Quantum Computation

Lecture 3: Quantum Circuits

Lecturer: Umesh Vazirani *Scribes: Victor Wen and Nemanja Isailovic*

1 Clarification on Homework 2: Problem 3

Consider an arbitrary quantum circuit with input $|x>$ consisting of basis states. Now observe the qubits at some point in the circuit (i.e. after some set of the gates has acted upon them). The general form of the qubits at this arbitrary point is

$$\sum_{y\in\{0,1\}^n} \alpha_y |y>, \alpha_y \in \mathbb{C}$$

Note that the α coefficients are generally complex. The problem in the homework asks for a redesign of such an arbitrary circuit such that the α coefficients are always real at any point in the circuit. (Of course, you are allowed to increase the complexity of the circuit by a fraction in order to accomplish this.)

2 Review

If we have n qubits, their collective state is the unit vector $|\phi>$:

$$|\phi> \in \underbrace{\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2}_{ntimes} = \mathbb{C}^{2^n}$$

Several gates were also introduced last time, the most important of which are the single-bit Hadamard gate and the double-bit CNOT gate (see previous week's notes for their matrix representations).

3 Operations on Subsets of the Input

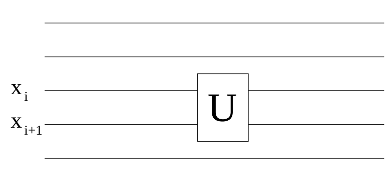

Figure 1: Unitary transformation gate acting on a pair of qubits.

3.1 General Case

Suppose we are given n qubits as input, but we only wish to use a single two-qubit gate (corresponding to a unitary transformation) on a single pair of the qubits (as in Figure 1). The inputs and outputs are just superpositions of the same elements, so we write the input as

$$\sum_{x\in\{0,1\}^n} \alpha_x|x>$$

and we write the output as

$$\sum_{x\in\{0,1\}^n} \beta_x|x>$$

The gate in question acts on qubits x_i and x_{i+1}. We wish to compute the output in terms of the input. Notice that the input can be written as

$$\sum_{x\in\{0,1\}^n} \alpha_x|x_1x_2\cdots x_ix_{i+1}\cdots x_n>$$

But since the two-qubit gate U acts only on qubits x_i and x_{i+1}, the result is

$$\sum_{x\in\{0,1\}^n} \alpha_x|x_1x_2\cdots U|x_ix_{i+1}>\cdots x_n>$$

Now we can simply expand this in order to get the β values in the output.

Notice that, since U acts only on two qubits, we can treat it as if it applies the identity operator to all the other qubits. Thus, we can apply the following matrix to the complete input, and the result is the same as if we only applied U to qubits x_i and x_{i+1}.

$$2^n\left\{\underbrace{\begin{pmatrix} U & 0 & 0 & \cdots & 0 \\ 0 & U & 0 & \cdots & 0 \\ 0 & 0 & U & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & U \end{pmatrix}}_{2^n}\right.$$

3.2 Example

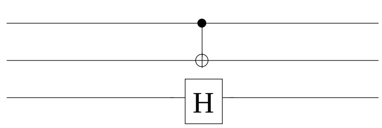

Figure 2: Applying CNOT gate and a Hadamard gate to three qubits.

Let us find the matrix corresponding to the circuit shown in Figure 2. This circuit consists of applying the CNOT gate to the first two qubits and independently applying a Hadamard transform to the third qubit.

Remember that the matrix for the CNOT gate is

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

and the matrix for the Hadamard transform is

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

The matrix representing the circuit in Figure 2 is actually the Tensor product of these two matrices:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \end{pmatrix}$$

In order to find the Tensor product of two matrices A and B, replace each element of matrix A with the product of that element and matrix B. Thus, each element of matrix A is replaced by something the size of matrix B. So if A is an axa matrix and B is a bxb matrix, then the Tensor product of A and B will be a cxc matrix where $c = ab$.

3.3 Definitions

A *local* unitary transformation acts as the identity on all but a constant number of qubits, regardless of the size of the input. Only local unitary transformations may be made in a real circuit.

A unitary transformation is *simple* if you can write it as $I \otimes V_{ij}$ (i.e. as the Tensor product of the identity matrix and a two-qubit gate). In other words, a simple unitary transformation acts on at most two qubits, regardless of the size of the input.

A quantum circuit U consists of a sequence of *simple* gates $U_1, U_2, \ldots U_k$ applied sequentially:

$$U = U_1 \cdot U_2 \cdots U_k$$

The value of k (that is, the number of gates in the circuit) is a measure of the circuit's complexity. Ideally, we want k to be polynomial in n, the number of qubits in the input.

4 Theorems

4.1 Universal Families of Gates

Two examples of a universal family of gates are:

- CNOT plus a one-qubit gate
- CNOT plus Hadamard gate plus phase flip gate

4.2 A Theorem

Given an m-dimensional unitary transformation U, $\exists$ a sequence simple gates $U_1, U_2, \ldots, U_k$ such that

$$\| U - U_1 U_2 \cdots U_k \| \leq \epsilon$$

where

$$k = polynomial(2^m, log\frac{1}{\epsilon})$$

This theorem means that any U with any number of input qubits can be approximated by a series of two-qubit gates, with a polynomial upper bound on the number of gates.

4.3 Solovay-Kitaev Theorem

Consider a finite set of gates $G \subseteq SU(d)$. (Note: $SU(d)$ denotes the set of unitary transformations in a d-dimensional Hilbert space.) Let G be closed under inverses ($g \in G \Leftrightarrow g^{-1} \in G$), and let G be dense in $SU(d)$. Then any $U \in SU(d)$ has a sequence $g_1, g_2, \ldots, g_k$ where $g_i \in G$ such that

$$\| g_1 g_2 \cdots g_k - U \| \leq \epsilon$$

and

$$k = O(log^2 \frac{1}{\epsilon})$$

The bottom line of this theorem is that local unitary operations can be done efficiently.

4.4 Gate Accuracy: Digital v. Analog

When we implement a gate, how accurate does it need to be? Even a gate as simple as the Hadamard gate involves the value $1/\sqrt{2}$. Do we need infinite precision to build this gate properly? A paper by Shamir showed that, if we assume infinite precision arithmetic, then a large class of otherwise difficult problems becomes easy. However, we obviously cannot have infinite precision, so we must digitize quantum computation in order to approximate values such as $1/\sqrt{2}$.

Suppose that we want to build a gate that rotates the input by θ, but the best accuracy we can actually build is rotation by $\theta \pm \delta\theta$ (finite precision). Let $U_1, \ldots, U_m$ be a set of ideal gates that implement an exact rotation by θ. Let $V_1, \ldots, V_m$ be a set of actual (constructible) gates that implement rotation by $\theta \pm \delta\theta$. Let $|\phi>$ be the initial state. Let $|\psi>$ be the ideal output

$$|\psi> = U_m U_{m-1} \ldots U_2 U_1 |\phi>$$

and let $|\psi'>$ be the actual output

$$|\psi'> = V_m V_{m-1} \ldots V_2 V_1 |\phi>$$

The closer $|\psi>$ and $|\psi'>$ are to each other, the better the approximation.

Theorem: If $\| U_i - V_i \| \leq \frac{\epsilon}{4m}$, then $\| |\psi> - |\psi'> \| \leq \frac{\epsilon}{4}$.

Proof: Consider two hybrid states

$$|\psi_k> = U_m \cdots U_{k+1} V_k \cdots V_1 |\phi>$$

and

$$|\psi_{k+1}> = U_m \cdots U_{k+2} V_{k+1} \cdots V_1 |\phi>$$

Subtract $|\psi_{k+1}>$ from $|\psi_k>$ to get

$$|\psi_k> - |\psi_{k+1}> = U_m \cdots U_{k+2}(U_{k+1} - V_{k+1})V_k \cdots V_1|\phi>$$

Since the unitary transformations don't change the norm of the vector, the only term we need to consider is $U_{k+1} - V_{k+1}$. But we have an upper bound on this, so we can conclude that

$$\| \, |\psi_k> - |\psi_{k+1}> \| \leq \frac{epislon}{4m}$$

Now we use the triangle inequality to finish the proof:

$$\| \, |\psi> - |\psi'> \| = \| \, |\psi_0> - |\psi_m> \| \leq \sum_i \| \, |\psi_i> - |\psi_{i+1}> \| \leq m \cdot \frac{\epsilon}{4m} \leq \frac{\epsilon}{4}$$

4.5 Random Example

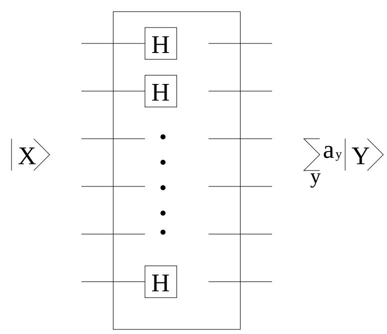

Figure 3:

The circuit in Figure 3 simply consists of a Hadamard gate applied independently to each input qubit. For simplicity, let us consider such a circuit consisting of only two inputs. The matrix representation is

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

The input $|x>$ is a Tensor product. The output is

$$\sum_y \alpha_y |y> = \sum_y \frac{\pm 1}{2^{n/2}} |y> = \sum_y \frac{(-1)^{x \cdot y}}{2^{n/2}} |y>$$

where

$$x \cdot y = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

So the Hadamard gates have generated all 2^n different bitstrings, with all information about the original input $|x>$ stored in the phases.

5 Algorithm/Circuit Complexity

5.1 Polynomial Time (P)

Let "PRIMES" denote the problem of determining whether a positive integer input N is prime or composite. It was recently proven (by Agrawal, Kayal and Saxena) that $PRIMES \in P$. More specifically, there exists a program which runs in time $O(logN)$ (polynomial in the *length* of the input N) and which determines with certainty whether N is a prime number.

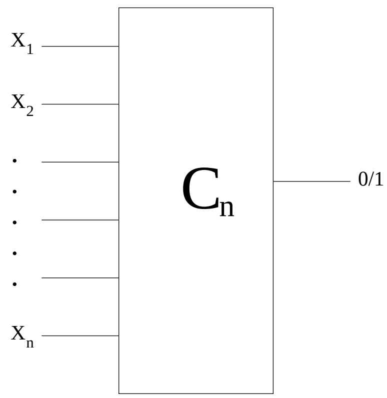

Figure 4:

This classical circuit is shown in Figure 4. There are two conditions necessary for polynomial time. First, we put a polynomial bound on the circuit size:

$$|C_n| = O(poly(n))$$

Second, we need the *Uniformity Condition*: It must be possible for the description of the circuit C_n to be computed in time $poly(n)$ on some Turing machine.

5.2 Bounded-error Probabilistic Polynomial Time (BPP)

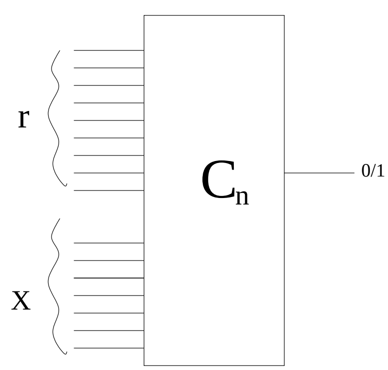

Figure 5:

Figure 5 shows a circuit corresponding to a quantum BPP solution of PRIMES. The output still consists of a single bit signifying prime or not-prime. The input consists of the integer N being tested for primality as well as a set of random qubits. The circuit C_n implements a probabilistic algorithm. That is,

$$if N \in PRIMES, C_n(N) = 1 \text{ with probability } \geq \frac{1}{2} + \epsilon$$

$$if N \notin PRIMES, C_n(N) = 0 \text{ with probability } \geq \frac{1}{2} + \epsilon$$

This simply means that the algorithm is correct within a bounded error probability. In order for the algorithm to be usable, it must be correct more than half the time ($\frac{1}{2} + \epsilon$). If this is true, then the algorithm may be run many times (with different random qubits) to achieve any desired level of accuracy. This method is called boosting. (Note: The two conditions from above, polynomial circuit size and the Uniformity Condition, are still necessary in BPP.)

5.3 Bounded-error Quantum Polynomial Time (BQP)

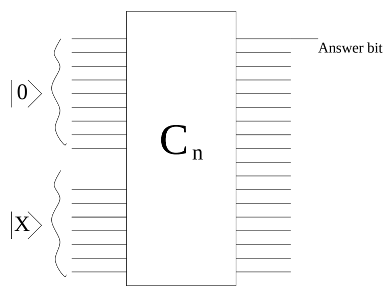

Figure 6:

This is the same as BPP, but we consider the realities of quantum computation. As shown in Figure 6, instead of random qubits, we input $|0>$'s. Additionally, all of the incoming qubits are still there at output, but we designate one of them as the "answer."

But now we need to consider whether boosting is still possible. We need to make certain that we are truly executing several *independent* and identical tests. Since $|x>$ is a classical input, we can copy it using CNOT gates (as shown in Figure 7), so we can generate as many independent copies of $|x>$ as we need. Additionally, we notice that the inputs are in Tensor product form:

$$|0^k> \otimes |x> \otimes |0^k> \otimes |x>$$

By definition of the Tensor product, if $|\phi>$ passes through circuit C_1 and $|\psi>$ passes through circuit C_2, we know

$$|\phi> \otimes |\psi> \rightarrow C_1|\phi> \otimes C_2|\psi>$$

Thus, if the inputs $|x>$ are independent, then the outputs will also be independent, so boosting is still possible in this model.

5.4 P $\subseteq$ BQP

Anything that can be computed in polynomial time can be computed in quantum polynomial time. If we have a classical input (consisting entirely of basis states), then any unitary transformation simply permutes

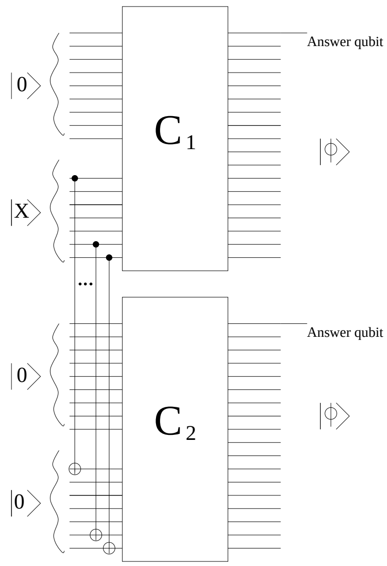

Figure 7:

the inputs. Thus, the output also consists of purely basis states, and there's no erasure. So quantum computation (on classical inputs) is reversible computation.

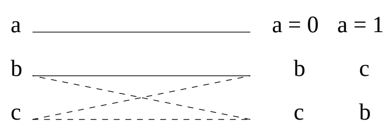

Figure 8:

This is easy to show with the help of the Fredkin gate shown in Figure 8. The inputs are three qubits a, b and c, each in a basis state (0 or 1). Qubit a passes through unchanged, while qubits b and c are swapped if $a = 1$ (otherwise, they pass through unchanged). We can simulate two classical gates using the Fredkin gate:

$$NOT : a = input, b = 0, c = 1, output = c$$

$$AND : a\&b = inputs, c = 0, output = c$$

But AND and NOT form a universal family. Using arbitrary quantities of these two gates, we can devise any classical circuit. Thus, anything that can be done classically can also be done in BQP.

5.5 BPP $\subseteq$ BQP

In BPP, we input a number of random qubits into the circuit. In BQP, we input all $|0>$'s. We want to show that it is possible to get random qubits from the $|0>$'s. A simple solution is shown in Figure 9. We simply apply a Hadamard gate in order to convert

$$|0> \to \frac{1}{2}(|0> + |1>)$$

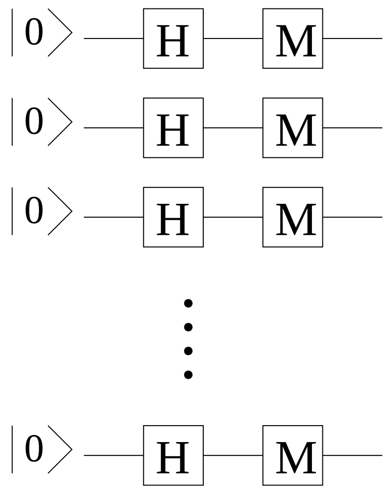

Figure 9:

Then we measure that, resulting in equal chance of getting either a $|0>$ or a $|1>$. This gives us the desired random qubit at each input.

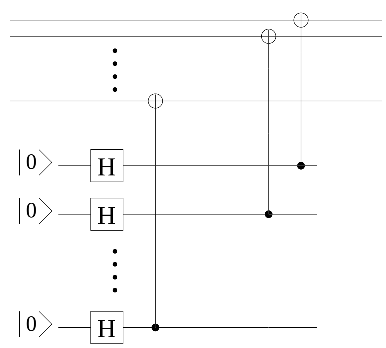

Figure 10:

The drawback of this first approach is the measurement. Suppose that we wish to entirely avoid making a measurement. Such a solution is shown in Figure 10. We now need twice as many $|0>$'s as before. Half of them are passed through Hadamard gates and then connected by CNOT gates to the other half. This fixes the first half of the qubits to either $|0>$ or $|1>$, even though no measurement was made. It is very important to note, however, that since the second half of the qubits are now entagled with the first half, we must be certain not to make any measurements on them either. This is called the Principle of Deferred Measurement.