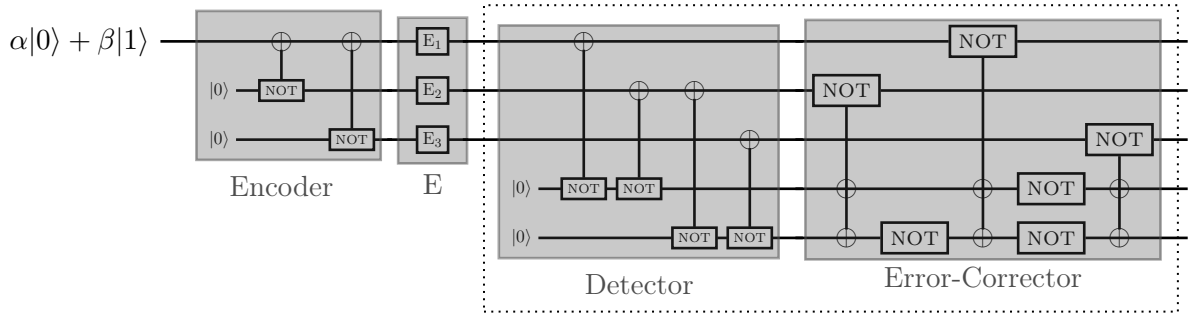
Measuring the state of the ancillae produces a different pure state for each possibile corrupted state of our 3-qubit encoded state. Depending on the outcome, we detect if there was an error, and if so which bit was flipped. We can then unflip that bit. For example, if we measure the ancillae as $|11\rangle$, we know qubit 2 in the encoded state was flipped, so we unflip it by applying NOT. This recovers the uncorrupted state. We have managed detected and corrected the bit-flip error without damaging the state. We also still do not know anything about α, β.

That could be the end of the story with respect to bit-flip errors, but there is some value in embelishing a little. We don't actually need to measure the ancillae. The reason is we know the

ancillae are in specific pure states, depending on the corrupted 3-qubit encoded state. Hence, we can use the ancillae to appropriately control NOTs on the corrupted 3-qubit state. This will restore the 3-qubit code to its uncorrupted state without the need for disruptive measuring. The logic we need to implement is

1: **if** ancillae = $|00\rangle$ **then**
2: Do nothing
3: **else if** ancillae = $|10\rangle$ **then**
4: Apply NOT_1
5: **else if** ancillae = $|11\rangle$ **then**
6: Apply NOT_2
7: **else if** ancillae = $|01\rangle$ **then**
8: Apply NOT_3

This is accomplished by the error-corrector circuit in the gray box,



(591)

The output of this circuit for any α, β is

$$(\alpha|0\rangle + \beta|1\rangle) \otimes |\text{ancillae}\rangle. \tag{592}$$

The two ancillae are stored in some pure state, but we do not know what that state is. The full bit-flip quantum error correction circuit enclosed in the dotted rectangle above is applied periodicallyto correct bitflip error. To reuse the ancillae one has to reset them to $|00\rangle$ which is most easily done by measuring them and applying NOT if needed.

24.5 Generalizing to Other Errors

25 Quantum Factoring

The input is an integer $N > 2$. The output is a non-trivial factor D of N if one exists ($1 < D < N$ and D divides N). If no such factor exists, then the algorithm should declare N to be prime. A simple classical algorithm tests if any of $2, 3, \ldots, \lfloor \sqrt{N} \rfloor$ divide N. If yes, you have a factor and if no, N is prime. Testing if i divides N is in $\Theta(\log^3 N)$ using simple grade school algorithms, so the total runtime is in $\Theta(\sqrt{N} \log^3 N)$, which is an exponential algorithm. Exponential because the input size is $O(\log_2 N)$, the number of bits needed to specify N.

Shor's quantum algorithm for factoring runs in poly($\log N$) time. It is the one example we have of exponential speedup against the best known classical algorithm. (Grover's search algorithm only gives quadratic speedup.) This does not, however, mean that quantum computing gives exponential speedups over classical computing because we do not know if a factor can be found in poly($\log N$) time on a classical computer. Factoring is a complex number theoretic task. It is not clear how a quantum computer, by somehow using quantum parallelism, could help with factoring. We know how to test properties of functions using quantum parallelism. The general idea is to evaluate the function on a massive superposition, producing a state that has all the information you need to test the given property of the function. The remaining quantum-magic is to unravel all that information to get at the property you want. It turns out that the property of a function we will need is its period. Given a periodic function $f : \mathbb{N} \mapsto [0, N-1]$, suppose

$$f(x + r) = f(x), \tag{593}$$

for some $r \in [1, N-1]$. We want to identify r. A classical algorithm could compare $f(x)$ to $f(x+1), f(x+1), \ldots, f(x+N-1)$ to identify r. The runtine is $\Theta(N)$ (exponential), which is even worse than the simple classical algorithm for factoring above. The plan is

(1) Reduce factoring to finding the period of some function. [Number Theory]
(2) Build the mathematical tools for period finding. [Fourier Transform]
(3) Quantum Fourier Transform makes period finding poly-time. [QFT]

The FFT is a landmark algorithm, both as a primitive in other algorithms and also as a tool for fast signal processing. Its impact has been immense. So, even if you have no interest in factoring or quantum computing, you will want to learn these mathematical tools.

25.1 Factoring and Period Finding

This section uses basic number theory to reformulate the factoring problem into one of testing the periodicity of a function defined on the natural numbers. One may skip the number theory and focus on the algorithmic details without sacrificing the logical flow (go to Section 25.2).

Since one can test if N is even, in which case 2 is a nontrivial factor, assume N is odd. As a working example, let $N = 15$. Consider an integer $a > 1$ and let

$$\gcd(N, a) = D. \tag{594}$$

If $1 < D < N$, then D is a nontrivial factor. For example, if $a = 25$, then $\gcd(15, 25) = 5$ and 5 is a factor of 15. If $D = 1$, then we cannot immediately get a factor from D. For example suppose

$a = 26$. There is some progress we can make even in this case. Notice that $26^2 = 676$ and 675 is divisible by 15, so

$$26^2 \equiv 1 \pmod{15}, \tag{595}$$

where $b \equiv c \pmod{d}$ if and only if d divides $b - c$, written $d|(b-c)$. Further, 26 ± 1 is not divisible by 15, but $\gcd(26 \pm 1, 15) > 1$, and either case gives a factor of 15. Indeed, $\gcd(25, 15) = 5$ and $\gcd(27, 15) = 3$. This example is not an isolated case. Indeed, consider any $a > 1$ for which

$$\begin{aligned} a^2 &\equiv 1 \pmod{N}; & (596)\\ a &\not\equiv \pm 1 \pmod{N}. & (597) \end{aligned}$$

We claim that $\gcd(a \pm 1, N) > 1$. Intuitively, if $N|(a^2 - 1)$ then $N|(a + 1)(a - 1)$ and N does not divide either term in the product, then the nontrivial factors of N must be spread across both terms. Let us formulate this explicitly as a lemma.

Lemma 25.1. Suppose $a, N > 1$ with

$$a^2 \equiv 1 \pmod{N} \qquad \text{and} \qquad a \not\equiv \pm 1 \pmod{N}. \tag{598}$$

Then $1 < \gcd(a + 1, N) < N$ and $1 < \gcd(a - 1, N) < N$.

The impact of this lemma is that if some a satisfies the conditions, then computing $\gcd(a-1, N)$ gives a nontrivial factor of N. Using Euclid's algorithm, one can compute the gcd in $O(\log N)$. We assume the reader is familiar with or can look up Euclid's algorithm.

Proof. Since $a^2 \equiv 1 \pmod{N}$, this means $N|(a^2-1)$, or $N|(a+1)(a-1)$. If $\gcd(N, a+1) = 1$, then by Euclid's Lemma[6], $N|(a - 1)$ which contradicts $a \not\equiv 1 \pmod{N}$. Therefore $\gcd(N, a + 1) > 1$. Similarly, if $\gcd(N, a - 1) = 1$, then $N|(a + 1)$ contradicting $a \not\equiv -1 \pmod{N}$. The upper bound $\gcd(a \pm 1, N) < N$ follows because N does not divide $a \pm 1$. ∎

The goal is to find an a satisfying the conditions in Lemma 25.1. We then get a factor efficiently by computing $\gcd(a - 1, N)$. To find such an a, we introduce the period of a number modulo N. Let us compute the powers of 2 modulo $N = 15$,

	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}
(mod 15)	2	4	8	1	2	4	8	1	2	4	8	1

(599)

Observe that $2^m \pmod{15}$ as a function of m is periodic and the period is 4. That is, $2^{m+4} \equiv 2^m \pmod{15}$. This is immediate from $2^4 \equiv 1 \pmod{15}$, because[7],

$$\begin{aligned} 2^4 &\equiv 1 && \pmod{15}\\ 2^m &\equiv 2^m && \pmod{15}\\ \hline 2^4 \times 2^m &\equiv 1 \times 2^m && \pmod{15} \end{aligned} \tag{600}$$

[6] *Euclid's Lemma*: If $d|bc$ and $\gcd(d, b) = 1$ then $d|c$. Proof: By Bezout's identity, $\gcd(d, b) = 1 = xd+yb$. Multiply both sides by c to get $c = xdc + ybc$. The RHS is divisible by d since $d|bc$, hence the LHS is divisible by d and $d|c$.

[7] The reader should verify that $a \equiv b \pmod{d}$ and $c \equiv d \pmod{d}$ implies $ac \equiv bd \pmod{d}$.

Let's play this same game with $x = 3$,

	3^1	3^2	3^3	3^4	3^5	3^6	3^7	3^8	3^9	3^{10}	3^{11}	3^{12}
(mod 15)	3	9	12	6	3	9	12	6	3	9	12	6

(601)

Again, the function $3^m \pmod{15}$ is periodic, and again with period 4. The crucial difference between 2^m and 3^m is that 2^m is congruent to 1 modulo 15 for some $m \geq 1$ but 3^m is never congruent to 1 modulo 15. This is because $\gcd(3, 15) = 3 > 1$.[8] We say that the *order* of 2 modulo 15 is 4,

$$\text{order}(2) = 4 \pmod{15}. \tag{602}$$

For a general a with $1 < a < N$, the order of a modulo N is the smallest number $r \geq 1$ for which

$$a^r \equiv 1 \pmod{N}. \tag{603}$$

Let's compute the orders of $2, 3, \ldots, 14$ modulo $N = 15$. The reader should verify,

a	2	3	4	5	6	7	8	9	10	11	12	13	14
order(a)	4	✓	2	✓	✓	4	4	✓	✓	2	✓	4	2

(604)

The order does not exist when $\gcd(a, N) > 1$, which we already proved. The ✓-marks indicate these cases. Why a checkmark will become clear soon.

- When $\gcd(a, N) = 1$ the order always exists and is less than N.
- The order appears to be always even.

The first claim is true. The second claim is almost true. Here is an example of odd order,

$$\text{order}(2) = 3 \pmod{7}, \tag{605}$$

because $2^3 \equiv 1 \pmod{7}$. To prove the first claim, consider thr first N powers of z modulo N,

$$a^1, a^2, a^3, \ldots, a^N \pmod{N}. \tag{606}$$

None of these are congruent to 0 because $\gcd(a, N) = 1$. Therefore $1 \leq a^i \pmod{N} \leq N - 1$. By pigeonhole, two of these powers are congruent to each other modulo N, that is for $i < j$,

$$a^j \equiv a^i \pmod{N}. \tag{607}$$

This means $N|(a^j - a^i)$, or $N|a^i(a^{j-i} - 1)$. Since[9] Since $\gcd(a^i, N) = 1$, by Euclid's Lemma $N|(a^{j-i} - 1)$. This means

$$a^{j-i} \equiv 1 \pmod{N} \qquad \rightarrow \qquad \text{order}(a) \leq j - i \pmod{N}. \tag{608}$$

We have proved the following lemma,

[8] As an exercise, prove that $\gcd(x, N) = D > 1$ implies $\gcd(x^m, N) \geq D > 1$. Also prove that if $x^m \equiv 1 \pmod{N}$ then $\gcd(x^m, N) = 1$ (use Bezout's identity). Therefore if $\gcd(x, N) > 1$ then $x^m \not\equiv 1 \pmod{N}$ for any $m \geq 1$.

[9] $\gcd(a, N) = 1$ implies $\gcd(a^i, N) = 1$, which the reader can prove by induction or directly using Bezout's identity.

Lemma 25.2. If $\gcd(a, N) = 1$ then $r = \text{order}(a) \pmod N$ exists and $r < N$.

We went out of our way to highlight that the order was even, a fact which we conceded was not always true. An even order is useful. Indeed, if a has even order r, that is $a^r \equiv 1 \pmod N$, then $z = a^{r/2}$ is an integer. Then,

$$z^2 = a^r \equiv 1 \pmod N. \tag{609}$$

We have found a z that satisfies the first condition in Lemma 25.1. We also know that $z \not\equiv 1 \pmod N$ because r is the order of a, that is the *smallest* power congruent to 1. If it also the case that $z \not\equiv -1 \pmod N$, then all the conditions of Lemma 25.1 are satisfied, and we can find a factor of N using $\gcd(z - 1, N)$. Let's work through another example, $N = 21$,

a	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
order, r	6	✓	3	6	✓	✓	2	✓	6	6	✓	2	✓	✓	3	6	✓	6	2
$z = a^{r/2}$	8		✘	125			8		1000	1331		13			✘	4913		6859	20
$z \not\equiv -1$?	✓		✘	✘			✓		✓	✓		✓			✘	✘		✓	✘
factor?	✓	✓	✘	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✘	✘	✓	✓	✘

(610)

The first row shows that of the 19 possibilities for a, eight of them immediately give a factor by computing $\gcd(a, N) > 1$. The other 11 give $\gcd(a, N) = 1$ and one can find the order. The third row shows that in 6 of those cases where $\gcd(a, N) = 1$, the order is even and $a^{r/2} \not\equiv -1 \pmod N$. The conclusion is shown in the last row. Of the 19 possibilities for z, five of them cannot be used to conveniently get a factor for $N = 21$. The other 14 possibilities for a give a nontrivial factor in one of two ways:

- $\gcd(a, N) > 1$ in which case $\gcd(a, N)$ is a nontrivial factor.
- $\gcd(a, N) = 1$, the order r of a is even and $a^{r/2} \not\equiv -1 \pmod N$, in which case $\gcd(a^{r/2} - 1, N)$ gives a nontrivial factor.

For $N = 21$, the chances are $14/19 \approx 74\%$ that a random $a \in [2, N - 1]$ produces a nontrivial factor. This situation is not isolated. More generally, the probability is at least $\frac{1}{2}$.

Lemma 25.3. Given an odd composite N, let $G_N \subseteq [1, N - 1]$ be those numbers less than N which are coprime with N. Pick an $a \in G_N$ randomly. Let $r = \text{order}(a) \pmod N$. Then,

$$\mathbb{P}\left[r \text{ is even and } a^{r/2} \not\equiv -1 \pmod N\right] \geq 1/2. \tag{611}$$

Lemma 25.3 implies the following result which leads to the algorithm in the next section. Pick an a randomly from $[2, N - 1]$ and let $p = \mathbb{P}[\gcd(a, N) > 1]$. Success occurs if either $\gcd(a, N) > 1$ or a has even order r, with $a^{r/2} \not\equiv -1 \pmod N$. Then,

$$\mathbb{P}[\text{Success}] = \underbrace{\mathbb{P}[\text{Success} \mid \gcd(a, N) > 1]}_{1} \times p + \underbrace{\mathbb{P}[\text{Success} \mid \gcd(a, N) = 1]}_{\geq \frac{1}{2} \text{ by Lemma 25.3}} \times (1 - p) \tag{612}$$

$$\geq \frac{1 + p}{2}. \tag{613}$$

The chance of failure is at most $1/2$, so by repeatedly trying k independent $a \in [2, N - 1]$, we fail to get a factor with probability at most $1/2^k$. The most intense number theory of this section is about to come in the proof of Lemma 25.3. This proof is essential for the algorithm but not instructive. On a first reading you may skip it and focus on the algorithm in Section 25.2.

25.2 Algorithm for Factoring

The previous section suggests the following algorithm to get a factor of N. Generate a random $a \in [2, N-1]$. If $\gcd(a, N) > 1$, you have a nontrivial factor. If not, but $r = \text{order}(a) \mod N$ is even, then $\gcd(a^{r/2} - 1, N)$ may give a nontrivial factor. If all this fails, with probability at most $1/2$, you have to repeat the whole process. Here is the algorithm.

```
1: Input: N > 1.
2: Test if N is prime (AKS or Miller-Rabin primality test). If so, return N is prime.
3: Test if 2 divides N. If so, return 2.
4: Test if N is a perfect power, N = a^k, a ≥ 3, k ≤ ⌊ log_3 N ⌋. If so, return a.
5: Continue if N is odd having at least two distinct prime factors.
6: repeat
7:    Pick a random a ∈ [2, N − 1].
8:    Compute D = gcd(a, N).
9:    if D = 1 then
10:      Compute r = order(a)  mod N.
11:      if r is even then
12:         Compute D = gcd(a^{r/2} − 1, N)
13: until 1 < D < N
14: return D.
```
(614)

Steps 2,3,4 are efficient. In step 4, one has to compute $O(\log N)$ k-th roots, and computing k-th roots of N can be done efficiently with classical algorithms. The algorithm makes it into the randomized loop if N is odd and has at least two distinct prime factors. The output is D. Each repetition fails with probability at most $1/2$ so the expected number of repetitions to success is 2. The only potentially inefficient steps in the algorithm are steps 7 and 9. Step 9 can be implemented as follows. First compute $q \in [0, N-1]$ such that

$$a^{r/2} \equiv q \pmod{N}. \tag{615}$$

This can be done efficiently using fast modular exponentiation that uses squaring. The number of squaring steps needed is $O(\log_2(r/2)) \in O(\log_2 N)$. In each step one performs a constant number of multiplications of $\log_2 N$-bit numbers, so the runtime is $O(\log_2^3 N)$. Finally one can compute $\gcd(q-1, N)$ to complete step 9. So, the full algorithm is polynomial if step 7, finding the period of a number, can be performed in $O(\text{poly}(\log_2 N))$ time. On a classical computer, we do not know how to do it. The entire contribution of quantum computing to factoring is to efficiently solve the period finding problem in step 7. The solution involves the Quantum Fourier Transform., We discuss these details in the next section. We also mention that period finding can be used to directly crack the RSA cryptosystem, as opposed to indirectly doing so via factoring. Also note that period finding is a special case of the general problem of "phase estimation", that is to find the eigenvalue of a unitary matrix for a given eigenvector.[10]

25.3 Proof of Lemma 25.3

[10] Since a unitary matrix preserves the norm, all eigenvalues are phases.

26 Quantum Period Finding

We reduced factoring to finding the order of an a modulo the number to be factored N, where $\gcd(a, N) = 1$. The order r is the smallest power such that $a^r \equiv 1 \pmod{N}$. Define a function

$$f(x) \equiv a^x \pmod{N}. \tag{616}$$

Note that

$$f(x+r) \equiv a^{x+r} \equiv a^x a^r \equiv a^x \equiv f(x) \pmod{N}. \tag{617}$$

Hence, the order r is the period of the periodic function f. We consider a general periodic function $f(x)$ with period r. The task is to find the period r, given a quantum black box access to U_f. Even though several optimizations are possible for the specific f in (616), for example the circuit that implements U_f can be optimized for (616), the general problem is not harder than the specific case in (616). This is because the function in (616) is quite erratic. Also, period finding is worth studying in general as it is of interest, independent of factoring.

In the general setup of period finding, $f : \{0, 1, 2, \ldots\} \mapsto [0, 2^m - 1]$ is a periodic function with period $r_0 \leq r \leq 2^m - 1$. This means $f(x + y) = f(x)$ if and only if y is an integer multiple of r. In particular this means

$$(f(0), \ldots, f(r-1)) \tag{618}$$

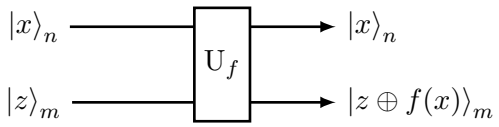
are all distinct and the entire sequence of $f(x)$ is just repetitions of the r-sequence above. We are to find the period r. Note that the assumption $r_0 \leq r$ is at the expense of a constant amount of classical overhead, for we can simply test the periods $1, 2, \ldots, r_0$ by checking $f(0) \stackrel{?}{=} f(r)$ for $r \in [1, r_0]$. For factoring, $f(x) = a^x \pmod{N}$ and $m = 1 + \lfloor \log_2 N \rfloor$. In the classical setting, a brute force approach is to compute

$$f(0), f(1), f(2), f(3), \ldots, f(r), \tag{619}$$

and in this way discover r, but the runtime is $O(2^m)$, since r can be as large as $2^m - 1$. This, in the factoring problem, is already exponential in the number of bits required to specify N. No polynomial classical algorithm is known, nor is any algorithm that is substantially better than exponential known. The big bang from quantum computing is to somehow compute all $f(i)$ simultaneously in a superposition. Of course we do not have access to those values, but we don't need all those values. We just need to infer r. By skillful untangling of the superposition, we can indeed recover r. Magic!

26.1 Quantum Black Box for f, U_f

Our function f is not Boolean, but rather the output has m bits, since it is in the range $[0, 2^m - 1]$. The quantum black box U_f is a straightforward generalization of (210) on page 54. The definition of the quantum black box unitary operator U_f for computational basis vectors is given by



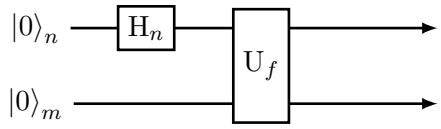
(620)

As usual, once you have specified the action of U_f on the computational basis, the full unitary operator is known. In the above representation, $z \oplus f(x)$ is bitwise XOR. The reader may show that the definition of U_f gives a unitary operator and indeed its inverse is itself.

The input registers are the top n qubits. The output registers are the bottom m qubits, which store the information about f. One interesting thing to note is that the quantum black box U_f takes an input x of n-qubits while the function f is defined for all $x \in \mathbb{N}$. This means that the quantum blackbox can only compute f for inputs $x = 0, 1, 2, \ldots, 2^n - 1$. That is we build the circuit for U_f assuming an upper bound on the x that we need to evaluate. This is related to how much information about f the quantum algorithm needs to compute the period. As we will see, we can set $n = 2m$, so the quantum circuit to determine the period of an m-bit function uses $3m$ qubits in total, in the general case. For the specific function in (616), this requirement can be optimized down to $2m$ qubits, which can make a difference in practice.

26.2 Period Finding Algorithm

To begin, let us encode global information about f in a superposition, in the usual way. Specifically, consider the circuit



(621)

The input to U_f is

$$\left(\frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle_n \right) \otimes |0\rangle_m = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle_n \otimes |0\rangle_m. \tag{622}$$

We leave the reader to use the usual maneuvers (linearity and the definition of U_f) and show that the output from U_f is the quatnum state

$$\frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle_n \otimes |f(x)\rangle_m. \tag{623}$$

We have produced the requisite superposition that contains global information about f. Now comes the trick which is not strictly necessary, but it makes the derivation simpler. Measure the bottom m qubits in the output registers. You will measure some value f_0, and the state collapses to a uniform superposition of all the pure states $|x\rangle|f_0\rangle$ that are consistent with this measurement, that is where $f(x) = f_0$. Let $x_0 < r$ be the first state satisfying this condition. There are L pure x-states in this superposition:

$$|x_0\rangle, |x_0 + r\rangle, |x_0 + 2r\rangle, |x_0 + 3r\rangle, \ldots, |x_0 + (L-1)r\rangle, \tag{624}$$

where L is the largest integer that satisfies $x_0 + (L-1)r \leq 2^n - 1$, that is

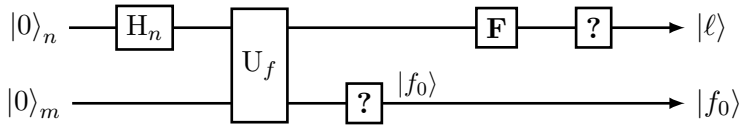
$$L = 1 + \left\lfloor \frac{2^n - 1 - x_0}{r} \right\rfloor. \tag{625}$$

The normalized state after the measurement is

$$|\psi\rangle_{n+m} = \frac{1}{\sqrt{L}} \sum_{k=0}^{L-1} |x_0 + kr\rangle_n \otimes |f_0\rangle_m. \tag{626}$$

The bottom m qubits now become irrelevant to the algorithm, so we just focus on the top n-qubit state. If you measure the top n qubits you will measure $x_0 + kr$ for some random $k \in [0, L-1]$.

This is of not much help because we don't know x_0 so we can't infer r. If we could replicate this state and then make multiple measurements, by subtracting two measurements we get rid of the x_0 and can find $(k-k')r$, an integer multiple of r. We can then infer r as the gcd of all this differences between pairs. But, by no cloning, we can replicate $|\psi\rangle$. What we need is some way to get rid of the x_0, and the Fourier Transform does exactly that, by converting it into a phase. We are going to apply the Quantum Fourier Transform to the top n-qubits, and then measure the top n-qubits. Here is the full period finding circuit.



(627)

The rest of the discussion is to compute the quantum state of the top n qubits after applying the Quantum Fourier transform, analyze what happens when you then measure, and finally to show how to use the result of the measurement to identify period r.

26.3 Applying the Quantum Fourier Transform

Let us recall the action of the Fourier Transform on a computational basis state $|j\rangle$ from (471),

$$\mathbf{F}|j\rangle = \frac{1}{2^{n/2}} \sum_{\ell=0}^{2^n-1} e^{(2\pi i)\ell j/2^n} |\ell\rangle. \tag{628}$$

Applying $\mathbf{F}$ to the state in (626) using linearity gives (we dropped the bottom m qubits)

$$\begin{aligned} \mathbf{F}|\psi\rangle &= \frac{1}{\sqrt{L}} \sum_{k=0}^{L-1} \mathbf{F}|x_0 + kr\rangle \\ &= \frac{1}{\sqrt{L}} \sum_{k=0}^{L-1} \frac{1}{2^{n/2}} \sum_{\ell=0}^{2^n-1} e^{(2\pi i)\ell(x_0+kr)/2^n} |\ell\rangle \\ &= \sum_{\ell=0}^{2^n-1} |\ell\rangle \underbrace{\frac{e^{(2\pi i)\ell x_0/2^n}}{\sqrt{2^n L}} \sum_{k=0}^{L-1} e^{(2\pi i)\ell k r/2^n}}_{\text{amplitude to measure } |l\rangle}. \end{aligned} \tag{629}$$

Notice that x_0 appears as a phase in the amplitude to measure $|\ell\rangle$. This phase will disappear when we compute the probability to measure $|\ell\rangle$. Further, the sum over k in the amplitude is a geometric sum which we can do in closed form. Therefore,

$$\mathbb{P}[\ell] = \frac{1}{2^n L} \left| \frac{e^{(2\pi i)\ell r L/2^n} - 1}{e^{(2\pi i)\ell r/2^n} - 1} \right|^2. \tag{630}$$

Let us now see what happens when we measure. To simplify the analysis, let us consider the fortuitious case where 2^n happens to be a multiple of the period r. Of course, this cannot be expected in general, but this simpler case will give all the main ideas on what is going on. The

general case follows a similar but more complicated analysis. When a whole number of periods of f are contained in the measurement window $[0, 2^n - 1]$, it means that L, the number of states making up the superposition in (624), is precisely $2^n/r$. That is $rL = 2^n$. Let us analyze the state in (629). The normalizing constant is $\sqrt{2^n L} = L\sqrt{r}$. When $\ell r/2^n = j$ for some integer j,

$$2^{(2\pi i)\ell k r/2^n} = 1, \tag{631}$$

and the amplitude for ℓ, up to a phase, is $1/\sqrt{r}$. When $\ell r/2^n$ is not an integer, then the phase is zero, as can be verified by looking at the numerator of the probability in (630). Therefore, the possible measurements with nonzero probability are of the form $j2^n/r$ for an integer j. Since $2^n/r = L$, the possible measurements for ℓ are jL. The maximum possible value for ℓ is $2^n - 1$, hence

$$jL \leq 2^n - 1 < 2^n = rL. \tag{632}$$

This means $j \leq (r-1)$ and the possible values of j are $0, 1, 2, \ldots, (r-1)$. Hence, the possible measurements for ℓ are

$$\ell = 0, L, 2L, 3L, \ldots, (r-1)L, \tag{633}$$

where $L = 2^n/r$. Each of these measurements has equal probability $1/r$. The measurement gives us an integer multiple of L, that is, it gives us some information about L. Knowing L, we can compute $r = 2^n/L$, so the measurement gives us some information about r. The game plan is to make many measurements, say T measurements. Each measurement gives a little information about r, and presumably after some detective work, we can infer r with probability close to 1 as T gets large.

Let's put this plan into action. Run the circuit in (627) independently T times and each time measure the top n qubits, producing measurements

$$\ell_1, \ldots, \ell_T, \tag{634}$$

where each ℓ_i is a multiple of L, $\ell_i = d_i L$ with $d_i \in [0, r-1]$. The gcd of the ℓ_i is

$$D = \gcd(\ell_1, \ldots, \ell_T) = L \gcd(d_1, \ldots, d_T), \tag{635}$$

which is a multiple of L. Let us write $D = dL$, where $d \geq 1$ is the gcd of the d_i,

$$d = \gcd(d_1, \ldots, d_T). \tag{636}$$

If all the $\ell_i = 0$, we will set $d = \infty$. Since $r = 2^n/L$ we have that $r = d2^n/D$. We observe D. Suppose that $D \leq \infty$. Let us assume that $d = 1$ and estimate r by

$$\hat{r} = 2^n/D. \tag{637}$$

If $d > 1$, then our estimate $\hat{r}$ is too small. That is okay, we can test $\hat{r}$ to see if it works by checking

$$f(0) \stackrel{?}{=} f(\hat{r}). \tag{638}$$

If the test works, then $d = 1$ and $\hat{r} = r$. If the test fails, we set $d = 2$ and test $\hat{r} = 2 \times 2^n/D$. At the expense of a constant classical overhead, we can continue testing in this way up to some value $d_0 - 1$, e.g. $d_0 = 4$ suffices. If all these $(d_0 - 1)$ tests fail, then we know $d \geq d_0$.

Our algorithm is as follows.

1: **Input:** Black Box quantum circuit U_f and Black box classical function f.
2: Classically test $f(0) \stackrel{?}{=} f(\hat{r})$ for $\hat{r} \in [1, r_0]$. Stop and return the smallest period if it is found. Otherwise continue.
3: Run the circuit in (627) T times. Each run involves one n-qubit Hadamard, one call to U_f and one n-qubit Quantum Fourier Transform. The result is T measurements

$$\ell_1, \ldots, \ell_T. \tag{639}$$

4: Let $D = \gcd(\ell_1, \ldots, \ell_T)$.
5: Test periods

$$\hat{r} = \frac{2^n}{D}, \frac{2 \times 2^n}{D}, \frac{3 \times 2^n}{D}, \ldots, \frac{(d_0 - 1) \times 2^n}{D}.$$

6: **return** The smallest $\hat{r}$ that works or NULL if none of the $\hat{r}$ work.

The algorithm fails if $d \geq d_0$. Let us compute the probability of failure. The reader may verify that

$$d \geq d_0 \text{ if and only if some integer } q \geq d_0 \text{ divides } d_1, \ldots, d_T.$$

Fix $q \geq d_0$. By independence of the d_i, the probability that q divides each of $d_1, \ldots, d_T$ is

$$\mathbb{P}[q \text{ divides } d_1, \ldots, d_T] = \prod_{i=1}^{T} \mathbb{P}[d_i \in \{0, q, 2q, \ldots, \kappa q\}], \tag{640}$$

where κ is the largest integer such that $\kappa q \leq (r-1)$. That means $\kappa = \lfloor (r-1)/q \rfloor$ and there are $\kappa + 1$ choices for d_i. Since d_i are randomly picked uniformly from $[0, r-1]$,

$$\begin{aligned}
\mathbb{P}[q \text{ divides } d_1, \ldots, d_T] &= \left(\frac{\kappa+1}{r}\right)^T \\
&\leq \left(\frac{1 + (r-1)/q}{r}\right)^T \\
&\leq \left(\frac{1}{q} + \frac{1}{r}\right)^T \\
&\leq \left(\frac{1}{d_0} + \frac{1}{r_0}\right)^T. \\
&\leq 2^{-T},
\end{aligned} \tag{641}$$

where the last step follows by choosing $d_0, r_0 \geq 4$. Since $d_0 \leq q \leq (r-1)$, there are at most r possible choices for q. By a union bound,

$$\mathbb{P}[\text{some } q \in [d_0, r-1] \text{ divides } d_1, \ldots, d_T] \leq r 2^{-T} \leq 2^{m-T}, \tag{642}$$

where the last step follows because $r \leq 2^m - 1$. by choosing $T = 2m$, the probability of failure is at most 2^{-m}. In a factoring application, for example to factorize the public key in RSA, $m \sim 1500$, so the probability of failure is less than 2^{-1500}. We can live with that.

Let us summarize the runtime, assuming $T = 2m$. There are $r_0 + Td_0 \in O(m)$ classical function evaluations. We run the circuit in (627) T times so the total quantum computation is

$$O(m) \text{ } n\text{-qubit Hadamards, } O(m) \text{ calls to } \mathrm{U}_f \text{ and } O(m) \text{ } n\text{-qubit QFTs.} \tag{643}$$

There is one more classical step which computes the gcd of $O(m)$ n-bit numbers, which is an $O(mn^3)$ classical computation, which is in $O(m^4)$ assuming $n \in O(m)$, which for the factoring application is polynomial in $\log N$. What remains to be done?

1. The simple case $2^n = rL$ gives all the intuition but we can't make this assumption in general. More generally, $2^n = rL + \rho$. The fact that a whole number of periods does not neatly fall into 2^n produces some aliasing which produces small amplitudes to measure states other than the ones in (633). This is okay, because the states in (633) are vastly more probable when $\rho \ll 2^n$ and the hangover piece is relatively speaking small. This requires us to choose n large enough and $n = 2m$ suffices. So we basically run the algorithm as is and there is a small additional failure probability due to the aliasing. Failure can be detected because any estimated period can be tested, in which event the entire algorithm is run again. A logarithmic number of repeats suffices to boost the success probability arbitrarily high. So, all the intuition basically holds, but the math gets a bit hairy. Nevertheless, we can do it.
2. The period estimation circuit requires the application of the QFT, which as you recall uses phase gates. The phases required are of the form (integer)$/2^n$ and with $n \sim 3000$, this requires very high precision phase gates, bordering on infeasible. Luckily, the algorithm is robust to small errors in the phase gates, in the sense that it does not deteriorate the probability of failure drastically. There is an added benefit. This robustness to phase errors allows us to use a linear number of gates in the construction of the QFT instead of the quadratic number required for the exact QFT, making the quantum computation significantly more efficient and feasible.
3. We need to efficiently construct the quatnum black box circuit for the the factoring application where $f(x) = a^x \pmod N$.
4. It turns out that period finding is a special case of phase estimation, so for intellectual completeness it behoves us to show how phase estimation can be used for period finding.

26.4 Application to Factoring

We wish to find the period r of $f(x) = a^x \pmod N$, where $\gcd(a, N) = 1$ and N is an m bit integer, $m = 1 + \lfloor \log_2 N \rfloor$. Let us consider the case $N = pq$, a product of two primes. [11]

26.4.1 Quantum Black Box Circuit for $f(x) = a^x \pmod N$

26.5 General Case: $2^n = \alpha r + \rho$, where $\alpha, \rho \in \mathbb{N}$ and $0 < \rho < r$

Let us recap the situation when the the number of possible measurements is not a multiple of the period. After measurement of the bottom m qubits, the top n qubits are in a superposition of L pure states (624), repeated here:

$$|x_0\rangle, |x_0 + r\rangle, |x_0 + 2r\rangle, |x_0 + 3r\rangle, \ldots, |x_0 + (L-1)r\rangle. \tag{644}$$

[11] The number of possible a is $(p-1)(q-1)$, the order of the group $G_N = \{x | 0 \le x \le N-1, \gcd(x, N) = 1\}$. Since $a, a^2, \ldots, a^r$ is a subgroup of G_N with order r, it follows that r divides $(p-1)(q-1)$ by Lagrange's Theorem.

Depending on x_0, $L = \alpha$ or $L = \alpha + 1$. After the QFT on the top n qubits, their state becomes

$$\sum_{\ell=0}^{2^n-1} a(\ell)|\ell\rangle, \tag{645}$$

where, up to a phase,

$$a(\ell) = \frac{1}{\sqrt{2^n L}} \sum_{k=0}^{L-1} e^{(2\pi i)\ell k r/2^n}. \tag{646}$$

The reader can verify by computing $|a(\ell)|^2$ that

$$\mathbb{P}[\ell] = \begin{cases} L/2^n & \ell r/2^n \text{ is an integer;} \\ \dfrac{1}{2^n L} \dfrac{\sin^2(\pi \ell r L/2^n)}{\sin^2(\pi \ell r/2^n)} & \text{otherwise.} \end{cases} \tag{647}$$

We now measure ℓ according to the probabilities in (647) and need to perform some detective work to infer r. This detective work requires some number theory. Recall that in the ideal case when $2^n = rL$, the measured value ℓ is an integer multiple of $2^n/r$ (see (633)). In the general case, let us assume that ℓ is close to some integer multiple of $2^n/r$ and write

$$\ell = k\frac{2^n}{r} + \delta_k. \tag{648}$$

Naturally, we can choose k to make δ_k as small as possible. Then,

$$\left| \frac{\ell}{2^n} - \frac{k}{r} \right| = \frac{|\delta_k|}{2^n}. \tag{649}$$

That is, when δ_k is small, $\ell/2^n$ is a good estimate for k/r. The plan is to use a rational approximation of $\ell/2^n$ to extract k/r and hence r. The question is, when can this be done. Recall that $r < N = 2^m$, where m is the number of qubits in the lower output register that computes U_f. The next theorem shows that if the approximation error is small enough, then the rationial approximant achieving that error is essentially unique.

Theorem 26.1. Let $x \in (0, 1)$ be approximated by a rational number a/b satisfying $b \leq N$ and

$$\left| x - \frac{a}{b} \right| < \frac{1}{2N^2}. \tag{650}$$

Then, up to a constant factor, a and b are unique. Specifically, if another rational number a'/b' with denominator $b' \leq N$ has an approximation error less than $1/2N^2$, then $a'/b' = a/b$.

Proof. Define the integer $z = a'b - b'a$. Then,

$$\frac{1}{2N^2} \;>\; \left| x - \frac{a'}{b'} \right| \;=\; \left| x - \frac{a}{b} + \frac{a}{b} - \frac{a'}{b'} \right| \;\geq\; \left| \frac{a}{b} - \frac{a'}{b'} \right| - \left| x - \frac{a}{b} \right| \;\geq\; \frac{|z|}{bb'} - \frac{1}{2N^2} \;\geq\; \frac{2\,|z| - 1}{2N^2}. \tag{651}$$

The last step uses $b, b' \leq N$. Since z is an integer, z must be 0. ∎

Theorem 26.1 is useful for estimating r. Suppose that for some k,

$$\left|\frac{\ell}{2^n} - \frac{k}{r}\right| \leq \frac{1}{2N^2}. \tag{652}$$

Let a/b be the rational with smallest denominator (reduced to lowest form) that approximates $\ell/2^n$ to within $1/2N^2$. So $b \leq r < N$ and $\gcd(a,b) = 1$. By Theorem 26.1, $a/b = k/r$, or

$$ar = bk. \tag{653}$$

Since b divides the RHS, this means $b|ar$ and since $\gcd(a,b) = 1$, by Euclid's Lemma, $b|r$. That is, we have found a divisor of r. Our plan is coming together. Two remaining questions are:

1. How likely is it that $\ell/2^n$ is close enough to an integer multiple of $1/r$. That is, what is

$$\mathbb{P}\left[\min_{k\in[1,r-1]}\left|\frac{\ell}{2^n} - \frac{k}{r}\right| \leq \frac{1}{2N^2}\right]? \tag{654}$$

2. To get b, a divisor of r, we need to find the appropriate rational approximation a/b of $\ell/2^n$. How? If we search all rational approximations with denominator less than N until we find one, the runtime would be quadratic in N. We need $O(m)$ runtime, where $m = \log_2(N)$.

The answers are technical. The probability in (654) can be made close to 1 by choosing n, the number of qubits in the input registers to be large enough. It suffices to choose $n = 2m + q$ for some small constant q. To efficiently compute an optimal rational approximation with smallest denominator, we need to appeal to the theory of continued fractions. This is a beautiful theory that is not at the forefront of application today, but does come in to save us here. Alas, you never know when and where pure mathematics will save the day, so learn it all. We will discuss these technicalities shortly.

For the moment, assume that with high probability we identify a divisor d_1 of r. Let us assume we run this process T times, so we obtain divisors $d_1, \ldots, d_T$. Their least common multiple $\hat{r} = \mathrm{lcm}(d_1, \ldots, d_T)$ is also a divisor of r. Since the d_i are random (each d_i comes from a random measurement ℓ on each run), with high probability, $\hat{r} = r$. So we can test $\hat{r}, 2\hat{r}, 3\hat{r}, 4\hat{r}, \ldots$, for a small number of multiples of $\hat{r}$. If all these tests fail, the entire process failed and we repeat it. Since the probability of success is high, only a few repeats of the entire process will yield the period. Let us now fill in the technical details.

26.5.1 Probability ℓ Yields a Divisor of r

Let us consider the $r-1$ integer multiples of $2^n/r$ which are less than 2^n, $y_k = k \times 2^n/r$ for $r \in [1, r-1]$:

$$y_1, y_2, y_3, \ldots y_{r-1} = \frac{2^n}{r},\ 2\times\frac{2^n}{r},\ 3\times\frac{2^n}{r}, \cdots,\ (r-1)\times\frac{2^n}{r}. \tag{655}$$

Let $u_k = \lfloor y_k \rfloor$ and $v_k = u_k + 1$. Note, u_k, v_k are possible values of ℓ that could be measured. Also,

$$\begin{aligned} u_k &= y_k - \delta_k, \\ v_k &= y_k + (1-\delta_k), \end{aligned} \tag{656}$$

where $0 \le \delta_k < 1$. Let $z \ge 0$ be an integer and $\ell = u_k - z$. Then $\ell/2^n = k/r - (\delta_k + z)/2^n$ and

$$\left| \frac{\ell}{2^n} - \frac{k}{r} \right| = \frac{\delta_k + z}{2^n} \le \frac{1+z}{2^n} \tag{657}$$

Similarly, if $\ell = v_k + z$

$$\left| \frac{\ell}{2^n} - \frac{k}{r} \right| = \frac{1 - \delta_k + z}{2^n} \le \frac{1+z}{2^n}. \tag{658}$$

Let us pick $n = 2m + q + 1$. Then $2^n = 2^{q+1}2^{2m} = 2^{q+1}N^2$. If $z \le 2^q - 1$, then $(1+z)/2^n \le 1/2N^2$. Further, if $\ell = u_k - z$ or $\ell = v_k + z$ for any $k \in [1, r-1]$ and any $z \in [0, 2^q - 1]$, then $\ell/2^n$ satisfies (652), and indeed these are the only choices of ℓ that satisfy (652). We can now compute the probability in (654),

$$\mathbb{P}\left[\min_{k \in [1, r-1]} \left| \frac{\ell}{2^n} - \frac{k}{r} \right| \le \frac{1}{2N^2} \right] = \sum_{z=0}^{2^q - 1} \sum_{k=1}^{r-1} (\mathbb{P}[\ell = u_k - z] + \mathbb{P}[\ell = v_k + z]). \tag{659}$$

This is the probability that a rational approximant to $\ell/2^n$ in lowest terms which achieves an approximation error at most $1/2N^2$ gives a divisor of r. Since y_k is an integer multiple of $2^n/r$ and $u_k - z = y_k - (\delta_k + z)$, using (647) we have that

$$\begin{aligned} \mathbb{P}[\ell = u_k - z] &= \frac{1}{2^n L} \frac{\sin^2(\pi(u_k - z) r L / 2^n)}{\sin^2(\pi(u_k - z) r / 2^n)} \\ &= \frac{1}{2^n L} \frac{\sin^2(\pi(y_k - (\delta_k + z)) r L / 2^n)}{\sin^2(\pi(y_k - (\delta_k + z)) r / 2^n)} \\ &= \frac{1}{2^n L} \frac{\sin^2(\pi(\delta_k + z) r L / 2^n)}{\sin^2(\pi(\delta_k + z) r / 2^n)}. \end{aligned} \tag{660}$$

Similarly,

$$\mathbb{P}[\ell = v_k + z] = \frac{1}{2^n L} \frac{\sin^2(\pi(1 - \delta_k + z) r L / 2^n)}{\sin^2(\pi(1 - \delta_k + z) r / 2^n)}. \tag{661}$$

Therefore the probability of success in (659) becomes

$$\mathbb{P}[\text{success}] = \frac{1}{2^n L} \sum_{k=1}^{r-1} \sum_{z=0}^{2^q - 1} \frac{\sin^2(\pi(z + \delta_k) r L / 2^n)}{\sin^2(\pi(z + \delta_k) r / 2^n)} + \frac{\sin^2(\pi(z + 1 - \delta_k) r L / 2^n)}{\sin^2(\pi(z + 1 - \delta_k) r / 2^n)}. \tag{662}$$

Note that L is either α or $\alpha + 1$. The inner sum over z is minimized when $\delta_k = 1/2$. Thus,

$$\mathbb{P}[\text{success}] \ge \frac{2(r-1)}{2^n L} \sum_{z=0}^{2^q - 1} \frac{\sin^2(\pi(z + \frac{1}{2}) r L / 2^n)}{\sin^2(\pi(z + \frac{1}{2}) r / 2^n)}. \tag{663}$$

The analysis of the sum is technically challenging, but we can approximate it. Since $rL \approx 2^n$, the numerator in the sum is approximately 1. Let us assume that $zr \ll 2^n$, in which case the argument

of the sine in the denominator is small and so,

$$\begin{aligned}\mathbb{P}[\text{success}] &\geq \frac{2(r-1)}{2^n L}\sum_{z=0}^{2^q-1}\frac{\sin^2(\pi(z+\frac{1}{2})rL/2^n)}{\sin^2(\pi(z+\frac{1}{2})r/2^n)} \\ &\approx \frac{2(r-1)}{2^n L}\sum_{z=0}^{2^q-1}\frac{1}{(\pi(z+\frac{1}{2})r/2^n)^2} \\ &= \frac{r-1}{r}\times\frac{2^n}{rL}\times\frac{8}{\pi^2}\sum_{z=0}^{2^q-1}\frac{1}{(1+2z)^2}. \end{aligned} \tag{664}$$

Using, again, $rL \approx 2^n$ and the fact that the sum converges rapidly to $\pi^2/8$ even for small q, e.g. $q = 5$, we have that

$$\mathbb{P}[\text{success}] \approx \frac{r-1}{r}. \tag{665}$$

Since $r \geq r_0$ and r_0 can be made large using a small constant amount of classical overhead (testing periods up to r_0), the probability of success can be made arbitrarily close to 1.

Example 26.2. As an example we consider $r = 127$. It is an instructive exercise for the reader to write a program to evaluate the true probability of success in (662) as a function of q, and compare that with the lower bound in (663) as well as the approximations in (664) and (665).

26.5.2 Rational Approximation Via Continued Fractions

We now discuss finding a and b so that a/b is a rational approximation to $\ell/2^n$ to within $1/2N^2$. The resulting b is a divisor of r. We want the smallest b and a corresponding a such that

$$\left|\frac{\ell}{2^n} - \frac{a}{b}\right| \leq \frac{1}{2N^2}. \tag{666}$$

We use the theory of simple continued fractions. Let us begin with some examples. Let $x \in (0,1)$ be a real number, for example $x = \pi - 3$,

$$x = 0.14159265358979323846264338327950288419 71\cdots \tag{667}$$

We iteratively use continued fractions to get rational estimates for x. Let $x_1 = x$, $0 < x_1 < 1$. Write $1/x_1 = \lfloor 1/x_1 \rfloor + x_2$, where $0 \leq x_2 < 1$. Let $a_1 = \lfloor 1/x_1 \rfloor$, then

$$x_1 = \frac{1}{a_1 + x_2}, \tag{668}$$

where $a_1 \geq 1$ is an integer. For $x = \pi - 3$, $a_1 = 7$. Let r_1 be our first rational approximation to x_1. To get r_1, ignore the small term x_2 giving

$$r_1 = \frac{1}{a_1} = \frac{1}{7}, \tag{669}$$

which corresponds to the famous approximation $\pi \approx 22/7$. The important fact is that $\pi - 3$ has no better rational approximation with a denominator at most 7. To see this, note that $x_1 < 1/a_1$, so any better rational approximation β/α has to be smaller than $1/a_1$. That is, $\beta/\alpha < 1/a_1$ or

$$\beta < \frac{\alpha}{a_1}. \tag{670}$$

Since $\alpha \leq a_1$, it follows that $\beta < 1$ which is not possible. We can now apply the same logic to x_2. Write $1/x_2 = \lfloor 1/x_2 \rfloor + x_3$, where $0 \leq x_3 < 1$. With $a_2 = \lfloor 1/x_2 \rfloor$, $x_2 = 1/(a_2 + x_3)$ and

$$x_1 = \frac{1}{a_1 + x_2} = \frac{1}{a_1 + \dfrac{1}{a_2 + x_3}}. \tag{671}$$

Here, $x_2 = 1/(\pi - 3) - \lfloor 1/(\pi - 3) \rfloor$ and the reader can verify that $a_2 = 15$. Our next rational approximation given by $[a_1, a_2]$ is

$$r_2 = \frac{1}{a_1 + \dfrac{1}{a_2}} = \frac{15}{106}. \tag{672}$$

This corresponds to approximation $\pi \approx 333/106$. The next rational approximant comes from writing $1/x_3 = a_3 + x_4$, where $a_3 = \lfloor 1/x_3 \rfloor = 1$. This gives

$$r_3 = \frac{1}{a_1 + \dfrac{1}{a_2 + \dfrac{1}{a_3}}} = \frac{1}{7 + \dfrac{1}{15 + \dfrac{1}{1}}} = \frac{16}{113}. \tag{673}$$

This gives the famous approximation $\pi \approx 355/113$. The algorithm for constructing continued fraction approximations should now be clear. Define two sequences recursively,

$$\begin{array}{cccccc} x_1, & x_2, & x_3, & x_4, & x_5, & \dots \\ a_1, & a_2, & a_3, & a_4, & a_5, & \dots \end{array} \tag{674}$$

where $x_1 = x$ and for $i > 1$

$$x_{i+1} = 1/x_i - \lfloor 1/x_i \rfloor, \tag{675}$$

and for $i \geq 1$

$$a_i = \lfloor 1/x_i \rfloor. \tag{676}$$

Note that if at some point $x_i = 0$, then the process stops, with $a_i = \infty$. From $a_1, \dots, a_k$, we then obtain the rational approximation r_k, called the kth convergent,

$$r_k = \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\cdots + \cfrac{1}{a_{k-1} + \frac{1}{a_k}}}}}}. \tag{677}$$

Note that by definition of x_k in the continued fraction algorithm,

$$x = \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\cdots + \cfrac{1}{a_{k-1} + x_k}}}}} = \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\cdots + \cfrac{1}{a_{k-1} + \frac{1}{\alpha_k}}}}}}, \tag{678}$$

where $\alpha_k = 1/x_k$ and a_k is the integer part of α_k.

26.5.3 Computing The Rational Approximation

The simple continued fraction defined by n positive numbers $[\alpha_1, \alpha_2, \ldots, \alpha_n]$ is

$$[\alpha_1, \alpha_2, \ldots, \alpha_n] = \cfrac{1}{\alpha_1 + \cfrac{1}{\alpha_2 + \cfrac{1}{\alpha_3 + \cfrac{1}{\cdots + \cfrac{1}{\alpha_{n-1} + \frac{1}{\alpha_n}}}}}}. \tag{679}$$

In (677), each α_i is the positive integer a_i. The expression on the right is some ratio p_n/q_n. We will need to efficiently compute this ratio for all the convergents, $[\alpha_1], [\alpha_1, \alpha_2], \ldots, [\alpha_1, \alpha_2, \ldots, \alpha_n]$. That is, for $k = 1, \ldots, n$, we need to find p_k and q_k such that

$$[\alpha_1, \alpha_2, \ldots, \alpha_k] = \frac{p_k}{q_k}. \tag{680}$$

Luckily there is a recursive algorithm that computes p_k and q_k. Let us define $p_0 = 0, q_0 = 1$. For a single positive number, $[\alpha_1] = 1/\alpha_1$ and we can choose

$$p_1 = 1 \text{ and } q_1 = \alpha_1. \tag{681}$$

Also,

$$[\alpha_1, \alpha_2] = \frac{\alpha_2}{\alpha_1\alpha_2 + 1}, \tag{682}$$

so

$$p_2 = \alpha_2 \text{ and } q_2 = \alpha_1\alpha_2 + 1. \tag{683}$$

We show by induction that for any choice of $\alpha_1, \alpha_2, \ldots, \alpha_k$, with the initial conditions

$$\begin{array}{rclcrcl} p_0 & = & 0 & & p_1 & = & 1 \\ q_0 & = & 1 & & q_1 & = & \alpha_1 \end{array}, \tag{684}$$

for all $k \geq 2$,

$$\begin{aligned} p_k &= \alpha_k p_{k-1} + p_{k-2} \\ q_k &= \alpha_k q_{k-1} + q_{k-2}. \end{aligned} \tag{685}$$

That is,

$$[\alpha_1, \alpha_2, \ldots, \alpha_{k-1}, \alpha_k] = \frac{p_k}{q_k} = \frac{\alpha_k p_{k-1} + p_{k-2}}{\alpha_k q_{k-1} + q_{k-2}}. \tag{686}$$

The base case can be verified from (683). For the induction, assume (685) for $k = 2, \ldots, K$. Then,

$$\begin{aligned} [\alpha_1, \alpha_2, \ldots, \alpha_{K-1}, \alpha_K, \alpha_{K+1}] &= [\alpha_1, \alpha_2, \ldots, \alpha_{K-1}, \alpha_K + \frac{1}{\alpha_{K+1}}] \\ &\overset{(*)}{=} \frac{\left(\alpha_K + \frac{1}{\alpha_{K+1}}\right) p_{K-1} + p_{K-2}}{\left(\alpha_K + \frac{1}{\alpha_{K+1}}\right) q_{K-1} + q_{K-2}}. \\ &= \frac{\alpha_{K+1}(\alpha_K p_{K-1} + p_{K-2}) + p_{K-1}}{\alpha_{K+1}(\alpha_K q_{K-1} + q_{K-2}) + q_{K-1}} \qquad (687) \\ &\overset{(*)}{=} \frac{\alpha_{K+1} p_K + p_{K-1}}{\alpha_{K+1} q_K + q_{K-1}}. \qquad (688) \end{aligned}$$

In $(*)$, we used the induction hypothesis. The last step proves (685) holds for $K+1$, concluding the induction. For the simple continued fraction in (677), the a_i are all positive integers. Applying (684) and (685) to $[a_1, a_2, a_3, \ldots]$ you can show by induction that p_k and q_k are positive integers. Further, all the convergents give rational approximations

$$r_k = \frac{p_k}{q_k} \tag{689}$$

that are fractions in their lowest terms. That is,

$$\gcd(p_k, q_k) = 1. \tag{690}$$

To prove this, let us compute $p_k q_{k-1} - p_{k-1} q_k$. You can verify from (683) and (684) that

$$\begin{aligned} p_1 q_0 - q_1 p_0 &= 1 \\ p_2 q_1 - q_2 p_1 &= -1. \end{aligned} \tag{691}$$

In general, as we will show by induction,

$$p_k q_{k-1} - p_{k-1} q_k = (-1)^{k+1}. \tag{692}$$

In the inductive step, using the recursions for p_{k+1} and q_{k+1} from (685),

$$\begin{aligned} p_{k+1} q_k - p_k q_{k+1} &= (a_{k+1} p_k + p_{k-1}) q_k - p_k (a_{k+1} q_k + q_{k-1}) \\ &= -(p_k q_{k-1} - p_{k-1} q_k) \\ &= (-1)^{k+2}. \end{aligned} \tag{693}$$

The last step uses the inductive hypothesis. Suppose $D > 1$ divides p_k and q_k. Then D divides the LHS in (692), so it divides the RHS, which is ± 1. This can't be, hence $\gcd(p_k, q_k) = 1$.

Another important consequence of (692) is that it gives an estimate of the accuracy for each convergent. Indeed, dividing (692) by $q_k q_{k-1}$ gives

$$\left| \frac{p_k}{q_k} - \frac{p_{k-1}}{q_{k-1}} \right| = \frac{1}{q_k q_{k-1}}. \tag{694}$$

That is, $|r_k - r_{k-1}| = 1/q_k q_{k-1}$. The reader may show by induction that $q_k \geq k$ and therefore for $k \geq 2$, $|r_k - r_{k-1}| \leq 1/k(k-1)$.

26.5.4 Optimality of the Continued Fraction Convergents

Let us begin with the motonicity of the convergents $r_1, r_2, r_3, \ldots$. Recall that

$$x = x_1 = \frac{1}{a_1 + x_2}. \tag{695}$$

Ignoring the x_2 term and setting $r_1 = 1/a_1$, we have that $x \leq r_1$. This is true of the first convergent for any $0 < x < 1$. Since $1/a_2$ is the first convergent for x_2, it follows that $x_2 < 1/a_2$. Therefore,

$$r_2 = \frac{1}{a_1 + \dfrac{1}{a_2}} \leq \frac{1}{a_1 + x_2} = x. \tag{696}$$

We have therefore proved that

$$r_2 \leq x \leq r_1. \tag{697}$$

Indeed, much more is true. The even convergents are increasing and all at most x. The odd convergents are decreasing and all at least x. For $k \geq 1$,

$$r_{2k-2} < r_{2k} \leq x \qquad \text{and} \qquad x \leq r_{2k+1} < r_{2k-1}. \tag{698}$$

The proof is by induction. The reader may verify the base cases for small k, e.g. $k = 1, 2$. For the induction, assume the above for $k \leq K$ and consider and consider $K + 1$.

$$\begin{aligned} r_{2K} = [a_1, a_2, \ldots, a_{2K}] &= \frac{1}{a_1 + [a_2, \ldots, a_{2K}]} \\ r_{2K+2} = [a_1, a_2, \ldots, a_{2K+2}] &= \frac{1}{a_1 + [a_2, \ldots, a_{2K+2}]} \end{aligned} \tag{699}$$

Note that $[a_2, \ldots, a_{2K}]$ is the $2K-1$ convergent for x_2 and $[a_2, \ldots, a_{2K+2}]$ is the $2K+1$ convergent for x_2. So, by the induction hypothesis,

$$x_2 \leq [a_2, \ldots, a_{2K+2}] < [a_2, \ldots, a_{2K}]. \tag{700}$$

Since $x = 1/(a_1 + x_2)$, it follows that

$$r_{2K} < r_{2K+2} \leq x. \tag{701}$$

For the odd convergent r_{2K+3}, we have

$$\begin{aligned} r_{2K+1} = [a_1, a_2, \ldots, a_{2K+1}] &= \frac{1}{a_1 + [a_2, \ldots, a_{2K+1}]} \\ r_{2K+3} = [a_1, a_2, \ldots, a_{2K+3}] &= \frac{1}{a_1 + [a_2, \ldots, a_{2K+3}]} \end{aligned} \tag{702}$$

Now, $[a_2, \ldots, a_{2K+1}]$ is the $2K$ convergent for x_2 and $[a_2, \ldots, a_{2K+3}]$ is the $2K+2$ convergent for x_2. So, by (701)

$$[a_2, \ldots, a_{2K+1}] < [a_2, \ldots, a_{2K+3}] \leq x_2, \tag{703}$$

and now it follows that

$$x \leq r_{2K+3} < r_{2K+1}. \tag{704}$$

This proves monotonicity for $K + 1$ and concludes the induction. Therefore, for any $k \geq 1$,

$$r_2 < r_4 < r_6 < \cdots < r_{2k} \leq x \leq r_{2k-1} < \cdots < r_5 < r_3 < r_1. \tag{705}$$

Let k be odd (a similar argument holds if k is even). Then $[r_{k-1}, r_k]$ contains x, and r_k is a rational approximant to x with denominator q_k. Let us estimate the approximation error of r_k. Define $\epsilon_k = |x - p_k/q_k|$. By definition of x_{k+1} in the continued fraction algorithm, see (678),

$$x = [a_1, a_2, \ldots, a_k, 1/x_{k+1}] = \frac{\frac{1}{x_{k+1}} p_k + p_{k-1}}{\frac{1}{x_{k+1}} q_k + q_{k-1}}. \tag{706}$$

In the last step we used (686).

$$\begin{aligned}\epsilon_k = \left| x - \frac{p_k}{q_k} \right| &= \left| \frac{\frac{1}{x_{k+1}} p_k + p_{k-1}}{\frac{1}{x_{k+1}} q_k + q_{k-1}} - \frac{p_k}{q_k} \right| \\ &= \left| \frac{p_{k-1} q_k - p_k q_{k-1}}{q_k (\frac{1}{x_{k+1}} q_k + q_{k-1})} \right| \\ &= \frac{1}{q_k (\frac{1}{x_{k+1}} q_k + q_{k-1})} \end{aligned} \tag{707}$$

In the last step we used $p_k q_{k-1} - p_{k-1} q_k = (-1)^{k+1}$. Since $a_{k+1} = \lfloor 1/x_{k+1} \rfloor \leq 1/x_{k+1}$,

$$\epsilon_k = \left| x - \frac{p_k}{q_k} \right| \leq \frac{1}{q_k (a_{k+1} q_k + q_{k-1})} = \frac{1}{q_k q_{k+1}}. \tag{708}$$

As long as $x_k \neq 0$, it should be no surprise that the approximation error strictly decreases, $\epsilon_k < \epsilon_{k-1}$. To see this, look at the denominator in (707). This denominator can be written

$$q_k (a_{k+1} q_k + q_{k-1}) + \left(\frac{1}{x_{k+1}} - a_{k+1} \right) q_k^2 = q_k q_{k+1} + x_{k+2} q_k^2, \tag{709}$$

where we have used the recursion for the q_k and by definition $x_{k+2} = 1/x_{k+1} - a_{k+1}$. Using (707) for ϵ_k and (709) for ϵ_{k-1} gives

$$\epsilon_k = \frac{1}{q_k q_{k-1} + \frac{1}{x_{k+1}} q_k^2}; \qquad \epsilon_{k-1} = \frac{1}{q_k q_{k-1} + x_{k+1} q_{k-1}^2}. \tag{710}$$

Since $x_{k+1} < 1/x_{k+1}$ and $q_{k-1} < q_k$, it follows that

$$\epsilon_k < \epsilon_{k-1}. \tag{711}$$

Indeed, we have a proved a stronger result,

$$q_k \epsilon_k = \frac{1}{q_{k-1} + \frac{1}{x_{k+1}} q_k}; \qquad q_{k-1} \epsilon_{k-1} = \frac{1}{q_k + x_{k+1} q_{k-1}}. \tag{712}$$

Since $x_{k+1} < 1$, we have that $q_k \epsilon_k < q_{k-1} \epsilon_{k-1}$, or

$$\epsilon_k < \frac{q_{k-1}}{q_k} \epsilon_{k-1}. \tag{713}$$

Iteratively applying this formula gives $\epsilon_k < (q_{k-t}/q_k) \epsilon_{k-t}$ for $t \geq 1$.

Optimality. After all this effort, we are ready for the main result, that r_k is the best rational approximation to x with denominator at most q_k. To see this, suppose some other estimator p/q is better. Since $r_{k-1} < x < r_k$ and r_{k-1} is a rational estimator with denominator $q_{k-1} < q_k$, it follows that p/q lies in (r_{k-1}, r_k). Note that $p/q \neq r_{k-1}$ because we know that ϵ_{k-1} is worse than ϵ_k.

$$\frac{p_{k-1}}{q_{k-1}} < \frac{p}{q} < \frac{p_k}{q_k}. \tag{714}$$

We have that

$$\frac{p}{q} - \frac{p_{k-1}}{q_{k-1}} = \frac{pq_{k-1} - p_{k-1}q}{qq_{k-1}} \geq \frac{1}{qq_{k-1}}. \tag{715}$$

The last step follows because the numerator is positive and an integer. Because $p/q < p_k/q_k$,

$$\frac{p}{q} - \frac{p_{k-1}}{q_{k-1}} < \frac{p_k}{q_k} - \frac{p_{k-1}}{q_{k-1}} = \frac{1}{q_k q_{k-1}}. \tag{716}$$

We have proved that $1/qq_{k-1} < 1/q_kq_{k-1}$ or $q > q_k$, a contradiction. Therefore, such a p/q does not exist and so p_k/q_k is the best approximator to x with denominator at most q_k. Note, that since p_k/q_k is in its lowest terms, it is unique.

A stronger result is true. Let q be any denominator that lies between two consecutive convergent denominators, $q_{k-1} < q < q_k$, and let p/q be a rational approximation to x. Assuming $x \neq p_k/q_k$,

$$q\left|x - \frac{p}{q}\right| > q_{k-1}\left|x - \frac{p_{k-1}}{q_{k-1}}\right| > q_k\left|x - \frac{p_k}{q_k}\right|. \tag{717}$$

The second inequality is just (713). Suppose k is odd (k even is similar). We prove

$$|qx - p| > |q_{k-1}x - p_{k-1}|. \tag{718}$$

When k is odd, $p_kq_{k-1} - p_{k-1}q_k = (-1)^{k+1} = 1$ and $p_{k-1}/q_{k-1} < x < p_k/q_k$. If $p/q \leq p_{k-1}/q_{k-1}$, then (718) follows from $q > q_{k-1}$ and

$$x - \frac{p}{q} \geq x - \frac{p_{k-1}}{q_{k-1}} \geq 0. \tag{719}$$

So, we may assume $p/q > p_{k-1}/q_{k-1}$, in which case the integer $pq_{k-1} - p_{k-1}q$ is positive,

$$pq_{k-1} - p_{k-1}q \geq 1. \tag{720}$$

The reader can verify, that p and and q are the linear combinations

$$\begin{aligned} p &= (pq_{k-1} - p_{k-1}q)p_k - (pq_k - p_kq)p_{k-1} \\ q &= (pq_{k-1} - p_{k-1}q)q_k - (pq_k - p_kq)q_{k-1}. \end{aligned} \tag{721}$$

Since $q < q_k$ and $pq_{k-1} - p_{k-1}q \geq 1$, it must be that $pq_k - p_kq \geq 1$. Using (721),

$$\begin{aligned} qx - p &= (pq_{k-1} - p_{k-1}q)(xq_k - p_k) - (pq_k - p_kq)(xq_{k-1} - p_{k-1}) \\ &= -(pq_{k-1} - p_{k-1}q)\,|xq_k - p_k| - (pq_k - p_kq)\,|xq_{k-1} - p_{k-1}|, \end{aligned} \tag{722}$$

where the last step follows because $p_{k-1}/q_{k-1} < x < p_k/q_k$. Since both $(pq_{k-1} - p_{k-1}q)$ and $(pq_k - p_kq)$ are at least 1 (positive), we have

$$\begin{aligned} |qx - p| &= (pq_{k-1} - p_{k-1}q)\,|xq_k - p_k| + (pq_k - p_kq)\,|xq_{k-1} - p_{k-1}| \\ &\geq |xq_{k-1} - p_{k-1}|. \end{aligned} \tag{723}$$

The last step follows because $pq_k - p_kq \geq 1$. When $x \neq p_k/q_k$, the inequality becomes strict because $pq_{k-1} - p_{k-1}q \geq 1$, hence the first term is positive.

A corollary of this optimality result is that if $x = a/b$ is rational, then the continued fraction must be finite. This is because once q_k exceeds b, the optimal approximator is x itself, this means the x_{k+1} will be zero. The reader can fill in the details. There are other ways to prove that then continued fraction of a rational number is finite, including directly from Euclid's GCD algorithm. We end with an application of the previous results that is relevant to our period finding setting.

Theorem 26.3. Let $x \in (0,1)$ have a rationial approximant p/q satisfying

$$\left| x - \frac{p}{q} \right| < \frac{1}{2q^2}. \tag{724}$$

Then $p/q = p_k/q_k$ for some convergent p_k/q_k of x, with $q_k \leq q$.

Proof. Suppose q is some convergent denominator, $q = q_k$. Then p_k/q_k is a better approximant than p/q, by the optimality of p_k/q_k. By Theorem 26.1 $p/q = p_k/q_k$ implying $p = p_k$.

So, suppose q lies strictly between two consecutive convergent denominators, $q_{k-1} < q < q_k$, and $p/q \neq p_{k-1}/q_{k-1}$. Using (723) and (724),

$$\left| x - \frac{p_{k-1}}{q_{k-1}} \right| \leq \frac{q}{q_{k-1}} \left| x - \frac{p}{q} \right| < \frac{1}{2qq_{k-1}}, \tag{725}$$

Therefore, by the triangle inequality,

$$\left| \frac{p}{q} - \frac{p_{k-1}}{q_{k-1}} \right| \leq \left| \frac{p}{q} - x \right| + \left| x - \frac{p_{k-1}}{q_{k-1}} \right| \leq < \frac{1}{2q^2} + \frac{1}{2qq_{k-1}} < \frac{1}{qq_{k-1}}. \tag{726}$$

The last step is because $q_{k-1} < q$. But, this is a cotradiction because

$$\left| \frac{p}{q} - \frac{p_{k-1}}{q_{k-1}} \right| = \left| \frac{pq_{k-1} - p_{k-1}q}{qq_{k-1}} \right| \geq \frac{1}{qq_{k-1}}. \tag{727}$$

The last step uses $p/q \neq p_{k-1}/q_{k-1}$, hence $pq_{k-1} - p_{k-1}q$ is a non-zero integer. ∎

26.5.5 Full Algorithm

The period finding algorithm produces a measurement ℓ from which we compute $x = \ell/2^n$. We know that with high probability a rational k/r is within $1/2N^2$ of x. Since $r < N$, by Theorem 26.3 $k/r = p_k/q_k$ for some convergent of x with $q_k \leq r < N$. We start by computing the convergents

$$r_1, r_2, r_3, \ldots, \tag{728}$$

where $r_k = p_k/q_k$. One of two things happens.

1. We get to a first $q_k < N$ for which p_k/q_k is within $1/2N^2$ of x. In this case, we output q_k as a divisor of r.

2. Case 1 never happens and we get to the first $q_k \geq N$. In this case, the sampled $\ell/2^n$ failed to be withing $1/2N^2$ of k/r. We start over and sample again.

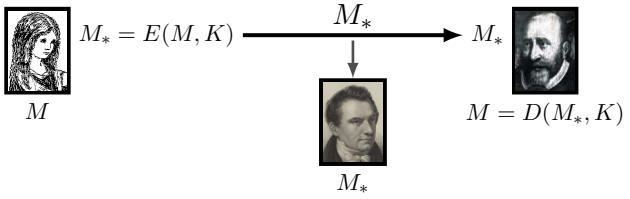
26.6 Sensitivity to Phase Errors in the QFT

26.7 Period Finding Via Phase Estimation

27 Secure Communication – Cryptography

The setup is that Alice wishes to send a message M to Bob over a public channel. For example, M could be the location of Charlie's surprise party. Since the channel is public, anyone could be eavesdropping, in particular Charlie. Can Alice use the public channel to communicate with Bob so that Bob can understand the message but no eavesdroppers can. Clearly Alice cannot send the message M, as anyone can see it. So the only alternative is for Alice to send some other message M_*, which again everyone can see. It should be possible for Bob to recover the intended message M from M_*, but noone else should be able to do so. As far as we are concerned, M is just some large prime number. The setup is as follows.

Alice must send M to Bob.
Charlie eavesdrops.
Alice sends M_* instead.
Bob and Charlie see M_*.
Bob recovers M. Charlie can't.



The original message M is called plain text and the encrypted message M_* is called cyphertext. Alice uses some side information K_E, called a key to encode M into M_*,

$$M_* = E(M, K_E). \tag{729}$$

The function $E(\cdot, \cdot)$ is the encoding function. Bob can recover the plain text M from M_* because Bob has some side information K_D, that is related to K_E, the side information that Alice used to encode M (often $K_E = K_D$, that is the side information used by Bob and Alice are the same). Bob decodes using a decoding function D,

$$M = D(M_*, K_D). \tag{730}$$

The decoding function $D(\cdot, \cdot)$ is a sort-of inverse of the encoding function $E(\cdot, \cdot)$. The side information Bob needs, K_D, depends on E, D and K_E. Assuming Alice and Bob have agreed on the cryptographic protocol, the encoding and decoding functions E and D are known to everyone, including Charlie. THe keys K_E, K_D are private. However, K_D depends on K_E. That is, the side information Bob needs to decode the message depends on the side information Alice uses to encode the message. So, at some point, K_E, K_D were created together by *Bob* and then sent K_E to Alice; or, K_E, K_D were created together by Alice and then Alice sent K_D to Bob. This sending of side information from one party to another is called *key exchange*.

Is the information secure. No. Suppose the decoding key is ℓ bits long. Here is what Charlie can do. Try all possible keys in lexicographic order,

$$\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, 0001, \ldots\}. \tag{731}$$

For each possible key K_{guess} compute decode the message M_* to get

$$M_{\text{guess}} = D(M_*, K_{\text{guess}}). \tag{732}$$

With probability essentially 1, if the guessed key does not match K_D, the guessed message will be gibberish. Assuming Charlie can recognize gibberish, Charlie continues guessing in this lexicographic

order until he gets a non-gibberish message, at which point Charlie has deciphered M. The point is that it takes Charlie at least

$$1 + 2 + 4 + \cdots + 2^{\ell-1} + 1 = 2^{\ell} \tag{733}$$

guesses and at most

$$1 + 2 + 4 + \cdots + 2^{\ell-1} + 2^{\ell} = 2^{\ell+1} - 1 \tag{734}$$

guesses. So it takes Charlie $\Theta(2^{\ell})$ time to decode the message. By this time the message is stale, if ℓ is large enough. It is also possible to thwart such a gibberish attack by Charlie by carefully inserting gibberish into the message M. The question is whether there is a faster way for Charlie to guess the message. This is one goal of cryptography. To create the functions E, D so that there is provably no way to guess the message other than by using this brute force approach. In some cases we are satisfied with no known approach to guess the message other than brute force. There exist such pairs of functions E, D, called one-way or trap-door functions. One-way because there is no efficient way to invert E without knowing the side information.

The sense in which the communication is secure is computational. It is only secure if Charlie has bounded computing capability, because it would take Charlie an inordinate amount of time to decipher. However, if the side information is ever made available to Charlie, in particular K_D, then all communication is open to Charlie. This becomes an issue because at some point a key exchange occured. The problem is the key exchange cannot be encrypted, otherwise you have a chicken and egg problem. In the good old days, this key exchange happened in a private meeting of Alice and Bob, say in the supermarket. Today, the key exchange needs to happen on a regular basis between often unknown parties like you and your online bank. Here are some examples of secure communication protocols, together with their pitfalls.

27.1 Classical Protocols

The Caesar Cipher is the earliest known systematic use of a cryptographic protocol to accomplish secure communication between military generals. In those days the channel was a rider on a horse with a message written on paper. If the rider gets captures, the enemy would know (say) your troops movements. The Caesar Cipher was an attempt to obfuscate the message by fairly simple letter substitution (reflection and translation).

$$\begin{array}{c} A\ B\ C\ D\ E\ F\ G\ H\ I\ J\ K\ L\ M\ N\ O\ P\ Q\ R\ S\ T\ U\ V\ W\ X\ Y\ Z \\ \updownarrow \\ N\ M\ L\ K\ J\ I\ H\ G\ F\ E\ D\ C\ B\ A\ Z\ Y\ X\ W\ V\ U\ T\ S\ R\ Q\ P\ 0 \end{array}$$

(735)

The brute force attack on this encoding will succeed very quickly. There are only 26 possibilities.

27.1.1 One Time Pad

The one-time pad is a simple robust secure communication protocol. Suppose the message

$$M = \begin{array}{cccccccc} 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \end{array}. \tag{736}$$

Assume that Alice and Bob have exchanged a random string of 8 bits, the key

$$K = \begin{array}{cccccccc} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}. \tag{737}$$

In this case, the side information used by Alice and Bob to encode and decode are the same, $K_E = K_D = K$. Alice computes the bitwise XOR of her message M and the key K to get

$$M_* = M \oplus K \tag{738}$$

$$\begin{array}{r|cccccccc} M & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ K & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline M_* & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{array} \tag{739}$$

Bob, who also has K decodes using bitwise XOR of the cyphertext M_* with the key K to get

$$M = M_* \oplus K \tag{740}$$

$$\begin{array}{r|cccccccc} M_* & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ K & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline M & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \end{array} \tag{741}$$

This always decodes correctly because

$$M_* \oplus K = (M \oplus K) \oplus K = M \oplus (K \oplus K) = M \oplus \mathbf{0} = M. \tag{742}$$

The encrytped message is information-theoretically secure because each bit of M_* is independently random, assuming the bits of K are independently random,

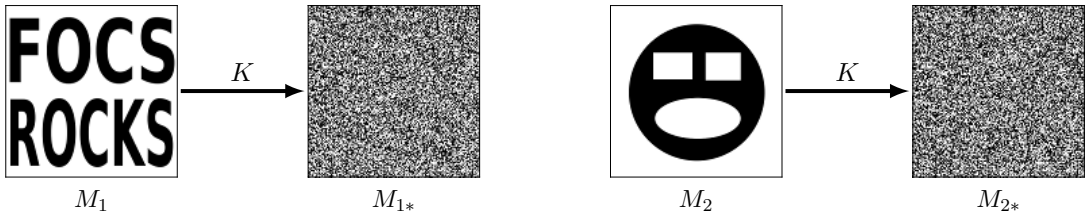
$$\mathbb{P}[M_*[i] = 1] = \mathbb{P}[M[i] \neq K[i]] = \frac{1}{2}. \tag{743}$$

The only way for Charlie to recover the message M is to try all $2^{|M_*|}$ possible keys checking for gibberish each time.

The security of the communication relies on Alice and Bob having privately shared the key K. There is a problem with the one time pad. As the name indicates, Alice can only send one message to Charlie using the key K. That is, the key cannot be reused. The reason is that if Alice sends two encrypted messages, M_{1*}, M_{2*}, then Charlie who sees both encrypted messages can learn something about the original messages M_1, M_2. In particular, Charlie can learn $M_1 \oplus M_2$. For the communication to be secure, Charlie must learn *nothing* about M_1 and M_2. Charlie learns $M_1 \oplus M_2$ by computing $M_{1*} \oplus M_{2*}$,

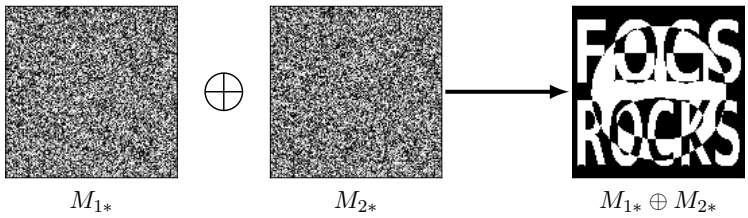
$$\begin{aligned} M_{1*} \oplus M_{2*} &= (M_1 \oplus K) \oplus (M_2 \oplus K) \\ &= M_1 \oplus M_2 \oplus K \oplus K \\ &= M_1 \oplus M_2 \oplus \mathbf{0} \\ &= M_1 \oplus M_2. \end{aligned} \tag{744}$$

What does Charlie gain by knowing $M_1 \oplus M_2$? Here is a striking visual example to illustrate this attack, called a prior information attack[12]. Suppose M_1 and M_2 are the images and they are encoded using the same one time pad K,



(745)

As you can see, each encoded image is just random noise. On their own, the encoded messages M_{1*} and M_{2*} are secure. But if Charlie has both M_{1*} and M_{2*}, here is what happens,



To the eye, having the "prior" information $M_1 \oplus M_2$ virtually reveals the individual messages M_1, M_2.

27.1.2 RSA

Rivest, Shamir and Adleman made a breakthrough in 1977 developing the RSA secure communication protocol which addressed the need to share the key. Here is the high-level idea without any details. Bob randomly picks two huge, comparably sized primes p and q and computes $n = pq$. bob also computes a public encoding key e and a decoding key d from p, q. Bob publishes to the world, including Alice and Charlie $K_E = (n, e)$. Alice encodes using

$$M_* \equiv M^e \pmod{n}. \tag{746}$$

Bob decodes using

$$M \equiv M_*^d \pmod{n}. \tag{747}$$

e and d are chosen carefully to ensure that Bob always decodes correctly. The fact that Charlie has K_E means that Charlie can encode and send messages to Bob. So what? The important thing is that Charlie should not be able to recover the decoding key d from (n, e). Bob has to keep d secret. Hence, e is called the public key and d the private key. RSA has remarkably arranged for the exchange of the key K_E over a public channel.

There are two problems with RSA. The first is computational. Encoding a message requires a heavy modular exponentionation calculation, which becomes an issue when M, n, e are all huge, which is the case in practice, and this only gets amplified by the need to send many messages. The second main problem is that if Charlie get hold of the original primes p, q, then Charlie can compute

[12]You can think of $M_1 \oplus M_2$ being given prior information and you are now trying to guess M_1 and M_2.

d, given (n, e). This means Charlie can decode just as easily as Bob can. Charlie can find p, q by factoring n. We don't know if factoring can be done efficiently on a classical computer. We do know it can be done efficiently on a Quantum computer (see Sections **??** and **??**). In conclusion, we don't really know if RSA is secure on a classical computer. We know it is not secure on a quantum computer. We don't have quantum computers yet, so for the moment we are safe, or are we?

The upshot of all this discussion is that RSA is inefficient and may or may not be secure. We would rather use more trusted encoding and decoding functions as they are provably secure and efficient, but these methods need key exchange. Hence, the professionals use a hybrid solution. RSA to share the keys (K_E, K_D) once and then encoding and decoding functions based on (K_E, K_D). The only weak-spot is now in the key sharing, which is just a single communication.

27.2 Quantum Protocols

Fundamentally, we see that the secure communication is solved down to one key step, namely key exchange. Quantum computing gives a quantum mechanically protected means for key exchange. Once the keys are exchanged, we can use them in classical encoding and decoding protocols. Specifically, we will see that by sending qubits on a *public* channel, it is possible to privately share a key of random bits. At a high-level let us see how quantum computing helps.

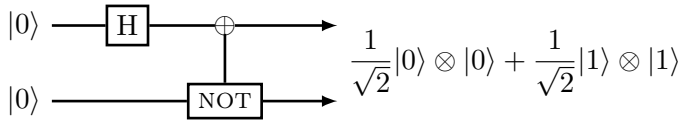
When Alice sends qubits to Bob, what can Charlie do? Well, he might try to copy them and store them, for later analysis. This is prevented by the no-cloning theorem. Charlie might measure the qubits, which essentially destroys them because their state collapses. This looks like a problem for key exchange, because while Charlie can't discover the qubits, he can certainly sabotage the key-exchange process. This is tantamount to Charlie preventing communication between Alice and Bob. This is always possible. Charlie can simply cut the wire. We assume that Charlie does not want to prevent Alice and Bob from communicating. Charlie just wants in on the conversation. So it will turn out that Charlies inability to measure the state without destroying it will turn into an opportunity for us. This is because, in addition to key-exchange, there are other challenges to secure communication. Here are three important problems:

- Key exchange.
- Intrusion detection. Can one detect if there is an eavesdropper like Charlie?
- Robustness to impersonation. Charlie intercepts the communication from Alice to Bob, and sends his own message to Bob, impersonating Alice. This is called a man-in-the-middle attack.

We will see that quantum protocols can help with all these challenges. Before we proceed, it is worth mentioning a quantum-mechanical solution to key exchange that is based on entanglement. The basic idea is that Alice creates a pair of entangled cubits in the state

$$\frac{1}{\sqrt{2}}|0\rangle \otimes |0\rangle + \frac{1}{\sqrt{2}}|1\rangle \otimes |1\rangle. \tag{748}$$

This is done from the pure state $|0\rangle|0\rangle$ by applying a Hadamard to the first qubit and using it to control a NOT on the second qubit,



(749)

The reader should verify using that

$$\frac{1}{\sqrt{2}}|0\rangle \otimes |0\rangle + \frac{1}{\sqrt{2}}|1\rangle \otimes |1\rangle = \text{CNOT}_{12}(H \otimes I_2)|0\rangle \otimes |0\rangle.$$

Alice keeps one qubit and sends the other to Bob. For example, Alice could create a pair of entangled photons, sending one to Bob. When Bob gets his qubit, both Alice and Bob measures their qubit (it does not matter who measures first). By state collapse, they both have the same (random) bit. This basic idea needs further embelishing because if Charlie intercepts the qubit, measures it and sends it to Bob, all three will have the same qubit. In the next section we will see a quantum computing approach to key exchange that does not rely on such entanglement.

27.3 Bell States

We digress to mention the four possible entangled states that can be obtained using the unitary operator $\text{CNOT}_{12}(H \otimes I_2)$. Since the operator is unitary, applying the operator to any orthonormal basis produces another orthonormal basis. Applying the operator to the standard basis $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ produces the maximally entangled orthonormal basis known as the Bell basis:

$$\text{CNOT}_{12}(H \otimes I_2)|0\rangle \otimes |0\rangle = \frac{1}{\sqrt{2}}|0\rangle \otimes |0\rangle + \frac{1}{\sqrt{2}}|1\rangle \otimes |1\rangle \quad (750)$$

$$\text{CNOT}_{12}(H \otimes I_2)|0\rangle \otimes |1\rangle = \frac{1}{\sqrt{2}}|0\rangle \otimes |1\rangle + \frac{1}{\sqrt{2}}|1\rangle \otimes |0\rangle \quad (751)$$

$$\text{CNOT}_{12}(H \otimes I_2)|1\rangle \otimes |0\rangle = \frac{1}{\sqrt{2}}|0\rangle \otimes |0\rangle - \frac{1}{\sqrt{2}}|1\rangle \otimes |1\rangle \quad (752)$$

$$\text{CNOT}_{12}(H \otimes I_2)|1\rangle \otimes |1\rangle = \frac{1}{\sqrt{2}}|0\rangle \otimes |1\rangle - \frac{1}{\sqrt{2}}|1\rangle \otimes |0\rangle \quad (753)$$

in principle, any of these entangled states could be used to exchange a bit. The Bell basis is also useful in superdense coding and teleportation.

28 Quantum Key Exchange

We discuss the BB84 protocol for quantum key exchange, named after its inventors Bennett & Brassard and the year of invention 1984. There have since been improvements, but all the basic ideas are in the simple protocol. Let us first be careful to about stating what we can do. Ideally, we would like to use quantum computing to let Alice securely send a specific message to Bob, for example the decoding key K_D,

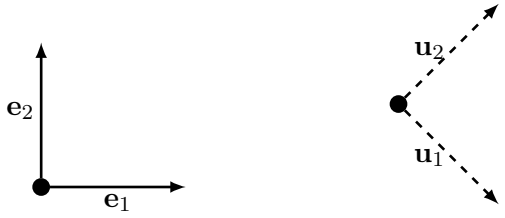
$$\text{Alice} \xrightarrow{K_D} \text{Bob}. \tag{754}$$

We don't know how to do this securely using quantum computing. The problem is that K_D is a fixed string, known to Alice. What we can do is arrange for Alice and Bob to securely share an arbitrarily long random string of bits,

$$\text{Alice} \xrightarrow{\text{random string of bits}} \text{Bob}. \tag{755}$$

Appreciate the subtle difference between these two tasks. For the task we can achieve above, the string shared is not *apriori* known to either party until the end. It turns out that this suffices because the shared random bits can be used as a one time pad to securely share K_D.

The basic idea is that Alice sends Bob a random pure state $|0\rangle$ or $|1\rangle$ in one of two bases.



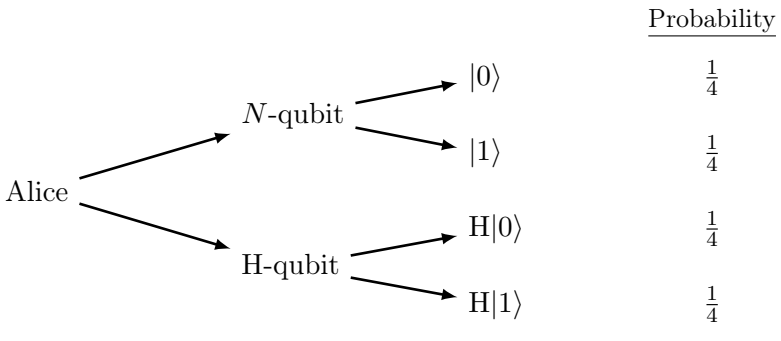
(756)

Bob now guesses the basis and measures. If Bob guesses the basis correctly, he recovers the bit. Alice now calls Bob over a public channel and tells Bob which basis she picked. If Bob guessed right, he tells Alice and now both parties know they share the same random bit. The bit shared is random because Alice picked a random pure state to start with. An eavesdropper on the public channel can tell whether or not Bob guessed the correct basis, and also what that basis was, but cannot ascertain what bit measured was. There are some physical challenges with implementing this basic idea. How does Alice send the qubit to Bob. It is very difficult to store and transport qubits. One could use photons. In this case the standard $\mathbf{e}_1, \mathbf{e}_2$ basis can be vertical or horizontal polarization. The other basis can be diagonal-up and diagonal-down polarization. There is still the challenge of Bob receiving the qubit, i.e. "catching" the photon possibly applying some unitary operation to it and then measuring. Notwithstanding these practical issues, we can still proceed with the theory.

28.1 Sending a Random Pure State in a Random Basis

Let us go through the mathematics using our familiar language. Alice starts with $|0\rangle$ and randomly decides whether to apply NOT to get a random pure state $|0\rangle$ or $|1\rangle$. Now Alice decides, again

randomly, whether or not to apply a Hadamard to the pure state. Alice then sends the qubit over a public channel to Bob. If Alice did apply the Hadamard, we call the qubit sent an H-qubit (H for Hadamard-qubit). Otherwise we call the qubit sent an N-qubit (N for Normal-qubit). Here are the possible outcomes for the qubit Alice sends Bob,



(757)

The stage now shifts to Bob. If Bob can guess which kind of qubit was sent, Bob can recover the qubit. If an N-qubit was sent, Bob can simply measure it. If an H-qubit was sent, Bob must first apply a Hadamard to undo Alice's Hadamard, then he can measure. If Bob measures the qubit directly, we call this an N-measurement. If Bob applies a Hadamard before measuring, it is an H-measurement. The problem is that Bob does not know which type of qubit was sent. So Bob guesses randomly which type of measurement to make. Let us consider what happens in a measurement. Suppose Alice sends the N-qubit $|1\rangle$. In an N-measurement, Bob recovers $|1\rangle$. In an H-measurement Bob applies a Hadamard first and then measures the state

$$H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle. \tag{758}$$

Bob measures $|0\rangle$ or $|1\rangle$ with equal probability and so cannot guarantee recovering Alice's state $|1\rangle$. Now suppose Alice sends the H-qubit

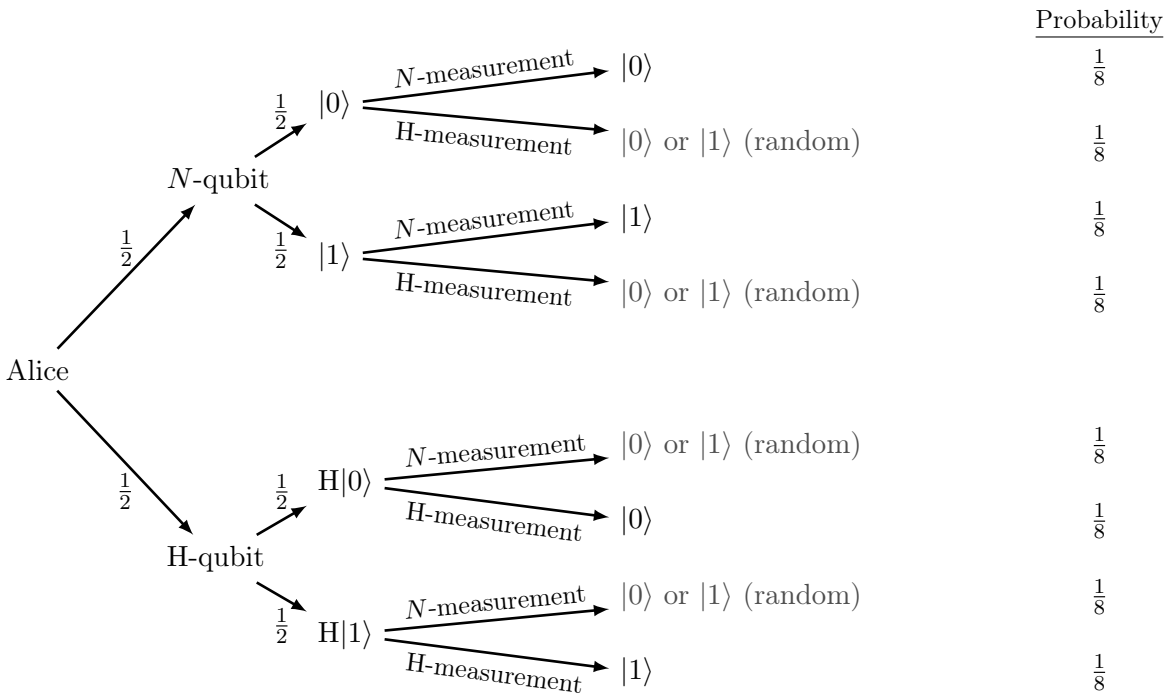
$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle. \tag{759}$$

In an N-measurement, Bob measures $|0\rangle$ or $|1\rangle$ with equal probability and so cannot guarantee recovering Alice's state $|0\rangle$. In an H-measurement Bob applies a Hadamard first to $\mathrm{H}|0\rangle$. Since $\mathrm{H}^2 = \mathrm{I}$, Bob measures the state

$$H \cdot H|0\rangle = |0\rangle. \tag{760}$$

In an H-measurement of an H-qubit, Bob always recovers the pure state Alice sent. If Bob guesses the type of qubit, N or H, then he always recovers Alices pure state. The last step of the algorithm is that Bob calls Alice on a public channel after measuring and Alice reveals the type of qubit sent and Bob reveals the type of measurement he made. If the type of qubit and type of measurement match, Alice and Bob know they agree on the pure qubit and they have successfully privately shared one random bit. If they type of qubit and measurement mismatch, they cannot guarantee matching pure states. They just throw away the qubit and start again. The process is summarized below. In four out of the 8 cases, Alice and Bob do not agree on a pure state. These four cases are highlighted

in red. In the other four cases, Alice and Bob agree on a pure state.



(761)

The probability that Alice and Bob can guarantee agreeing on a pure state is

$$\mathbb{P}[\text{Alice and Bob agree}] = \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} = \frac{1}{2}. \tag{762}$$

If Alice and Bob repeat this entire process 2ℓ times, they will expect to have shared a one time pad of ℓ bits. Ofcourse Bob does not need to call Alice for each time the process is repeated. Alice can send all her 2ℓ qubits at once, approximately half will be N-qubits and the rest H-qubits. Then Bob can do all his measuring. When Bob is done, he calls Alice, and provided they can be indexed, Alice can announce the sequence types and Bob can announce the sequence of measurement types. Or they can just exchange this information in a file by email (insecure).

28.2 Attacking the Quantum Key Exchange Protocol

Assuming Charlie (the eavesdropper) can intercept the qubit sent by Alice, Charlie cannot clone it and store it for later processing (no cloning theorem). So whatever Charlie does has to be done right then and there and then a qubit has to be forwarded on to Bob. Charlies two main options are to measure the qubit or do something more sophisticated like entangle the qubit with his own local qubit before sending.

28.2.1 Measure and Send

28.3 Entangle and Send

28.4 Bit Commitment

29 Quantum Teleportation

To set up the teleportation problem, we introduce two players, Alice and Bob. Alice is in control of some qubits and Bob is in control of some other qubits. We will use $|\psi\rangle_a$ and $|\psi\rangle_b$ to indicate the quantum state on Alice's and Bob's qubits respectively. So, for example, the quantum state

$$\frac{1}{\sqrt{2}}|001\rangle_a|00\rangle_b + \frac{1}{\sqrt{2}}|000\rangle_a|11\rangle_b \tag{763}$$

is a superposition of two classical states $|00100\rangle$ and $|00011\rangle$. The notation in (763) indicates that Alice is in possession of the first 3 qubits and Bob has the last 2 qubits. Let us focus on a single qubit teleportation. We want to transfer a quantum state from Alice to Bob,

$$|\psi\rangle_a|0\rangle_b \to |\psi\rangle_a|\psi\rangle_b, \tag{764}$$

but this is forbidden by no-cloning. An alternative is

$$|\psi\rangle_a|0\rangle_b \to |0\rangle_a|\psi\rangle_b, \tag{765}$$

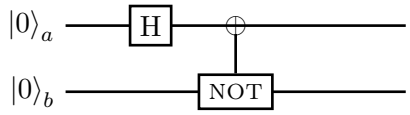
which is basically a swap operation and we have a circuit for swap (3 c-NOTs). A swap gate can only be used if Alice and Bob are colocated and a quantum gate can be implemented on their qubits. But in this case, Alice can just give her qubit to Bob. The goal of teleportation is to perform (765) when Alice and Bob are physically separated and a quantum circuit cannot be applied to their joint qubits simultaneously. We don't know a way to do this. Here is what we can do.

1. Build a quantum link from Alice to Bob that will allow instantaneous transfer of a quantum state, no matter how far Bob is from Alice.
   - This link formation requires Alice and Bob to have been together at some point.
   - The link entangles Alice and Bob together.
   - The link (entanglement) is created between qubits Alice owns and qubits Bob owns.
2. Bob now takes his qubits and goes far away from Alice.
3. Alice at some later time gets a quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Alice could call up Bob and tell him α and β. The problem is Alice does not know α, β. Even if Alice knew α, β and sent them to Bob, how would Bob create the state $|\psi\rangle$.
4. Instead, Alice applies a quantum circuit to her qubits and makes a measurement of the result. The measurement produces a pure state.
5. Alice sends the measured pure state (bits) to Bob far away. This communication is on a classical channel (email or phone).
6. Upon getting the classical bits, Bob applies a quantum circuit to his qubits from step 1.
7. Miraculously, Bob recieves the quantum state $|\psi\rangle$ "instantaneously".

This teleportation protocol communicates the information in $|\psi\rangle$ (potentially infinitely many bits of information because the amplitudes are complex numbers), but it requires a communication of classical bits over a classical channel for Bob to actually receive the information. So there is no violation of the speed of light here. The classical bits can be copied and sent via email. Even if an eavesdropper receives the classical information, there is nothing they can do with it.

What is the usecase for teleportation? The Star-Trek usecase is to beam Captain Kirk down to a planet below the enterprise. In one sense Star Trek got it right. When Kirk appears on the planet below, the version of Kirk on the enterprise must be deleted due to no-cloning. This use case is a bit far fetched, or at least far into the future. A more likely use case is that Alice is working on a quantum computation on her quantum computer and arrives at state $|\psi\rangle$. Bob, who has more powerful quantum circuits, must continue the computation so Alice needs to send her state to Bob.

The first step in the teleportation protocol is to build a quantum link between Alice and Bob. To do this we create an entangled state with two qubits and give one qubit to Alice and one to Bob. This is done when Alice and Bob are colocated. Here is the circuit,



(766)

The controlled-NOT operates on

$$\frac{|0\rangle_a + |1\rangle_a}{\sqrt{2}} \otimes |0\rangle_b = \frac{1}{\sqrt{2}}|0\rangle_a \otimes |0\rangle_b + \frac{1}{\sqrt{2}}|1\rangle_a \otimes |0\rangle_b. \tag{767}$$

The a and b subscripts simply indicates who "owns" the respective qubits. The actual quantum state is a standard superposition of $|00\rangle$ and $|10\rangle$. The controlled-NOT being a linear operator operates separately on each pure state in the superposition to give

$$\text{c-NOT}\left(\frac{1}{\sqrt{2}}|0\rangle_a \otimes |0\rangle_b + \frac{1}{\sqrt{2}}|1\rangle_a \otimes |0\rangle_b\right) = \frac{1}{\sqrt{2}}|0\rangle_a \otimes |0\rangle_b + \frac{1}{\sqrt{2}}|1\rangle_a \otimes |1\rangle_b. \tag{768}$$

The qubits at Alice and Bob are now correlated. A measurement collapses the state to either $|00\rangle$ or $|11\rangle$. This means that if Alice measures $|0\rangle$ the state collapses to the pure state $|00\rangle$. Now if Bob measures, he must see $|0\rangle$. Bob takes his qubit and moves far away from Alice. This benign operation requires some finesse. How does Bob store and transport his qubit. This is a challenge. One alternative is that the two qubits are created using photons. Two photons are created in this entagled quantum state with one photon being shot toward Alice and one toward Bob. But how do Alice and Bob "catch" exactly the right photon and then store it. Quantum states are fragile. These practical considerations not withstanding, we continue with the theory. Now Alice gets the quantum state $|\psi\rangle = \alpha|0\rangle_a + \beta|1\rangle_a$. The full quantum state of the three qubits is

$$(\alpha|0\rangle_a + \beta|1\rangle_a) \otimes \left(\frac{1}{\sqrt{2}}|0\rangle_a \otimes |0\rangle_b + \frac{1}{\sqrt{2}}|1\rangle_a \otimes |1\rangle_b\right). \tag{769}$$

The goal is to apply some operator to this 3-qubit state and convert it to something of the form

$$|\phi\rangle_a \otimes (\alpha|0\rangle_b + \beta|1\rangle_b). \tag{770}$$

$|\phi\rangle$ is some 2-qubit state at Alice, and Bob now has the original state $|\psi\rangle$. The constraint is that whatever operators we apply must either be applied by Alice on her two qubits or by Bob on his single qubit. An operator cannot be applied simultaneously to all three qubits because the qubits are not colocated. Let us label the qubits q_1, q_2, q_3. The qubits q_1 and q_2 are with Alice, while qubit q_3 is with Bob. The full quantum state, ignoring who owns which qubit is

$$\frac{\alpha}{\sqrt{2}}(|000\rangle + |011\rangle) + \frac{\beta}{\sqrt{2}}(|100\rangle + |111\rangle). \tag{771}$$

Alice first applies cNOT_{12} on her two qubits. The full 3-qubit operator is $\text{cNOT}_{12} \otimes \text{I}$.

$$\text{cNOT}_{12}\left(\frac{\alpha}{\sqrt{2}}(|000\rangle + |011\rangle) + \frac{\beta}{\sqrt{2}}(|100\rangle + |111\rangle)\right) = \frac{\alpha}{\sqrt{2}}(|000\rangle + |011\rangle) + \frac{\beta}{\sqrt{2}}(|110\rangle + |101\rangle). \tag{772}$$

Alice now applies a Hadamard on her qubit q_1. The full operator is $\text{H} \otimes \text{I} \otimes \text{I}$, applied on the state

$$\frac{\alpha}{\sqrt{2}}(|0\rangle_a|0\rangle_a|0\rangle_b + |0\rangle_a|1\rangle_a|1\rangle_b) + \frac{\beta}{\sqrt{2}}(|1\rangle_a|1\rangle_a|0\rangle_b + |1\rangle_a|0\rangle_a|1\rangle_b). \tag{773}$$

By linearity and using $\text{H}|0\rangle_a = (|0\rangle_a + |1\rangle_a)/\sqrt{2}$, $\text{H}|1\rangle_a = (|0\rangle_a - |1\rangle_a)/\sqrt{2}$, we get

$$\begin{aligned} & \frac{\alpha}{2}(|0\rangle_a|0\rangle_a|0\rangle_b + |1\rangle_a|0\rangle_a|0\rangle_b + |0\rangle_a|1\rangle_a|1\rangle_b + |1\rangle_a|1\rangle_a|1\rangle_b) \\ + \quad & \frac{\beta}{2}(|0\rangle_a|1\rangle_a|0\rangle_b - |1\rangle_a|1\rangle_a|0\rangle_b + |0\rangle_a|0\rangle_a|1\rangle_b - |1\rangle_a|0\rangle_a|1\rangle_b). \end{aligned} \tag{774}$$

Collecting terms, we get the 3-qubit quantum state

$$\begin{aligned} & \frac{1}{2}|00\rangle_a \otimes (\alpha|0\rangle_b + \beta|1\rangle_b) \\ + \quad & \frac{1}{2}|01\rangle_a \otimes (\beta|0\rangle_b + \alpha|1\rangle_b) \\ + \quad & \frac{1}{2}|10\rangle_a \otimes (\alpha|0\rangle_b - \beta|1\rangle_b) \\ + \quad & \frac{1}{2}|11\rangle_a \otimes (-\beta|0\rangle_b + \alpha|1\rangle_b). \end{aligned} \tag{775}$$

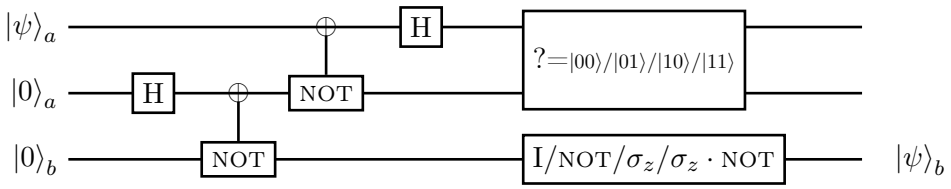
The information about α and β has been transferred to the third qubit. If Alice measures her qubits, the full 3-qubit state collapses to a pure state at Alice and a superposition at Bob.

Alice sees	Bob Has	Bob Applies
$\|00\rangle$	$\frac{\alpha\|0\rangle_b + \beta\|1\rangle_b}{\sqrt{2}}$	I
$\|01\rangle$	$\frac{\beta\|0\rangle_b + \alpha\|1\rangle_b}{\sqrt{2}}$	NOT
$\|10\rangle$	$\frac{\alpha\|0\rangle_b - \beta\|1\rangle_b}{\sqrt{2}}$	$\sigma_3 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
$\|11\rangle$	$\frac{-\beta\|0\rangle_b + \alpha\|1\rangle_b}{\sqrt{2}}$	$\sigma_3 \cdot \text{NOT}$

(776)

In each case, depending on what Alice sees, Bob can apply a standard unitary gate to his qubit and recover $|\psi\rangle$. Alice, therefore, needs to tell Bob her measurement, which is where the classical

channel comes in. However, Alice only needs to classically communicate two bits of information to transfer infinitely many bits "instantaneously". The full circuit is



(777)

The only part of this circuit that requires colocation is the first controlled not. In a futuristic world where objects are being "beamed" to different places, to teleport from Alice's location to Bob's location, Alice and Bob must have been colocated at some time. Imagine a syncing station where Alice and Bob come, they each pickup their respective entangled qubits and move on. Now Alice can teleport to Bob. Alice cannot teleport to some random location, e.g. some random plannet below the Enterprise. That random point and Alice must have at some point been colocated. So, Star Trek got that part wrong.

30 Primer On Machine Learning

31 Kernel Methods

31.1 PCA

31.2 Support Vector Machine (SVM)

31.3 What can Quantum Do For Machine Learning

32 Encoding Data into Quantum Circuits

33 Quantum Kernels

34 Quantum Machine Learning

34.1 Quantum PCA

34.2 Quantum SVM

35 Quantum Variational Encoder

36 Optimization

36.1 The Ising Hamiltonian

36.2 Binary Programming and The Ising Hamiltonion

36.3 Application to Max-Cut

37 Quantum Computing For Optimization

37.1 Optimization as Finding the Ground State of the Hamiltonian

37.2 Quantum Computing for Estimating the Ground State

37.3 Ising Hamiltonian: Quantum Approximate Optimization Algorithm (QAOA)

38 A Truly Quantum Support Vector Machine

References

Magdon-Ismail, M. (2020). *Discrete Mathematics and Computing: A Set of Lectures*. dmc-book.com.