Quantum algorithms (CO 781, Winter 2008)
Prof. Andrew Childs, University of Waterloo

LECTURE 1: Quantum circuits and the abelian QFT

This is a course on quantum algorithms. It is intended for graduate students who have already taken an introductory course on quantum information. Such an introductory course typically covers only the early breakthroughs in quantum algorithms, namely Shor's factoring algorithm (1994) and Grover's searching algorithm (1996). The purpose of this course is to show that there is more to quantum computing than Shor and Grover by exploring some of the many quantum algorithms that have been developed in the intervening decade.

The course will consist of three parts, corresponding to what I think are the three most active research areas in quantum algorithms:

- The *hidden subgroup problem* is a group-theoretic problem that generalizes the core problem solved by Shor's algorithm. Thinking about the HSP has led to new quantum algorithms, and also presents some important open questions.
- *Quantum walk* is a quantum generalization of random walk. There are several different search algorithms based on quantum walk that can be viewed as generalizing Grover's algorithm, and also some indications that quantum walk may be useful for more dramatic speedups.
- *Adiabatic quantum computing* is a general approach to optimization problems in a similar spirit to simulated annealing. Related ideas may also provide insights into how one might build a quantum computer.

We will spend the most time on the HSP (since these algorithms tend to have the most dramatic advantage over classical computation) and the least time on adiabatic computation (since this is a newer topic with fewer results to discuss).

Unfortunately, due to time limitations (and the limited expertise of the instructor), we will not cover every aspect of quantum algorithms. Notable omissions will include

- Quantum algorithms for approximating the Jones polynomial and related quantities
- Lower bounds on quantum query complexity
- Quantum complexity beyond polynomial time (QMA, quantum interactive proofs, etc.)

In this lecture, we will briefly review some background material on quantum computation. If you plan to take this course, most of this material should be familiar to you.

Quantum data A quantum computer is a device that uses a quantum mechanical representation of information to perform calculations. Information is stored in quantum bits, the states of which can be represented as ℓ_2-normalized vectors in a complex vector space. So for example, we can write the state of n qubits as

$$|\psi\rangle = \sum_{x\in\{0,1\}^n} a_x|x\rangle \tag{1}$$

where the $a_x \in \mathbb{C}$ satisfy $\sum_x |a_x|^2 = 1$. We refer to the basis of states $|x\rangle$ as the *computational basis*.

It will often be useful to think of quantum states as storing data in a more abstract form. For example, given a group G, we could write $|g\rangle$ for a basis state corresponding to the group element

$g \in G$, and

$$|\phi\rangle = \sum_{g \in G} b_g |g\rangle \tag{2}$$

for an arbitrary superposition over the group. We assume that there is some canonical way of efficiently representing group elements using bit strings; it is usually unnecessary to make this representation explicit.

If a quantum computer stores the state $|\psi\rangle$ *and* the state $|\phi\rangle$, its overall state is given by the tensor product of those two states. This may be denoted $|\psi\rangle \otimes |\phi\rangle = |\psi\rangle|\phi\rangle = |\psi, \phi\rangle$.

Quantum circuits The allowed operations on (pure) quantum states are those that map normalized states to normalized states, namely *unitary operators* U, satisfying $UU^\dagger = U^\dagger U = 1$. (You probably know that there are more general quantum operations, but for the most part we will not need to use them in this course.)

To have a sensible notion of *efficient* computation, we require that the unitary operators appearing in a quantum computation are realized by *quantum circuits*. We are given a set of gates, each of which acts on one or two qubits at a time (meaning that it is a tensor product of a one- or two-qubit operator with the identity operator on the remaining qubits). A quantum computation begins in the $|0\rangle$ state, applies a sequence of one- and two-qubit gates chosen from the set of allowed gates, and finally reports an outcome obtained by measuring in the computational basis.

Universal gate sets In principle, any unitary operator on n qubits can be implemented using only 1- and 2-qubit gates. Thus we say that the set of all 1- and 2-qubit gates is *(exactly) universal*. (Of course, some unitary operators may take many more 1- and 2-qubit gates to realize than others, and indeed, a counting argument shows that most unitary operators on n qubits can only be realized using an exponentially large circuit.)

In general, we are content to give circuits that give good approximations of our desired unitary transformations. We say that a circuit with gates $U_1, U_2, \ldots, U_t$ approximates U with precision ϵ if

$$\|U - U_t \cdots U_2 U_1\| \leq \epsilon \tag{3}$$

where $\|\cdot\|$ denotes the operator norm, the largest singular value. We call a set of elementary gates *universal* if any unitary operator on a fixed number of qubits can be approximated to precision ϵ using $\mathrm{poly}(\log \frac{1}{\epsilon})$ elementary gates. It turns out that there are finite sets of gates that are universal: for example, the set $\{H, T, C\}$ with

$$H := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad T := \begin{pmatrix} e^{i\pi/8} & 0 \\ 0 & e^{-i\pi/8} \end{pmatrix} \qquad C := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{4}$$

There are situations in which we say a set of gates is *effectively* universal, even though it cannot actually approximate any unitary operator on n qubits. For example, the basis $\{H, T^2, T^{-2}, C, \mathrm{Tof}\}$,

where

$$\text{Tof} := \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{5}$$

is universal, but only if we allow the use of ancilla qubits (qubits that start and end in the $|0\rangle$ state). Similarly, the basis $\{H, \text{Tof}\}$ is universal in the sense that, with ancillas, it can approximate any *orthogonal* matrix. It clearly cannot approximate complex unitary matrices, since the entries of H and Tof are real; but the effect of arbitrary unitary transformations can be simulated using orthogonal ones by simulating the real and imaginary parts separately.

Efficiently universal gate sets Are some universal gate sets better than others? It turns out that the answer is essentially no: a unitary operator that can be realized efficiently with one set of 1- and 2-qubit gates can also be realized efficiently with another such set. This is a consequence of

Theorem (Solovay-Kitaev)**.** *Fix two bases that allow universal quantum computation. Then any t-gate circuit in the first basis can be implemented to precision ϵ using a circuit of $t \cdot \text{poly}(\log(t/\epsilon))$ gates from the second basis (indeed, there is an efficient classical algorithm for finding this circuit).*

Note that if we cared about *exactly* implementing unitary operators, the situation could be quite different.

Reversible computation Unitary matrices are invertible: in particular, $U^{-1} = U^\dagger$. Thus any unitary transformation is a reversible operation. This may seem at odds with how we often define classical circuits, using irreversible gates such as AND and OR. But in fact, any classical computation can be made reversible by replacing any irreversible gate $x \mapsto g(x)$ by the reversible gate $(x, y) \mapsto (x, y \oplus g(x))$, and running it on the input $(x, 0)$, producing $(x, g(x))$. In other words, by storing all intermediate steps of the computation, we make it reversible.

On a quantum computer, storing all intermediate computational steps could present a problem, since two identical results obtained in different ways would not be able to interfere. However, there is an easy way to remove the accumulated information. After performing the classical computation with reversible gates, we simply XOR the answer into an ancilla register, and then perform the computation in reverse. Thus we can implement the map $(x, y) \mapsto (x, y \oplus f(x))$ even when f is a complicated circuit consisting of many gates.

Using this trick, any computation that can be performed efficiently on a classical computer can be performed efficiently on a quantum computer, even on a superposition of computational basis states. For example, if we can efficiently implement the map $x \mapsto f(x)$ on a classical computer, we can efficiently perform the transformation

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x, 0\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x, f(x)\rangle \tag{6}$$

on a quantum computer.

Uniformity When we give an algorithm for a computational problem, we consider inputs of varying sizes. Typically, the circuits for instances of different sizes with be related to one another in a simple way. But this need not be the case; and indeed, given the ability to choose an arbitrary circuit for each input size, we could have circuits computing uncomputable languages. Thus we require that our circuits be *uniformly generated*: say, that there exists a fixed (classical) Turing machine that, given a tape containing the symbol '1' n times, outputs a description of the nth circuit in time poly(n).

Quantum complexity We say that an algorithm for a problem is *efficient* if the circuit describing it contains a number of gates that is polynomial in the number of bits needed to write down the input. For example, if the input is a number modulo N, the input size is $\lceil \log_2 N \rceil$.

With a quantum computer, as with a randomized (or noisy) classical computer, the final result of a computation may not be correct with certainty. Instead, we are typically content with an algorithm that can produce the correct answer with high enough probability (for a decision problem, bounded above 1/2; for a non-decision problem for which we can check a correct solution, $\Omega(1)$). By repeating the computation many times, we can make the probability of outputting an incorrect answer arbitrarily small.

In addition to considering explicit computational problems, in which the input is a string, we will also consider the concept of *query complexity*. Here the input is a black box transformation, and our goal is to discover some property of the transformation by making as few queries as possible. For example, in Simon's problem, we are given a transformation $f : \mathbb{Z}_2^n \to S$ satisfying $f(x) = f(y)$ iff $y = x \oplus t$ for some unknown $t \in \mathbb{Z}_2^n$, and the goal is to learn t. The main advantage of considering query complexity is that it allows us to prove lower bounds on the number of queries required to solve a given problem. Furthermore, if we find an efficient algorithm for a problem in query complexity, then if we are given an explicit circuit realizing the black-box transformation, we will have an efficient algorithm for an explicit computational problem.

Sometimes, we care not just about the size of a circuit for implementing a particular unitary operation, but also about its *width*, the maximum number of gates on any path from an input to an output. The width of a circuit tells us how long it takes to implement if we can perform gates in parallel. In the problem set, you will get a chance to think about parallel circuits for implementing the quantum Fourier transform.

Fault tolerance In any real computer, operations cannot be performed perfectly. Quantum gates and measurements may be performed imprecisely, and errors may happen even to stored data that is not being manipulated. Fortunately, there are protocols for dealing with faults that may occur during the execution of a quantum computation. Specifically, the *threshold theorem* states that as long as the noise level is below some threshold (depending on the noise model, but typically in the range of 10^{-3} to 10^{-4}, an arbitrarily long computation can be performed with an arbitrarily small amount of error.

In this course, we will always assume implicitly that fault-tolerant protocols have been applied, such that we can effectively assume a perfectly functioning quantum computer. (Then only possible exception is that we may briefly discuss the inherent robustness of adiabatic quantum computation.)

Quantum Fourier transform Perhaps the most important unitary transformation in quantum computing is the *quantum Fourier transform* (QFT). Later, we will discuss the QFT over arbitrary

finite groups; but in this lecture we will focus on the case of an abelian group G. Here the transformation is

$$F_G := \frac{1}{\sqrt{|G|}} \sum_{x \in G} \sum_{y \in \hat{G}} \chi_y(x) |y\rangle\langle x| \tag{7}$$

where $\hat{G}$ is a complete set of characters of G, and $\chi_y(x)$ denotes the yth character of G evaluated at x. (You can verify that this is a unitary operator using the orthogonality of characters.) Since G and $\hat{G}$ are isomorphic, we can label the elements of $\hat{G}$ using elements of G, and it is often useful to do so.

The simplest QFT over a family of groups is the QFT over $\mathbb{Z}_2^n$; it is simply

$$F_{\mathbb{Z}_2^n} = \frac{1}{\sqrt{2^n}} \sum_{x,y \in \mathbb{Z}_2^n} (-1)^{x \cdot y} |y\rangle\langle x| = H^{\otimes n}. \tag{8}$$

You have presumably seen how this transformation is used in the solution of Simon's problem.

QFT over $\mathbb{Z}_{2^n}$ A more complex quantum Fourier transform is the QFT over $\mathbb{Z}_{2^n}$:

$$F_{\mathbb{Z}_{2^n}} = \frac{1}{\sqrt{2^n}} \sum_{x,y \in \mathbb{Z}_{2^n}} \omega_{2^n}^{xy} |y\rangle\langle x| \tag{9}$$

where $\omega_m := \exp(2\pi \mathrm{i}/m)$ is a primitive mth root of unity. To see how to realize this transformation by a quantum circuit, it is helpful to represent the input x as a string of bits, $x = x_{n-1} \ldots x_1 x_0$, and to consider how an input basis vector is transformed:

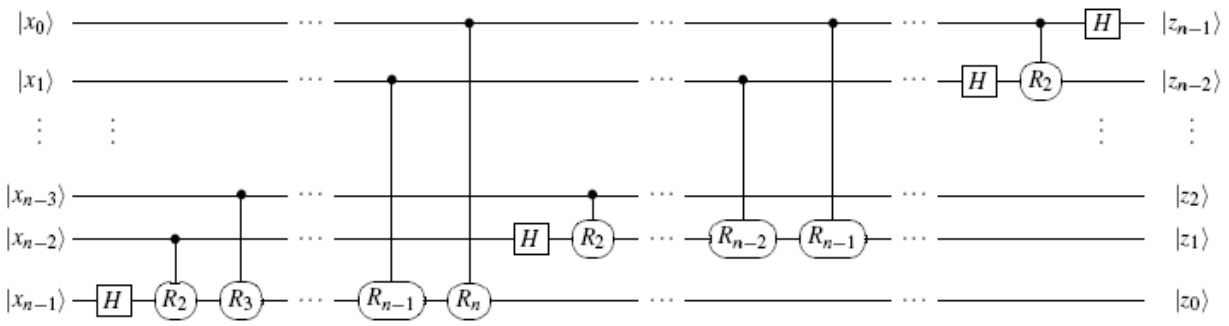
$$\begin{aligned}
|x\rangle &= \bigotimes_{j=0}^{n-1} |x_j\rangle && (10)\\
&\mapsto \frac{1}{\sqrt{2^n}} \sum_{y \in \mathbb{Z}_{2^n}} \omega_{2^n}^{(\sum_{j=0}^{n-1} x_j 2^j)(\sum_{k=0}^{n-1} y_k 2^k)} |y_{n-1} \ldots y_1 y_0\rangle && (11)\\
&= \frac{1}{\sqrt{2^n}} \sum_{y \in \mathbb{Z}_{2^n}} \prod_{k=0}^{n-1} \omega_{2^n}^{\sum_{j=0}^{n-1} x_j y_k 2^{j+k}} |y_{n-1} \ldots y_1 y_0\rangle && (12)\\
&= \frac{1}{\sqrt{2^n}} \bigotimes_{k=0}^{n-1} \sum_{y_k \in \mathbb{Z}_2} \omega_{2^n}^{\sum_{j=0}^{n-1} x_j y_k 2^{j+k}} |y_k\rangle && (13)\\
&= \bigotimes_{k=0}^{n-1} \frac{1}{\sqrt{2}} (|0\rangle + \omega_{2^n}^{\sum_{j=0}^{n-1} x_j 2^{j+k}} |1\rangle) && (14)\\
&= \bigotimes_{k=0}^{n-1} \frac{1}{\sqrt{2}} (|0\rangle + \omega_{2^n}^{\sum_{j=0}^{n-1-k} x_j 2^{j+k}} |1\rangle) && (15)\\
&= \bigotimes_{k=0}^{n-1} \frac{1}{\sqrt{2}} (|0\rangle + \prod_{j=0}^{n-1-k} \omega_{2^{n-k}}^{x_j} |1\rangle) && (16)\\
&=: \bigotimes_{k=0}^{n-1} |z_k\rangle. && (17)
\end{aligned}$$

In other words, $F|x\rangle$ is a tensor product of single-qubit states, where the kth qubit only depends on the k least significant bits of x.

This decomposition immediately gives a circuit for the QFT over $\mathbb{Z}_{2^n}$. Let R_k denote the single-qubit unitary operator

$$R_k := \begin{pmatrix} 1 & 0 \\ 0 & \omega_{2^k} \end{pmatrix}. \tag{18}$$

Then the circuit can be written as follows:



This circuit uses $O(n^2)$ gates. However, there are many rotations by small angles that do not affect the final result very much. If we simply omit the gates R_k with $k = \Omega(\log n)$, then we obtain a circuit with $O(n \log n)$ gates that implements the QFT with precision $1/\operatorname{poly}(n)$.

Phase estimation Aside from being directly useful in quantum algorithms, such as Shor's algorithm, The QFT over $\mathbb{Z}_{2^n}$ provides a useful quantum computing primitive called *phase estimation*. In the phase estimation problem, we are given a unitary operator U (either as an explicit circuit, or as a black box that allows applying a controlled-U^j operation for integer values of j). We are also given a state $|\phi\rangle$ that is promised to be an eigenvector of U, namely $U|\phi\rangle = e^{i\phi}|\phi\rangle$ for some $\phi \in \mathbb{R}$. The goal is to output an estimate of ϕ to some desired precision.

The procedure for phase estimation is straightforward. To get an n-bit estimate of ϕ, prepare the quantum computer in the state

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \mathbb{Z}_{2^n}} |x, \phi\rangle, \tag{19}$$

apply the operator

$$\sum_{x \in \mathbb{Z}_{2^n}} |x\rangle\langle x| \otimes U^x, \tag{20}$$

apply an inverse Fourier transform on the first register, and measure. If the binary expansion of $2\pi\phi$ terminates after at most n bits, then the result is guaranteed to be the binary expansion of $2\pi\phi$. In general, we obtain a good approximation with high probability.

QFT over $\mathbb{Z}_N$ and over a general finite abelian group One useful application of phase estimation is to implement the QFT over an arbitrary cyclic group $\mathbb{Z}_N$:

$$F_{\mathbb{Z}_N} = \frac{1}{\sqrt{N}} \sum_{x,y \in \mathbb{Z}_N} \omega_N^{xy} |y\rangle\langle x|. \tag{21}$$

The circuit we derived using the binary representation of the input and output only works when N is a power of two (or, with a slight generalization, some other small integer). But there is a simple way to realize $F_{\mathbb{Z}_N}$ (approximately) using phase estimation.

We would like to perform the transformation that maps $|x\rangle \mapsto |\tilde{x}\rangle$, where $|\tilde{x}\rangle := F_{\mathbb{Z}_N}|x\rangle$ denotes a Fourier basis state. (By linearity, if the transformation acts correctly on a basis, it acts correctly on all states.) It is straightforward to perform the transformation $|x, 0\rangle \mapsto |x, \tilde{x}\rangle$; but then the difficulty is to erase the register $|x\rangle$ from such a state.

Consider the unitary operator that adds 1 modulo N:

$$U := \sum_{x \in \mathbb{Z}_N} |x+1\rangle\langle x|. \tag{22}$$

The eigenstates of this operator are precisely the Fourier basis states $|\tilde{x}\rangle := F_{\mathbb{Z}_N}|x\rangle$, since (as a simple calculation shows)

$$F_{\mathbb{Z}_N}^\dagger U F_{\mathbb{Z}_N} = \sum_{x \in \mathbb{Z}_N} \omega_N^x |x\rangle\langle x|. \tag{23}$$

Thus, using phase estimation on U (with n bits of precision where $n = O(\log N)$), we can perform the transformation

$$|\tilde{x}, 0\rangle \mapsto |\tilde{x}, x\rangle \tag{24}$$

(actually, phase estimation only gives an approximation of x, so we implement this transformation only approximately). By running this operation in reverse, we can erase $|x\rangle$, and thereby produce the desired QFT.

Given the Fourier transform over $\mathbb{Z}_N$, it is straightforward to implement the QFT over an arbitrary finite abelian group: any finite abelian group can be written as a direct product of cyclic factors, and the QFT over a direct product of groups is simply the tensor product of QFTs over the individual groups.