Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 1: Quantum circuits

This is a course on quantum algorithms. It is intended for graduate students who have already taken an introductory course on quantum information. Such a 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 since then.

The course will cover several major topics in quantum algorithms, as follows:

- We will discuss algorithms that generalize the main idea of Shor's algorithm. These algorithms make use of the quantum Fourier transform, and typically achieve an exponential (or at least superpolynomial) speedup over classical computers. In particular, we will explore a group-theoretic problem called the *hidden subgroup problem*. We will see how a solution of this problem for abelian groups leads to several applications, and we will also discuss what is known about the nonabelian case.
- We will explore the concept of *quantum walk*, a quantum generalization of random walk. This concept leads to a powerful framework for solving search problems, generalizing Grover's search algorithm.
- We will discuss lower bounds on quantum query complexity, demonstrating limitations on the power of quantum algorithms. We will cover the two main quantum lower bound techniques, the adversary method and the polynomial method.
- We will see how, through the concept of *span programs*, the quantum adversary method can in fact be turned into an upper bound on quantum query complexity. We will also see how these ideas lead to optimal quantum algorithms for evaluating Boolean formulas.
- Time permitting, we will cover other recent topics in quantum algorithms, such as adiabatic optimization and the approximation of the Jones polynomial.

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 (except for the details of the Solovay-Kitaev theorem).

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. 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 $UU^\dagger = U^\dagger U = I$. (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 of 1- and 2-qubit gates.

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

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

Here $\|\cdot\|$ denotes some appropriate matrix norm, which should have the property that if $\|U - V\|$ is small, then U should be hard to distinguish from V no matter what quantum state they act on. A natural choice (which will be suitable for our purposes) is the spectral norm

$$\|A\| := \max_{|\psi\rangle} \frac{\|A|\psi\rangle\|}{\||\psi\rangle\|}, \tag{4}$$

(where $\||\psi\rangle\| = \sqrt{\langle\psi|\psi\rangle}$ denotes the vector 2-norm of $|\psi\rangle$), i.e., the largest singular value of A. Then we call a set of elementary gates *universal* if any unitary operator on a fixed number of qubits can be approximated to any desired precision ϵ using elementary gates.

It turns out that there are finite sets of gates that are universal: for example, the set $\{H, T, C\}$

with

$$H := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad T := \begin{pmatrix} e^{i\pi/8} & 0 \\ 0 & e^{-i\pi/8} \end{pmatrix} \qquad C := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{5}$$

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 set $\{H, T^2, \text{Tof}\}$, where

$$\text{Tof} := \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{6}$$

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.

Equivalence between different universal gate sets

Are some universal gate sets better than others? Classically, this is not an issue: the set of possible operations is discrete, so any gate acting on a constant number of bits can be simulated exactly using a constant number of gates from any given universal gate set. But we might imagine that some quantum gates are much more powerful than others. For example, given two rotations about strange axes by strange angles, it may not be obvious how to implement a Hadamard gate, and we might worry that implementing such a gate to high precision could take a very large number of elementary operations, scaling badly with the required precision.

Fortunately, it turns out that this is not the case: 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. In particular, we have the following.

Theorem (Solovay-Kitaev)**.** *Fix two universal gate sets that are closed under inverses. Then any t-gate circuit using one gate set can be implemented to precision ϵ using a circuit of $t \cdot \text{poly}(\log \frac{t}{\epsilon})$ gates from other set (indeed, there is a classical algorithm for finding this circuit in time $t \cdot \text{poly}(\log \frac{t}{\epsilon})$).*

Thus, not only are the two gate sets equivalent under polynomial-time reduction, but the running time of an algorithm using one gate set is the same as that using the other gate set up to logarithmic factors. This means that even polynomial quantum speedups are robust with respect to the choice of gate set.

To establish this, we first note the basic fact that errors in the approximation of one quantum circuit by another accumulate linearly.

Lemma. *Let U_i, V_i be unitary matrices satisfying $\|U_i - V_i\| \leq \epsilon$ for all $i \in \{1, 2, \ldots, t\}$. Then $\|U_t \ldots U_2 U_1 - V_t \ldots V_2 V_1\| \leq t\epsilon$.*

Proof. We use induction on t. For $t = 1$ the lemma is trivial. Now suppose the lemma holds for a particular value of t. Then by the triangle inequality and the fact that the norm is unitarily invariant ($\|UAV\| = \|A\|$ for any unitary matrices U, V),

$$\begin{aligned}
&\|U_{t+1}U_t \dots U_1 - V_{t+1}V_t \dots V_1\| \\
&\quad = \|U_{t+1}U_t \dots U_1 - U_{t+1}V_t \dots V_1 + U_{t+1}V_t \dots V_1 - V_{t+1}V_t \dots V_1\| && (7)\\
&\quad \leq \|U_{t+1}U_t \dots U_1 - U_{t+1}V_t \dots V_1\| + \|U_{t+1}V_t \dots V_1 - V_{t+1}V_t \dots V_1\| && (8)\\
&\quad = \|U_{t+1}(U_t \dots U_1 - V_t \dots V_1)\| + \|(U_{t+1} - V_{t+1})V_t \dots V_1\| && (9)\\
&\quad = \|U_t \dots U_1 - V_t \dots V_1\| + \|U_{t+1} - V_{t+1}\| && (10)\\
&\quad \leq (t+1)\epsilon, && (11)
\end{aligned}$$

so the lemma follows by induction. □

Thus, in order to simulate a t-gate quantum circuit with total error at most ϵ, it suffices to simulate each individual gate with error at most ϵ/t.

To simulate an arbitrary individual gate, the strategy is to first construct a very fine net covering a very small ball around the identity using the *group commutator*,

$$[\![U, V]\!] := UVU^{-1}V^{-1}. \tag{12}$$

To approximate general unitaries, we will effectively translate them close to the identity.

Note that it suffices to consider unitary gates with determinant 1 (i.e., elements of SU(2)) since a global phase is irrelevant. Let

$$S_\epsilon := \{U \in \mathrm{SU}(2) \colon \|I - U\| \leq \epsilon\} \tag{13}$$

denote the ϵ-ball around the identity. Given sets $\Gamma, S \subseteq \mathrm{SU}(2)$, we say that Γ is an *ϵ-net* for S if for any $A \in S$, there is a $U \in \Gamma$ such that $\|A - U\| \leq \epsilon$. The following result (to be proved later on) indicates how the group commutator helps us to make a fine net around the identity.

Lemma. *If Γ is an ϵ^2-net for S_ϵ, then $[\![\Gamma, \Gamma]\!] := \{[\![U, V]\!] \colon U, V \in \Gamma\}$ is an $O(\epsilon^3)$-net for S_{ϵ^2}.*

To make an arbitrarily fine net, we apply this idea recursively. But first it is helpful to derive a consequence of the lemma that is more suitable for recursion. We would like to maintain the quadratic relationship between the size of the ball and the quality of the net. If we aim for a $k^2\epsilon^3$-net (for some constant k), we would like it to apply to arbitrary points in $S_{k\epsilon^{3/2}}$, whereas the lemma only lets us approximate points in S_{ϵ^2}. To handle an arbitrary $A \in S_{k\epsilon^{3/2}}$, we first let W be the closest gate in Γ to A. For sufficiently small ϵ we have $k\epsilon^{3/2} < \epsilon$, so $S_{k\epsilon^{3/2}} \subset S_\epsilon$, and therefore $A \in S_\epsilon$. Since Γ is an ϵ^2-net for S_ϵ, we have $\|A - W\| \leq \epsilon^2$, i.e., $\|AW^\dagger - I\| \leq \epsilon^2$, so $AW^\dagger \in S_{\epsilon^2}$. Then we can apply the lemma to find $U, V \in \Gamma$ such that $\|AW^\dagger - [\![U, V]\!]\| = \|A - [\![U, V]\!]W\| \leq k^2\epsilon^3$. In other words, if Γ is an ϵ^2-net for S_ϵ, then $[\![\Gamma, \Gamma]\!]\Gamma := \{[\![U, V]\!]W \colon U, V, W \in \Gamma\}$ is a $k^2\epsilon^3$-net for $S_{k\epsilon^{3/2}}$.

Now suppose that Γ_0 is an ϵ_0^2-net for S_{ϵ_0}, and let $\Gamma_i := [\![\Gamma_{i-1}, \Gamma_{i-1}]\!]\Gamma_{i-1}$ for all positive integers i. Then Γ_i is an ϵ_i^2-net for S_{ϵ_i}, where $\epsilon_i = k\epsilon_{i-1}^{3/2}$. Solving this recursion gives $\epsilon_i = (k^2\epsilon_0)^{(3/2)^i}/k^2$.

With these tools in hand, we are prepared to establish the main result.

Proof of the Solovay-Kitaev Theorem. It suffices to consider how to approximate an arbitrary $U \in \mathrm{SU}(2)$ to precision ϵ by a sequence of gates from a given universal gate set Γ.

First we take products of elements of Γ to form a new universal gate set Γ_0 that is an ϵ_0^2-net for SU(2), for some sufficiently small constant ϵ_0. We know this can be done since Γ is universal. Since ϵ_0 is a constant, the overhead in constructing Γ_0 is constant.

Now we can find $V_0 \in \Gamma_0$ such that $\|U - V_0\| \leq \epsilon_0^2$. Since $\|U - V_0\| = \|UV_0^\dagger - I\|$, we have $UV_0^\dagger \in S_{\epsilon_0^2}$. If ϵ_0 is sufficiently small, then $\epsilon_0^2 < k\epsilon_0^{3/2} = \epsilon_1$, so $UV_0^\dagger \in S_{\epsilon_1}$.

Since Γ_0 is an ϵ_0^2-net for SU(2), in particular it is an ϵ_0^2-net for S_{ϵ_0}. Thus by the above argument, Γ_1 is an ϵ_1^2-net for S_{ϵ_1}, so we can find $V_1 \in \Gamma_1$ such that $\|UV_0^\dagger - V_1\| \leq \epsilon_1^2 < k\epsilon_1^{3/2} = \epsilon_2$, i.e., $UV_0^\dagger V_1^\dagger - I \in S_{\epsilon_2}$.

In general, suppose we are given $V_0, V_1, \ldots, V_{i-1}$ such that $UV_0^\dagger V_1^\dagger \ldots V_{i-1}^\dagger \in S_{\epsilon_i}$. Since Γ_i is an ϵ_i^2-net for S_{ϵ_i}, we can find $V_i \in \Gamma_i$ such that $\|UV_0^\dagger V_1^\dagger \ldots V_{i-1}^\dagger - V_i\| \leq \epsilon_i^2$. In turn, this implies that $UV_0^\dagger V_1^\dagger \ldots V_i^\dagger \in S_{\epsilon_{i+1}}$.

Repeating this process t times gives a very good approximation of U by $V_t \ldots V_1 V_0$: we have $\|U - V_t \ldots V_1 V_0\| \leq \epsilon_t^2$. Suppose we consider a gate from Γ_0 to be elementary. (These gates can be implemented using only a constant number of gates from Γ, so there is a constant factor overhead if only count gates in Γ as elementary.) The number of elementary gates needed to implement a gate from Γ_i is 5^i, so the total number of gates in the approximation is $\sum_{i=0}^t 5^i = (5^{t+1} - 1)/4 = O(5^t)$. To achieve an overall error at most ϵ, we need $\epsilon_t^2 = ((k^2\epsilon_0)^{(3/2)^t}/k^2)^2 \leq \epsilon$, i.e.,

$$\left(\frac{3}{2}\right)^t > \frac{\frac{1}{2}\log(k^2\epsilon)}{\log(k^2\epsilon_0)}. \tag{14}$$

Thus the number of gates used is $O(\log^\nu \frac{1}{\epsilon})$ where $\nu = \log 5 / \log \frac{3}{2}$.

At this point, it may not be clear that the approximation can be found quickly, since Γ_i contains a large number of points, so we need to be careful about how we find a good approximation $V_i \in \Gamma_i$ of $UV_0^\dagger V_1^\dagger \ldots V_{i-1}^\dagger$. However, by constructing the approximation recursively, it can be shown that the running time of this procedure is poly($\log \frac{1}{\epsilon}$). It will be clearer how to do this after we prove the lemma, but we leave the details as an exercise. □

It remains to prove the lemma. A key idea is to move between the Lie group SU(2) and its Lie algebra, i.e., the Hamiltonians generating these unitaries. In particular, we can represent any $A \in \mathrm{SU}(2)$ as $A = e^{i\vec{a}\cdot\vec{\sigma}}$, where $\vec{a} \in \mathbb{R}^3$ and $\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_z)$ is a vector of Pauli matrices. Note that we can choose $\|\vec{a}\| \leq \pi$ without loss of generality.

In the proof, the following basic facts about SU(2) will be useful.

(i) $\|I - e^{i\vec{a}\cdot\vec{\sigma}}\| = 2\sin\frac{\|\vec{a}\|}{2} = \|\vec{a}\| + O(\|\vec{a}\|^3)$

(ii) $\|e^{i\vec{b}\cdot\vec{\sigma}} - e^{i\vec{c}\cdot\vec{\sigma}}\| \leq \|\vec{b} - \vec{c}\|$

(iii) $[\vec{b}\cdot\vec{\sigma}, \vec{c}\cdot\vec{\sigma}] = 2i(\vec{b}\times\vec{c})\cdot\vec{\sigma}$

(iv) $\|[\![e^{i\vec{b}\cdot\vec{\sigma}}, e^{i\vec{c}\cdot\vec{\sigma}}]\!] - e^{-[\vec{b}\cdot\vec{\sigma}, \vec{c}\cdot\vec{\sigma}]}\| = O(\|\vec{b}\|\|\vec{c}\|(\|\vec{b}\| + \|\vec{c}\|))$

Here the big-O notation is with respect to $\|\vec{a}\| \to 0$ in (i) and with respect to $\|\vec{b}\|, \|\vec{c}\| \to 0$ in (iv).

Proof of Lemma. Let $A \in S_{\epsilon^2}$. Our goal is to find $U, V \in \Gamma$ such that $\|A - [\![U, V]\!]\| = O(\epsilon^3)$.

Choose $\vec{a} \in \mathbb{R}^3$ such that $A = e^{i\vec{a}\cdot\vec{\sigma}}$. Since $A \in S_{\epsilon^2}$, by (i) we can choose $\vec{a}$ so that $\|\vec{a}\| = O(\epsilon^2)$.

Then choose $\vec{b}, \vec{c} \in \mathbb{R}^3$ such that $2\vec{b}\times\vec{c} = \vec{a}$. We can choose these vectors to be orthogonal and of equal length, so that $\|\vec{b}\| = \|\vec{c}\| = \sqrt{\|\vec{a}\|/2} = O(\epsilon)$. Let $B = e^{i\vec{b}\cdot\vec{\sigma}}$ and $C = e^{i\vec{c}\cdot\vec{\sigma}}$. Then

the only difference between A and $[\![B, C]\!]$ is the difference between the commutator and the group commutator, which is $O(\epsilon^3)$ by (iv).

However, we need to choose points from the net Γ. So let $U = e^{i\vec{u}\cdot\vec{\sigma}}$ be the closest element of Γ to B, and let $V = e^{i\vec{v}\cdot\vec{\sigma}}$ be the closest element of Γ to C. Since Γ is an ϵ^2-net for S_ϵ, we have $\|U - B\| \leq \epsilon^2$ and $\|V - C\| \leq \epsilon^2$, so in particular $\|\vec{u} - \vec{b}\| = O(\epsilon^2)$ and $\|\vec{v} - \vec{c}\| = O(\epsilon^2)$.

Now by the triangle inequality, we have

$$\|A - [\![U, V]\!]\| \leq \|A - e^{2i(\vec{u}\times\vec{v})\cdot\vec{\sigma}}\| + \|e^{2i(\vec{u}\times\vec{v})\cdot\vec{\sigma}} - [\![U, V]\!]\|. \tag{15}$$

For the first term, using (ii), we have

$$\begin{aligned}
\|A - e^{2i(\vec{u}\times\vec{v})\cdot\vec{\sigma}}\| &= \|e^{2i(\vec{b}\times\vec{c})\cdot\vec{\sigma}} - e^{2i(\vec{u}\times\vec{v})\cdot\vec{\sigma}}\| && (16)\\
&\leq 2\|\vec{b}\times\vec{c} - \vec{u}\times\vec{v}\| && (17)\\
&= 2\|(\vec{b} - \vec{u} + \vec{u})\times(\vec{c} - \vec{v} + \vec{v}) - \vec{u}\times\vec{v}\| && (18)\\
&= 2\|(\vec{b} - \vec{u})\times(\vec{c} - \vec{v}) + (\vec{b} - \vec{u})\times\vec{v} + \vec{u}\times(\vec{c} - \vec{v})\| && (19)\\
&= O(\epsilon^3). && (20)
\end{aligned}$$

For the second term, using (iii) and (iv) gives

$$\|e^{2i(\vec{u}\times\vec{v})\cdot\vec{\sigma}} - [\![U, V]\!]\| = \|e^{-[\vec{u}\cdot\vec{\sigma}, \vec{v}\cdot\vec{\sigma}]} - [\![U, V]\!]\| = O(\epsilon^3) \tag{21}$$

The lemma follows. □

Note that it is possible to improve the construction somewhat over the version described above. Furthermore, it can be generalized to $\mathrm{SU}(N)$ for arbitrary N. In general, the cost is exponential in N^2, but for any fixed N this is just a constant.

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: if we can efficiently implement the map $x \mapsto f(x)$ on a classical computer, we can efficiently perform the transformation $|x, y\rangle \mapsto |x, y \oplus f(x)\rangle$ on a quantum computer. This transformation can be applied to any superposition of computational

basis states, so for example, we can perform the transformation

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x, 0\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x, f(x)\rangle. \tag{22}$$

Note that this does not necessarily mean we can efficiently implement the map $|x\rangle \mapsto |f(x)\rangle$, even when f is a bijection (so that this is indeed a unitary transformation). However, if we can efficiently invert f, then we can indeed do this efficiently.

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 $f\colon \mathbb{Z}_2^n \to S$ 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 *depth*, the maximum number of gates on any path from an input to an output. The depth 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.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 2: The abelian QFT, phase estimation, and discrete log

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 for now we will focus on the case of an abelian group G. Here the transformation is

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

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 $G = \mathbb{Z}_2^n$. The characters of this group are $\chi_y(x) = (-1)^{x\cdot y}$, so the QFT is simply

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

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 $G = \mathbb{Z}_{2^n}$:

$$F_{\mathbb{Z}_{2^n}} = \frac{1}{\sqrt{2^n}} \sum_{x,y\in\mathbb{Z}_{2^n}} \omega_{2^n}^{xy} |y\rangle\langle x| \tag{3}$$

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, $x = x_{n-1}\dots x_1 x_0$, and to consider how an input basis vector is transformed:

$$|x\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{y\in\mathbb{Z}_{2^n}} \omega_{2^n}^{xy} |y\rangle \tag{4}$$

$$= \frac{1}{\sqrt{2^n}} \sum_{y\in\mathbb{Z}_{2^n}} \omega_{2^n}^{x(\sum_{k=0}^{n-1} y_k 2^k)} |y_{n-1}\dots y_1 y_0\rangle \tag{5}$$

$$= \frac{1}{\sqrt{2^n}} \sum_{y\in\mathbb{Z}_{2^n}} \prod_{k=0}^{n-1} \omega_{2^n}^{x y_k 2^k} |y_{n-1}\dots y_1 y_0\rangle \tag{6}$$

$$= \frac{1}{\sqrt{2^n}} \bigotimes_{k=0}^{n-1} \sum_{y_k\in\mathbb{Z}_2} \omega_{2^n}^{x y_k 2^k} |y_k\rangle \tag{7}$$

$$= \bigotimes_{k=0}^{n-1} |z_k\rangle \tag{8}$$

where

$$|z_k\rangle := \frac{1}{\sqrt{2}} \sum_{y_k \in \mathbb{Z}_2} \omega_{2^n}^{x y_k 2^k} |y_k\rangle \tag{9}$$

$$= \frac{1}{\sqrt{2}} (|0\rangle + \omega_{2^n}^{x 2^k} |1\rangle) \tag{10}$$

$$= \frac{1}{\sqrt{2}} (|0\rangle + \omega_{2^n}^{\sum_{j=0}^{n-1} x_j 2^{j+k}} |1\rangle) \tag{11}$$

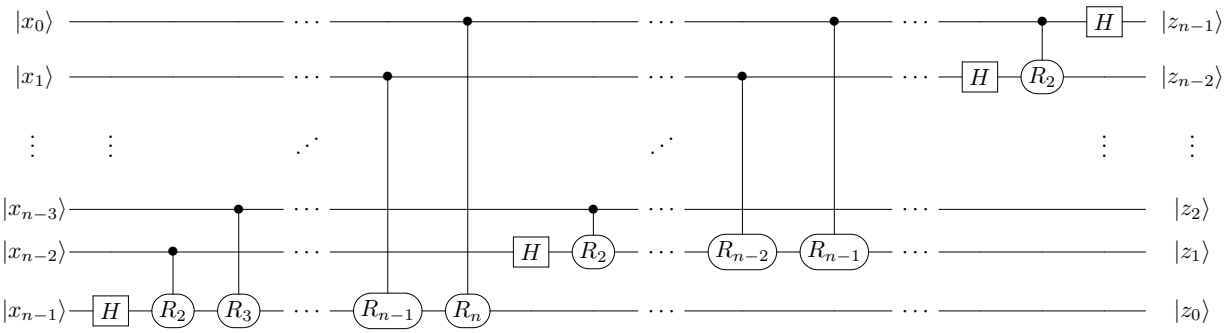
$$= \frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi \mathrm{i}(x_0 2^{k-n} + x_1 2^{k+1-n} + \cdots + x_{n-1-k} 2^{-1})} |1\rangle). \tag{12}$$

(A more succinct way to write this is $|z_k\rangle = \frac{1}{\sqrt{2}}(|0\rangle + \omega_{2^{n-k}}^{x} |1\rangle)$, but the above expression is more helpful for understanding the circuit.) 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

$$R_k := \begin{pmatrix} 1 & 0 \\ 0 & \omega_{2^k} \end{pmatrix}. \tag{13}$$

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 lets us apply 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 $U|\phi\rangle = e^{\mathrm{i}\phi}|\phi\rangle$ 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

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \mathbb{Z}_{2^n}} |x, \phi\rangle, \tag{14}$$

apply the operator

$$\sum_{x \in \mathbb{Z}_{2^n}} |x\rangle\langle x| \otimes U^x \tag{15}$$

to give the state

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \mathbb{Z}_{2^n}} e^{\mathrm{i}\phi x} |x, \phi\rangle, \tag{16}$$

apply an inverse Fourier transform on the first register, and measure. If the binary expansion of $\phi/2\pi$ terminates after at most n bits (i.e., if $\phi = 2\pi y/2^n$ for some $y \in \mathbb{Z}_{2^n}$), then the state (16) is $F_{2^n}|y\rangle \otimes |\phi\rangle$, so the result is guaranteed to be the binary expansion of $\phi/2\pi$. In general, we obtain a good approximation with high probability. In particular, the probability of obtaining the result y (corresponding to the estimate $2\pi y/2^n$ for the phase) is

$$\Pr(y) = \frac{1}{2^{2n}} \cdot \frac{\sin^2(2^{n-1}\phi)}{\sin^2(\frac{\phi}{2} - \frac{\pi y}{2^n})}, \tag{17}$$

which is strongly peaked around the best n-bit approximation (in particular, it gives the best n-bit approximation with probability at least $4/\pi^2$). We will see the details of a similar calculation when we discuss period finding.

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 $\mathbb{Z}_N$:

$$F_{\mathbb{Z}_N} = \frac{1}{\sqrt{N}} \sum_{x,y \in \mathbb{Z}_N} \omega_N^{xy} |y\rangle\langle x|. \tag{18}$$

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 $|x, 0\rangle \mapsto |x, \tilde{x}\rangle$; then it remains to erase the register $|x\rangle$ from such a state.

Consider the unitary operator that adds 1 modulo N:

$$U := \sum_{x \in \mathbb{Z}_N} |x+1\rangle\langle x|. \tag{19}$$

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)

$$F_{\mathbb{Z}_N}^\dagger U F_{\mathbb{Z}_N} = \sum_{x \in \mathbb{Z}_N} \omega_N^x |x\rangle\langle x|. \tag{20}$$

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

$$|\tilde{x}, 0\rangle \mapsto |\tilde{x}, x\rangle \tag{21}$$

(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.

Discrete log

One of the applications of the QFT over a cyclic group is to the solution of the discrete log problem. This problem is defined as follows. Let $G = \langle g \rangle$ be a cyclic group generated by g, written multiplicatively. Given an element $x \in G$, the *discrete logarithm of x in G with respect to g*, denoted $\log_g x$, is the smallest non-negative integer α such that $g^\alpha = x$. The *discrete logarithm problem* is the problem of calculating $\log_g x$.

Here are some simple examples of discrete logarithms:

- For any $G = \langle g \rangle$, $\log_g 1 = 0$
- For $G = \mathbb{Z}_7^\times$, $\log_3 2 = 2$
- For $G = \mathbb{Z}_{541}^\times$, $\log_{126} 282 = 101$

The discrete logarithm seems like a good candidate for a one-way function. We can efficiently compute g^α, even if α is exponentially large (in $\log |G|$), using repeated squaring. But given x, it is not immediately clear how to compute $\log_g x$, other than by checking exponentially many possibilities. (There are better algorithms than brute force search, but none is known that works in polynomial time.) This is the basis of the well-known Diffie-Hellman key exchange protocol.

Shor's algorithm

Now we will see how Shor's algorithm can be used to calculate discrete logarithms. This is a nice example because it's simpler than the factoring algorithm, but the problem it solves is actually at least as hard: factoring N can be reduced to calculating discrete log in $\mathbb{Z}_N^\times$. (Unfortunately, this does not by itself give a quantum algorithm for factoring, because Shor's algorithm for discrete log in G requires us to know the order of G—but computing $|\mathbb{Z}_N^\times| = \phi(N)$ is as hard as factoring N.)

Given some element x of a cyclic group $G = \langle g \rangle$, we would like to calculate $\log_g x$, the smallest integer α such that $g^\alpha = x$. For simplicity, let us assume that the order of G, $N := |G|$, is known. (For example, if $G = \mathbb{Z}_p^\times$ for prime p, then we know $N = p - 1$. In general, if we do not know N, we can learn it using Shor's period-finding algorithm, which we'll review later.) We can also assume that $x \neq g$ (i.e., $\log_g x \neq 1$), since it is easy to check whether this is the case.

To approach this problem, consider the function $f \colon \mathbb{Z}_N \times \mathbb{Z}_N \to G$ as follows:

$$f(\alpha, \beta) = x^\alpha g^\beta. \tag{22}$$

Since $f(\alpha, \beta) = g^{\alpha \log_g x + \beta}$, f is constant on the lines

$$L_\gamma := \{(\alpha, \beta) \in \mathbb{Z}_N^2 \colon \alpha \log_g x + \beta = \gamma\}. \tag{23}$$

Shor's algorithm for finding $\log_g x$ proceeds as follows. We start from the uniform superposition over $\mathbb{Z}_N \times \mathbb{Z}_N$ and compute the hiding function in another register:

$$|\mathbb{Z}_N \times \mathbb{Z}_N\rangle := \frac{1}{N} \sum_{\alpha,\beta \in \mathbb{Z}_N} |\alpha, \beta\rangle \mapsto \frac{1}{N} \sum_{\alpha,\beta \in \mathbb{Z}_N} |\alpha, \beta, f(\alpha, \beta)\rangle. \tag{24}$$

Then we discard the third register. To see what this does, it may be conceptually helpful to imagine that we actually measure the third register. Then the post-measurement state is a superposition over group elements consistent with the observed function value (say, g^δ), which is simply the set of points on some line L_δ. In other words, we get the state

$$|L_\delta\rangle = \frac{1}{\sqrt{N}} \sum_{\alpha \in \mathbb{Z}_N} |\alpha, \delta - \alpha \log_g x\rangle \tag{25}$$

However, note that the measurement outcome is unhelpful: each possible value occurs with equal probability, and we cannot obtain δ from g^δ unless we know how to take discrete logarithms. Thus we may as well simply discard the third register, leaving the system in the mixed state described by the ensemble of pure states (25) where δ is uniformly random and unknown.

Now we can exploit the symmetry of the quantum state by performing a QFT over $\mathbb{Z}_N \times \mathbb{Z}_N$; then the state becomes

$$\frac{1}{N^{3/2}} \sum_{\alpha,\mu,\nu \in \mathbb{Z}_N} \omega_N^{\mu\alpha + \nu(\delta - \alpha \log_g x)} |\mu, \nu\rangle = \frac{1}{N^{3/2}} \sum_{\mu,\nu \in \mathbb{Z}_N} \omega_N^{\nu\delta} \sum_{\alpha \in \mathbb{Z}_N} \omega_N^{\alpha(\mu - \nu \log_g x)} |\mu, \nu\rangle, \tag{26}$$

and using the identity $\sum_{\alpha \in \mathbb{Z}_N} \omega_N^{\alpha\beta} = N\delta_{\beta,0}$, we have

$$\frac{1}{\sqrt{N}} \sum_{\nu \in \mathbb{Z}_N} \omega_N^{\nu\delta} |\nu \log_g x, \nu\rangle. \tag{27}$$

Now suppose we measure this state in the computational basis. Then we obtain some pair $(\nu \log_g x, \nu)$ for uniformly random $\nu \in \mathbb{Z}_N$. If ν has a multiplicative inverse modulo N, we can divide the first register by ν to get the desired answer. If ν does not have a multiplicative inverse, we simply repeat the entire procedure again. The probability of success for each independent attempt is $\phi(N)/N = \Omega(1/\log\log N)$ (where $\phi(N)$ denotes the number of positive integers less than and relatively prime to n), so we don't have to repeat the procedure many times before we find an invertible ν.

This algorithm can be carried out for any cyclic group G so long as we have a unique representation of the group elements, and we are able to efficiently compute products in G. (We need to be able to compute high powers of a group element, but recall that this can be done quickly by repeated squaring.) In particular, it can also be used to solve the discrete log problem for elliptic curves, thus compromising most elliptic curve cryptography.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 3: The abelian hidden subgroup problem

In this lecture, we will introduce the general hidden subgroup problem (HSP). We'll see how Shor's discrete log algorithm solves a particular instance of the HSP in an Abelian group. Finally, we'll see how to solve the HSP in any finite abelian group of known structure.

The hidden subgroup problem

In the general HSP, we are given a black-box function $f\colon G \to S$, where G is a known group and S is a finite set. The function is promised to satisfy

$$f(x) = f(y) \text{ if and only if } x^{-1}y \in H \quad \text{i.e., } y = xh \text{ for some } h \in H \tag{1}$$

for some unknown subgroup $H \leq G$. We say that such a function *hides* H. The goal of the HSP is to learn H (say, specified in terms of a generating set) using queries to f.

It's clear that H can in principle be reconstructed if we are given the entire truth table of f. Notice in particular that $f(1) = f(x)$ if and only if $x \in H$: the hiding function is constant on the hidden subgroup, and does not take that value anywhere else.

But the hiding function has a lot more structure than this. If we fix some element $g \in G$ with $g \notin H$, we see that $f(g) = f(x)$ if and only if $x \in gH$, a left coset of H in G with coset representative g. So f is constant on the left cosets of H in G, and distinct on different left cosets.

In the above definition of the HSP, we have made an arbitrary choice to multiply by elements of H on the right, which is why the hiding function is constant on left cosets. We could just as well have chosen to multiply by elements of H on the left, in which case the hiding function would be constant on right cosets; the resulting problem would be equivalent. Of course, in the case where G is abelian, we don't need to make such a choice. For reasons that we will see later, this case turns out to be considerably simpler than the general case; indeed, there is an efficient quantum algorithm for the HSP in any abelian group, whereas there are only a few nonabelian groups for which efficient algorithms are known.

You should be familiar with Simon's problem, which is simply the HSP with $G = \mathbb{Z}_2^n$ and $H = \{0, s\}$ for some $s \in \mathbb{Z}_2^n$. There is a straightforward quantum algorithm for this problem, yet one can prove that any classical algorithm for finding s must query the hiding function exponentially many times (in n). The gist of the argument is that, since the set S is unstructured, we can do no better than querying random group elements so long as we do not know two elements x, y for which $f(x) = f(y)$. But by the birthday problem, we are unlikely to see such a collision until we make $\Omega(\sqrt{|G|/|H|})$ random queries.

A similar argument applies to any HSP with a large number of trivially intersecting subgroups. More precisely, we have

Theorem. *Suppose that G has a set $\mathcal{H}$ of N subgroups whose only common element is the identity. Then a classical computer must make $\Omega(\sqrt{N})$ queries to solve the HSP.*

Proof. Suppose the oracle does not a priori hide a particular subgroup, but instead behaves adversarially, as follows. On the ℓth query, the algorithm queries g_ℓ, which we assume to be different

from $g_1, \ldots, g_{\ell-1}$ without loss of generality. If there is any subgroup $H \in \mathcal{H}$ for which $g_k \notin g_j H$ for all $1 \leq j < k \leq \ell$ (i.e., there is some consistent way the oracle could assign g_ℓ to an as-yet-unqueried coset of a hidden subgroup from $\mathcal{H}$), then the oracle simply outputs ℓ; otherwise the oracle conceeds defeat and outputs a generating set for some $H \in \mathcal{H}$ consistent with its answers so far (which must exist, by construction).

The goal of the algorithm is to force the oracle to conceed, and we want to lower bound the number of queries required. (Given an algorithm for the HSP in G, there is clearly an algorithm that forces this oracle to conceed using only one more query.) Now consider an algorithm that queries the oracle t times before forcing the oracle to conceed. This algorithm simply sees a fixed sequence of responses $1, 2, \ldots, t$, so for the first t queries, the algorithm cannot be adaptive. But observe that, regardless of which t group elements are queried, there are at most $\binom{t}{2}$ values of $g_k g_j^{-1}$, whereas there are N possible subgroups in $\mathcal{H}$. Thus, to satisfy the N conditions that for all $H \in \mathcal{H}$, there is some pair j, k such that $g_k g_j^{-1} \in H$, we must have $\binom{t}{2} \geq N$, i.e., $t = \Omega(\sqrt{N})$. $\square$

Note that there are cases where a classical algorithm *can* find the hidden subgroup with a polynomial number of queries. In particular, since a classical computer can easily test whether a certain subgroup is indeed the hidden one, the HSP is easy for a group with only polynomially many subgroups. For example, a classical computer can easily solve the HSP in $\mathbb{Z}_p$ for p prime (since it has only 2 subgroups) and in $\mathbb{Z}_{2^n}$ (since it has only $n+1$ subgroups).

Discrete log as a hidden subgroup problem

The discrete log problem is easily recognized as an HSP. Recall that Shor's algorithm for computing $\log_g x$ involves the function $f\colon \mathbb{Z}_N \times \mathbb{Z}_N \to \langle g \rangle$ defined by $f(\alpha, \beta) = x^\alpha g^\beta$. This function is constant on the lines $L_\gamma = \{(\alpha, \beta) \in \mathbb{Z}_N^2 \colon \alpha \log_g x + \beta = \gamma\}$. Observe that $H = L_0$ is a subgroup of $G = \mathbb{Z}_N \times \mathbb{Z}_N$, and the sets $L_\gamma = L_0 + (0, \gamma)$ are its cosets. Shor's algorithm for discrete log works by making the coset state $|L_\gamma\rangle$ for a uniformly random γ and measuring in the Fourier basis.

The abelian HSP

We now consider the HSP for a general abelian group. When the group elements commute, it often makes more sense to use additive notation for the group operation. We use this convention here, writing the condition that f hides H as $f(x) = f(x)$ iff $x - y \in H$.

The strategy for the general abelian HSP closely follows the algorithm for the discrete log problem. We begin by creating a uniform superposition over the group,

$$|G\rangle := \frac{1}{\sqrt{|G|}} \sum_{x \in G} |x\rangle. \tag{2}$$

Then we compute the function value in another register, giving

$$\frac{1}{\sqrt{|G|}} \sum_{x \in G} |x, f(x)\rangle. \tag{3}$$

Discarding the second register then gives a uniform superposition over the elements of some randomly chosen coset $x + H := \{x + h \colon h \in H\}$ of H in G,

$$|x + H\rangle = \frac{1}{\sqrt{|H|}} \sum_{h \in H} |x + h\rangle. \tag{4}$$

Such a state is commonly called a *coset state*. Equivalently, since the coset is unknown and uniformly random, the state can be described by the density matrix

$$\rho_H := \frac{1}{|G|} \sum_{x \in G} |x+H\rangle\langle x+H|. \tag{5}$$

Next we apply the QFT over G. Then we obtain the state

$$\begin{aligned} |\widehat{x+H}\rangle &:= F_G |x+H\rangle & (6)\\ &= \frac{1}{\sqrt{|H| \cdot |G|}} \sum_{y \in \hat{G}} \sum_{h \in H} \chi_y(x+h)|y\rangle & (7)\\ &= \sqrt{\frac{|H|}{|G|}} \sum_{y \in \hat{G}} \chi_y(x)\chi_y(H)|y\rangle & (8) \end{aligned}$$

where

$$\chi_y(H) := \frac{1}{|H|} \sum_{h \in H} \chi_y(h). \tag{9}$$

Note that applying the QFT was the right thing to do because the state ρ_H is G-invariant. In other words, it commutes with the regular representation of G, the unitary matrices $U(x)$ satisfying $U(x)|y\rangle = |x+y\rangle$ for all $x, y \in G$: we have

$$\begin{aligned} U(x)\rho_H &= \frac{1}{|G|} \sum_{y \in G} |x+y+H\rangle\langle y+H| & (10)\\ &= \frac{1}{|G|} \sum_{z \in G} |z+H\rangle\langle z-x+H| & (11)\\ &= \rho_H U(-x)^\dagger & (12)\\ &= \rho_H U(x). & (13) \end{aligned}$$

It follows that $\hat{\rho}_H := F_G \rho_H F_G^\dagger$ is diagonal (indeed, we verify this explicitly below), so we can measure without losing any information. We will talk about this phenomenon more when we discuss nonabelian Fourier sampling.

Note that χ_y is a character of H if we restrict our attention to that subgroup. If $\chi_y(h) = 1$ for all $h \in H$, then clearly $\chi_y(H) = 1$. On the other hand, if there is any $h \in H$ with $\chi_y(h) \neq 1$ (i.e., if the restriction of χ_y to H is not the trivial character of H), then by the orthogonality of distinct characters,

$$\frac{1}{|H|} \sum_{x \in H} \chi_y(x)\chi_{y'}(x)^* = \delta_{y,y'} \tag{14}$$

(equivalent to unitarity of the QFT), we have $\chi_y(H) = 0$. Thus we have

$$|\widehat{x+H}\rangle = \sqrt{\frac{|H|}{|G|}} \sum_{y:\ \chi_y(H)=1} \chi_y(x)|y\rangle \tag{15}$$

or, equivalently, the mixed quantum state

$$\hat{\rho}_H = \frac{|H|}{|G|^2} \sum_{x \in G} \sum_{y,y' \colon \chi_y(H) = \chi_{y'}(H) = 1} \chi_y(x) \chi_{y'}(x) |y\rangle\langle y'| = \frac{|H|}{|G|} \sum_{y \colon \chi_y(H) = 1} |y\rangle\langle y|. \tag{16}$$

Next we measure in the computational basis. Then we obtain some character χ_y that is trivial on the hidden subgroup H. This information narrows down the possible elements of the hidden subgroup: we can restrict our attention to those elements $g \in G$ satisfying $\chi_y(g) = 1$. The set of such elements is called the *kernel* of χ_y,

$$\ker \chi_y := \{g \in G \colon \chi_y(g) = 1\}; \tag{17}$$

it is a subgroup of G. Now our strategy is to repeat the entire sampling procedure many times and compute the intersection of the kernels of the resulting characters. After only polynomially many steps, we claim that the resulting subgroup is H with high probability. It clearly cannot be smaller than H (since the kernel of every sampled character contains H), so it suffices to show that each sample is likely to reduce the size of H by a substantial fraction until H is reached.

Suppose that at some point in this process, the intersection of the kernels is $K \leq G$ with $K \neq H$. Since K is a subgroup of G with $H < K$, we have $|K| \geq 2|H|$ (by Lagrange's theorem). Because each character χ_y of G satisfying $\chi_y(H)$ has probability $|H|/|G|$ of appearing, the probability that we see some χ_y for which $K \leq \ker \chi_y$ is

$$\frac{|H|}{|G|} |\{y \in \hat{G} \colon K \leq \ker \chi_y\}|. \tag{18}$$

But the number of such ys is precisely $|G|/|K|$, since we know that if the subgroup K were hidden, we would sample such ys uniformly, with probability $|K|/|G|$. Therefore the probability that we see a y for which $K \leq \ker \chi_y$ is precisely $|H|/|K| \leq 1/2$. Now if we observe a y such that $K \not\leq \ker \chi_y$, then $|K \cap \ker \chi_y| \leq |K|/2$; furthermore, this happens with probability at least $1/2$. Thus, if we repeat the process $O(\log |G|)$ times, it is extremely likely that the resulting subgroup is in fact H.

Decomposing abelian groups

To apply the above algorithm, we must understand the structure of the group G; in particular, we must be able to apply the Fourier transform F_G. For some applications, we might not know the structure of G a priori. But if we assume only that we have a unique encoding of each element of G, the ability to perform group operations on these elements, and a generating set for G, then there is an efficient quantum algorithm (due to Mosca) that decomposes the group as

$$G = \langle \gamma_1 \rangle \oplus \langle \gamma_2 \rangle \oplus \cdots \oplus \langle \gamma_t \rangle \tag{19}$$

in terms of generators $\gamma_1, \gamma_2, \ldots, \gamma_t$. Here $\oplus$ denotes an internal direct sum, meaning that the groups $\langle \gamma_i \rangle$ intersect only in the identity element; in other words, we have

$$G \cong \mathbb{Z}_{|\langle \gamma_1 \rangle|} \times \mathbb{Z}_{|\langle \gamma_2 \rangle|} \times \cdots \times \mathbb{Z}_{|\langle \gamma_t \rangle|}. \tag{20}$$

Given such a decomposition, it is straightforward to implement F_G and thereby solve HSPs in G. We might also use this tool to decompose the structure of the hidden subgroup H output by the HSP algorithm, e.g., to compute $|H|$.

This algorithm is based on Shor's algorithm for order finding, together with standard tools from group theory. We will not have time to cover the algorithm in detail; for more, see the lecture notes from 2011.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 4: Period finding from $\mathbb{Z}$ to $\mathbb{R}$

In this lecture, we will explore quantum algorithms for determining the period of a function. Shor's factoring algorithm is based on a solution of the period-finding problem for a function over the integers. More recently, Hallgren considered the problem of solving a quadratic Diophantine equation known as *Pell's equation*, together with related problems involving number fields. Hallgren gave efficient quantum algorithms for these problems by generalizing period finding over the integers to period finding over the real numbers.

Factoring and order finding

Shor's factoring algorithm is based on a reduction of factoring to order finding (observed by Miller in the 1970s). This reduction is typically covered in a first course on quantum computing, so we will not discuss the details here.

In the order finding problem for a group G, we are given an element $g \in G$ and our goal is to find the order of g, the smallest $r \in \mathbb{N}$ such that $g^r = 1$. (Factoring L reduces to order finding in $G = \mathbb{Z}_L$.) One way to approach this problem is to consider the function $f\colon \mathbb{Z} \to G$ defined by $f(x) = g^x$. This function is periodic with period r, and there is an efficient quantum algorithm to find this period, which we review below.

Pell's equation

Given a squarefree integer d (i.e., an integer not divisible by any perfect square), the Diophantine equation

$$x^2 - dy^2 = 1 \tag{1}$$

is known as *Pell's equation*. This equation was already studied in ancient India and Greece, and is closely related to concepts in algebraic number theory.

The left-hand side of Pell's equation can be factored as

$$x^2 - dy^2 = (x + y\sqrt{d})(x - y\sqrt{d}). \tag{2}$$

Note that a solution of the equation $(x, y) \in \mathbb{Z}^2$ can be encoded uniquely as the real number $x+y\sqrt{d}$: since $\sqrt{d}$ is irrational, $x + y\sqrt{d} = w + z\sqrt{d}$ if and only if $(x, y) = (w, z)$. (Proof: $\frac{x-w}{z-y} = \sqrt{d}$.) Thus we can also refer to the number $x + y\sqrt{d}$ as a solution of Pell's equation.

There is clearly no loss of generality in restricting our attention to *positive* solutions of the equation, namely those for which $x > 0$ and $y > 0$. It is straightforward to show that if $x_1 + y_1\sqrt{d}$ is a positive solution, then $(x_1 + y_1\sqrt{d})^n$ is also a positive solution for any $n \in \mathbb{N}$. In fact, one can show that *all* positive solutions are obtained in this way, where $x_1 + y_1\sqrt{d}$ is the *fundamental solution*, the smallest positive solution of the equation. Thus, even though Pell's equation has an infinite number of solutions, we can in a sense find them all by finding the fundamental solution.

Unfortunately, it is not feasible to find the fundamental solution explicitly. The solutions can be very large—the size of $x_1 + y_1\sqrt{d}$ is only upper bounded by $2^{O(\sqrt{d}\log d)}$. Thus it is not even possible to *write down* the fundamental solution with $\mathrm{poly}(\log d)$ bits.

To get around this difficulty, we define the *regulator* of the fundamental solution,

$$R := \ln(x_1 + y_1\sqrt{d}). \tag{3}$$

Since $R = O(\sqrt{d}\log d)$, we can write down $\lceil R \rceil$ using $O(\log d)$ bits. Now R is an irrational number, so determining only its integer part may seem unsatisfactory. But in fact, given the integer part of R, there is a classical algorithm to compute n digits of R in time $\mathrm{poly}(\log d, n)$. Thus it suffices to give an algorithm that finds the integer part of R in time $\mathrm{poly}(\log d)$. The best known classical algorithm for this problem takes time $2^{O(\sqrt{\log d \log\log d})}$ assuming the generalized Riemann hypothesis, or time $O(d^{1/4}\,\mathrm{poly}(\log d))$ with no such assumptions.

Hallgren's algorithm for solving Pell's equation is based on defining an efficiently computable periodic function whose period is the regulator. Defining this function would require us to introduce a substantial amount of algebraic number theory, so we omit the details here (for a partial account, see the lecture notes from 2011; for a more thorough treatment, see the review article by Jozsa). Instead, we will focus on the quantum part of the algorithm, which solves the period-finding problem.

Period finding over the integers

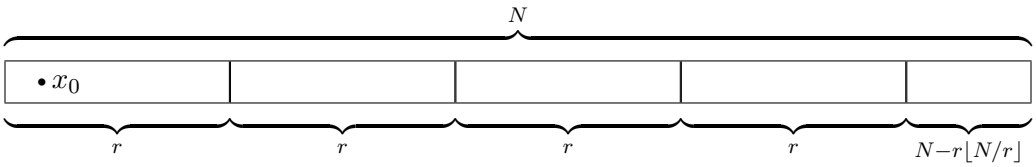
Recall that Shor's algorithm for factoring the number L works by finding the period of the function $f\colon \mathbb{Z} \to \mathbb{Z}_L$ defined by $f(x) = a^x \bmod L$ (where a is chosen at random). In other words, we are trying to find the smallest positive integer r such that $a^x \bmod L = a^{x+r} \bmod L$ for all $x \in \mathbb{Z}$. Note that since the period does not, in general, divide a known number N, we cannot simply reduce this task to period finding over $\mathbb{Z}_N$; rather, we should really think of it as period finding over $\mathbb{Z}$ (or, equivalently, the hidden subgroup problem over $\mathbb{Z}$).

Of course, we cannot hope to represent arbitrary integers on a computer with finitely many bits of memory. Instead, we will consider the function only on the inputs $\{0, 1, \ldots, N-1\}$ for some chosen N, and we will perform Fourier sampling over $\mathbb{Z}_N$. We will see that this procedure can work even when the function is not precisely periodic over $\mathbb{Z}_N$. Of course, this can only have a chance of working if the period is sufficiently small, since otherwise we could miss the period entirely. Later, we will see how to choose N if we are given an a priori upper bound of M on the period. If we don't initially have such a bound, we can simply start with $M = 2$ and repeatedly double M until it's large enough for period finding to work. The overhead incurred by this procedure is only $\mathrm{poly}(\log r)$.

Given a value of N, we prepare a uniform superposition over $\{0, 1, \ldots, N-1\}$ and compute the function in another register, giving

$$\frac{1}{\sqrt{N}} \sum_{x \in \{0,\ldots,N-1\}} |x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{x \in \{0,\ldots,N-1\}} |x, f(x)\rangle. \tag{4}$$

Next we measure the second register, leaving the first register in a uniform superposition over those values consistent with the measurement outcome. When f is periodic with minimum period r, we obtain a superposition over points separated by the period r. The number of such points, n, depends on where the first point, $x_0 \in \{0, 1, \ldots, r-1\}$, appears. When restricted to $\{0, 1, \ldots, N-1\}$, the function has $\lfloor N/r \rfloor$ full periods and $N - r\lfloor N/r \rfloor$ remaining points, as depicted below. Thus $n = \lfloor N/r \rfloor + 1$ if $x_0 < N - r\lfloor N/r \rfloor$ and $n = \lfloor N/r \rfloor$ otherwise.



Discarding the measurement outcome, we are left with the quantum state

$$\frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} |x_0 + jr\rangle \tag{5}$$

where x_0 occurs nearly uniformly random (it appears with probability n/N) and is unknown. To obtain information about the period, we apply the Fourier transform over $\mathbb{Z}_N$, giving

$$\frac{1}{\sqrt{nN}} \sum_{j=0}^{n-1} \sum_{k \in \mathbb{Z}_N} \omega_N^{k(x_0+jr)} |k\rangle = \frac{1}{\sqrt{nN}} \sum_{k \in \mathbb{Z}_N} \omega_N^{kx_0} \sum_{j=0}^{n-1} \omega_N^{jkr} |k\rangle. \tag{6}$$

Now if we were lucky enough to choose a value of N for which $r \mid N$, then in fact $n = N/r$ regardless of the value of x_0, and the sum over j above is

$$\sum_{j=0}^{n-1} \omega_N^{jkr} = \sum_{j=0}^{n-1} \omega_n^{jk} \tag{7}$$

$$= n\delta_{k \bmod n, 0}. \tag{8}$$

In this especially simple case, the quantum state is

$$\frac{n}{\sqrt{nN}} \sum_{k \in \mathbb{Z}_N} \omega_N^{kx_0} \delta_{k \bmod n, 0} = \frac{1}{\sqrt{r}} \sum_{k \in n\mathbb{Z}_r} \omega_N^{kx_0} |k\rangle, \tag{9}$$

and measurement of k is guaranteed to give an integer multiple of $n = N/r$, with each of the r multiples occurring with probability $1/r$. But more generally, the sum over j in (6) is the geometric series

$$\sum_{j=0}^{n-1} \omega_N^{jkr} = \frac{\omega_N^{krn} - 1}{\omega_N^{kr} - 1} \tag{10}$$

$$= \omega_N^{(n-1)kr/2} \frac{\sin \frac{\pi krn}{N}}{\sin \frac{\pi kr}{N}}. \tag{11}$$

The probability of seeing a particular value k is given by the normalization factor $1/nN$ times the magnitude squared of this sum, namely

$$\Pr(k) = \frac{\sin^2 \frac{\pi krn}{N}}{nN \sin^2 \frac{\pi kr}{N}}. \tag{12}$$

From the case where $n = N/r$, we expect this distribution to be strongly peaked around values of k that are close to integer multiples of N/r. The probability of seeing $k = \lfloor jN/r \rceil = jN/r + \epsilon$ for some $j \in \mathbb{Z}$, where $\lfloor x \rceil$ denotes the nearest integer to x, is

$$\Pr(k = \lfloor jN/r \rceil) = \frac{\sin^2(\pi jn + \frac{\pi \epsilon rn}{N})}{nN \sin^2(\pi j + \frac{\pi \epsilon r}{N})} \tag{13}$$

$$= \frac{\sin^2 \frac{\pi \epsilon rn}{N}}{nN \sin^2 \frac{\pi \epsilon r}{N}}. \tag{14}$$

Now using the inequalities $4x^2/\pi^2 \leq \sin^2 x \leq x^2$ (where the lower bound holds for $|x| \leq \pi/2$, and can be applied since $|\epsilon| \leq 1/2$), we have

$$\Pr(k = \lfloor jN/r \rceil) \geq \frac{4(\frac{\epsilon r n}{N})^2}{nN(\frac{\pi \epsilon r}{N})^2} \tag{15}$$

$$= \frac{4n}{\pi^2 N} \tag{16}$$

$$= \frac{4}{\pi^2 r}. \tag{17}$$

This bound shows that Fourier sampling produces a value of k that is the closest integer to one of the r integer multiples of N/r with probability lower bounded by a constant.

To discover r given one of the values $\lfloor jN/r \rceil$, we can divide by N to obtain a rational approximation to j/r that deviates by at most $1/2N$. Then consider the continued fraction expansion

$$\frac{\lfloor jN/r \rceil}{N} = \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}}. \tag{18}$$

Truncating this expansion after a finite number of terms gives a *convergent* of the expansion. The convergents provide a sequence of successively better approximations to $\lfloor jN/r \rceil/N$ by fractions that can be computed in polynomial time (see for example Knuth's *The Art of Computer Programming*, volume 2). Furthermore, it can be shown that any fraction p/q with $|p/q - \lfloor jN/r \rceil/N| < 1/2q^2$ will appear as one of the convergents (see for example Hardy and Wright, Theorem 184). Since j/r differs by at most $1/2N$ from $\lfloor jN/r \rceil/N$, the fraction j/r will appear as a convergent provided $r^2 < N$. By taking N is sufficiently large, this gives an efficient means of recovering the period.

Period finding over the reals

Now suppose we are given a function $f\colon \mathbb{R} \to S$ satisfying $f(x + r) = f(x)$ for some $r \in \mathbb{R}$, and as usual, assume that f is injective within each (minimal) period. Now we'll see how to adapt Shor's procedure to find an approximation to r, even if it happens to be irrational.

To perform period finding on a digital computer, we must of course discretize the function. We have to be careful about how we perform this discretization. For example, suppose that $S = \mathbb{R}$. If we simply evaluate f at equally spaced points and round the resulting values (perhaps rescaled) to get integers, there is no reason for the function values corresponding to inputs separated by an amount close to the period to be related in any way whatsoever. It could be that the discretized function is injective, carrying absolutely no information about the period.

Instead we will discretize in such a way that the resulting function is *pseudoperiodic*. We say that $f : \mathbb{Z} \to S$ is *pseudoperiodic at $k \in \mathbb{Z}$ with period $r \in \mathbb{R}$* if for each $\ell \in \mathbb{Z}$, either $f(k) = f(k + \lfloor \ell r \rfloor)$ or $f(k) = f(k - \lceil \ell r \rceil)$. We say that f is *ϵ-pseudoperiodic* if it is pseudoperiodic for at least an ϵ fraction of the values $k = 0, 1, \ldots, \lfloor r \rfloor$. We assume that the discretized function is ϵ-pseudoperiodic for some constant ϵ, and that it is injective on the subset of inputs where it is pseudoperiodic. Note that the periodic function encoding the regulator of Pell's equation can be constructed so that it satisfies these conditions.

Now let's consider what happens when we apply Fourier sampling to a pseudoperiodic function. As before, we will Fourier sample over $\mathbb{Z}_N$, with N to be determined later (again, depending on

some a priori upper bound M on the period r). We start by computing the pseudoperiodic function on a uniform superposition:

$$\sum_{x\in\{0,\ldots,N-1\}} |x\rangle \mapsto \sum_{x\in\{0,\ldots,N-1\}} |x, f(x)\rangle. \tag{19}$$

Now measuring the second register gives, with constant probability, a value for which f is pseudoperiodic. Say that this value is $f(x_0)$ where $0 \le x_0 \le r$. As before, we see $n = \lfloor N/r \rfloor + 1$ points if $x_0 < N - r\lfloor N/r \rfloor$ or $n = \lfloor N/r \rfloor$ points otherwise (possibly offset by 1 depending on how the rounding occurs for the largest value of x, but let's not be concerned with this detail). We will write $[\ell]$ to denote an integer that could be either $\lfloor \ell \rfloor$ or $\lceil \ell \rceil$. With this notation, we obtain

$$\frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} |x_0 + [jr]\rangle. \tag{20}$$

Next, performing the Fourier transform over $\mathbb{Z}_N$ gives

$$\frac{1}{\sqrt{nN}} \sum_{j=0}^{n-1} \sum_{k\in\mathbb{Z}_N} \omega_N^{k(x_0+[jr])} |k\rangle = \frac{1}{\sqrt{nN}} \sum_{k\in\mathbb{Z}_N} \omega_N^{kx_0} \sum_{j=0}^{n-1} \omega_N^{k[jr]} |k\rangle. \tag{21}$$

Now we have $[jr] = jr + \delta_j$, where $-1 < \delta_j < 1$, so the sum over j is

$$\sum_{j=0}^{n-1} \omega_N^{k[jr]} = \sum_{j=0}^{n-1} \omega_N^{kjr} \omega_N^{k\delta_j}. \tag{22}$$

We would like this to be close to the corresponding sum in the case where the offsets δ_j are zero (which, when normalized, is $\Omega(1/\sqrt{r})$ by the same calculation as in the case of period finding over $\mathbb{Z}$). Consider the deviation in amplitude,

$$\left| \sum_{j=0}^{n-1} \omega_N^{kjr} \omega_N^{k\delta_j} - \sum_{j=0}^{n-1} \omega_N^{kjr} \right| \le \sum_{j=0}^{n-1} |\omega_N^{k\delta_j} - 1| \tag{23}$$

$$= \frac{1}{2} \sum_{j=0}^{n-1} \left| \sin \frac{\pi k \delta_j}{N} \right| \tag{24}$$

$$\le \frac{1}{2} \sum_{j=0}^{n-1} \left| \frac{\pi k \delta_j}{N} \right| \tag{25}$$

$$\le \frac{\pi k n}{2N}. \tag{26}$$

At least insofar as this bound is concerned, the amplitudes may not be close for all values of k. However, suppose we only consider values of k less than $N/\log r$. (We will obtain such a k with probability about $1/\log r$, so we can condition on this event with only polynomial overhead.) For such a k, we have

$$\left| \frac{1}{\sqrt{nN}} \sum_{j=0}^{n-1} \omega_N^{k[jr]} \right| = \Omega(1/\sqrt{r}) - O(\tfrac{1}{\sqrt{nN}} \cdot \tfrac{n}{\log r}) \tag{27}$$

$$= \Omega(1/\sqrt{r}) - O(\tfrac{1}{\sqrt{r}\log r}) \tag{28}$$

$$= \Omega(1/\sqrt{r}). \tag{29}$$

Thus, as in the case of period finding over $\mathbb{Z}$, Fourier sampling allows us to sample from a distribution for which some value $k = \lfloor jN/r \rceil$ (with $j \in \mathbb{Z}$) appears with reasonably large probability (now $\Omega(1/\operatorname{poly}(\log r))$ instead of $\Omega(1)$).

Finally, we must obtain an approximation to r using these samples. Since r is not an integer, the procedure used in Shor's period-finding algorithm does not suffice. However, we can perform Fourier sampling sufficiently many times that we obtain two values $\lfloor jN/r \rceil, \lfloor j'N/r \rceil$ such that j and j' are relatively prime, again with only polynomial overhead. We prove below that if $N \geq 3r^2$, then j/j' is guaranteed to be one of the convergents in the continued fraction expansion for $\lfloor jN/r \rceil / \lfloor j'N/r \rceil$. Thus we can learn j, and hence compute $jN/\lfloor jN/r \rceil$, which gives a good approximation to r: in particular, $|r - \lfloor jN/\lfloor jN/r \rceil \rceil| \leq 1$.

Lemma. *If $N \geq 3r^2$, then j/j' appears a convergent in the continued fraction expansion of $\lfloor jN/r \rceil / \lfloor j'N/r \rceil$. Furthermore, $|r - \lfloor jN/\lfloor jN/r \rceil \rceil| \leq 1$.*

Proof. A standard result on the theory of approximation by continued fractions says that if $a, b \in \mathbb{Z}$ with $|x - \frac{a}{b}| \leq \frac{1}{2b^2}$, then a/b appears as a convergent in the continued fraction expansion of x (see for example Hardy and Wright, *An Introduction to the Theory of Numbers*, Theorem 184.) Thus it is sufficient to show that

$$\left| \frac{\lfloor jN/r \rceil}{\lfloor j'N/r \rceil} - \frac{j}{j'} \right| < \frac{1}{2j'^2}. \tag{30}$$

Letting $\lfloor jN/r \rceil = jN/r + \mu$ and $\lfloor j'N/r \rceil = jN/r + \nu$ with $|\mu|, |\nu| \leq 1/2$, we have

$$\left| \frac{\lfloor jN/r \rceil}{\lfloor j'N/r \rceil} - \frac{j}{j'} \right| = \left| \frac{jN/r + \mu}{j'N/r + \nu} - \frac{j}{j'} \right| \tag{31}$$

$$= \left| \frac{jN + \mu r}{j'N + \nu r} - \frac{j}{j'} \right| \tag{32}$$

$$= \left| \frac{r(\mu j' - \nu j)}{j'(j'N + \nu r)} \right| \tag{33}$$

$$\leq \left| \frac{r(j + j')}{2j'^2 N - j'r} \right| \tag{34}$$

$$\leq \frac{r}{j'N - r/2} \tag{35}$$

where in the last step we have assumed $j < j'$ wlog. This is upper bounded by $1/2j'^2$ provided $j'N \geq r/2 + 2j'^2 r$, which certainly holds if $N \geq 3r^2$ (using the fact that $j' < r$).

Finally

$$r - \frac{jN}{\left\lfloor \frac{jN}{r} \right\rceil} = r - \frac{jN}{\frac{jN}{r} + \mu} \tag{36}$$

$$= r - \frac{jNr}{jN + \mu r} \tag{37}$$

$$= \frac{\mu r^2}{jN + \mu r} \tag{38}$$

which is at most 1 in absolute value since $N \geq 3r^2$, $|\mu| \leq 1/2$, and $j \geq 1$. □

Other algorithms for number fields

To conclude, we mention some further applications of quantum computing to computational algebraic number theory.

Hallgren's original paper on Pell's equation also solves another problem, the *principal ideal problem*, which is the problem of deciding whether an ideal is principal, and if so, finding a generator of the ideal. Factoring reduces to the problem of solving Pell's equation, and Pell's equation reduces to the principal ideal problem; but no reductions in the other direction are known. Motivated by the possibility that the principal ideal problem is indeed harder than factoring, Buchmann and Williams designed a key exchange protocol based on it. Hallgren's algorithm shows that quantum computers can break this cryptosystem.

Subsequently, further related algorithms for problems in algebraic number theory have been found by Hallgren and, independently, by Schmidt and Vollmer. Specifically, they found polynomial-time algorithms for computing the unit group and the class group of a number field of constant degree. These algorithms require generalizing period finding over $\mathbb{R}$ to a similar problem over $\mathbb{R}^d$.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 5: Quantum query complexity of the HSP

So far, we have considered the hidden subgroup problem in abelian groups. We now turn to the case where the group might be nonabelian. We will look at some of the potential applications of the HSP, and then show that the general problem has polynomial quantum query complexity.

The nonabelian HSP and its applications

Recall that in the hidden subgroup problem for a group G, we are given a black box function $f\colon G \to S$, where S is a finite set. We say that f *hides* a subgroup $H \leq G$ provided

$$f(x) = f(y) \text{ if and only if } x^{-1}y \in H. \tag{1}$$

In other words, f is constant on left cosets $H, g_1H, g_2H, \ldots$ of H in G, and distinct on different left cosets. When G is a nonabelian group, we refer to this problem as the nonabelian HSP.

The nonabelian HSP is of interest not only because it generalizes the abelian case in a natural way, but because a solution of certain nonabelian hidden subgroup problems would have particularly useful applications. The most well-known (and also the most straightforward) applications are to the *graph automorphism problem* and the *graph isomorphism problem*, problems for which no efficient classical algorithm is currently known.

In the graph automorphism problem, we are given a graph Γ on n vertices, and the goal is to determine whether it has some nontrivial automorphism. In other words, we would like to know whether there is any nontrival permutation $\pi \in S_n$ such that $\pi(\Gamma) = \Gamma$. The automorphisms of Γ form a subgroup $\operatorname{Aut}\Gamma \leq S_n$; if $\operatorname{Aut}\Gamma$ is trivial then we say Γ is *rigid*. We may cast the graph automorphism problem as an HSP over S_n by considering the function $f(\pi) := \pi(\Gamma)$, which hides $\operatorname{Aut}\Gamma$. If we could solve the HSP in S_n, then by checking whether or not the automorphism group is trivial, we could decide graph automorphism.

In the graph isomorphism problem, we are given two graphs Γ, Γ', each on n vertices, and our goal is to determine whether there is any permutation $\pi \in S_n$ such that $\pi(\Gamma) = \Gamma'$, in which case we say that Γ and Γ' are isomorphic. We can cast graph isomorphism as an HSP in the wreath product $S_n \wr S_2 \leq S_{2n}$, the subgroup of S_{2n} generated by permutations of the first n points, permutations of the second n points, and swapping the two sets of points. Writing elements of $S_n \wr S_2$ in the form (σ, τ, b) where $\sigma, \tau \in S_n$ represent permutations of Γ, Γ', respectively, and $b \in \{0, 1\}$ denotes whether to swap the two graphs, we can define a function

$$f(\sigma, \tau, b) := \begin{cases} (\sigma(\Gamma), \tau(\Gamma')) & b = 0 \\ (\sigma(\Gamma'), \tau(\Gamma)) & b = 1. \end{cases} \tag{2}$$

This function hides the automorphism group of the disjoint union of Γ and Γ', which contains an element that swaps the two graphs if and only if they are isomorphic. In particular, if Γ and Γ' are rigid (which seems to be the hardest case for the HSP approach to graph isomorphism), the hidden subgroup is trivial when Γ, Γ' are non-isomorphic; and has order two, with its nontrival element the involution $(\pi, \pi^{-1}, 1)$, when $\Gamma = \pi(\Gamma')$.

The second major potential application of the hidden subgroup problem is to lattice problems. An *n-dimensional lattice* is the set of all integer linear combinations of n linearly independent

vectors in $\mathbb{R}^n$ (a *basis* for the lattice). In the *shortest vector problem*, we are asked to find a shortest nonzero vector in the lattice. In particular, in the *g(n)-unique shortest vector problem*, we are promised that the shortest nonzero vector is unique (up to its sign), and is shorter than any other non-parallel vector by a factor $g(n)$. This problem can be solved in polynomial time on a classical computer if $g(n)$ is sufficiently large (say, if it is exponentially large), and is NP-hard if $g(n) = O(1)$. Less is known about intermediate cases, but the problem is suspected to be classically hard even for $g(n) = \text{poly}(n)$, to the extent that cryptosystems have been designed based on this assumption.

Regev showed that an efficient quantum algorithm for the dihedral hidden subgroup problem based on the so-called *standard method* (described below) could be used to solve the poly(n)-unique shortest vector problem. Such an algorithm would be significant since it would break lattice cryptosystems, which are some of the few proposed cryptosystems that are not compromised by Shor's algorithm.

So far, only the symmetric and dihedral hidden subgroup problems are known to have significant applications. Nevertheless, there has been considerable interest in understanding the complexity of the HSP for general groups. There are at least three reasons for this. First, the problem is simply of fundamental interest: it appears to be a natural setting for exploring the extent of the advantage of quantum computers over classical ones. Second, techniques developed for other HSPs may eventually find application to the symmetric or dihedral groups. Finally, exploring the limitations of quantum computers for HSPs may suggest cryptosystems that could be robust even to quantum attacks.

The standard method

Nearly all known algorithms for the nonabelian hidden subgroup problem use the black box for f in essentially the same way as in the abelian HSP. This approach has therefore come to be known as the *standard method*.

In the standard method, we begin by preparing a uniform superposition over group elements:

$$|G\rangle := \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle. \tag{3}$$

We then compute the value $f(g)$ in an ancilla register, giving the state

$$\frac{1}{\sqrt{|G|}} \sum_{g \in G} |g, f(g)\rangle. \tag{4}$$

Finally, we measure the second register and discard the result (or equivalently, simply discard the second register). If we obtain the outcome $s \in S$, then the state is projected onto the uniform superposition of those $g \in G$ such that $f(g) = s$, which by the definition of f is simply some left coset of H. Since every coset contains the same number of elements, each left coset occurs with equal probability. Thus this procedure produces the *coset state*

$$|gH\rangle := \frac{1}{\sqrt{|H|}} \sum_{h \in H} |gh\rangle \text{ with } g \in G \text{ uniformly random} \tag{5}$$

(or, equivalently, we can view g as being chosen uniformly at random from some left transversal of H in G).

Depending on context, it may be more convenient to view the outcome either as a random pure state, or equivalently, as the mixed quantum state

$$\rho_H := \frac{1}{|G|} \sum_{g \in G} |gH\rangle\langle gH| \tag{6}$$

which we refer to as a *hidden subgroup state*. In the standard approach to the hidden subgroup problem, we attempt to determine H using samples of this hidden subgroup state. In other words, given $\rho_H^{\otimes k}$ for some $k = \mathrm{poly}(\log |G|)$, we try to find a generating set for H.

Query complexity of the HSP

As a first step toward understanding the quantum complexity of the HSP, we can ask how many queries of the hiding function are required to solve the problem. If we could show that an exponential number of quantum queries were required, then we would know that there was no efficient quantum algorithm. But it turns out that this is not the case: as shown by Ettiner, Høyer, and Knill, $\mathrm{poly}(\log |G|)$ queries to f suffice to determine H. In particular, they showed this within the framework of the standard method: $\rho_H^{\otimes \mathrm{poly}(\log |G|)}$ contains enough information to recover H. Of course, this does not necessarily mean that the quantum *computational complexity* of the HSP is polynomial, since it is not clear in general how to perform the quantum post-processing of the hidden subgroup states efficiently. Nevertheless, this is an important observation since it already shows a difference between quantum and classical computation, and offers some clues as to how we might design efficient quantum algorithms.

To show that the query complexity of the HSP is polynomial, it is sufficient to show that the (single-copy) hidden subgroup states are pairwise statistically distinguishable, as measured by the quantum fidelity

$$F(\rho, \rho') := \mathrm{tr}\, |\sqrt{\rho}\sqrt{\rho'}|. \tag{7}$$

This follows from a result of Barnum and Knill, who showed the following.

Theorem. *Suppose ρ is drawn from an ensemble $\{\rho_1, \ldots, \rho_N\}$, where each ρ_i occurs with some fixed prior probability p_i. Then there exists a quantum measurement (namely, the so-called* pretty good measurement*) that identifies ρ with probability at least*

$$1 - N \sqrt{\max_{i \neq j} F(\rho_i, \rho_j)}. \tag{8}$$

In fact, by the minimax theorem, this holds even without assuming a prior distribution for the ensemble.

Given only one copy of the hidden subgroup state, (8) will typically give only a trivial bound. However, by taking multiple copies of the hidden subgroup states, we can ensure that the overall states are nearly orthogonal, and hence distinguishable. In particular, using k copies of ρ, we see that there is a measurement for identifying ρ with probability at least

$$1 - N \sqrt{\max_{i \neq j} F(\rho_i^{\otimes k}, \rho_j^{\otimes k})} = 1 - N \sqrt{\max_{i \neq j} F(\rho_i, \rho_j)^k} \tag{9}$$

(since the fidelity is multiplicative under tensor products). Setting this expression equal to $1-\epsilon$ and solving for k, we see that arbitrarily small error probability ϵ can be achieved provided we use

$$k \geq \left\lceil \frac{2(\log N - \log \epsilon)}{\log\left(1/\max_{i \neq j} F(\rho_i, \rho_j)\right)} \right\rceil \tag{10}$$

copies of ρ.

Provided that G does not have too many subgroups, and that the fidelity between two distinct hidden subgroup states is not too close to 1, this shows that polynomially many copies of ρ_H suffice to solve the HSP. The total number of subgroups of G is $2^{O(\log^2 |G|)}$, which can be seen as follows. Any group K can be specified in terms of at most $\log_2 |K|$ generators, since every additional (non-redundant) generator increases the size of the group by at least a factor of 2. Since every subgroup of G can be specified by a subset of at most $\log_2 |G|$ elements of G, the number of subgroups of G is upper bounded by $|G|^{\log_2 |G|} = 2^{(\log_2 |G|)^2}$. This shows that we can take $\log N = \mathrm{poly}(\log |G|)$ in (10). Thus $k = \mathrm{poly}(\log |G|)$ copies of ρ_H suffice to identify H with constant probability provided the maximum fidelity is bounded away from 1 by at least $1/\mathrm{poly}(\log |G|)$.

To upper bound the fidelity between two states ρ, ρ', consider the two-outcome measurement that projects onto the support of ρ or its orthogonal complement. The classical fidelity of the resulting distribution is an upper bound on the quantum fidelity, so

$$F(\rho, \rho') \leq \sqrt{\operatorname{tr} \Pi_\rho \rho \operatorname{tr} \Pi_\rho \rho'} + \sqrt{\operatorname{tr}((1-\Pi_\rho)\rho) \operatorname{tr}((1-\Pi_\rho)\rho')} \tag{11}$$

$$= \sqrt{\operatorname{tr} \Pi_\rho \rho'}. \tag{12}$$

where Π_ρ denotes the projector onto the support of ρ.

Now consider the fidelity between ρ_H and $\rho_{H'}$ for two distinct subgroups $H, H' \leq G$. Let $|H| \geq |H'|$ without loss of generality. We can write (6) as

$$\rho_H = \frac{1}{|G|} \sum_{g \in G} |gH\rangle\langle gH| = \frac{|H|}{|G|} \sum_{g \in T_H} |gH\rangle\langle gH|. \tag{13}$$

where T_H denotes some left transversal of H in G. Since the right hand expression is a spectral decomposition of ρ_H, we have

$$\Pi_{\rho_H} = \sum_{g \in T_H} |gH\rangle\langle gH| = \frac{1}{|H|} \sum_{g \in G} |gH\rangle\langle gH|. \tag{14}$$

Then we have

$$F(\rho_H, \rho_{H'})^2 \leq \operatorname{tr} \Pi_{\rho_H} \rho_{H'} \tag{15}$$

$$= \frac{1}{|H| \cdot |G|} \sum_{g, g' \in G} |\langle gH | g'H' \rangle|^2 \tag{16}$$

$$= \frac{1}{|H| \cdot |G|} \sum_{g, g' \in G} \frac{|gH \cap g'H'|^2}{|H| \cdot |H'|} \tag{17}$$

$$= \frac{1}{|G| \cdot |H|^2 \cdot |H|'} \sum_{g, g' \in G} |gH \cap g'H'|^2. \tag{18}$$

Now

$$|gH \cap g'H'| = |\{(h,h') \in H \times H' : gh = g'h'\}| \tag{19}$$
$$= |\{(h,h') \in H \times H' : hh' = g^{-1}g'\}| \tag{20}$$
$$= \begin{cases} |H \cap H'| & \text{if } g^{-1}g' \in HH' \\ 0 & \text{if } g^{-1}g' \notin HH', \end{cases} \tag{21}$$

so

$$\sum_{g,g' \in G} |gH \cap gH'|^2 = |G| \cdot |HH'| \cdot |H \cap H'|^2 \tag{22}$$
$$= |G| \cdot |H| \cdot |H'| \cdot |H \cap H'|. \tag{23}$$

Thus we have

$$F(\rho_H, \rho_{H'})^2 = \frac{|G| \cdot |H| \cdot |H'| \cdot |H \cap H'|}{|G| \cdot |H|^2 \cdot |H'|} \tag{24}$$
$$= \frac{|H \cap H'|}{|H|} \tag{25}$$
$$\leq \frac{1}{2}. \tag{26}$$

This shows that $F(\rho_H, \rho_{H'}) \leq 1/\sqrt{2}$, thereby establishing that the query complexity of the HSP is $\text{poly}(\log |G|)$.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 6: Fourier analysis in nonabelian groups

We have seen that hidden subgroup states contain sufficient information to determine the hidden subgroup. Now we would like to know whether this information can be extracted efficiently. In this lecture, we will introduce the theory of Fourier analysis over general groups, an important tool for getting a handle on this problem.

A brief introduction to representation theory

To understand nonabelian Fourier analysis, we first need to introduce some notions from group representation theory. For further information on this subject, a good basic reference is the book *Linear Representations of Finite Groups* by Serre.

A *linear representation* (or simply *representation*) of a group G over the vector space $\mathbb{C}^n$ is a *homomorphism* $\sigma\colon G \to \mathrm{GL}(\mathbb{C}^n)$, i.e., a map from group elements to nonsingular $n \times n$ complex matrices satisfying $\sigma(x)\sigma(y) = \sigma(xy)$ for all $x, y \in G$. Clearly, $\sigma(1) = 1$ and $\sigma(x^{-1}) = \sigma(x)^{-1}$. We call $\mathbb{C}^n$ the *representation space* of σ, where n is called its *dimension* (or *degree*), denoted d_σ.

Two representations σ and σ' with representation spaces $\mathbb{C}^n$ are called *isomorphic* (denoted $\sigma \sim \sigma'$) if there is an invertible linear transformation $M \in \mathbb{C}^{n\times n}$ such that $M\sigma(x) = \sigma'(x)M$ for all $x \in G$. Otherwise they are called *non-isomorphic* (denoted $\sigma \not\sim \sigma'$). In particular, representations of different dimensions are non-isomorphic. Every representation of a finite group is isomorphic to a *unitary representation*, i.e., one for which $\sigma(x)^{-1} = \sigma(x)^\dagger$ for all $x \in G$. Thus we can restrict our attention to unitary representations without loss of generality.

The simplest representations are those of dimension one, such that $\sigma(x) \in \mathbb{C}$ with $|\sigma(x)| = 1$ for all $x \in G$. Every group has a one-dimensional representation called the *trivial representation*, defined by $\sigma(x) = 1$ for all $x \in G$.

Two particularly useful representations of a group G are the *left regular representation* and the *right regular representation*. Both of these representations have dimension $|G|$, and their representation space is the *group algebra* $\mathbb{C}G$, the $|G|$-dimensional complex vector space spanned by basis vectors $|x\rangle$ for $x \in G$. The left regular representation L satisfies $L(x)|y\rangle = |xy\rangle$, and the right regular representation R satisfies $R(x)|y\rangle = |yx^{-1}\rangle$. In particular, both regular representations are *permutation representations*: each of their representation matrices is a permutation matrix.

Given two representations $\sigma\colon G \to V$ and $\sigma'\colon G \to V'$, we can define their *direct sum*, a representation $\sigma \oplus \sigma'\colon G \to V \oplus V'$ of dimension $d_{\sigma\oplus\sigma'} = d_\sigma + d_{\sigma'}$. The representation matrices of $\sigma \oplus \sigma'$ are block diagonal, of the form

$$(\sigma \oplus \sigma')(x) = \begin{pmatrix} \sigma(x) & 0 \\ 0 & \sigma'(x) \end{pmatrix} \tag{1}$$

for all $x \in G$.

A representation is called *irreducible* if it cannot be decomposed as the direct sum of two other representations. Any representation of a finite group G can be written as a direct sum of irreducible representations (or *irreps*) of G.

Another way to combine two representations is with the *tensor product*. The tensor product of $\sigma\colon G \to V$ and $\sigma'\colon G \to V'$ is $\sigma\otimes\sigma'\colon G \to V\otimes V'$, a representation of G of dimension $d_{\sigma\otimes\sigma'} = d_\sigma d_{\sigma'}$.

The *character* of a representation σ is the function $\chi_\sigma \colon G \to \mathbb{C}$ defined by $\chi_\sigma(x) := \operatorname{tr} \sigma(x)$. We have

- $\chi_\sigma(1) = d_\sigma$ (since $\sigma(1)$ is I_d, the d-dimensional identity matrix)
- $\chi_\sigma(x^{-1}) = \chi_\sigma(x)^*$ (since we can assume that σ is unitary), and
- $\chi_\sigma(yx) = \chi_\sigma(xy)$ for all $x, y \in G$ (since the trace is cyclic).

In particular, $\chi_\sigma(yxy^{-1}) = \chi_\sigma(x)$, so characters are constant on conjugacy classes. For two representations σ, σ', we have $\chi_{\sigma \oplus \sigma'} = \chi_\sigma + \chi_{\sigma'}$ and $\chi_{\sigma \otimes \sigma'} = \chi_\sigma \cdot \chi_{\sigma'}$.

The most useful result in representation theory is probably *Schur's Lemma*, which can be stated as follows:

Theorem (Schur's Lemma)**.** *Let σ and σ' be two irreducible representations of G, and let $M \in \mathbb{C}^{d_\sigma \times d_{\sigma'}}$ be a matrix satisfying $\sigma(x)M = M\sigma'(x)$ for all $x \in G$. Then if $\sigma \not\sim \sigma'$, $M = 0$; and if $\sigma = \sigma'$, M is a scalar multiple of the identity matrix.*

Schur's Lemma can be used to prove the following orthogonality relation for irreducible representations:

Theorem (Orthogonality of irreps)**.** *For two irreps σ and σ' of G, we have*

$$\frac{d_\sigma}{|G|} \sum_{x \in G} \sigma(x)^*_{i,j}\, \sigma'(x)_{i',j'} = \delta_{\sigma,\sigma'} \delta_{i,i'} \delta_{j,j'}, \tag{2}$$

where we interpret $\delta_{\sigma,\sigma'}$ to mean 1 if $\sigma \sim \sigma'$, and 0 otherwise.

This implies a corresponding orthogonality relation for the *irreducible characters* (i.e., the characters of the irreducible representations):

Theorem (Orthogonality of characters)**.** *For two irreps σ and σ' of G, we have*

$$(\chi_\sigma, \chi_{\sigma'}) := \frac{1}{|G|} \sum_{x \in G} \chi_\sigma(x)^*\, \chi_{\sigma'}(x) = \delta_{\sigma,\sigma'}. \tag{3}$$

The characters of G supply an orthonormal basis for the space of *class functions*, functions that are constant on conjugacy classes of G. (Recall that the characters themselves are class functions.) This is expressed by the orthonormality of the *character table* of G, the square matrix whose rows are labeled by irreps, whose columns are labeled by conjugacy classes, and whose entries are the corresponding characters. The character orthogonality theorem says that the rows of this matrix are orthonormal, provided each entry is weighted by the square root of the size of the corresponding conjugagcy class divided by $|G|$. In fact the columns are orthonormal in the same sense.

Any representation of G can be broken up into its irreducible components. The regular representations of G are useful for understanding such decompositions, since they contain every possible irreducible representation of G, with each irrep occuring a number of times equal to its dimension. Let $\hat{G}$ denote a complete set of irreps of G (which are unique up to isomorphism). Then we have

$$L \cong \bigoplus_{\sigma \in \hat{G}} \left(\sigma \otimes I_{d_\sigma}\right), \quad R \cong \bigoplus_{\sigma \in \hat{G}} \left(I_{d_\sigma} \otimes \sigma^*\right). \tag{4}$$

In fact, this holds with the same isomorphism for both L and R, since the left and right regular representations commute. This isomorphism is simply the Fourier transform over G, which we discuss further below.

Considering $\chi_L(1) = \chi_R(1) = |G|$ and using this decomposition, we find the well-known identity

$$\sum_{\sigma \in \hat{G}} d_\sigma^2 = |G|. \tag{5}$$

Also, noting that $\chi_L(x) = \chi_R(x) = 0$ for any $x \in G \setminus \{1\}$, we see that

$$\sum_{\sigma \in \hat{G}} d_\sigma \, \chi_\sigma(x) = 0. \tag{6}$$

In general, the multiplicity of the irrep $\sigma \in \hat{G}$ in an arbitrary representation τ of G is given by $\mu_\sigma^\tau := (\chi_\sigma, \chi_\tau)$. This gives the decomposition

$$\tau \cong \bigoplus_{\sigma \in \hat{G}} \sigma \otimes I_{\mu_\sigma^\tau}. \tag{7}$$

Characters also provide a simple test for irreducibility: for any representation σ, $(\chi_\sigma, \chi_\sigma)$ is a positive integer, and is equal to 1 if and only if σ is irreducible.

Any representation σ of G can also be viewed as a representation of any subgroup $H \leq G$, simply by restricting its domain to elements of H. We denote the resulting *restricted representation* by $\mathrm{Res}_H^G \, \sigma$. Even if σ is irreducible over G, it may not be irreducible over H.

Fourier analysis for nonabelian groups

The *Fourier transform* is a unitary transformation from the group algebra, $\mathbb{C}G$, to a complex vector space whose basis vectors correspond to matrix elements of the irreps of G, $\bigoplus_{\sigma \in \hat{G}} (\mathbb{C}^{d_\sigma} \otimes \mathbb{C}^{d_\sigma})$. These two spaces have the same dimension by (5).

The Fourier transform of the basis vector $|x\rangle \in \mathbb{C}G$ corresponding to the group element $x \in G$ is a weighted superposition over all irreducible representations $\sigma \in \hat{G}$, namely

$$|\hat{x}\rangle := \sum_{\sigma \in \hat{G}} \frac{d_\sigma}{\sqrt{|G|}} |\sigma, \sigma(x)\rangle, \tag{8}$$

where $|\sigma\rangle$ is a state that labels the irreducible representation, and $|\sigma(x)\rangle$ is a normalized, d_σ^2-dimensional state whose amplitudes correspond to the entries of the matrix $\sigma(x)/\sqrt{d_\sigma}$:

$$|\sigma(x)\rangle := \sum_{j,k=1}^{d_\sigma} \frac{\sigma(x)_{j,k}}{\sqrt{d_\sigma}} |j, k\rangle. \tag{9}$$

(If σ is one-dimensional, then $|\sigma(x)\rangle$ is simply a phase factor $\sigma(x) = \chi_\sigma(x) \in \mathbb{C}$ with $|\sigma(x)| = 1$.) The Fourier transform over G is the unitary matrix

$$F_G := \sum_{x \in G} |\hat{x}\rangle\langle x| \tag{10}$$

$$= \sum_{x \in G} \sum_{\sigma \in \hat{G}} \sqrt{\frac{d_\sigma}{|G|}} \sum_{j,k=1}^{d_\sigma} \sigma(x)_{j,k} \, |\sigma, j, k\rangle\langle x|. \tag{11}$$

Note that the Fourier transform over G is not uniquely defined, but rather, depends on a choice of basis for each irreducible representation.

It is straightforward to check that F_G is indeed a unitary transformation. Using the identity

$$\begin{aligned}\langle\sigma(y)|\sigma(x)\rangle &= \operatorname{tr}\sigma^\dagger(y)\sigma(x)/d_\sigma &(12)\\ &= \operatorname{tr}\sigma(y^{-1}x)/d_\sigma &(13)\\ &= \chi_\sigma(y^{-1}x)/d_\sigma, &(14)\end{aligned}$$

we have

$$\begin{aligned}\langle\hat{y}|\hat{x}\rangle &= \sum_{\sigma\in\hat{G}}\frac{d_\sigma^2}{|G|}\langle\sigma(y)|\sigma(x)\rangle &(15)\\ &= \sum_{\sigma\in\hat{G}}\frac{d_\sigma}{|G|}\chi_\sigma(y^{-1}x). &(16)\end{aligned}$$

Hence by (5–6) above, we see that $\langle\hat{y}|\hat{x}\rangle = \delta_{x,y}$.

F_G is precisely the transformation that decomposes both the left and right regular representations of G into their irreducible components. Let us check this explicitly for the left regular representation L. Recall that this representation satisfies $L(x)|y\rangle = |xy\rangle$, so we have

$$\begin{aligned}\hat{L}(x) &:= F_G\,L(x)\,F_G^\dagger &(17)\\ &= \sum_{y\in G}|\widehat{xy}\rangle\langle\hat{y}| &(18)\\ &= \sum_{y\in G}\sum_{\sigma,\sigma'\in\hat{G}}\sum_{j,k=1}^{d_\sigma}\sum_{j',k'=1}^{d_{\sigma'}}\frac{\sqrt{d_\sigma d_{\sigma'}}}{|G|}\sigma(xy)_{j,k}\,\sigma'(y)^*_{j',k'}\,|\sigma,j,k\rangle\langle\sigma',j',k'| &(19)\\ &= \sum_{y\in G}\sum_{\sigma,\sigma'\in\hat{G}}\sum_{j,k,\ell=1}^{d_\sigma}\sum_{j',k'=1}^{d_{\sigma'}}\frac{\sqrt{d_\sigma d_{\sigma'}}}{|G|}\sigma(x)_{j,\ell}\,\sigma(y)_{\ell,k}\,\sigma'(y)^*_{j',k'}\,|\sigma,j,k\rangle\langle\sigma',j',k'| &(20)\\ &= \sum_{\sigma\in\hat{G}}\sum_{j,k,\ell=1}^{d_\sigma}\sigma(x)_{j,\ell}\,|\sigma,j,k\rangle\langle\sigma,\ell,k| &(21)\\ &= \bigoplus_{\sigma\in\hat{G}}\big(\sigma(x)\otimes I_{d_\sigma}\big), &(22)\end{aligned}$$

where in the fourth line we have used the orthogonality relation for irreducible representations.

A similar calculation can be done for the right regular representation defined by $R(x)|y\rangle = |yx^{-1}\rangle$, giving

$$\begin{aligned}\hat{R}(x) &:= F_G\,R(x)\,F_G^\dagger &(23)\\ &= \bigoplus_{\sigma\in\hat{G}}\big(I_{d_\sigma}\otimes\sigma(x)^*\big). &(24)\end{aligned}$$

This identity will be useful when analyzing the application of the quantum Fourier transform to the hidden subgroup problem.

To use the Fourier transform as part of a quantum computation, we must be able to implement it efficiently by some quantum circuit. Efficient quantum circuits for the quantum Fourier transform are known for many, but not all, nonabelian groups. Groups for which an efficient QFT is known include metacyclic groups (i.e., semidirect products of cyclic groups), such as the dihedral group; the symmetric group; and many families of groups that have suitably well-behaved towers of subgroups. There are a few notable groups for which efficient QFTs are *not* known, such as the general linear group $\mathrm{GL}_n(q)$ of $n \times n$ invertible matrices over $\mathbb{F}_q$, the finite field with q elements.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 7: Fourier sampling

In this lecture, we will see how the Fourier transform can be used to simplify the structure of the states obtained in the standard approach to the hidden subgroup problem. In particular, we will see how *weak Fourier sampling* is sufficient to identify any normal hidden subgroup (generalizing the solution of the abelian HSP). We will also briefly discuss the potential of *strong Fourier sampling* to go beyond the limitations of weak Fourier sampling.

Weak Fourier sampling

Recall that the standard approach to the HSP allows us to produce a *coset state*

$$|gH\rangle := \frac{1}{\sqrt{|H|}} \sum_{h\in H} |gh\rangle \tag{1}$$

where each $g \in G$ occurs uniformly at random; or equivalently, the *hidden subgroup state*

$$\rho_H := \frac{1}{|G|} \sum_{g\in G} |gH\rangle\langle gH|. \tag{2}$$

The symmetry of such a state can be exploited using the quantum Fourier transform. In particular, we have

$$|gH\rangle = \frac{1}{\sqrt{|H|}} \sum_{h\in H} R(h)|g\rangle \tag{3}$$

where R is the right regular representation of G. Thus the hidden subgroup state can be written

$$\rho_H = \frac{1}{|G|\cdot|H|} \sum_{g\in G} \sum_{h,h'\in H} R(h)|g\rangle\langle g|R(h')^\dagger \tag{4}$$

$$= \frac{1}{|G|\cdot|H|} \sum_{h,h'\in H} R(hh'^{-1}) \tag{5}$$

$$= \frac{1}{|G|} \sum_{h\in H} R(h). \tag{6}$$

Since the right regular representation is block-diagonal in the Fourier basis, the same is true of ρ_H. In particular, we have

$$\hat{\rho}_H := F_G \, \rho_H \, F_G^\dagger \tag{7}$$

$$= \frac{1}{|G|} \bigoplus_{\sigma\in\hat{G}} \left(I_{d_\sigma} \otimes \sigma(H)^*\right) \tag{8}$$

where

$$\sigma(H) := \sum_{h\in H} \sigma(h). \tag{9}$$

Since $\hat{\rho}_H$ is block diagonal, with blocks labeled by irreducible representations, we may now measure the irrep label without loss of information. This procedure is referred to as *weak Fourier sampling*. The probability of observing representation $\sigma \in \hat{G}$ under weak Fourier sampling is

$$\Pr(\sigma) = \frac{1}{|G|} \operatorname{tr}\left(I_{d_\sigma} \otimes \sigma(H)^*\right) \tag{10}$$

$$= \frac{d_\sigma}{|G|} \sum_{h \in H} \chi_\sigma(h)^* \tag{11}$$

$$= \frac{d_\sigma |H|}{|G|} (\chi_\sigma, \chi_1)_H, \tag{12}$$

or in other words, $d_\sigma |H|/|G|$ times the number of times the trivial representation appears in $\operatorname{Res}^G_H \sigma$, the restriction of σ to H. We may now ask whether polynomially many samples from this distribution are sufficient to determine H, and if so, whether H can be reconstructed from this information efficiently.

Normal subgroups

If G is abelian, then all of its representations are one-dimensional, so weak Fourier sampling reveals all of the available information about ρ_H. (In this case there is no difference between weak Fourier sampling and strong Fourier sampling, which we will discuss later.) Indeed, for an abelian group, we saw that the information provided by Fourier sampling can be used to efficiently determine H.

Weak Fourier sampling succeeds for a similar reason whenever H is a *normal subgroup* of G (denoted $H \trianglelefteq G$), i.e., whenever $gHg^{-1} = H$ for all $g \in G$. In this case, the hidden subgroup state within the irrep $\sigma \in \hat{G}$ is proportional to

$$\sigma(H)^* = \frac{1}{|G|} \sum_{g \in G, h \in H} \sigma(ghg^{-1})^*. \tag{13}$$

This commutes with $\sigma(g)^*$ for all $g \in G$, so by Schur's Lemma, it is a multiple of the identity. Thus $\hat{\rho}_H$ is proportional to the identity within each block, and again weak Fourier sampling reveals all available information about H.

Furthermore, when $H \trianglelefteq G$, the distribution under weak Fourier sampling is a particularly simple generalization of the abelian case: we have

$$\Pr(\sigma) = \begin{cases} d_\sigma^2 |H|/|G| & H \leq \ker \sigma \\ 0 & \text{otherwise,} \end{cases} \tag{14}$$

where $\ker \sigma := \{g \in G : \sigma(g) = I_{d_\sigma}\}$ is the kernel of the representation σ (a normal subgroup of G). To see this, note that if $H \not\leq \ker \sigma$, then there is some $h' \in H$ with $\sigma(h') \neq 1$; but then $\sigma(h')\sigma(H) = \sum_{h \in H} \sigma(h'h) = \sigma(H)$, and since $\sigma(h')$ is unitary and $\sigma(H)$ is a scalar multiple of the identity, this can only be satisfied if in fact $\sigma(H) = 0$. On the other hand, if $H \leq \ker \sigma$, then $\chi_\sigma(h) = d_\sigma$ for all $h \in H$, and the result is immediate.

To find H, we can simply proceed as in the abelian case: perform weak Fourier sampling $O(\log |G|)$ times and compute the intersection of the kernels of the resulting irreps (assuming this can be done efficiently). Again, it is clear that the resulting subgroup contains H, and we claim that it is equal to H with high probability. For suppose that at some stage during this process, the

intersection of the kernels is $K \trianglelefteq G$ with $K \neq H$; then the probability of obtaining an irrep σ for which $K \leq \ker \sigma$ is

$$\frac{|H|}{|G|} \sum_{\sigma:\, K \leq \ker \sigma} d_\sigma^2 = \frac{|H|}{|K|} \leq \frac{1}{2} \tag{15}$$

where we have used the fact that the distribution (14) remains normalized if H is replaced by any normal subgroup of G. Since each repetition of weak Fourier sampling has a probability of at least $1/2$ of cutting the intersection of the kernels at least in half, $O(\log |G|)$ repetitions suffice to converge to H with substantial probability. In fact, applying the same approach when H is not necessarily normal in G gives an algorithm to find the *normal core* of H, the largest subgroup of H that is normal in G.

Strong Fourier sampling

Despite the examples we have just discussed, weak Fourier sampling does *not* provide sufficient information to recover the hidden subgroup for the majority of hidden subgroup problems. For example, weak Fourier sampling fails to solve the HSP in the symmetric group and the dihedral group.

To obtain more information about the hidden subgroup, we can perform a measurement on the d_σ^2-dimensional state that results when weak Fourier sampling returns the outcome σ. Such an approach is referred to as *strong Fourier sampling*.

Recall that the state $\hat{\rho}_H$ from (8) is maximally mixed over the row register, as a consequence of the fact that the left and right regular representations commute. Thus we may discard this register without loss of information, so that strong Fourier sampling is effectively faced with the d_σ-dimensional state

$$\hat{\rho}_{H,\sigma} := \frac{\sigma(H)^*}{\sum_{h \in H} \chi_\sigma(h)^*}. \tag{16}$$

In fact, this state is proportional to a projector whose rank is simply the number of times the trivial representation appears in $\operatorname{Res}_H^G \sigma^*$. This follows because

$$\sigma(H)^2 = \sum_{h,h' \in H} \sigma(hh') = |H|\, \sigma(H), \tag{17}$$

which gives

$$\hat{\rho}_{H,\sigma}^2 = \frac{|H|}{\sum_{h \in H} \chi_\sigma(h)^*} \hat{\rho}_{H,\sigma}, \tag{18}$$

so that $\hat{\rho}_{H,\sigma}$ is proportional to a projector with $\operatorname{rank}(\hat{\rho}_{H,\sigma}) = \sum_{h \in H} \chi_\sigma(h)^*/|H|$.

It is not immediately clear how to choose a good basis for strong Fourier sampling, so a natural first approach is to consider the effect of measuring in a random basis (i.e., a basis chosen uniformly with respect to the Haar measure over $\mathbb{C}^{d_\sigma}$). There are a few cases in which such *random strong Fourier sampling* produces sufficient information to identify the hidden subgroup—in particular, Sen showed that it succeeds whenever $\operatorname{rank}(\hat{\rho}_{H,\sigma}) = \operatorname{poly}(\log |G|)$ for all $\sigma \in \hat{G}$.

However, in many cases random strong Fourier sampling is unhelpful. For example, Grigni et al. showed that if H is sufficiently small and G is sufficiently non-Abelian (in a certain precise sense), then random strong Fourier sampling is not very informative. In particular, they showed this for the problem of finding hidden involutions in the symmetric group. Another example was

provided by Moore et al., who showed that random strong Fourier sampling fails in the metacyclic groups $\mathbb{Z}_p \rtimes \mathbb{Z}_q$ (subgroups of the affine group $\mathbb{Z}_p \rtimes \mathbb{Z}_p^\times$) when $q < p^{1-\epsilon}$ for some $\epsilon > 0$.

Even when measuring in a random basis is information-theoretically sufficient, it does not give an efficient quantum algorithm, since it is not possible to efficiently measure in a random basis. It would be interesting to find informative pseduo-random bases that can be implemented efficiently. However, in the absence of such techniques, we can instead hope to find explicit bases in which strong Fourier sampling can be performed efficiently, and for which the results give a solution of the HSP. The first such algorithm was provided by Moore et al., for the aforementioned metacyclic groups, but with $q = p/\operatorname{poly}(\log p)$. Note that for these values of p, q, unlike the case $q < p^{1-\epsilon}$ mentioned above, measurement in a random basis is information-theoretically sufficient. Indeed, we do not know of *any* example of an HSP for which strong Fourier sampling succeeds, yet random strong Fourier sampling fails; it would be interesting to find any such example (or to prove that none exists).

Note that simply finding an informative basis is not sufficient; it is also important that the measurement results can be efficiently post-processed. This issue arises not only in the context of measurement in a pseudo-random basis, but also in the context of certain explicit bases. For example, Ettinger and Høyer gave a basis for the dihedral HSP in which a measurement gives sufficient classical information to infer the hidden subgroup, but no efficient means of post-processing this information is known.

For some groups, it turns out that strong Fourier sampling simply fails. Moore, Russell, and Schulman showed that, regardless of what basis is chosen, strong Fourier sampling provides insufficient information to solve the HSP in the symmetric group. Specifically, they showed that for any measurement basis (indeed, for any POVM applied to a hidden subgroup state), the distribution of outcomes in the cases where the hidden subgroup is trivial and where the hidden subgroup is an involution are exponentially close. Thus, in general one has to consider *entangled measurements* on multiple copies of the hidden subgroup states. (Indeed, entangled measurements on $\Omega(\log |G|)$ copies may be necessary, as Hallgren et al. showed for the symmetric group.) In the next two lectures, we will see some examples of quantum algorithms for the HSP that make use of entangled measurements.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 8: Kuperberg's algorithm for the dihedral HSP

In this lecture, we will discuss a quantum algorithm for the dihedral hidden subgroup problem. No polynomial-time algorithm for this problem is known. However, Kuperberg gave a quantum algorithm that runs in subexponential (though superpolynomial) time—specifically, it runs in time $2^{O(\sqrt{\log |G|})}$.

The HSP in the dihedral group

The dihedral group of order $2N$, denoted D_N, is the group of symmetries of a regular N-gon. It has the presentation

$$D_N = \langle r, s \colon r^2 = s^N = 1,\ rsr = s^{-1} \rangle. \tag{1}$$

Here r can be thought of as a reflection about some fixed axis, and s can be thought of as a rotation of the N-gon by an angle $2\pi/N$.

Using the defining relations, we can write any group element in the form $s^x r^a$ where $x \in \mathbb{Z}_N$ and $a \in \mathbb{Z}_2$. Thus we can equivalently think of the group as consisting of elements $(x, a) \in \mathbb{Z}_N \times \mathbb{Z}_2$. Since

$$\begin{aligned} (s^x r^a)(s^y r^b) &= s^x r^a s^y r^a r^{a+b} && (2)\\ &= s^x s^{(-1)^a y} r^{a+b} && (3)\\ &= s^{x+(-1)^a y} r^{a+b}, && (4) \end{aligned}$$

the group operation '$\cdot$' on such elements can be expressed as

$$(x, a) \cdot (y, b) = (x + (-1)^a y, a + b). \tag{5}$$

(In particular, this shows that the dihedral group is the semidirect product $\mathbb{Z}_N \rtimes_\varphi \mathbb{Z}_2$, where $\varphi \colon \mathbb{Z}_2 \to \mathrm{Aut}(\mathbb{Z}_N)$ is defined by $\varphi(a)(y) = (-1)^a y$.) It is also easy to see that the group inverse is

$$(x, a)^{-1} = (-(-1)^a x, a). \tag{6}$$

The subgroups of D_N are either cyclic or dihedral. The possible cyclic subgroups are of the form $\langle (x, 0) \rangle$ where $x \in \mathbb{Z}_N$ is either 0 or some divisor of N. The possible dihedral subgroups are of the form $\langle (y, 1) \rangle$ where $y \in \mathbb{Z}_N$, and of the form $\langle (x, 0), (y, 1) \rangle$ where $x \in \mathbb{Z}_N$ is some divisor of N and $y \in \mathbb{Z}_x$. A result of Ettinger and Høyer reduces the general dihedral HSP, in which the hidden subgroup could be any of these possibilities, to the dihedral HSP with the promise that the hidden subgroup is of the form $\langle (y, 1) \rangle = \{(0, 0), (y, 1)\}$, i.e., a subgroup of order 2 generated by the reflection $(y, 1)$.

The basic idea of the Ettinger-Høyer reduction is as follows. Suppose that $f \colon D_N \to S$ hides a subgroup $H = \langle (x, 0), (y, 1) \rangle$. Then we can consider the function f restricted to elements from the abelian group $\mathbb{Z}_N \times \{0\} \leq D_N$. This restricted function hides the subgroup $\langle (x, 0) \rangle$, and since the restricted group is abelian, we can find x efficiently using Shor's algorithm. Now $\langle (x, 0) \rangle \trianglelefteq D_N$ (since $(z, a)(x, 0)(z, a)^{-1} = (z + (-1)^a x, a)(-(-1)^a z, a) = ((-1)^a x, 0) \in \mathbb{Z}_N \times \{0\}$), so we can define the quotient group $D_N / \langle (x, 0) \rangle$. But this is simply a dihedral group (of order N/x), and if we now define a function f' as f evaluated on some coset representative, it hides the subgroup $\langle (y, 1) \rangle$. Thus, in the rest of this lecture, we will assume that the hidden subgroup is of the form $\langle (y, 1) \rangle$ for some $y \in \mathbb{Z}_N$ without loss of generality.

Fourier sampling in the dihedral group

When the hidden subgroup is $H = \langle (y, 1) \rangle$, one particular left transversal of H in G consists of the left coset representatives $(z, 0)$ for all $z \in \mathbb{Z}_N$. The coset state corresponding to the coset $(z, 0)H$ is

$$|(z, 0)\{(0, 0), (y, 1)\}\rangle = \frac{1}{\sqrt{2}}(|z, 0\rangle + |y + z, 1\rangle). \tag{7}$$

We would like to determine y using samples of this state.

We have seen that to distinguish coset states in general, one should start by performing weak Fourier sampling: apply a Fourier transform over G and then measure the irrep label. However, in this case we will instead simply Fourier transform the first register over $\mathbb{Z}_N$, leaving the second register alone. It is possible to show that measuring the first register of the resulting state is essentially equivalent to performing weak Fourier sampling over D_N (and discarding the row register), but for simplicity we will just consider the abelian procedure.

Fourier transforming the first register over $\mathbb{Z}_N$, we obtain

$$(F_{\mathbb{Z}_N} \otimes I_2)|(z, 0)H\rangle = \frac{1}{\sqrt{2N}} \sum_{k \in \mathbb{Z}_N} (\omega_N^{kz}|k, 0\rangle + \omega_N^{k(y+z)}|k, 1\rangle \tag{8}$$

$$= \frac{1}{\sqrt{N}} \sum_{k \in \mathbb{Z}_N} \omega_N^{kz}|k\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + \omega_N^{ky}|1\rangle). \tag{9}$$

If we then measure the first register, we obtain one of the N values of k uniformly at random, and we are left with the post-measurement state

$$|\psi_k\rangle := \frac{1}{\sqrt{2}}(|0\rangle + \omega_N^{yk}|1\rangle). \tag{10}$$

Thus we are left with the problem of determining y given the ability to produce single-qubit states $|\psi_k\rangle$ of this form (where k is known).

Combining states

It would be very useful if we could prepare states $|\psi_k\rangle$ with particular values of k. For example, if we could prepare the state $|\psi_{N/2}\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^y|1\rangle)$, then we could learn the parity of y (i.e., its least significant bit) by measuring in the basis of states $|\pm\rangle := (|0\rangle \pm |1\rangle)/\sqrt{2}$. The main idea of Kuperberg's algorithm is to combine states of the form (10) to produce new states of the same form, but with more desirable values of k.

To combine states, we can use the following procedure. Given two states $|\psi_p\rangle$ and $|\psi_q\rangle$, perform a controlled-not gate from the former to the latter, giving

$$|\psi_p, \psi_q\rangle = \frac{1}{2}(|0, 0\rangle + \omega_N^{yp}|1, 0\rangle + \omega_N^{yq}|0, 1\rangle + \omega_N^{y(p+q)}|1, 1\rangle) \tag{11}$$

$$\mapsto \frac{1}{2}(|0, 0\rangle + \omega_N^{yp}|1, 1\rangle + \omega_N^{yq}|0, 1\rangle + \omega_N^{y(p+q)}|1, 0\rangle) \tag{12}$$

$$= \frac{1}{\sqrt{2}}(|\psi_{p+q}, 0\rangle + \omega_N^{yq}|\psi_{p-q}, 1\rangle). \tag{13}$$

Then a measurement on the second qubit leaves the first qubit in the state $|\psi_{p\pm q}\rangle$ (up to an irrelevant global phase), with the $+$ sign occurring when the outcome is 0 and the $-$ sign occurring when the outcome is 1, each outcome occurring with probability $1/2$.

This combination operation has a nice representation-theoretic interpretation: the state indices p and q can be viewed as labels of irreducible representations of D_N, and the extraction of $|\psi_{p\pm q}\rangle$ can be viewed as decomposing their tensor product (a reducible representation of D_N) into one of two irreducible components.

The Kuperberg sieve

Now we are ready to describe how the algorithm works. For simplicity, we will assume from now on that $N = 2^n$ is a power of 2. For such a dihedral group, it is actually sufficient to be able to determine the least significant bit of y, since such an algorithm could be used recursively to determine all the bits of y. This can be seen as follows. The group D_N contains two subgroups isomorphic to $D_{N/2}$, namely $\{(2x, 0), (2x, 1) \colon x \in \mathbb{Z}_{N/2}\}$ and $\{(2x, 0), (2x + 1, 1) \colon x \in \mathbb{Z}_{N/2}\}$. The hidden subgroup is a subgroup of the former if y has even parity, and of the latter if y has odd parity. Thus, once we learn the parity of y, we can restrict our attention to the appropriate $D_{N/2}$ subgroup. The elements of either $D_{N/2}$ subgroup can be represented using only $n - 1$ bits, and finding the least significant bit of the hidden reflection within this subgroup corresponds to finding the second least significant bit of y in D_N. Continuing in this way, we can learn all the bits of y with only n iterations of an algorithm for finding the least significant bit of the hidden reflection.

The idea of Kuperberg's algorithm is to start with a large number of states, and collect them into pairs $|\psi_p\rangle, |\psi_q\rangle$ that share many of their least significant bits, such that $|\psi_{p-q}\rangle$ is likely to have many of its least significant bits equal to zero. Trying to zero out all but the most significant bit in one shot would require an exponential running time, so instead we will proceed in stages, only trying to zero some of the least significant bits in each stage; this will turn out to give an improvement.

Specifically, the algorithm proceeds as follows:

1. Prepare $\Theta(16^{\sqrt{n}})$ coset states of the form (10), where each copy has $k \in \mathbb{Z}_{2^n}$ chosen independently and uniformly at random.
2. For each $j = 0, 1, \ldots, m - 1$ where $m := \lceil \sqrt{n} \rceil$, assume the current coset states are all of the form $|\psi_k\rangle$ with at least mj of the least significant bits of k equal to 0. Collect them into pairs $|\psi_p\rangle, |\psi_q\rangle$ that share at least m of the next least significant bits, discarding any qubits that cannot be paired. Create a state $|\psi_{p\pm q}\rangle$ from each pair, and discard it if the $+$ sign occurs. Notice that the resulting states have at least $m(j + 1)$ significant bits equal to 0.
3. The remaining states are of the form $|\psi_0\rangle$ and $|\psi_{2^{n-1}}\rangle$. Measure one of the latter states in the $|\pm\rangle$ basis to determine the least significant bit of y.

Since this algorithm requires $2^{O(\sqrt{n})}$ initial queries and proceeds through $O(\sqrt{n})$ stages, each of which takes at most $2^{O(\sqrt{n})}$ steps, the overall running time is $2^{O(\sqrt{n})}$.

Analysis of the Kuperberg sieve

To show that this algorithm works, we need to prove that some qubits survive to the final stage of the process with non-negligible probability. Let's analyze a more general version of the algorithm to see why we should try to zero out $\sqrt{n}$ bits at a time, starting with $2^{O(\sqrt{n})}$ states.

Suppose we try to cancel m bits in each stage, so that there are n/m stages (not yet assuming any relationship between m and n), starting with 2^ℓ states. Each combination operation succeeds with probability $1/2$, and turns 2 states into 1, so at each step we retain only about $1/4$ of the

states that can be paired. Now when we pair states that allow us to cancel m bits, there can be at most 2^m unpaired states, since that is the number of values of the m bits to be canceled. Thus if we ensure that there are at least $2 \cdot 2^m$ states at each stage, we expect to retain at least a $1/8$ fraction of the states for the next stage. Since we begin with 2^ℓ states,we expect to have at least $2^{\ell-3j}$ states left after the jth stage. Thus, to have $2 \cdot 2^m$ states remaining at the last stage of the algorithm, we require $2^{\ell-3n/m} > 2^{m+1}$, or $\ell > m + 3n/m + 1$. This is minimized by choosing $m \approx \sqrt{n}$, so we see that $\ell \approx 4\sqrt{n}$ suffices.

This analysis is not quite correct because we do not obtain precisely a $1/8$ fraction of the paired states for use in the next stage. For most of the stages, we have many more than $2 \cdot 2^m$ states, so nearly all of them can be paired, and the expected fraction remaining for the next stage is close to $1/4$. Of course, the precise fraction will experience statistical fluctuations. However, since we are working with a large number of states, the deviations from the expected values are very small, and a more careful analysis (using the Chernoff bound) shows that the procedure succeeds with high probability. For a detailed argument, see section 3.1 of Kuperberg's paper (SICOMP version). That paper also gives an improved algorithm that runs faster and that works for general N.

Note that this algorithm uses not only superpolynomial time, but also superpolynomial space, since all $\Theta(16^{\sqrt{n}})$ coset states are present at the start of the algorithm. However, by creating a smaller number of coset states at a time and combining them according to the solution of a subset sum problem, Regev showed how to make the space requirement polynomial with only a slight increase in the running time.

Entangled measurements

Although this algorithm acts on pairs of coset states at a time, the overall algorithm effectively implements a highly entangled measurement on all $\Theta(\sqrt{16^n})$ registers, since the combination operation that produces $|\psi_{p\pm q}\rangle$ entangles the coset states $|\psi_p\rangle$ and $|\psi_q\rangle$. The same is true of Regev's polynomial-space variant.

It is natural to ask whether a similar sieve could be applied to other hidden subgroup problems, such as in the symmetric group, for which highly entangled measurements are necessary. Alagic, Moore, and Russell used a similar approach to give a subexponential-time algorithm for the hidden subgroup problem in the group G^n, where G is a fixed non-Abelian group. (Note that the HSP in G^n can be much harder than solving n instances of the HSP in G, since G^n has many subgroups that are not direct products of subgroups of G.) But unfortunately, this kind of sieve does not seem well-suited to the symmetric group. In particular, Moore, Russell, and Sniady gave the following negative result for the HSP in $S_n \wr \mathbb{Z}_2$, where the hidden subgroup is promised to be either trivial or an involution. Consider any algorithm that works by combining pairs of hidden subgroup states to produce a new state in the decomposition of their tensor product into irreps (i.e., in their *Clebsch-Gordan decomposition*), and uses the sequence of measurement results to guess whether the hidden subgroup is trivial or nontrivial. Any such algorithm must use $2^{\Omega(\sqrt{n})}$ queries. Thus it is not possible to give a significantly better-than-classical algorithm for graph isomorphism in this way, since there are classical algorithms for graph isomorphism that run in time $2^{O(\sqrt{n/\log n})}$.

Note that entangled measurements are *not* information-theoretically necessary for the dihedral HSP: Ettinger and Høyer gave an explicit measurement (i.e., an explicit basis for strong Fourier sampling) from which the measurement results give sufficient information to determine the hidden subgroup. Suppose that, given the state (10), we simply measure in the $|\pm\rangle$ basis. Then we obtain

the result $|+\rangle$ with probability

$$\left|\left(\frac{\langle 0| + \langle 1|}{\sqrt{2}}\right)\left(\frac{|0\rangle + \omega_N^{yk}|1\rangle}{\sqrt{2}}\right)\right|^2 = \left|\frac{1+\omega_N^{yk}}{2}\right|^2 = \cos^2\frac{\pi yk}{N}. \tag{14}$$

If we postselect on obtaining this outcome (which happens with probability $1/2$ over the uniformly random value of k, assuming $y \neq 0$), then we effectively obtain each value $k \in \mathbb{Z}_N$ with probability $\Pr(k|+) = \frac{2}{N}\cos^2\frac{\pi yk}{N}$. It is not hard to show that these distributions are statistically far apart for different values of k, so that they can in principle be distinguished with only polynomially many samples. However, no efficient (or even subexponential time) classical (or even quantum) algorithm for doing so is known.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 9: Simulating Hamiltonian dynamics

So far, we have focused on quantum algorithms for the hidden subgroup problem, with applications to number-theoretic problems such as factoring, computing discrete logarithms, and performing calculations in number fields. Another major potential application of quantum computers is the simulation of quantum dynamics. Indeed, this was the idea that first led Feynman to propose the concept of a quantum computer. In this lecture we will see how a universal quantum computer can efficiently simulate several natural families of Hamiltonians. These simulation methods could be used either to simulate actual physical systems, or to implement quantum algorithms defined in terms of Hamiltonian dynamics (such as continuous-time quantum walks and adiabatic quantum algorithms).

Hamiltonian dynamics

In quantum mechanics, time evolution of the wave function $|\psi(t)\rangle$ is governed by the Schrödinger equation,

$$i\hbar \frac{\mathrm{d}}{\mathrm{d}t}|\psi(t)\rangle = H(t)|\psi(t)\rangle. \tag{1}$$

Here $H(t)$ is the *Hamiltonian*, an operator with units of energy, and $\hbar$ is Planck's constant. For convenience it is typical to choose units in which $\hbar = 1$. Given an initial wave function $|\psi(0)\rangle$, we can solve this differential equation to determine $|\psi(t)\rangle$ at any later (or earlier) time t.

For H independent of time, the solution of the Schrödinger equation is $|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$. For simplicity we will only consider this case. There are many situations in which time-dependent Hamiltonians arise, not only in physical systems but also in computational applications such as adiabatic quantum computing. In such cases, the evolution cannot in general be written in such a simple form, but nevertheless similar ideas can be used to simulate the dynamics.

Efficient simulation

We will say that a Hamiltonian H acting on n qubits can be *efficiently simulated* if for any $t > 0$, $\epsilon > 0$ there is a quantum circuit U consisting of $\mathrm{poly}(n, t, 1/\epsilon)$ gates such that $\|U - e^{-iHt}\| < \epsilon$. Clearly, the problem of simulating Hamiltonians in general is BQP-hard, since we can implement any quantum computation by a sequence of Hamiltonian evolutions. In fact, even with natural restrictions on the kind of Hamiltonians we consider, it is easy to specify Hamiltonian simulation problems that are BQP-complete (or more precisely, PromiseBQP-complete).

You might ask why we define the notion of efficient simulation to be polynomial in t; if t is given as part of the input, this means that the running time is, strictly speaking, not polynomial in the input size. However, one can show that a running time polynomial in $\log t$ is impossible; running time $\Omega(t)$ is required in general (intuitively, one cannot "fast forward" the evolution according to a generic Hamiltonian). The dependence on ϵ is more subtle. There are no nontrivial lower bounds on the running time in terms of ϵ, and it is an open question to better understand the complexity of quantum simulation as a function of simulation error.

We would like to understand the conditions under which a Hamiltonian can be efficiently simulated. Of course, we cannot hope to efficiently simulate arbitrarily Hamiltonians, just as we cannot

hope to efficiently implement arbitrary unitaries. Instead, we will simply describe a few classes of Hamiltonian that can be efficiently simulated. Our strategy will be to start from simple Hamiltonians that are easy to simulate and define ways of combining the known simulations to give more complicated ones.

There are a few cases where a Hamiltonian can obviously simulated efficiently. For example, this is the case if H only acts nontrivially on a constant number of qubits, simply because any unitary evolution on a constant number of qubits can be approximated with error at most ϵ using $\text{poly}(\log \frac{1}{\epsilon})$ one- and two-qubit gates, using the Solovay-Kitaev theorem.

Note that since we require a simulation for an arbitrary time t (with $\text{poly}(t)$ gates), we can rescale the evolution by any polynomial factor: if H can be efficiently simulated, then so can cH for any $c = \text{poly}(n)$. This holds even if $c < 0$, since any efficient simulation is expressed in terms of quantum gates, and can simply be run in reverse.

In addition, we can rotate the basis in which a Hamiltonian is applied using any unitary transformation with an efficient decomposition into basic gates. In other words, if H can be efficiently simulated and the unitary transformation U can be efficiently implemented, then $UHU^\dagger$ can be efficiently simulated. This follows from the simple identity

$$e^{-iUHU^\dagger t} = Ue^{-iHt}U^\dagger. \tag{2}$$

Another simple but useful trick for simulating Hamiltonians is the following. Suppose H is diagonal in the computational basis, and any diagonal element $d(a) = \langle a|H|a\rangle$ can be computed efficiently. Then H can be simulated efficiently using the following sequence of operations, for any input computational basis state $|a\rangle$:

$$\begin{aligned} |a,0\rangle &\mapsto |a,d(a)\rangle && (3)\\ &\mapsto e^{-itd(a)}|a,d(a)\rangle && (4)\\ &\mapsto e^{-itd(a)}|a,0\rangle && (5)\\ &= e^{-iHt}|a\rangle|0\rangle. && (6) \end{aligned}$$

By linearity, this process simulates H for time t on an arbitrary input.

Note that if we combine this simulation with the previous one, we have a way to simulate any Hamiltonian that can be efficiently diagonalized, and whose eigenvalues can be efficiently computed.

Product formulas

Many natural Hamiltonians have the form of a sum of terms, each of which can be simulated by the techniques described above. For example, consider the Hamiltonian of a particle in a potential:

$$H = \frac{p^2}{2m} + V(x).$$

To simulate this a digital quantum computer, we can imagine discretizing the x coordinate. The operator $V(x)$ is diagonal, and natural discretizations of $p^2 = -\mathrm{d}^2/\mathrm{d}x^2$ are diagonal in the discrete Fourier basis. Thus we can efficiently simulate both $V(x)$ and $p^2/2m$. Similarly, consider the Hamiltonian of a spin system, say of the form

$$H = \sum_i h_i X_i + \sum_{ij} J_{ij} Z_i Z_j$$

(or more generally, any k-local Hamiltonian, a sum of terms that each act on at most k qubits). This consists of a sum of terms, each of which acts only only a constant number of qubits and hence is easy to simulate.

In general, if H_1 and H_2 can be efficiently simulated, then $H_1 + H_2$ can also be efficiently simulated. If the two Hamiltonians commute, then this is trivial, since $e^{-iH_1t}e^{-iH_2t} = e^{-i(H_1+H_2)t}$. However, in the general case where the two Hamiltonians do not commute, we can still simulate their sum as a consequence of the Lie product formula

$$e^{-i(H_1+H_2)t} = \lim_{m\to\infty} \left(e^{-iH_1t/m}e^{-iH_2t/m}\right)^m. \tag{7}$$

A simulation using a finite number of steps can be achieved by truncating this expression to a finite number of terms, which introduces some amount of error that must be kept small. In particular, if we want to have

$$\left\| \left(e^{-iH_1t/m}e^{-iH_2t/m}\right)^m - e^{-i(H_1+H_2)t} \right\| \leq \epsilon, \tag{8}$$

it suffices to take $m = O((\nu t)^2/\epsilon)$, where $\nu := \max\{\|H_1\|, \|H_2\|\}$. (The requirement that H_1 and H_2 be efficiently simulable means that ν can be at most $\mathrm{poly}(n)$.)

It is somewhat unappealing that to simulate an evolution for time t, we need a number of steps proportional to t^2. Fortunately, the situation can be improved if we use higher-order approximations of (7). For example, one can show that

$$\left\| \left(e^{-iH_1t/2m}e^{-iH_2t/m}e^{-iH_1t/2m}\right)^m - e^{-i(H_1+H_2)t} \right\| \leq \epsilon \tag{9}$$

with a smaller value of m. In fact, by using even higher-order approximations, it is possible to show that $H_1 + H_2$ can be simulated for time t with only $O(t^{1+\delta})$, for any fixed $\delta > 0$, no matter how small.

A Hamiltonian that is a sum of polynomially many terms can be efficiently simulated by composing the simulation of two terms, or by directly using an approximation to the identity

$$e^{-i(H_1+\cdots+H_k)t} = \lim_{m\to\infty} \left(e^{-iH_1t/m}\cdots e^{-iH_kt/m}\right)^m. \tag{10}$$

Another way of combining Hamiltonians comes from commutation: if H_1 and H_2 can be efficiently simulated, then $i[H_1, H_2]$ can be efficiently simulated. This is a consequence of the identity

$$e^{[H_1,H_2]t} = \lim_{m\to\infty} \left(e^{-iH_1\sqrt{t/m}}e^{-iH_2\sqrt{t/m}}e^{iH_1\sqrt{t/m}}e^{iH_2\sqrt{t/m}}\right)^m, \tag{11}$$

which can again be approximated with a finite number of terms. However, I don't know of any algorithmic application of such a simulation.

Sparse Hamiltonians

We will say that an $N \times N$ Hermitian matrix is *sparse* (in a fixed basis) if, in any fixed row, there are only $\mathrm{poly}(\log N)$ nonzero entries. The simulation techniques described above allow us to efficiently simulate sparse Hamiltonians. More precisely, suppose that for any a, we can efficiently determine all of the bs for which $\langle a|H|b\rangle$ is nonzero, as well as the values of the corresponding matrix elements;

then H can be efficiently simulated. In particular, this gives an efficient implementation of the continuous-time quantum walk on any graph $G = (V, E)$ whose maximum degree is $\mathrm{poly}(\log |V|)$.

The basic idea of the simulation is to edge-color the graph, simulate the edges of each color separately, and combine these pieces using (7). The main new technical ingredient in the simulation is a means of coloring the edges of the graph of nonzero matrix elements of H. A classic result in graph theory (Vizing's Theorem) says that a graph of maximum degree d has an edge coloring with at most $d+1$ colors (in fact, the edge chromatic number is either d or $d+1$). If we are willing to accept a polynomial overhead in the number of colors used, then we can actually find an edge coloring using only local information about the graph.

Lemma. *Suppose we are given an undirected graph G with N vertices and maximum degree d, and that we can efficiently compute the neighbors of any given vertex. Then there is an efficiently computable function $c(a, b) = c(b, a)$ taking $\mathrm{poly}(d, \log N)$ values such that for all a, $c(a, b) = c(a, b')$ implies $b = b'$. In other words, $c(a, b)$ is a coloring of G.*

Here is a simple proof showing that $O(d^2 \log N)$ colors are sufficient (note that stronger results are possible):

Proof. Number the vertices of G from 1 through N. For any vertex a, let $\mathrm{idx}(a, b)$ denote the index of vertex b in the list of neighbors of a. Also, let $k(a, b)$ be the index of the first bit at which a and b differ. Note that $k(a, b) = k(b, a)$, and $k \leq \lceil \log_2 N \rceil$.

For $a < b$, define the color of the edge ab to be the 4-tuple

$$c(a, b) := \big(\mathrm{idx}(a, b), \mathrm{idx}(b, a), k(a, b), b_{k(a,b)}\big) \tag{12}$$

where b_k denotes the kth bit of b. For $a > b$, define $c(a, b) := c(b, a)$.

Now suppose $c(a, b) = c(a, b')$. There are four possible cases:

1. Suppose $a < b$ and $a < b'$. Then the first component of c shows that $\mathrm{idx}(a, b) = \mathrm{idx}(a, b')$, which implies $b = b'$.

2. Suppose $a > b$ and $a > b'$. Then the second component of c shows that $\mathrm{idx}(a, b) = \mathrm{idx}(a, b')$, which implies $b = b'$.

3. Suppose $a < b$ and $a > b'$. Then from the third and fourth components of c, $k(a, b) = k(a, b')$ and $a_{k(a,b)} = b_{k(a,b)}$, which is a contradiction.

4. Suppose $a > b$ and $a < b'$. Then from the third and fourth components of c, $k(a, b) = k(a, b')$ and $a_{k(a,b)} = b'_{k(a,b)}$, which is a contradiction.

Each case that does not lead to a contradiction gives rise to a valid coloring, which completes the proof. □

Given this lemma, the simulation proceeds as follows. Write H as a diagonal matrix plus a matrix with zeros on the diagonal. We have already shown how to simulate the diagonal part, so we can assume H has zeros on the diagonal without loss of generality.

It suffices to simulate the term corresponding to the edges of a particular color c. We show how to make the simulation work for any particular vertex x; then it works in general by linearity. By computing the complete list of neighbors of x and computing each of their colors, we can reversibly

compute $v_c(x)$, the vertex adjacent to x via an edge with color c, along with the associated matrix element:

$$|x\rangle \mapsto |x, v_c(x), H_{x,v_c(x)}\rangle. \tag{13}$$

Then we can simulate the H-independent Hamiltonian defined by the map

$$|x, y, h\rangle \mapsto h|y, x, h^*\rangle \tag{14}$$

since it is easily diagonalized, as it consists of a direct sum of two-dimensional blocks. Finally, we can uncompute the second and third registers. Before the uncomputation, the simulation produces a linear combination of the states $|x, v_c(x), H_{x,v_c(x)}\rangle$ and $|v_c(x), x, H^*_{x,v_c(x)}\rangle$. Since

$$|v_c(x), x, H^*_{x,v_c(x)}\rangle = |v_c(x), v_c(v_c(x)), H_{v_c(x),x}\rangle, \tag{15}$$

the uncomputation works identically for both components.

Measuring an operator

So far, we have focused on the simulation of Hamiltonian dynamics. However, it is also possible to view a Hermitian operator not as the generator of dynamics, but as a quantity to be measured. In a practical quantum simulation, the desired final measurement might be of this type. For example, we might want to measure the final energy of the system, and the final Hamiltonian could be a sum of noncommuting terms.

It turns out that any Hermitian operator that can be efficiently simulated (viewing it as the Hamiltonian of a quantum system) can also be efficiently *measured* using a formulation of the quantum measurement process given by von Neumann. In fact, von Neumann's procedure is essentially the same as quantum phase estimation!

In von Neumann's description of the measurement process, a measurement is performed by coupling the system of interest to an ancillary system, which we call the *pointer*. Suppose that the pointer is a one-dimensional free particle and that the system-pointer interaction Hamiltonian is $H \otimes p$, where p is the momentum of the particle. Furthermore, suppose that the mass of the particle is sufficiently large that we can neglect the kinetic term. Then the resulting evolution is

$$e^{-itH\otimes p} = \sum_a \left[|E_a\rangle\langle E_a| \otimes e^{-itE_a p} \right], \tag{16}$$

where $|E_a\rangle$ are the eigenstates of H with eigenvalues E_a. Suppose we prepare the pointer in the state $|x = 0\rangle$, a narrow wave packet centered at $x = 0$. Since the momentum operator generates translations in position, the above evolution performs the transformation

$$|E_a\rangle \otimes |x = 0\rangle \to |E_a\rangle \otimes |x = tE_a\rangle. \tag{17}$$

If we can measure the position of the pointer with sufficiently high precision that all relevant spacings $x_{ab} = t|E_a - E_b|$ can be resolved, then measurement of the position of the pointer—a fixed, easy-to-measure observable, independent of H—effects a measurement of H.

Von Neumann's measurement protocol makes use of a continuous variable, the position of the pointer. To turn it into an algorithm that can be implemented on a digital quantum computer, we can approximate the evolution (16) using r quantum bits to represent the pointer. The full Hilbert

space is thus a tensor product of a 2^n-dimensional space for the system and a 2^r-dimensional space for the pointer. We let the computational basis of the pointer, with basis states $\{|z\rangle\}$, represent the basis of momentum eigenstates. The label z is an integer between 0 and $2^r - 1$, and the r bits of the binary representation of z specify the states of the r qubits. In this basis, p acts as

$$p|z\rangle = \frac{z}{2^r}|z\rangle. \tag{18}$$

In other words, the evolution $e^{-itH\otimes p}$ can be viewed as the evolution e^{-itH} on the system for a time controlled by the value of the pointer.

Expanded in the momentum eigenbasis, the initial state of the pointer is

$$|x = 0\rangle = \frac{1}{2^{r/2}} \sum_{z=0}^{2^r-1} |z\rangle. \tag{19}$$

The measurement is performed by evolving under $H \otimes p$ for some appropriately chosen time t. After this evolution, the position of the simulated pointer can be measured by measuring the qubits that represent it in the x basis, i.e., the Fourier transform of the computational basis.

Note that this discretized von Neumann measurement procedure is equivalent to phase estimation. Recall that in the phase estimation problem, we are given an eigenvector $|\psi\rangle$ of a unitary operator U and asked to determine its eigenvalue $e^{i\phi}$. The algorithm uses two registers, one that initially stores $|\psi\rangle$ and one that will store an approximation of the phase ϕ. The first and last steps of the algorithm are Fourier transforms on the phase register. The intervening step is to perform the transformation

$$|\psi\rangle \otimes |z\rangle \to U^z|\psi\rangle \otimes |z\rangle, \tag{20}$$

where $|z\rangle$ is a computational basis state. If we take $|z\rangle$ to be a momentum eigenstate with eigenvalue z (i.e., if we choose a different normalization than in (18)) and let $U = e^{-iHt}$, this is exactly the transformation induced by $e^{-i(H\otimes p)t}$. Thus we see that the phase estimation algorithm for a unitary operator U is exactly von Neumann's prescription for measuring $i \ln U$.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 10: Continuous-time quantum walk

We now turn to our second major topic in quantum algorithms, the concept of quantum walk. In this lecture we will introduce continuous-time quantum walk as a natural analog of continuous-time classical random walk, and we'll see some examples of how the two kinds of processes differ.

Continuous-time quantum walk

Random walks come in two flavors: discrete- and continuous-time. It is easiest to define a quantum analog of a continuous-time random walk, so we consider this case first. Given a graph $G = (V, E)$, we define the continuous-time random walk on G as follows. Let A be the *adjacency matrix* of G, the $|V| \times |V|$ matrix with

$$A_{j,k} = \begin{cases} 1 & (j,k) \in E \\ 0 & (j,k) \notin E \end{cases} \tag{1}$$

for every pair $j, k \in V$. In particular, if we disallow self loops, then the diagonal of A is zero. There is another matrix associated with G that is nearly as important: the *Laplacian* of G, which has

$$L_{j,k} = \begin{cases} -\deg(j) & j = k \\ 1 & (j,k) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $\deg(j)$ denotes the degree of vertex j. (The Laplacian is sometimes defined differently than this—e.g., sometimes with the opposite sign. We use this definition because it makes L a discrete approximation of the Laplacian operator ∇^2 in the continuum.)

The continuous-time random walk on G is defined as the solution of the differential equation

$$\frac{\mathrm{d}}{\mathrm{d}t} p_j(t) = \sum_{k \in V} L_{jk}\, p_k(t). \tag{3}$$

Here $p_j(t)$ denotes the probability associated with vertex j at time t. This can be viewed as a discrete analog of the diffusion equation. Note that

$$\frac{\mathrm{d}}{\mathrm{d}t} \sum_{j \in V} p_j(t) = \sum_{j,k \in V} L_{jk}\, p_k(t) = 0 \tag{4}$$

(since the columns of L sum to 0), which shows that an initially normalized distribution remains normalized: the evolution of the continuous-time random walk for any time t is a *stochastic process*. The solution of the differential equation can be given in closed form as

$$p(t) = e^{Lt} p(0). \tag{5}$$

Now notice that the equation (3) is very similar to the Schrödinger equation

$$\mathrm{i}\frac{\mathrm{d}}{\mathrm{d}t}|\psi\rangle = H|\psi\rangle \tag{6}$$

except that it lacks the factor of i. If we simply insert this factor, and rename the probabilities $p_j(t)$ as quantum amplitudes $q_j(t) = \langle j|\psi(t)\rangle$ (where $\{|j\rangle \colon j \in V\}$ is an orthonormal basis for the Hilbert space), then we obtain the equation

$$\mathrm{i}\frac{\mathrm{d}}{\mathrm{d}t}q_j(t) = \sum_{k\in V} L_{jk}\, q_k(t), \tag{7}$$

which is simply the Schrödinger equation with the Hamiltonian given by the Laplacian of the graph. Since the Laplacian is a Hermitian operator, these dynamics preserve normalization in the sense that $\frac{\mathrm{d}}{\mathrm{d}t}\sum_{j\in V}|q_j(t)|^2 = 0$. Again the solution of the differential equation can be given in closed form, but here it is $|\psi(t)\rangle = e^{-iLt}|\psi(0)\rangle$.

We could also define a continuous-time quantum walk using any Hermitian Hamiltonian that respects the structure of G. For example, we could use the adjacency matrix A of G, even though this matrix cannot be used as the generator of a continuous-time classical random walk.

Random and quantum walks on the hypercube

Let's begin by investigating a simple, dramatic example of a difference between the behavior of random and quantum walks. Consider the Boolean hypercube, the graph with vertex set $V = \{0,1\}^n$ and edge set $E = \{(x,y) \in V^2 \colon \Delta(x,y) = 1\}$, where $\Delta(x,y)$ denotes the Hamming distance between the strings x and y. When $n = 1$, the hypercube is simply an edge, with adjacency matrix

$$\sigma_x := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{8}$$

For general n, the graph is the direct product of this graph with itself n times, and the adjacency matrix is

$$A = \sum_{j=1}^{n} \sigma_x^{(j)} \tag{9}$$

where $\sigma_x^{(j)}$ denotes the operator acting as σ_x on the jth bit, and as the identity on every other bit.

For simplicity, let's consider the quantum walk with the Hamiltonian given by the adjacency matrix. (In fact, since the graph is regular, the walk generated by the Laplacian would only differ by an overall phase.) Since the terms in the above expression for the adjacency matrix commute, the unitary operator describing the evolution of this walk is simply

$$e^{-\mathrm{i}At} = \prod_{j=1}^{n} e^{-\mathrm{i}\sigma_x^{(j)}t} \tag{10}$$

$$= \bigotimes_{j=1}^{n} \begin{pmatrix} \cos t & -\mathrm{i}\sin t \\ -\mathrm{i}\sin t & \cos t \end{pmatrix}. \tag{11}$$

After time $t = \pi/2$, this operator flips every bit of the state (up to an overall phase), mapping any input state $|x\rangle$ to the state $|\bar{x}\rangle$ corresponding to the opposite vertex of the hypercube.

In contrast, consider the continuous- or discrete-time random walk starting from the vertex x. It is not hard to show that the probability of reaching the opposite vertex $\bar{x}$ is exponentially small at any time, since the walk rapidly reaches the uniform distribution over all 2^n vertices of the hypercube. So this simple example shows that random and quantum walks can exhibit radically different behavior.

Random and quantum walks in one dimension

Perhaps the best-known example of a random walk is the case of an infinite path, with $V = \mathbb{Z}$ and $(j, k) \in E$ iff $|j - k| = 1$. It is well known that the random walk on this graph starting from the origin (in either continuous or discrete time) typically moves a distance proportional to $\sqrt{t}$ in time t. Now let's consider the corresponding quantum walk.

To calculate the behavior of the walk, it is helpful to diagonalize the Hamiltonian. The eigenstates of the Laplacian of the graph are the momentum states $|\hat{p}\rangle$ with components

$$\langle j|\hat{p}\rangle = e^{\mathrm{i}pj} \tag{12}$$

where $-\pi \leq p \leq \pi$. We have

$$\begin{aligned}
\langle j|L|\hat{p}\rangle &= \langle j+1|\hat{p}\rangle + \langle j-1|\hat{p}\rangle - 2\langle j|\hat{p}\rangle && (13)\\
&= (e^{\mathrm{i}p(j+1)} + e^{\mathrm{i}p(j-1)} - 2e^{\mathrm{i}pj}) && (14)\\
&= e^{\mathrm{i}pj}(e^{\mathrm{i}p} + e^{-\mathrm{i}p} - 2) && (15)\\
&= 2(\cos p - 1)\langle j|\hat{p}\rangle, && (16)
\end{aligned}$$

so the corresponding eigenvalue is $2(\cos p - 1)$. Thus the amplitude for the walk to move from j to k in time t is

$$\begin{aligned}
\langle k|e^{-\mathrm{i}Lt}|j\rangle &= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{-2\mathrm{i}t(\cos p - 1)}\langle k|\hat{p}\rangle\langle\hat{p}|j\rangle \,\mathrm{d}p && (17)\\
&= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{ip(k-j)-2it(\cos p-1)} \,\mathrm{d}p && (18)\\
&= e^{2\mathrm{i}t}(-\mathrm{i})^{k-j} J_{k-j}(2t) && (19)
\end{aligned}$$

where J_ν is the Bessel function of order ν. This expression can be understood using basic asymptotic properties of the Bessel function. For large values of ν, the function $J_\nu(t)$ is exponentially small in ν for $\nu \gg t$, of order $\nu^{-1/3}$ for $\nu \approx t$, and of order $\nu^{-1/2}$ for $\nu \ll t$. Thus (19) describes a wave propagating with speed 2.

We can use a similar calculation to exactly describe the corresponding continuous-time classical random walk, which is simply the analytic continuation of the quantum case with $t \to \mathrm{i}t$. Here the probability of moving from j to k in time t is

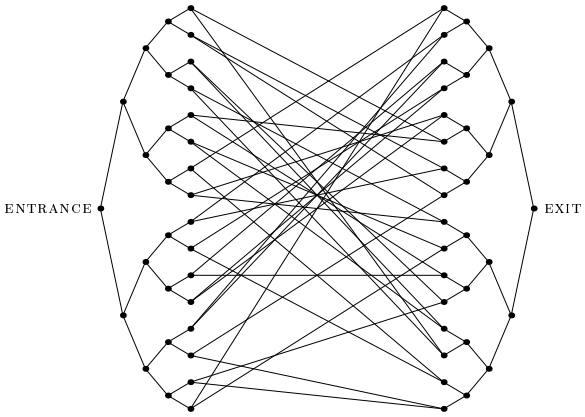
$$[e^{Lt}]_{kj} = e^{-2t} I_{k-j}(2t), \tag{20}$$

where I_ν is the modified Bessel function of order ν. For large t, this expression is approximately $\frac{1}{\sqrt{4\pi t}} \exp(-(k-j)^2/4t)$, a Gaussian of width $\sqrt{2t}$, in agreement with our expectations for a classical random walk in one dimension.

Black-box traversal of the glued trees graph

We have seen that the behavior of a quantum walk can be dramatically different from that of its classical counterpart. Next we will see an even stronger example of the power of quantum walk: a black-box problem that can be solved exponentially faster by a quantum walk than by *any* classical algorithm.

Consider a graph obtained by starting from two balanced binary trees of height n, and joining them by a random cycle of length $2 \cdot 2^n$ that alternates between the leaves of the two trees. For example, such a graph for $n = 4$ could look like the following:



Suppose we take a random walk on the graph starting from the root of the left tree. It is not hard to see that such a walk rapidly gets lost in the middle of the graph and never has a substantial probability of reaching the opposite root. In fact, by specifying the graph in such a way that it can only be explored locally, we can ensure that no classical procedure starting from the left root can efficiently reach the right root. However, a quantum walk starting from the left root produces a state with a large (lower bounded by $1/\operatorname{poly}(n)$) overlap on the right root in a short (upper bounded by $\operatorname{poly}(n)$) amount of time.

To establish a provable separation between classical and quantum strategies, we will formulate the graph traversal problem in terms of query complexity.

Let $G = (V, E)$ be a graph with N vertices. To represent G by a black box, let m be such that $2^m \geq N$, and let k be at least as large as the maximum degree of G. For each vertex $a \in V$, assign a distinct m-bit string (called the *name* of a), not assigning $11\ldots1$ as the name of any vertex. For each $b \in V$ with $(a, b) \in E$, assign a unique label from $\{1, 2, \ldots, k\}$ to the ordered pair (a, b). For $a \in \{0,1\}^m$ (identifying the vertex with its name) and $c \in \{1, 2, \ldots, k\}$, define $v_c(a)$ as the name of the vertex reached by following the outgoing edge of a labeled by c, if such an edge exists. If there is no vertex of G named a or no outgoing edge from a labeled c, then let $v_c(a) = 11\ldots1$. The black box for G takes $a \in \{0,1\}^m$ and $c \in \{1, 2, \ldots, k\}$ as input and returns $v_c(a)$.

The black box graph traversal problem is as follows. Let G be a graph and let ENTRANCE and EXIT be two vertices of G. Given a black box for G as described above, with the additional promise that the name of the ENTRANCE is $00\ldots0$, the goal is to output the name of the EXIT. We say an algorithm for this problem is efficient if its running time is polynomial in m.

Of course, a random walk is not necessarily the best classical strategy for this problem. For example, there is an efficient classical algorithm for traversing the n-dimensional hypercube (exercise: what is it?) even though a random walk does not work. However, no classical algorithm can efficiently traverse the glued trees, whereas a quantum walk can.

Quantum walk algorithm to traverse the glued trees graph

Given a black box for a graph G as specified above, we can efficiently compute a list of neighbors of any desired vertex, provided $k = \operatorname{poly}(m)$ (i.e., provided the maximum degree of the graph is not too large). Thus it is straightforward to simulate the dynamics of the continuous-time quantum walk on any such G, and in particular, on the glued trees graph (which has maximum degree 3). Our strategy for solving the traversal problem is simply to run the quantum walk and show that the resulting state has a substantial overlap on the EXIT for some $t = \operatorname{poly}(n)$.

Let G be the glued trees graph. The dynamics of the quantum walk on this graph are dramatically simplified because of symmetry. Consider the basis of states $|\mathrm{col}\, j\rangle$ that are uniform superpositions over the vertices at distance j from the ENTRANCE, i.e.,

$$|\mathrm{col}\, j\rangle := \frac{1}{\sqrt{N_j}} \sum_{\delta(a,\mathrm{ENTRANCE})=j} |a\rangle \tag{21}$$

where

$$N_j := \begin{cases} 2^j & 0 \le j \le n \\ 2^{2n+1-j} & n+1 \le j \le 2n+1 \end{cases} \tag{22}$$

is the number of vertices at distance j from the ENTRANCE, and where $\delta(a,b)$ denotes the length of the shortest path in G from a to b. It is straightforward to see that the subspace $\mathrm{span}\{|\mathrm{col}\, j\rangle : 0 \le j \le 2n+1\}$ is invariant under the action of the adjacency matrix A of G. At the ENTRANCE and EXIT, we have

$$A|\mathrm{col}\, 0\rangle = \sqrt{2}|\mathrm{col}\, 1\rangle \tag{23}$$

$$A|\mathrm{col}\, 2n+1\rangle = \sqrt{2}|\mathrm{col}\, 2n\rangle. \tag{24}$$

For any $0 < j < n$, we have

$$A|\mathrm{col}\, j\rangle = \frac{1}{\sqrt{N_j}} \sum_{\delta(a,\mathrm{ENTRANCE})=j} A|a\rangle \tag{25}$$

$$= \frac{1}{\sqrt{N_j}} \left(2 \sum_{\delta(a,\mathrm{ENTRANCE})=j-1} |a\rangle + \sum_{\delta(a,\mathrm{ENTRANCE})=j+1} |a\rangle \right) \tag{26}$$

$$= \frac{1}{\sqrt{N_j}} (2\sqrt{N_{j-1}}|\mathrm{col}\, j-1\rangle + \sqrt{N_{j+1}}|\mathrm{col}\, j+1\rangle) \tag{27}$$

$$= \sqrt{2}(|\mathrm{col}\, j-1\rangle + |\mathrm{col}\, j+1\rangle). \tag{28}$$

Similarly, for any $n+1 < j < 2n+1$, we have

$$A|\mathrm{col}\, j\rangle = \frac{1}{\sqrt{N_j}} (\sqrt{N_{j-1}}|\mathrm{col}\, j-1\rangle + 2\sqrt{N_{j+1}}|\mathrm{col}\, j+1\rangle) \tag{29}$$

$$= \sqrt{2}(|\mathrm{col}\, j-1\rangle + |\mathrm{col}\, j+1\rangle). \tag{30}$$

The only difference occurs at the middle of the graph, where we have

$$A|\mathrm{col}\, n\rangle = \frac{1}{\sqrt{N_n}} (2\sqrt{N_{n-1}}|\mathrm{col}\, n-1\rangle + 2\sqrt{N_{n+1}}|\mathrm{col}\, n+1\rangle) \tag{31}$$

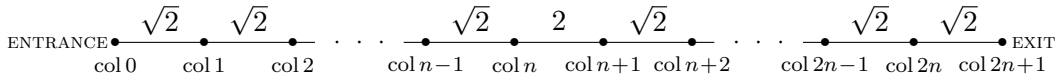
$$= \sqrt{2}|\mathrm{col}\, n-1\rangle + 2|\mathrm{col}\, n+1\rangle \tag{32}$$

and similarly

$$A|\mathrm{col}\, n+1\rangle = \frac{1}{\sqrt{N_{n+1}}} (2\sqrt{N_n}|\mathrm{col}\, n\rangle + 2\sqrt{N_{n+2}}|\mathrm{col}\, n+2\rangle) \tag{33}$$

$$= 2|\mathrm{col}\, n\rangle + \sqrt{2}|\mathrm{col}\, n+2\rangle. \tag{34}$$

In summary, the matrix elements of A between basis states for this invariant subspace can be depicted as follows:



By identifying the subspace of states $|\mathrm{col}\, j\rangle$, we have found that the quantum walk on the glued trees graph starting from the ENTRANCE is effectively the same as a quantum walk on a weighted path of $2n+2$ vertices, with all edge weights the same except for the middle one. Given our example of the quantum walk on the infinite path, we can expect this walk to reach the EXIT with amplitude $1/\operatorname{poly}(n)$ in time linear in n. To prove that the walk indeed reaches the EXIT in polynomial time, we will use the notion of the *mixing time* of a quantum walk.

Classical and quantum mixing

Informally, the mixing time of a random walk is the amount of time it takes to come close to a stationary distribution. Recall that the continuous-time random walk on a graph $G = (V, E)$ with Laplacian L is defined as the solution of the differential equation $\frac{\mathrm{d}p(t)}{\mathrm{d}t} = Lp(t)$, where $p(t) \in \mathbb{R}^{|V|}$ denotes a vector of probabilities for the walk to be at each vertex at time t. The uniform distribution over the vertices, $u := (1, 1, \ldots, 1)/|V|$, is an eigenvector of L with eigenvalue 0. Indeed, if G is connected, then this is the unique eigenvector with this eigenvalue. Letting v_λ denote a normalized eigenvector of L with eigenvalue λ (so that $L = \sum_{\lambda \neq 0} \lambda v_\lambda v_\lambda^T$), we have

$$p(t) = e^{Lt} p(0) \tag{35}$$

$$= \left(|V| uu^T + \sum_{\lambda \neq 0} e^{\lambda t} v_\lambda v_\lambda^T \right) p(0) \tag{36}$$

$$= \langle |V| u, p(0) \rangle u + \sum_{\lambda \neq 0} e^{\lambda t} \langle v_\lambda, p(0) \rangle v_\lambda \tag{37}$$

$$= u + \sum_{\lambda \neq 0} e^{\lambda t} \langle v_\lambda, p(0) \rangle v_\lambda \tag{38}$$

(where in exponentiating L we have used the fact that $\sqrt{|V|}u$ is a normalized eigenvector of L, so that $|V|uu^T$ is the projector onto the corresponding subspace). The Laplacian is a negative semidefinite operator, so the contributions $e^{\lambda t}$ for $\lambda \neq 0$ decrease exponentially in time; thus the walk asymptotically approaches the uniform distribution. The deviation from uniform is small when t is large compared to the inverse of the largest (i.e., least negative) nonzero eigenvalue of L.

Since a quantum walk is a unitary process, we should not expect it to approach a limiting quantum state, no matter how long we wait. Nevertheless, it is possible to define a notion of the limiting distribution of a quantum walk as follows. Suppose we pick a time t uniformly at random between 0 and T, run the quantum walk starting at $a \in V$ for a total time t, and then measure in the vertex basis. The resulting distribution is

$$p_{a \to b}(T) = \frac{1}{T} \int_0^T |\langle b | e^{-iHt} | a \rangle|^2 \mathrm{d}t \tag{39}$$

$$= \sum_{\lambda, \lambda'} \langle b|\lambda\rangle\langle\lambda|a\rangle\langle a|\lambda'\rangle\langle\lambda'|v\rangle \frac{1}{T} \int_0^T e^{-i(\lambda - \lambda')t} \mathrm{d}t \tag{40}$$

$$= \sum_{\lambda} |\langle a|\lambda\rangle\langle b|\lambda\rangle|^2 + \sum_{\lambda \neq \lambda'} \langle b|\lambda\rangle\langle\lambda|a\rangle\langle a|\lambda'\rangle\langle\lambda'|b\rangle \frac{1 - e^{-i(\lambda - \lambda')T}}{i(\lambda - \lambda')T} \tag{41}$$

where we have considered a quantum walk generated by an unspecified Hamiltonian H (it could be the Laplacian or the adjacency matrix, or some other operator as desired), and where we have assumed for simplicity that the spectrum of $H = \sum_\lambda \lambda|\lambda\rangle\langle\lambda|$ is nondegenerate. We see that the distribution $p_{a\to b}(T)$ tends toward a limiting distribution

$$p_{a\to b}(\infty) := \sum_\lambda |\langle a|\lambda\rangle\langle b|\lambda\rangle|^2. \tag{42}$$

The timescale for approaching this distribution is again governed by the spectrum of H, but now we see that T must be large compared to the inverse of the smallest gap between any pair of distinct eigenvalues, not just the smallest gap between a particular pair of eigenvalues as in the classical case.

Let's apply this notion of quantum mixing to the quantum walk on the glued trees. It will be simplest to consider the walk generated by the adjacency matrix A. Since the subspace of states $|\mathrm{col}\, j\rangle$ has dimension only $2n+1$, it should not be surprising that the limiting probability of traversing from ENTRANCE to EXIT is bigger than $1/\operatorname{poly}(n)$. To see this, notice that A commutes with the reflection operator R defined as $R|\mathrm{col}\, j\rangle = |\mathrm{col}\, 2n+1-j\rangle$, so these two operators can be simultaneously diagonalized. Now $R^2 = 1$, so it has eigenvalues ± 1, which shows that we can choose the eigenstates $|\lambda\rangle$ of A to satisfy $\langle\textsc{entrance}|\lambda\rangle = \pm\langle\textsc{exit}|\lambda\rangle$. Therefore,

$$\begin{aligned}
p_{\textsc{entrance}\to\textsc{exit}}(\infty) &= \sum_\lambda |\langle\textsc{entrance}|\lambda\rangle\langle\textsc{exit}|\lambda\rangle|^2 && (43)\\
&= \sum_\lambda |\langle\textsc{entrance}|\lambda\rangle|^4 && (44)\\
&\geq \frac{1}{2n+2}\left(\sum_\lambda |\langle\textsc{entrance}|\lambda\rangle|^2\right)^2 && (45)\\
&= \frac{1}{2n+2} && (46)
\end{aligned}$$

where the lower bound follows by the Cauchy-Schwarz inequality. Thus it suffices to show that the mixing time of the quantum walk is $\operatorname{poly}(n)$.

To see how long we must wait before the probability of reaching the EXIT is close to its limiting value, we can calculate

$$\begin{aligned}
&|p_{\textsc{entrance}\to\textsc{exit}}(\infty) - p_{\textsc{entrance}\to\textsc{exit}}(T)|\\
&\quad= \left|\sum_{\lambda\neq\lambda'} \langle\textsc{exit}|\lambda\rangle\langle\lambda|\textsc{entrance}\rangle\langle\textsc{entrance}|\lambda'\rangle\langle\lambda'|\textsc{exit}\rangle \frac{1-e^{-i(\lambda-\lambda')T}}{i(\lambda-\lambda')T}\right| && (47)\\
&\quad\leq \frac{2}{\Delta T}\sum_{\lambda,\lambda'} |\langle\textsc{exit}|\lambda\rangle\langle\lambda|\textsc{entrance}\rangle\langle\textsc{entrance}|\lambda'\rangle\langle\lambda'|\textsc{exit}\rangle| && (48)\\
&\quad= \frac{2}{\Delta T}\sum_{\lambda,\lambda'} |\langle\textsc{entrance}|\lambda\rangle|^2 |\langle\textsc{entrance}|\lambda'\rangle|^2 && (49)\\
&\quad= \frac{2}{\Delta T}, && (50)
\end{aligned}$$

where Δ denotes the smallest gap between any pair of distinct eigenvalues of A. All that remains is to lower bound Δ.

To understand the spectrum of A, recall that an infinite path has eigenstates of the form e^{ipj}. For any value of p, the state $|\lambda\rangle$ with amplitudes $\langle \mathrm{col}\, j|\lambda\rangle = e^{ipj}$ satisfies $\langle \mathrm{col}\, j|A|\lambda\rangle = \lambda\langle \mathrm{col}\, j|\lambda\rangle$, where the eigenvalue is $\lambda = 2\sqrt{2}\cos p$, for all values of j except $0, n, n+1, 2n+1$. We can satisfy the eigenvalue condition for $j = 0, 2n+1$ by taking linear combinations of $e^{\pm ipj}$ that vanish for $j = -1$ and $j = 2n+2$, namely

$$\langle \mathrm{col}\, j|\lambda\rangle = \begin{cases} \sin(p(j+1)) & 0 \le j \le n \\ \pm\sin(p(2n+2-j)) & n+1 \le j \le 2n+1. \end{cases} \tag{51}$$

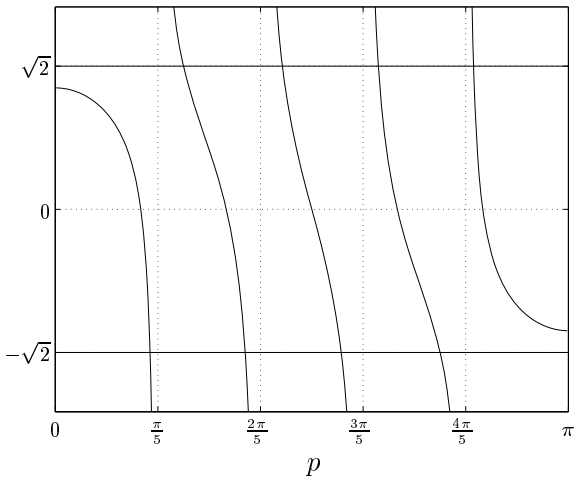
Finally, we can enforce the eigenvalue condition at $j = n$ (which automatically enforces it at $j = n+1$ by symmetry), which restricts the values of p to a finite set. We have

$$\sqrt{2}\sin(pn) \pm 2\sin(p(n+1)) = 2\sqrt{2}\cos(p)\sin(p(n+1)), \tag{52}$$

which can be simplified to

$$\frac{\sin(p(n+2))}{\sin(p(n+1))} = \pm\sqrt{2}. \tag{53}$$

The left hand side of this equation decreases monotonically, with poles at integer multiples of $\pi/(n+1)$. For example, with $n = 4$, we have the following:



With a bit of analysis (see quant-ph/0209131 for details), one can show that the solutions of this equation give $2n$ values of p, each of which is separated from the integer multiples of $\pi/(n+1)$ by $\Omega(1/n^2)$. The spacings between the corresponding eigenvalues of A, $\lambda = 2\sqrt{2}\cos p$, are $\Omega(1/n^3)$. The remaining two eigenvalues of A can be obtained by considering solutions with p imaginary, and it is easy to show that they are separated from the rest of the spectrum by a constant amount. By taking (say) $T = 5n/\Delta = O(n^4)$, we can ensure that the probability to reach the EXIT is $\Omega(1/n)$. Thus there is an efficient quantum algorithm to traverse the glued trees graph.

Classical lower bound

It remains to show that this problem is difficult for a classical computer. A formal proof of this fact can be given using a sequence of reductions to problems that are essentially no easier than the original one, but that restrict the nature of the allowed algorithms. Here we will simply sketch the main ideas.

First, note that if we name the vertices at random using strings of about, say, $2 \log |V|$ bits, then there are exponentially many more possible names than there are actual vertices. Since the probability that a randomly guessed name corresponds to a vertex of the graph is exponentially small, we can essentially restrict our attention to algorithms that query a connected set of vertices, starting from the ENTRANCE (the only vertex whose name is known initially).

Next, suppose we consider the algorithm to succeed not only if it reaches the EXIT, but also if it manages to find a cycle in the graph. This only makes it easier for the algorithm to succeed, but not significantly so, since it turns out to be hard even to find a cycle.

Now we can restrict our attention to the steps the algorithm takes before it finds a cycle. Notice that for such steps, the names supplied by the black box provide no information whatsoever about the structure of the graph: they could just as well be simulated by a sequence of random responses. Therefore, we can think of an algorithm as simply producing a rooted binary tree and embedding it into the glued trees graph at random. To show that the algorithm fails, it suffices to show that under such a random embedding, the probability of any rooted binary tree giving rise to a cycle or reaching the exit is small. By a fairly straightforward probabilistic argument, one can show that even for exponentially large trees (say, having at most $2^{n/6}$ vertiecs), the probability of the embedded tree giving rise to a cycle or reaching the exit is exponentially small. Thus any classical algorithm for solving the black box glued trees traversal problem must make exponentially many queries to succeed with more than exponentially small probability.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 11: Discrete-time quantum walk

In the last lecture we introduced the notion of continuous-time quantum walk. We now turn our attention to discrete-time quantum walk, which provides a convenient framework for quantum search algorithms.

Discrete-time quantum walk

It is trickier to define a quantum analog of a discrete-time random walk than of a continuous-time random walk. In the simplest discrete-time random walk on G, at each time step we simply move from any given vertex to each of its neighbors with equal probability. Thus the walk is governed by the $|V| \times |V|$ matrix M with entries

$$M_{jk} = \begin{cases} 1/\deg(k) & (j,k) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

for $j, k \in V$: an initial probability distribution p over the vertices evolves to $p' = Mp$ after one step of the walk.

To define a quantum analog of this process, we would like to specify a unitary operator U with the property that an input state $|j\rangle$ corresponding to the vertex $j \in V$ evolves to a superposition of the neighbors of j. We would like this to happen in essentially the same way at every vertex, so we are tempted to propose the definition

$$|j\rangle \stackrel{?}{\mapsto} |\partial_j\rangle := \frac{1}{\sqrt{\deg(j)}} \sum_{k:(j,k)\in E} |k\rangle. \tag{2}$$

However, a moment's reflection shows that this typically does not define a unitary transformation, since the orthogonal states $|j\rangle$ and $|k\rangle$ corresponding to adjacent vertices j, k with a common neighbor ℓ evolve to non-orthogonal states. We could potentially avoid this problem using a rule that sometimes introduces phases, but that would violate the spirit of defining a process that behaves in the same way at every vertex. In fact, even if we give that up, there are some graphs that simply do not allow local unitary dynamics.

We can get around this difficulty if we allow ourselves to enlarge the Hilbert space, an idea proposed by Watrous as part of a logarithmic-space quantum algorithm for deciding whether two vertices are connected in a graph. Let the Hilbert space consist of states of the form $|j,k\rangle$ where $(j,k) \in E$. We can think of the walk as taking place on the (directed) edges of the graph; the state $|j,k\rangle$ represents a walker at vertex j that will move toward vertex k. Each step of the walk consists of two operations. First, we apply a unitary transformation that operates on the second register conditional on the first register. This transformation is sometimes referred to as a "coin flip," as it modifies the next destination of the walker. A common choice is the Grover diffusion operator over the neighbors of j, namely

$$C := \sum_{j\in V} |j\rangle\langle j| \otimes \big(2|\partial_j\rangle\langle\partial_j| - I\big). \tag{3}$$

Next, the walker is moved to the vertex indicated in the second register. Of course, since the process must be unitary, the only way to do this is to swap the two registers using the operator

$$S := \sum_{(j,k)\in E} |j,k\rangle\langle k,j|. \tag{4}$$

Overall, one step of the discrete-time quantum walk is described by the unitary operator SC.

In principle, this construction can be used to define a discrete-time quantum walk on any graph (although care must be taken if the graph is not regular). However, in practice it is often more convenient to use an alternative framework introduced by Szegedy, as described in the next section.

How to quantize a Markov chain

A discrete-time classical random walk on an N-vertex graph can be represented by an $N \times N$ matrix P. The entry P_{jk} represents the probability of making a transition to k from j, so that an initial probability distribution $p \in \mathbb{R}^N$ becomes Pp after one step of the walk. To preserve normalization, we must have $\sum_{j=1}^N P_{jk} = 1$; we say that such a matrix is *stochastic*.

For any $N \times N$ stochastic matrix P (not necessarily symmetric), we can define a corresponding discrete-time quantum walk, a unitary operation on the Hilbert space $\mathbb{C}^N \otimes \mathbb{C}^N$. To define this walk, we introduce the states

$$|\psi_j\rangle := |j\rangle \otimes \sum_{k=1}^N \sqrt{P_{kj}}|k\rangle \tag{5}$$

$$= \sum_{k=1}^N \sqrt{P_{kj}}|j,k\rangle \tag{6}$$

for $j = 1, \ldots, N$. Each such state is normalized since P is stochastic. Now let

$$\Pi := \sum_{j=1}^N |\psi_j\rangle\langle\psi_j| \tag{7}$$

denote the projection onto $\mathrm{span}\{|\psi_j\rangle : j = 1, \ldots, N\}$, and let

$$S := \sum_{j,k=1}^N |j,k\rangle\langle k,j| \tag{8}$$

be the operator that swaps the two registers. Then a single step of the quantum walk is defined as the unitary operator $U := S(2\Pi - 1)$.

Notice that if $P_{jk} = A_{jk}/\deg(k)$ (i.e., if the walk simply chooses an outgoing edge of an underlying digraph uniformly at random), then this is exactly the coined quantum walk with the Grover diffusion operator as the coin flip.

If we take two steps of the walk, then the corresponding unitary operator is

$$[S(2\Pi - 1)][S(2\Pi - 1)] = [S(2\Pi - 1)S][2\Pi - 1] \tag{9}$$

$$= (2S\Pi S - 1)(2\Pi - 1), \tag{10}$$

which can be interpreted as the reflection about $\mathrm{span}\{|\psi_j\rangle\}$ followed by the reflection about $\mathrm{span}\{S|\psi_j\rangle\}$ (the states where we condition on the second register to do a coin operation on the first). To understand the behavior of the walk, we will now compute the spectrum of U; but note that it is also possible to compute the spectrum of a product of reflections more generally.

Spectrum of the quantum walk

To understand the behavior of a discrete-time quantum walk, it will be helpful to compute its spectral decomposition. Let us show the following:

Theorem. *Fix an $N \times N$ stochastic matrix P, and let $\{|\lambda\rangle\}$ denote a complete set of orthonormal eigenvectors of the $N \times N$ matrix D with entries $D_{jk} = \sqrt{P_{jk}P_{kj}}$ with eigenvalues $\{\lambda\}$. Then the eigenvalues of the discrete-time quantum walk $U = S(2\Pi - 1)$ corresponding to P are ± 1 and $\lambda \pm i\sqrt{1-\lambda^2} = e^{\pm i \arccos \lambda}$.*

Proof. Define an isometry

$$T := \sum_{j=1}^{N} |\psi_j\rangle\langle j| \tag{11}$$

$$= \sum_{j,k=1}^{N} \sqrt{P_{kj}}|j,k\rangle\langle j| \tag{12}$$

mapping states in $\mathbb{C}^n$ to states in $\mathbb{C}^n \otimes \mathbb{C}^n$, and let $|\tilde{\lambda}\rangle := T|\lambda\rangle$. Notice that

$$TT^\dagger = \sum_{j,k=1}^{N} |\psi_j\rangle\langle j|k\rangle\langle\psi_k| \tag{13}$$

$$= \sum_{j=1}^{N} |\psi_j\rangle\langle\psi_j| \tag{14}$$

$$= \Pi, \tag{15}$$

whereas

$$T^\dagger T = \sum_{j,k=1}^{N} |j\rangle\langle\psi_j|\psi_k\rangle\langle k| \tag{16}$$

$$= \sum_{j,k,\ell,m=1}^{N} \sqrt{P_{\ell j}P_{mk}}|j\rangle\langle j,\ell|k,m\rangle\langle k| \tag{17}$$

$$= \sum_{j,\ell=1}^{N} P_{\ell j}|j\rangle\langle j| \tag{18}$$

$$= I \tag{19}$$

and

$$T^\dagger S T = \sum_{j,k=1}^{N} |j\rangle\langle\psi_j|S|\psi_k\rangle\langle k| \tag{20}$$

$$= \sum_{j,k,\ell,m=1}^{N} \sqrt{P_{\ell j}P_{mk}}|j\rangle\langle j,\ell|S|k,m\rangle\langle k| \tag{21}$$

$$= \sum_{j=1}^{N} \sqrt{P_{jk}P_{kj}}|j\rangle\langle k| \tag{22}$$

$$= D. \tag{23}$$

Applying the walk operator U to $|\tilde{\lambda}\rangle$ gives

$$\begin{aligned} U|\tilde{\lambda}\rangle &= S(2\Pi - 1)|\tilde{\lambda}\rangle && (24)\\ &= S(2TT^\dagger - 1)T|\lambda\rangle && (25)\\ &= 2ST|\lambda\rangle - ST|\lambda\rangle && (26)\\ &= S|\tilde{\lambda}\rangle, && (27) \end{aligned}$$

and applying U to $S|\tilde{\lambda}\rangle$ gives

$$\begin{aligned} US|\tilde{\lambda}\rangle &= S(2\Pi - 1)S|\tilde{\lambda}\rangle && (28)\\ &= S(2TT^\dagger - 1)ST|\lambda\rangle && (29)\\ &= (2STD - T)|\lambda\rangle && (30)\\ &= 2\lambda S|\tilde{\lambda}\rangle - |\tilde{\lambda}\rangle. && (31) \end{aligned}$$

We see that the subspace $\mathrm{span}\{|\tilde{\lambda}\rangle, S|\tilde{\lambda}\rangle\}$ is invariant under U, so we can find eigenvectors of U within this subspace.

Now let $|\mu\rangle := |\tilde{\lambda}\rangle - \mu S|\tilde{\lambda}\rangle$, and let us choose $\mu \in \mathbb{C}$ so that $|\mu\rangle$ is an eigenvector of U. We have

$$\begin{aligned} U|\mu\rangle &= S|\tilde{\lambda}\rangle - \mu(2\lambda S|\tilde{\lambda}\rangle - |\tilde{\lambda}\rangle) && (32)\\ &= \mu|\tilde{\lambda}\rangle + (1 - 2\lambda\mu)S|\tilde{\lambda}\rangle. && (33) \end{aligned}$$

Thus μ will be an eigenvalue of U corresponding to the eigenvector $|\mu\rangle$ provided $(1-2\lambda\mu) = \mu(-\mu)$, i.e. $\mu^2 - 2\lambda\mu + 1 = 0$, so

$$\mu = \lambda \pm i\sqrt{1-\lambda^2}. \tag{34}$$

Finally, note that for any vector in the orthogonal complement of $\mathrm{span}\{|\tilde{\lambda}\rangle\} = \mathrm{span}\{|\psi_j\rangle\}$ (these spaces are the same since $\sum_\lambda |\tilde{\lambda}\rangle\langle\tilde{\lambda}| = \sum_\lambda T|\lambda\rangle\langle\lambda|T^\dagger = TT^\dagger = \Pi$), U simply acts as $-S$, which has eigenvalues ± 1. □

Hitting times

We can use random walks to formulate a generic search algorithm, and quantizing this algorithm gives a generic square root speedup. Consider a graph $G = (V, E)$, with some subset $M \subset V$ of the vertices designated as *marked*. We will compare classical and quantum walk algorithms for deciding whether any vertex in G is marked.

Classically, a straightforward approach to this problem is to take a random walk defined by some stochastic matrix P, stopping if we encounter a marked vertex. In other words, we modify the original walk P to give a walk P' defined as

$$P'_{jk} = \begin{cases} 1 & k \in M \text{ and } j = k \\ 0 & k \in M \text{ and } j \neq k \\ P_{jk} & k \notin M. \end{cases} \tag{35}$$

Let us assume from now on that the original walk P is symmetric, though the modified walk P' clearly is not provided M is non-empty. If we order the vertices so that the marked ones come last, the matrix P' has the block form

$$P' = \begin{pmatrix} P_M & 0 \\ Q & I \end{pmatrix} \tag{36}$$

where P_M is obtained by deleting the rows and columns of P corresponding to vertices in M.

Suppose we take t steps of the walk. A simple calculation shows

$$(P')^t = \begin{pmatrix} P_M^t & 0 \\ Q(I + P_M + \cdots + P_M^{t-1}) & I \end{pmatrix} \tag{37}$$

$$= \begin{pmatrix} P_M^t & 0 \\ Q\frac{P_M^t - I}{P_M - I} & I \end{pmatrix}. \tag{38}$$

Now if we start from the uniform distribution over unmarked items (if we start from a marked item we are done, so we might as well condition on this not happening), then the probability of not reaching a marked item after t steps is $\frac{1}{N-|M|}\sum_{j,k\notin M}[P_M^t]_{jk} \leq \|P_M^t\| = \|P_M\|^t$, where the inequality follows because the left hand side is the expectation of P_M^t in the normalized state $|V\setminus M\rangle = \frac{1}{\sqrt{N-|M|}}\sum_{j\notin M}|j\rangle$. Now if $\|P_M\| = 1-\Delta$, then the probability of reaching a marked item after t steps is at least $1 - \|P_M\|^t = 1-(1-\Delta)^t$, which is $\Omega(1)$ provided $t = O(1/\Delta) = O(\frac{1}{1-\|P_M\|})$.

It turns out that we can bound $\|P_M\|$ away from 1 knowing only the fraction of marked vertices and the spectrum of the original walk. Thus we can upper bound the *hitting time*, the time required to reach some marked vertex with constant probability.

Lemma. *If the second largest eigenvalue of P (in absolute value) is at most $1-\delta$ and $|M| \leq \epsilon N$, then $\|P_M\| \geq 1 - \delta\epsilon$.*

Proof. Let $|v\rangle \in \mathbb{R}^{N-|M|}$ be the principal eigenvector of P_M, and let $|w\rangle \in \mathbb{R}^N$ be the vector obtained by padding $|v\rangle$ with 0's for all the marked vertices.

We will decompose $|w\rangle$ in the eigenbasis of P. Since P is symmetric, it is actually doubly stochastic, and the uniform vector $|V\rangle = \frac{1}{\sqrt{N}}\sum_j |j\rangle$ corresponds to the eigenvalue 1. All other eigenvectors $|\lambda\rangle$ have eigenvalues at most $1-\delta$ by assumption. Now

$$\|P_M\| = \langle v|P_M|v\rangle \tag{39}$$

$$= \langle w|P|w\rangle \tag{40}$$

$$= |\langle V|w\rangle|^2 + \sum_{\lambda\neq 1}\lambda|\langle\lambda|w\rangle|^2 \tag{41}$$

$$\leq |\langle V|w\rangle|^2 + (1-\delta)\sum_{\lambda\neq 1}|\langle\lambda|w\rangle|^2 \tag{42}$$

$$= 1 - \delta\sum_{\lambda\neq 1}|\langle\lambda|w\rangle|^2 \tag{43}$$

$$= 1 - \delta(1 - |\langle V|w\rangle|^2). \tag{44}$$

But by the Cauchy-Schwarz inequality,

$$|\langle V|w\rangle|^2 = |\langle V|\Pi_{V\setminus M}|w\rangle|^2 \tag{45}$$

$$\leq \|\Pi_{V\setminus M}|V\rangle\|^2 \cdot \||w\rangle\|^2 \tag{46}$$

$$= \frac{N-|M|}{N} \tag{47}$$

$$= 1 - \epsilon \tag{48}$$

where $\Pi_{V\setminus M} = \sum_{j\notin M}|j\rangle\langle j|$. Thus $\|P_M\| \leq 1 - \delta\epsilon$ as claimed. □

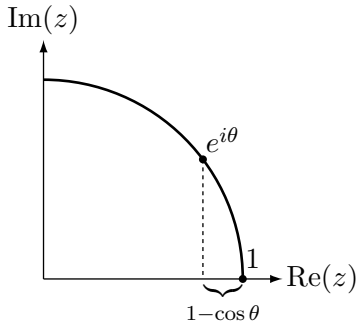


Figure 1: The classical gap, $1 - \lambda = 1 - \cos\theta$, appears on the real axis. The quantum phase gap, $\theta = \arccos\lambda$, is quadratically larger, since $\cos\theta \geq 1 - \theta^2/2$, i.e., $\arccos\lambda \geq \sqrt{2(1-\lambda)}$.

Thus we see that the classical hitting time is $O(1/\delta\epsilon)$.

Now we turn to the quantum case. Our strategy will be to perform phase estimation with sufficiently high precision on the operator U, the quantum walk corresponding to P', with the state

$$|\psi\rangle := \frac{1}{\sqrt{N}} \sum_{j \notin M} |\psi_j\rangle. \tag{49}$$

This state can easily be prepared by starting from the state

$$T|V\rangle = \frac{1}{\sqrt{N}} \sum_{j} |\psi_j\rangle \tag{50}$$

and measuring whether the first register corresponds to a marked vertex; if it does then we are done, and if not then we have prepared $|\psi\rangle$.

The matrix D for the walk P' is

$$\begin{pmatrix} P_M & 0 \\ 0 & I \end{pmatrix}, \tag{51}$$

so according to the spectral theorem, the eigenvalues of the resulting walk operator U are ± 1 and $e^{\pm i \arccos\lambda}$, where λ runs over the eigenvalues of P_M. If the marked set M is empty, then $P' = P$, and $|\psi\rangle$ is an eigenvector of U with eigenvalue 1, so phase estimation on U is guaranteed to return a phase of 0. But if M is non-empty, then the state $|\psi\rangle$ lives entirely within the subspace with eigenvalues $e^{\pm i \arccos\lambda}$. Thus if we perform phase estimation on U with precision $O(\min_\lambda \arccos\lambda)$, we will see a phase different from 0. Since $\arccos\lambda \geq \sqrt{2(1-\lambda)}$ (see Figure 1 for an illustration), we see that precision $O(\sqrt{1 - \|P_M\|})$ suffices. So the quantum algorithm can decide whether there is a marked vertex in time $O(1/\sqrt{1 - \|P_M\|}) = O(1/\sqrt{\delta\epsilon})$.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 12: Unstructured search

Now we begin to discuss applications of quantum walks to search algorithms. We start with the most basic of all search problems, the unstructured search problem (which is solved optimally by Grover's algorithm). We discuss how this problem fits into the framework of quantum walk search, and also describe amplitude amplification and quantum counting in this setting. We also discuss quantum walk algorithms for the search problem under locality constraints.

Unstructured search

In the unstructured search problem, we are given a black box function $f\colon S \to \{0,1\}$, where S is a finite set of size $|S| = N$. The inputs $x \in M$, where $M := \{x \in S\colon f(x) = 1\}$, are called *marked items*. In the decision version of the problem, our goal is to determine whether M is empty or not. We might also want to find a marked item when one exists.

It is quite easy to see that even the decision problem requires $\Omega(N)$ classical queries, and that N queries suffice, so the classical query complexity of unstructured search is $\Theta(N)$.

You should already be familiar with Grover's algorithm, which solves this problem using $O(\sqrt{N})$ quantum queries. Grover's algorithm works by starting from the state $|S\rangle := \sum_{x\in S} |x\rangle/\sqrt{N}$ and alternately applying the reflection about the set of marked items, $\sum_{x\in M} 2|x\rangle\langle x| - 1$, and the reflection about the state $|S\rangle$, $2|S\rangle\langle S| - 1$. The former can be implemented with two quantum queries to f, and the latter requires no queries to implement. It is straightforward to show that there is some $t = O(\sqrt{N/|M|})$ for which t steps of this procedure give a state with constant overlap on $|M\rangle$ (assuming M is non-empty), so that a measurement will reveal a marked item with constant probability.

It can be shown that unstructured search requires $\Omega(\sqrt{N/|M|})$ queries. We will prove this when we discuss adversary lower bounds.

Quantum walk algorithm

Consider the discrete-time random walk on the complete graph represented by the stochastic matrix

$$P = \frac{1}{N-1}\begin{pmatrix} 0 & 1 & 1 & \cdots & 1 \\ 1 & 0 & 1 & \cdots & 1 \\ 1 & 1 & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{pmatrix} \tag{1}$$

$$= \frac{N}{N-1}|S\rangle\langle S| - \frac{1}{N-1}I. \tag{2}$$

It has eigenvalues 1 (which is non-degenerate) and $-1/(N-1)$ (with degeneracy $N-1$). Since the graph is highly connected, its spectral gap is very large: we have $\delta = 1 - \frac{1}{N-1} = \frac{N}{N-1}$.

This random walk gives rise to a very simple classical algorithm for unstructured search. In this algorithm, we start from a uniformly random item and repeatedly choose a new item uniformly at

random from the other $N-1$ possibilities, stopping when we reach a marked item. The fraction of marked items is $\epsilon = |M|/N$, so the hitting time of this walk is

$$O\left(\frac{1}{\delta\epsilon}\right) = \frac{(N-1)N}{N|M|} = O(N/|M|) \tag{3}$$

(this is only an upper bound on the hitting time, but in this case we know it is optimal). Of course, if we have no a priori lower bound on $|M|$ in the event that M is non-empty, the best we can say is that $\epsilon \geq 1/N$, giving a running time $O(N)$.

The corresponding quantum walk search algorithm has a hitting time of

$$O\left(\frac{1}{\sqrt{\delta\epsilon}}\right) = O(\sqrt{N/|M|}), \tag{4}$$

corresponding to the running time of Grover's algorithm. To see that this actually gives an algorithm using $O(\sqrt{N/|M|})$ queries, we need to see that a step of the quantum walk can be performed using only $O(1)$ quantum queries. In the case where the first item is marked, the modified classical walk matrix is

$$P' = \frac{1}{N-1}\begin{pmatrix} N-1 & 1 & 1 & \cdots & 1 \\ 0 & 0 & 1 & \cdots & 1 \\ 0 & 1 & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 1 \\ 0 & 1 & \cdots & 1 & 0 \end{pmatrix}, \tag{5}$$

so that the vectors $|\psi_j\rangle$ are $|\psi_1\rangle = |1,1\rangle$ and $|\psi_j\rangle = |j, S \setminus \{j\}\rangle = \sqrt{\frac{N}{N-1}}|j,S\rangle - \frac{1}{\sqrt{N-1}}|j,j\rangle$ for $j = 2,\ldots,N$. With a general marked set M, the projector onto the span of these states is

$$\Pi = \sum_{j\in M} |j,j\rangle\langle j,j| + \sum_{j\notin M} |j, S\setminus\{j\}\rangle\langle j, S\setminus\{j\}|, \tag{6}$$

so the operator $2\Pi - 1$ acts as Grover diffusion over the neighbors when when the vertex is unmarked, and as a phase flip when the vertex is marked. (Note that since we start from the state $|\psi\rangle = \sum_{j\notin M}|\psi_j\rangle$, we stay in the subspace of states $\mathrm{span}\{|j,k\rangle : (j,k)\in E\}$, and in particular have zero support on any state $|j,j\rangle$ for $j \in V$, so $2\Pi - 1$ acts as -1 when the first register holds a marked vertex.) Each such step can be implemented using two queries of the black box, one to compute whether we are at a marked vertex and one to uncompute that information; the subsequent swap operation requires no queries. Thus the query complexity is indeed $O(\sqrt{N/|M|})$.

This algorithm is not exactly the same as Grover's; for example, it works in the Hilbert space $\mathbb{C}^N \otimes \mathbb{C}^N$ instead of $\mathbb{C}^N$. Nevertheless, it is clearly closely related. In particular, notice that in Grover's algorithm, the unitary operation $2|S\rangle\langle S| - 1$ can be viewed as a kind of discrete-time quantum walk on the complete graph, where in this particular case no coin is necessary to define the walk.

The algorithm we have described so far only solves the decision version of unstructured search. To find marked item, we could use bisection, but this would introduce a logarithmic overhead. In fact, it can be shown that the final state of the quantum walk algorithm actually encodes a marked item when one exists.

Amplitude amplification and quantum counting

We briefly mention some other concepts related to unstructured search that provide useful tools for quantum algorithms in general. These ideas are typically presented in the context of Grover's algorithm; he were describe them in the framework of quantum walk search. This is slightly less space efficient, but the essential ideas are the same.

Amplitude amplification is a general method for boosting the success probability of a (classical or quantum) subroutine. It can be implemented by quantum walk search as follows. Suppose we have a procedure that produces a correct answer with probability p (i.e., with an amplitude of magnitude $\sqrt{p}$ if we view it as a quantum process). From this procedure we can define a two-state Markov chain that, at each step, moves from the state where the answer is not known to the state where the answer is known with probability p, and then remains there. This walk has the transition matrix

$$P' = \begin{pmatrix} 1-p & 0 \\ p & 1 \end{pmatrix},$$

so $P_M = 1 - p$, giving a quantum hitting time of $O(1/\sqrt{1 - \|P_M\|}) = O(1/\sqrt{p})$.

For some applications, it may be desirable to estimate the value of p. Quantizing the above two-state Markov chain gives eigenvalues in the non-marked subspace of $e^{\pm i \arccos(1-p)} = e^{\pm i\sqrt{2p}+O(p^{3/2})}$. By applying phase estimation, we can determine $\sqrt{p}$ aproximately. Recall that phase estimation gives an estimate with precision μ using $O(1/\mu)$ applications of the given unitary (assuming we cannot apply high powers of the unitary any more efficiently than simply applying it repeatedly). An estimate of $\sqrt{p}$ with precision μ gives an estimate of p with precision $\mu\sqrt{p}$ (since $(\sqrt{p}+O(\mu))^2 = p + O(\mu\sqrt{p})$), so we can produce an estimate of p with precision ν in $O(\sqrt{p}/\nu)$ steps.

In particular, if the Markov chain is a search of the complete graph as described the previous section, with $|M|$ marked sites out of N, then $p = |M|/N$, and this allows us to count the number of marked items. We obtain an estimate of $|M|/N$ with precision ν in $O(\sqrt{|M|/N}/\nu)$ steps. If we want a multiplicative approximation of $|M|$ with precision ρ, this means we need $O(\sqrt{N/|M|}/\rho)$ steps.

Note that for exact counting, no speedup is possible in general. If $|M| = \Theta(N)$ then we need to estimate p within precision $O(1/N)$ to uniquely determine $|M|$, but then the running time of the above procedure is $O(N)$. In fact, it can be shown that exact counting requires $\Omega(N)$ queries.

Search on graphs

We can also consider a variant of unstructured search with additional locality constraints. Suppose we view the items in S as the vertices of a graph $G = (S, E)$, and we require the algorithm to be local with respect to the graph. More concretely, we require the algorithm to alternate between queries and unitary operations U constrained to satisfy $U|j, \psi\rangle = \sum_{k \in j \cup \partial(j)} \alpha_k |k, \phi_k\rangle$ for any $j \in S$ (where the second register represents possible ancillary space, and recall that $\partial(j)$ denotes the set of neighbors of j in G).

Since we have only added new restrictions that an algorithm must obey, the $\Omega(\sqrt{N})$ lower bound from the non-local version of the problem still applies. However, it is immediately clear that this bound cannot always be achieved. For example, if the graph is a cycle of N vertices, then simply propagating from one vertex of the cycle to an opposing vertex takes time $\Omega(N)$. So we would like to know, for example, how far from complete the graph can be such that we can still perform the search in $O(\sqrt{N})$ steps.

First, note that any expander graph (a graph with degree upper bounded by a constant and second largest eigenvalue bounded away from 1 by a constant) can be searched in time $O(\sqrt{N})$. Such graphs have $\delta = \Omega(1)$, and since $\epsilon \geq 1/N$ when there are marked items, the quantum hitting time is $O(1/\sqrt{\delta\epsilon}) = O(\sqrt{N})$ (whereas the classical hitting time is $O(1/\delta\epsilon) = O(N)$).

There are also many cases in which a quantum search can be performed in time $O(\sqrt{N})$ even though the eigenvalue gap of P is non-constant. For example, consider the n-dimensional hypercube (with $N = 2^n$ vertices). Recall that since the adjacency matrix acts independently as σ_x on each coordinate, the eigenvalues are equally spaced, and the gap of P is $2/n$. Thus the general bound in terms of the eigenvalues of P shows that the classical hitting time is $O(nN) = O(N \log N)$. In fact, this bound is loose; the hitting time is actually $O(N)$, which can be seen by directly computing $\|P_M\|$ with one marked vertex. So there is a local quantum algorithm that runs in the square root of this time, namely $O(\sqrt{N})$.

Perhaps the most interesting example is the d-dimensional square lattice with N sites (i.e., with linear size $N^{1/d}$). This case can be viewed as having N items distributed on a grid in d-dimensional space. For simplicity, suppose we have periodic boundary conditions; then the eigenstates of the adjacency matrix are given by

$$|\tilde{k}\rangle := \frac{1}{\sqrt{N}} \sum_x e^{2\pi i k \cdot x / N^{1/d}} |x\rangle \tag{7}$$

where k is a d-component vector of integers from 0 to $N^{1/d} - 1$. The corresponding eigenvalues are

$$2 \sum_{j=1}^{d} \cos \frac{2\pi k_j}{N^{1/d}}. \tag{8}$$

Normalizing to obtain a stochastic matrix, we simply divide these eigenvalues by $2d$. The 1 eigenvector has $k = (0, 0, \ldots, 0)$, and the second largest eigenvalue comes from (e.g.) $k = (1, 0, \ldots, 0)$, with an eigenvalue

$$\frac{1}{d} \left(d - 1 + \cos \frac{2\pi}{N^{1/d}} \right) \approx 1 - \frac{1}{2d} \left(\frac{2\pi}{N^{1/d}} \right)^2. \tag{9}$$

Thus the gap of the walk matrix P is about $\frac{2\pi^2}{2dN^{2/d}} = O(N^{-2/d})$. This is another case in which the bound on the classical hitting time in terms of eigenvalues of P is too loose (it gives only $O(N^{1+2/d})$), and instead we must directly estimate the gap of P_M. One can show that the classical hitting time is $O(N^2)$ in $d = 1$, $O(N \log N)$ in $d = 2$, and $O(N)$ for any $d \geq 3$. Thus there is a local quantum walk search algorithm that saturates the lower bound for any $d \geq 3$, and one that runs in time time $O(\sqrt{N \log N})$ for $d = 2$. We already argued that there could be no speedup for $d = 1$, and indeed we see that the quantum hitting time in this case is $O(N)$.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 13: Quantum walk search

In this lecture we will discuss the algorithm that cemented the importance of quantum walk as a tool for quantum query algorithms: Ambainis's algorithm for the element distinctness problem. The key new conceptual idea of this algorithm is to consider walks that store information obtained from many queries at each vertex, but that do not require many queries to update this information for an adjacent vertex. This idea leads to a general, powerful framework for quantum walk search.

Element distinctness

In the *element distinctness problem*, we are given a black-box function $f\colon \{1,\dots,n\} \to S$, where S is some finite set. The goal is to determine whether there are two distinct inputs $x, y \in \{1,\dots,n\}$ such that $f(x) = f(y)$.

It is clear that a classical algorithm must make $\Omega(n)$ queries to solve the problem, since deciding whether there is such a pair is at least as hard as unstructured search (suppose we give the additional promise that if there is a pair, it will be with $x = 1$ for which $f(1) = 1$; then we must search for a $y \in \{2,\dots,n\}$ for which $f(y) = 1$). By the same argument, there is a quantum lower bound for element distinctness of $\Omega(\sqrt{n})$.

There is a simple quantum algorithm that uses Grover's algorithm recursively to improve upon the trivial running time of $O(n)$. To see how this algorithm works, first consider the following subroutine. Query f in ℓ randomly chosen places, and check whether one of these ℓ places belongs to a pair of inputs that map to the same value by performing a Grover search on the remaining $n - \ell$ inputs. The initial setup takes ℓ queries, and the Grover search takes $O(\sqrt{n} - \ell) = O(\sqrt{n})$ queries, for a total of $\ell + O(\sqrt{n})$. This subroutine fails most of the time, since it is likely that the random choice of ℓ inputs will be unlucky, but it succeeds with probability at least ℓ/n. To boost the success probability, we can use amplitude amplification, which takes $O(\sqrt{n/\ell})$ steps to boost the success probability to a constant. Overall, we can obtain success probability $\Omega(1)$ using

$$(\ell + \sqrt{n})\sqrt{n/\ell} = \sqrt{n\ell} + n/\sqrt{\ell} \tag{1}$$

queries. To optimize the query complexity, we set the two terms to be equal, giving $\ell = \sqrt{n}$ and hence a query complexity of $O(n^{3/4})$. (Note that an analysis of the *running time* of this algorithm would include extra logarithmic factors, since the inner use of Grover's algorithm must check whether an element against ℓ queried function values, which can be done in time $O(\log \ell)$ provided S is ordered and we initially sort the queried values.)

So far, we have a quantum upper bound of $O(n^{3/4})$, and a quantum lower bound of $\Omega(n^{1/2})$. It turns out that both of these can be improved. On the lower bound side, Aaronson and Shi proved an $\Omega(n^{1/3})$ lower bound for the closely-related *collision problem*, in which the goal is to distinguish one-to-one from two-to-one functions. This implies an $\Omega(n^{2/3})$ lower bound for element distinctness by the following reduction. Suppose we randomly choose $\sqrt{n}$ inputs of the collision problem function and run the element distinctness algorithm on them. If the function is two-to-one, then there is some pair of elements in this set mapping to the same value with high probability (by the birthday problem), which the element distinctness algorithm will detect. Hence a k-query element distinctness algorithm implies an $O(\sqrt{k})$-query collision algorithm; or equivalently, a k-query collision lower bound implies an $\Omega(k^2)$ element distinctness lower bound.

Now the question remains, can we close the gap between the $O(n^{3/4})$ upper bound and this $\Omega(n^{2/3})$ lower bound? Ambainis's quantum walk algorithm does exactly this.

Quantum walk algorithm

The idea of Ambainis's algorithm is to quantize a walk on the Johnson graph $J(n, m)$, where m is chosen appropriately. This graph has $\binom{n}{m}$ vertices corresponding to subsets of $\{1, 2, \ldots, n\}$ of size m, and two vertices are connected by an edge if the subsets differ in exactly one element.

To simplify the analysis slightly, we will use a different graph, the Hamming graph $H(n, m)$. The vertices of this graph are the m-tuples of values from $\{1, 2, \ldots, n\}$ (so there are n^m vertices). Two vertices are connected by an edge if they differ in exactly one coordinate. There are two main differences between the Johnson and Hamming graphs: the Hamming graph allows for repeated elements, and the order of elements is significant. Neither of these differences significantly affects the performance of the algorithm.

At each vertex, we store the values of the function at the corresponding inputs. In other words, the vertex $(x_1, x_2, \ldots, x_m) \in \{1, 2, \ldots, n\}^m$ is represented by the state

$$|x_1, x_2, \ldots, x_m, f(x_1), f(x_2), \ldots, f(x_m)\rangle. \tag{2}$$

To prepare such states, we must query the black-box function. In particular, to prepare an initial superposition over vertices of this graph takes m queries. However, we can move from one vertex to an adjacent vertex using only two queries: to replace x by y in any particular coordinate, we use one query to erase $f(x)$ and another to compute $f(y)$.

In this search problem, the marked vertices are those containing some $x \neq y$ with $f(x) = f(y)$. Notice that, given the stored function values, we can check whether we are at a marked vertex with no additional queries. The total number of marked vertices (in the case where the elements are not all distinct) is at least $m(m-1)(n-2)^{m-2}$, so the fraction of marked vertices is

$$\epsilon \geq \frac{m(m-1)(n-2)^{m-2}}{n^m}. \tag{3}$$

To analyze the walk, we also need the eigenvalues of the relevant Markov chain. The adjacency matrix of the Hamming graph $H(n, m)$ is $A = \sum_{i=1}^{m} (J-I)^{(i)}$, where J denotes the $n \times n$ all 1s matrix, and the superscript indicates that this matrix acts on the ith coordinate. The eigenvalues of J are n and 0, so the eigenvalues of $J-I$ are $n-1$ and -1. Hence the largest eigenvalue of A is $m(n-1)$ (the degree of any vertex of $H(n, m)$) and the second largest eigenvalue is $(m-1)(n-1)-1 = m(n-1)-n$. Normalizing by the degree, we see that the second largest eigenvalue of the stochastic matrix $A/m(n-1)$ is $(m(n-1)-n)/m(n-1) = 1 - n/m(n-1)$. In other words, the spectral gap is

$$\delta = \frac{n}{m(n-1)}. \tag{4}$$

Finally, how many queries does this algorithm use? Taking into account the initial m queries used to prepare the starting state and the 2 queries per step of the walk, we have a total number of queries

$$m + 2 \cdot O\left(\frac{1}{\sqrt{\delta\epsilon}}\right) = m + O\left(\sqrt{\frac{m(n-1)}{n}} \sqrt{\frac{n^m}{m(m-1)(n-2)^{m-2}}}\right) \tag{5}$$

$$= m + O\left(\frac{n}{\sqrt{m}}\right). \tag{6}$$

Again we can set the two terms equal to optimize the performance. We have $m^{3/2} = O(n)$, so we should take $m = \Theta(n^{2/3})$. Then the total number of queries is $O(n^{2/3})$, which matches the lower bound, and hence is optimal.

Note that for the classical random walk search algorithm that we have quantized, the corresponding query complexity is $m + O(n^2/m)$, which is optimized by $m = n$. This gives no improvement over querying every input, as we knew must be the case.

Quantum walk search algorithms with auxiliary data

Algorithms based on similar ideas turn out to be useful for a wide variety of problems, including deciding whether a graph contains a triangle (or various other related graph properties), checking matrix multiplication, and testing whether a group is abelian. In general, as in the element distinctness case, we may need to store some data at each vertex, and we need to take into account the operations on this data when analyzing the walk.

Suppose we have a setup cost S, a cost U to update the state after one step of the walk, and a cost C to check whether a vertex is marked. For example, in Ambainis's algorithm for element distinctness, we had

$$S = m \qquad \text{to query } m \text{ positions} \tag{7}$$

$$U = 2 \qquad \text{to remove one of the items and add another} \tag{8}$$

$$C = 0 \qquad \text{since the function values for the subset are stored.} \tag{9}$$

In general, there is an algorithm to solve such a problem with total cost

$$S + \frac{1}{\sqrt{\delta\epsilon}}(U + C). \tag{10}$$

It turns out that for some problems, when the checking cost C is much larger than the update cost U, it is advantageous to take many steps of the walk on the unmarked graph before performing a phase flip on the marked sites. This is how Ambainis's algorithm originally worked, though for element distinctness it is not actually necessary. Using this idea, one can give a general quantum walk search algorithm with total cost

$$S + \frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}U + C\right). \tag{11}$$

In fact, it is also possible to modify the general algorithm so that it finds a marked item when one exists.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 13: Query complexity and the polynomial method

So far, we have discussed several different kinds of quantum algorithms. In the next few lectures, we will discuss ways of establishing limitations on the power of quantum algorithms. After reviewing the model of quantum query complexity, this lecture presents the *polynomial method*, an approach that relates quantum query algorithms to properties of polynomials.

Quantum query complexity

Many of the algorithms we have covered work in the setting of query complexity, where the input for a problem is provided by a black box. This setting is convenient since the black box provides a handle for proving lower bounds: we can often show that many queries are required to compute some given function of the black-box input. In contrast, it is notoriously difficult to prove lower bounds on the complexity of computing some function of explicit input data.

We briefly formalize the model of query complexity. Consider the computational task of computing a function $f\colon S \to T$, where $S \subset \Sigma^n$ is a set of strings over some input alphabet Σ. If $S = \Sigma^n$ then we say f is *total*; otherwise we say it is *partial*. The input string $x \in S$ is provided to us by a black box that computes x_i for any desired $i \in \{1, \ldots, n\}$. A query algorithm begins from a state that does not depend on the oracle string x. It then alternate between queries to the black box and other, non-query operations. Our goal is to compute $f(x)$ using as few queries to the black box as possible.

Of course, the minimum number of queries (which we call the *query complexity* of f) depends on the kind of computation we allow. There are at least three natural models:

- $D(f)$ denotes the deterministic query complexity, where the algorithm is classical and must always work correctly.
- R_ϵ denotes the randomized query complexity with error probability at most ϵ. Note that this it does not depend strongly on ϵ since we can boost the success probability by repeating the computation several times and take a majority vote. Therefore $R_\epsilon(f) = \Theta(R_{1/3}(f))$ for any constant ϵ, so sometimes we simply write $R(f)$.
- Q_ϵ denotes the quantum query complexity, again with error probability at most ϵ. Similarly to the randomized case, $Q_\epsilon(f) = \Theta(Q_{1/3}(f))$ for any constant ϵ, so sometimes we simply write $Q(f)$.

We know that $D(\textsc{or}) = n$ and $R(\textsc{or}) = \Theta(n)$. Grover's algorithm shows that $Q(\textsc{or}) = O(\sqrt{n})$. In this lecture we will use the polynomial method to show (among other things) that $Q(\textsc{or}) = \Omega(\sqrt{n})$, a tight lower bound.

Quantum queries

A quantum query algorithm begins from x-independent state $|\psi\rangle$ and applies a sequence of unitary operations $U_1, \ldots, U_t$ interspersed with queries O_x, resulting in the state

$$|\psi_x^t\rangle := U_t O_x \ldots U_2 O_x U_1 O_x |\psi\rangle. \tag{1}$$

To make this precise, we need to specify the action of the oracle O_x.

For simplicity, we will mostly consider the case where the input is a bit string, i.e., $\Sigma = \{0,1\}$. Perhaps the most natural oracle model is the bit flip oracle $\hat{O}_x$, which acts as

$$\hat{O}_x|i,b\rangle = |i, b \oplus x_i\rangle \qquad \text{for } i \in \{1,\ldots,n\},\ b \in \{0,1\}. \tag{2}$$

This is simply the linear extension of the natural reversible oracle mapping $(i,b) \mapsto (i, b \oplus x_i)$, which can be performed efficiently given the ability to efficiently compute $i \mapsto x_i$. Note that the algorithm may involve states in a larger Hilbert space; implicitly, the oracle acts as the identity on any ancillary registers.

It is often convenient to instead consider the phase oracle, which is obtained by conjugating the bit-flip oracle by Hadamard gates: by the well-known phase kickback trick, $O_x = (I \otimes H)\hat{O}_x(I \otimes H)$ satisfies

$$O_x|i,b\rangle = (-1)^{bx_i}|i,b\rangle \qquad \text{for } i \in \{1,\ldots,n\},\ b \in \{0,1\}. \tag{3}$$

Note that this is slightly wasteful since $O_x|i,0\rangle = |i,0\rangle$ for all i; we could equivalently consider a phase oracle O'_x defined by $O'_x|0\rangle = |0\rangle$ and $O'_x|i\rangle = (-1)^{x_i}|i\rangle$ for all $i \in \{1,\ldots,n\}$. However, it is essential to include the ability to not query the oracle by giving the oracle some eigenstate of known eigenvalue, independent of x. If we could only perform the phase flip $|i\rangle \mapsto (-1)^{x_i}|i\rangle$ for $i \in \{1,\ldots,n\}$, then we could not tell a string x from its bitwise complement $\bar{x}$.

These constructions can easily be generalized to the case of a d-ary input alphabet, say $\Sigma = \mathbb{Z}_d$ (identifying input symbols with integers modulo d). Then for $b \in \Sigma$, we can define an oracle $\hat{O}_x$ by

$$\hat{O}_x|i,b\rangle = |i, b + x_i\rangle \qquad \text{for } i \in \{1,\ldots,n\},\ b \in \mathbb{Z}_d. \tag{4}$$

Taking the Fourier transform of the second register gives a phase oracle $O_x = (I \otimes F^\dagger_{\mathbb{Z}_d})\hat{O}_x(I \otimes F_{\mathbb{Z}_d})$ satisfying

$$O_x|i,b\rangle = \omega_d^{bx_i}|i,b\rangle \qquad \text{for } i \in \{1,\ldots,n\},\ b \in \mathbb{Z}_d \tag{5}$$

where $\omega_d := e^{2\pi i/d}$.

Quantum algorithms and polynomials

The following shows a basic connection between quantum algorithms and polynomials.

Lemma. *The acceptance probability of a t-query quantum algorithm for a problem with black-box input $x \in \{0,1\}^n$ is a polynomial in $x_1,\ldots,x_n$ of degree at most $2t$.*

Proof. We claim that the amplitude of any basis state is a polynomial of degree at most t, so that the probability of any basis state (and hence the probability of success) is a polynomial of degree at most $2t$.

The proof is by induction on t. If an algorithm makes no queries to the input, then its success probability is independent of the input, so it is a constant, a polynomial of degree 0.

For the induction step, a query maps

$$|i,b\rangle \overset{O_x}{\mapsto} (-1)^{bx_i}|i,b\rangle \tag{6}$$

$$= (1 - 2bx_i)|i,b\rangle, \tag{7}$$

so it increases the degree of each amplitude by at most 1. □

Consider a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$. We say a polynomial $p \in \mathbb{R}[x_1, \ldots, x_n]$ represents f if $p(x) = f(x)$ for all $x \in \{0,1\}^n$. Letting $\deg(f)$ denote the smallest degree of any polynomial representing f, we have $Q_0(f) \geq \deg(f)/2$.

To handle bounded-error algorithms, we introduce the concept of *approximate degree*. We say a polynomial p ϵ-represents f if $|p(x) - f(x)| \leq \epsilon$ for all $x \in \{0,1\}^n$. Then the ϵ-approximate degree of f, denoted $\widetilde{\deg}_\epsilon(f)$, is the smallest degree of any polynomial that ϵ-represents f. Clearly, $Q_\epsilon(f) \geq \widetilde{\deg}_\epsilon(f)/2$. Since bounded-error query complexity does not depend strongly on the particular error probability ϵ, we can define, say, $\widetilde{\deg}(f) := \widetilde{\deg}_{1/3}(f)$.

Now to lower bound the quantum query complexity of a Boolean function, it suffices to lower bound its approximate degree.

Symmetrization

While polynomials are well-understood objects, the acceptance probability is a multivariate polynomial, so it can be rather complicated. Since $x^2 = x$ for $x \in \{0,1\}$, we can restrict our attention to multilinear polynomials, but it is still somewhat difficult to deal with such polynomials directly. Fortunately, for many functions it suffices to consider a related univariate polynomial obtained by symmetrization.

For a string $x \in \{0,1\}^n$, let $|x|$ denote the Hamming weight of x, the number of 1s in x.

Lemma. *Given any n-variate multilinear polynomial p, let $P(k) := \mathbb{E}_{|x|=k}[p(x)]$. Then P is a polynomial with $\deg(P) \leq \deg(p)$.*

Proof. Since p is multilinear, it can be written as a sum of monomials, i.e., as

$$p(x) = \sum_{S \subseteq \{1,\ldots,n\}} c_S \prod_{i \in S} x_i \tag{8}$$

for some coefficients c_S. Then we have

$$P(k) = \sum_{S \subseteq \{1,\ldots,n\}} c_S \mathop{\mathbb{E}}_{|x|=k} \left[\prod_{i \in S} x_i \right] \tag{9}$$

and it suffices to compute the expectation of each monomial. We find

$$\mathop{\mathbb{E}}_{|x|=k} \left[\prod_{i \in S} x_i \right] = \Pr_{|x|=k} [\forall\, i \in S,\ x_i = 1] \tag{10}$$

$$= \frac{\binom{n-|S|}{k-|S|}}{\binom{n}{k}} \tag{11}$$

$$= \frac{(n-|S|)!\, k!\, (n-k)!}{(k-|S|)!\, (n-k)!\, n!} \tag{12}$$

$$= \frac{(n-|S|)!}{n!} k(k-1)\cdots(k-|S|+1) \tag{13}$$

which is a polynomial in k of degree $|S|$. Since $c_S = 0$ whenever $|S| > \deg(p)$, we see that $\deg(P) \leq \deg(p)$. □

Thus the polynomial method is a particularly natural approach for symmetric functions, those that only depend on the Hamming weight of the input.

Parity

Let PARITY: $\{0,1\}^n \to \{0,1\}$ denote the symmetric function $\text{PARITY}(x) = x_1 \oplus \cdots \oplus x_n$. Recall that Deutsch's problem, which is the problem of computing the parity of 2 bits, can be solved exactly with only one quantum query. Applying this algorithm to a pair of bits at a time and then taking the parity of the results, we see that $Q_0(\text{PARITY}) \leq n/2$.

What can we say about lower bounds for computing parity? Symmetrizing PARITY gives the function $P\colon \{0,1,\ldots,n\} \to \mathbb{R}$ defined by

$$P(k) = \begin{cases} 0 & \text{if } k \text{ is even} \\ 1 & \text{if } k \text{ is odd.} \end{cases} \tag{14}$$

Since P changes direction n times, $\deg(P) \geq n$, so we see that $Q_0(\text{PARITY}) \geq n/2$. Thus Deutsch's algorithm is tight among zero-error algorithms.

What about bounded-error algorithms? To understand this, we would like to lower bound the approximate degree of PARITY. If $|p(x) - f(x)| \leq \epsilon$ for all $x \in \{0,1\}^n$, then

$$|P(k) - F(k)| = \left| \mathop{\mathbb{E}}_{|x|=k} (p(x) - f(x)) \right| \leq \epsilon \tag{15}$$

for all $k \in \{0,1,\ldots,n\}$, where P is the symmetrization of p and F is the symmetrization of f. Thus, a multilinear polynomial p that ϵ-approximates PARITY implies a univariate polynomial P satisfying $P(k) \leq \epsilon$ for k even and $P(k) \geq 1 - \epsilon$ for k odd. For any $\epsilon < 1/2$, this function still changes direction n times, so in fact we have $\widetilde{\deg}_\epsilon(f) \geq n$, and hence $Q_\epsilon(\text{PARITY}) \geq n/2$.

This shows that the strategy for computing parity using Deutsch's algorithm is optimal, even among bounded-error algorithms. This is an example of a problem for which a quantum computer cannot get a significant speedup—here the speedup is only by factor of 2. In fact, we need at least $n/2$ queries to succeed with any bounded error, even with very small advantage (e.g., even if we only want to be correct with probability $\frac{1}{2} + 10^{-100}$). In contrast, while the adversary method can prove an $\Omega(n)$ lower bound for parity, the constant factor that it establishes is error-dependent.

Note that this also shows we need $\Omega(n)$ queries to exactly count the number of marked items in an unstructured search problem, since exactly determining the number of 1s would in particular determine whether the number of 1s is odd or even.

Unstructured search

Next we will see how the polynomial method can be used to prove the $\Omega(\sqrt{n})$ lower bound for computing the logical OR of n bits. Symmetrizing OR gives a function $F(k)$ with $F(0) = 0$ and $F(1) = 1$. We also have $F(k) = 1$ for all $k > 1$, but we will not actually need to use this. This function is monotonic, so we cannot use the same simple argument we applied to parity. Nevertheless, we can prove that $\widetilde{\deg}(\text{OR}) = \Omega(\sqrt{n})$ using the following basic fact about polynomials, due to Markov.

Lemma. *Let $P\colon \mathbb{R} \to \mathbb{R}$ be a polynomial. Then*

$$\max_{x \in [0,n]} \frac{\mathrm{d}P(x)}{\mathrm{d}x} \leq \frac{\deg(P)^2}{n} \left(\max_{x \in [0,n]} P(x) - \min_{x \in [0,n]} P(x) \right). \tag{16}$$

In other words, if we let

$$h := \max_{x \in [0,n]} P(x) - \min_{x \in [0,n]} P(x) \tag{17}$$

denote the "height" of P in the range $[0, n]$, and

$$d := \max_{x \in [0,n]} \frac{\mathrm{d}P(x)}{\mathrm{d}x} \tag{18}$$

denote the largest derivative of P in that range, then we have $\deg(P) \geq \sqrt{nd/h}$.

Now let P be a polynomial that ϵ-approximates OR. Since $P(0) \leq \epsilon$ and $P(1) \geq 1 - \epsilon$, P must increase by at least $1 - 2\epsilon$ in going from $k = 0$ to $k = 1$, so $d \geq 1 - 2\epsilon$.

We have no particular bound on h, since we have no control over the value of P at non-integer points; the function could become arbitrarily large or small. However, since $P(k) \in [0, 1]$ for $k \in \{0, 1, \ldots, n\}$, a large value of h implies a large value of d, since P must change fast enough to start from and return to values in the range $[0, 1]$. In particular, P must change by at least $(h - 1)/2$ over a range of k of width at most $1/2$, so we have $d \geq h - 1$. Therefore,

$$\deg(P) \geq \sqrt{\frac{n \max\{1 - 2\epsilon, h - 1\}}{h}} \tag{19}$$

$$= \Omega(\sqrt{n}). \tag{20}$$

It follows that $Q(\text{OR}) = \Omega(\sqrt{n})$.

Note that the same argument applies for a function that takes the value 0 whenever $|x| = w$ and the value 1 whenever $|x| = w + 1$, for any w; in particular, it applies to any non-constant symmetric function. (Of course, we can do better for some symmetric functions, such as PARITY and also MAJORITY, among others.)

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 15: The adversary method

We now discuss a second approach to proving quantum query lower bounds, the *quantum adversary method*. In fact, we'll see later that the generalized version of the adversary method we consider here (allowing negative weights) turns out to be an *upper bound* on quantum query complexity, up to constant factors.

Quantum adversaries

Motivation for the quantum adversary method comes from the following construction. Suppose the oracle is operated by an adversarial party who holds a quantum state determining the oracle string, which is in some superposition $\sum_{x\in S} a_x|x\rangle$ over the possible oracles. To implement each query, the adversary performs the "super-oracle"

$$O := \sum_{x\in S} |x\rangle\langle x| \otimes O_x. \tag{1}$$

An algorithm does not have direct access to the oracle string, and hence can only perform unitary operations that act as the identity on the adversary's superposition. After t steps, an algorithm maps the overall state to

$$|\psi^t\rangle := (I\otimes U_t)O\ldots(I\otimes U_2)O(I\otimes U_1)O\left(\sum_{x\in S} a_x|x\rangle\otimes|\psi\rangle\right) \tag{2}$$

$$= \sum_{x\in S} a_x|x\rangle\otimes|\psi_x^t\rangle. \tag{3}$$

The main idea of the approach is that for the algorithm to learn x, this state must become very entangled. To measure the entanglement of the pure state $|\psi^t\rangle$, we can consider the reduced density matrix of the oracle,

$$\rho^t := \sum_{x,y\in S} a_x^* a_y \langle\psi_x^t|\psi_y^t\rangle\,|x\rangle\langle y|. \tag{4}$$

Initially, the state ρ^0 is pure. Our goal is to quantify how mixed it must become (i.e., how entangled the overall state must be) before we can compute f with error at most ϵ. To do this we could consider, for example, the entropy of ρ^t. However, it turns out that other measures are easier to deal with.

In particular, we have the following basic fact about the distinguishability of quantum states (for a proof, see for example section A.9 of KLM):

Fact. *Given one of two pure states $|\psi\rangle, |\phi\rangle$, we can make a measurement that determines which state we have with error probability at most $\epsilon \in [0, 1/2]$ if and only if $|\langle\psi|\phi\rangle| \le 2\sqrt{\epsilon(1-\epsilon)}$.*

Thus it is convenient to consider measures that are linear in the inner products $\langle\psi_x^t|\psi_y^t\rangle$.

The adversary method

To obtain an adversary lower bound, we choose a matrix $\Gamma \in \mathbb{R}^{|S|\times|S|}$ with rows and columns indexed by the possible black-box inputs. The entry $\Gamma_{x,y}$ is meant to characterize how hard it is to distinguish between x and y. We say Γ is an *adversary matrix* if

1. $\Gamma_{xy} = \Gamma_{yx}$ and
2. if $f(x) = f(y)$ then $\Gamma_{xy} = 0$.

The second condition reflects that we do not need to distinguish between x and y if $f(x) = f(y)$.

The original adversary method made the additional assumption that $\Gamma_{xy} \geq 0$, but it turns out that this condition is not actually necessary. Sometimes we refer to the *negative* or *generalized* adversary method to distinguish it from the original, positive-weighted method. While it may not be intuitively obvious what it would mean to give a negative weight to the entry characterizing distinguishability of two inputs, it turns out that this flexibility can lead to significantly improved lower bounds for some functions.

Given an adversary matrix Γ, we can define a weight function

$$W^j := \sum_{x,y\in S} \Gamma_{xy} a_x^* a_y \langle \psi_x^j | \psi_y^j \rangle. \tag{5}$$

Note that this is a simple function of the entries of ρ^j. The idea of the lower bound is to show that W^j starts out large, must become small in order to compute f, and cannot change by much if we make a query.

The initial value of the weight function is

$$W^0 = \sum_{x,y\in S} \Gamma_{xy} a_x^* a_y \langle \psi_x^0 | \psi_y^0 \rangle \tag{6}$$

$$= \sum_{x,y\in S} a_x^* \Gamma_{xy} a_y \tag{7}$$

since $|\psi_x^0\rangle$ cannot depend on x. To make this as large as possible, we take a to be a principal eigenvector of Γ, an eigenvector with eigenvalue $\pm\|\Gamma\|$. Then $|W^0| = \|\Gamma\|$.

The final value of the weight function is easier to bound if we assume a nonnegative adversary matrix. The final value is constrained by the fact that we must distinguish x from y with error probability at most ϵ whenever $f(x) \neq f(y)$. For this to hold after t queries, we need $|\langle \psi_x^t | \psi_y^t \rangle| \leq 2\sqrt{\epsilon(1-\epsilon)}$ for all pairs $x, y \in S$ with $f(x) \neq f(y)$ (by the above Fact). Thus we have

$$|W^t| \leq \sum_{x,y\in S} \Gamma_{xy} a_x^* a_y 2\sqrt{\epsilon(1-\epsilon)} \tag{8}$$

$$= 2\sqrt{\epsilon(1-\epsilon)}\|\Gamma\|. \tag{9}$$

Here we can include the terms where $f(x) = f(y)$ in the sum since $\Gamma_{xy} = 0$ for such pairs. We also used the fact that the principal eigenevector of a nonnegative matrix can be taken to have nonnegative entries (by the Perron-Frobenius theorem).

A similar bound holds if Γ has negative entries, but we need a different argument. In general, one can only show that $|W^t| \leq (2\sqrt{\epsilon(1-\epsilon)} + 2\epsilon)\|\Gamma\|$. But if we assume that $f\colon S \to \{0,1\}$ has Boolean output, then we can prove the same bound as in the non-negative case, and the proof is simpler than for a general output space. We use the following simple result, stated in terms of the Frobenius norm $\|X\|_F := \sum_{a,b} |X_{ab}|^2$:

Proposition. *For any $X \in \mathbb{C}^{m\times n}, Y \in \mathbb{C}^{n\times n}, Z \in \mathbb{C}^{n\times m}$, we have $|\mathrm{tr}(XYZ)| \leq \|X\|_F \|Y\| \|Z\|_F$.*

Proof. We have

$$\mathrm{tr}(XYZ) = \sum_{a,b,c} X_{ab} Y_{bc} Z_{ca} \tag{10}$$

$$= \sum_a (x^a)^\dagger Y z^a \tag{11}$$

where $(x^a)_b = X^*_{ab}$ and $(z^a)_c = Z_{ca}$. Thus

$$|\mathrm{tr}(XYZ)| \leq \sum_a \|x^a\| \|Y z^a\| \tag{12}$$

$$\leq \|Y\| \sum_a \|x^a\| \|z^a\| \tag{13}$$

$$\leq \|Y\| \sqrt{\sum_a \|x^a\|^2 \sum_{a'} \|z^{a'}\|^2} \tag{14}$$

$$= \|Y\| \|X\|_F \|Z\|_F \tag{15}$$

as claimed, where we used the Cauchy-Schwarz inequality in the second and third steps. □

To upper bound $|W^t|$ for the negative adversary with Boolean output, write $W^t = \mathrm{tr}(\Gamma V)$ where $V_{xy} := a^*_x a_y \langle \psi^t_x | \psi^t_y \rangle \delta[f(x) \neq f(y)]$. Define

$$C := \sum_{x\in S} a_x \Pi_{f(x)} |\psi^t_x\rangle\langle x| \tag{16}$$

$$\bar{C} := \sum_{x\in S} a_x \Pi_{1-f(x)} |\psi^t_x\rangle\langle x| \tag{17}$$

with Π_0, Π_1 denoting the projectors onto the subspaces indicating $f(x) = 0, 1$, respectively. Then

$$(C^\dagger \bar{C})_{xy} = a^*_x a_y \langle \psi^t_x | \Pi_{f(x)} \Pi_{1-f(y)} | \psi^t_y \rangle, \tag{18}$$

so

$$(C^\dagger \bar{C} + \bar{C}^\dagger C)_{xy} = a^*_x a_y \langle \psi^t_x | (\Pi_{f(x)} \Pi_{1-f(y)} + \Pi_{1-f(x)} \Pi_{f(y)}) | \psi^t_y \rangle \tag{19}$$

$$= a^*_x a_y \langle \psi^t_x | \psi^t_y \rangle \delta[f(x) \neq f(y)], \tag{20}$$

i.e., $V = C^\dagger \bar{C} + \bar{C}^\dagger C$. Thus we have

$$W^t = \mathrm{tr}(\Gamma (C^\dagger \bar{C} + \bar{C}^\dagger C)) \tag{21}$$

$$= \mathrm{tr}(\bar{C} \Gamma C^\dagger) + \mathrm{tr}(C \Gamma \bar{C}^\dagger). \tag{22}$$

By the Proposition, $|W^t| \leq 2\|\Gamma\| \|C\|_F \|\bar{C}\|_F$. Finally, we upper bound $\|C\|_F$ and $\|\bar{C}\|_F$. We have

$$\|C\|^2_F + \|\bar{C}\|^2_F = \sum_{x,y\in S} |a_x|^2 (|\langle y | \Pi_{f(x)} | \psi^t_x \rangle|^2 + |\langle y | \Pi_{1-f(x)} | \psi^t_x \rangle|^2) = 1 \tag{23}$$

$$\|\bar{C}\|^2_F = \sum_{x\in S} |a_x|^2 \| \Pi_{1-f(x)} |\psi^t_x\rangle \|^2 \leq \epsilon. \tag{24}$$

Therefore $\|C\|_F\|\bar{C}\|_F \leq \max_{x\in[0,\epsilon]} \sqrt{x(1-x)} = \sqrt{\epsilon(1-\epsilon)}$ (assuming $\epsilon \in [0, 1/2]$), and we find that $|W^t| \leq 2\sqrt{\epsilon(1-\epsilon)}\|\Gamma\|$, as claimed.

It remains to understand how much the weight function can decrease at each step of the algorithm. We have

$$W^{j+1} - W^j = \sum_{x,y\in S} \Gamma_{xy} a_x^* a_y (\langle \psi_x^{j+1} | \psi_y^{j+1} \rangle - \langle \psi_x^j | \psi_y^j \rangle). \tag{25}$$

Consider how the state changes when we make a query. We have $|\psi_x^{j+1}\rangle = U^{j+1} O_x |\psi_x^j\rangle$. Thus the elements of the Gram matrix of the states $\{|\psi_x^{j+1}\rangle \colon x \in S\}$ are

$$\begin{aligned} \langle \psi_x^{j+1} | \psi_y^{j+1} \rangle &= \langle \psi_x^j | O_x^\dagger (U^{j+1})^\dagger U^{j+1} O_y | \psi_y^j \rangle & (26) \\ &= \langle \psi_x^j | O_x O_y | \psi_y^j \rangle & (27) \end{aligned}$$

since U^{j+1} is unitary and $O_x^\dagger = O_x$. Therefore

$$W^{j+1} - W^j = \sum_{x,y\in S} \Gamma_{xy} a_x^* a_y \langle \psi_x^j | (O_x O_y - I) | \psi_y^j \rangle. \tag{28}$$

Observe that $O_x O_y |i, b\rangle = (-1)^{b(x_i \oplus y_i)} |i, b\rangle$. Let $P_0 = I \otimes |0\rangle\langle 0|$ denote the projection onto the $b = 0$ states, and let P_i denote the projection $|i, 1\rangle\langle i, 1|$. (As with O_x, the projections P_i implicitly act as the identity on any ancilla registers, so $\sum_{i=0}^n P_i = I$.) Then $O_x O_y = P_0 + \sum_{i=1}^n (-1)^{x_i \oplus y_i} P_i$, so $O_x O_y - I = -2 \sum_{i\colon x_i \neq y_i} P_i$. Thus we have

$$W^{j+1} - W^j = 2 \sum_{x,y\in S} \sum_{i\colon x_i \neq y_i} \Gamma_{xy} a_x^* a_y \langle \psi_x^j | P_i | \psi_y^j \rangle. \tag{29}$$

Now for each $i \in \{1, \ldots, n\}$, let Γ_i be a matrix with

$$(\Gamma_i)_{xy} := \begin{cases} \Gamma_{xy} & \text{if } x_i \neq y_i \\ 0 & \text{if } x_i = y_i \end{cases} \tag{30}$$

Then we have

$$\begin{aligned} W^{j+1} - W^j &= 2 \sum_{x,y\in S} \sum_{i=1}^n (\Gamma_i)_{xy} a_x^* a_y \langle \psi_x^j | P_i | \psi_y^j \rangle & (31) \\ &= 2 \sum_{i=1}^n \mathrm{tr}(Q_i \Gamma_i Q_i^\dagger) & (32) \end{aligned}$$

where $Q_i := \sum_x a_x P_i |\psi_x^j\rangle\langle x|$.

Using the triangle inequality and the above Proposition, we have

$$\begin{aligned} |W^{j+1} - W^j| &\leq 2 \sum_{i=1}^n |\mathrm{tr}(Q_i \Gamma_i Q_i^\dagger)| & (33) \\ &\leq 2 \sum_{i=1}^n \|\Gamma_i\| \|Q_i\|_F^2. & (34) \end{aligned}$$

Since

$$\sum_{i=1}^{n} \|Q_i\|_F^2 = \sum_{i=1}^{n} \sum_{x \in S} |a_x|^2 \|P_i |\psi_x^j\rangle\|^2 \tag{35}$$

$$\leq \sum_{x \in S} |a_x|^2 \tag{36}$$

$$= 1, \tag{37}$$

we have

$$|W^{j+1} - W^j| \leq 2 \max_{i \in \{1,\ldots,n\}} \|\Gamma_i\|. \tag{38}$$

Combining these three facts gives the adversary lower bound. Since $|W^0| = \|\Gamma\|$, we have

$$|W^t| \geq \|\Gamma\| - 2t \max_{i \in \{1,\ldots,n\}} \|\Gamma_i\|. \tag{39}$$

Thus, to have $|W^t| \leq 2\sqrt{\epsilon(1-\epsilon)}\|\Gamma\|$, we require

$$t \geq \frac{1 - 2\sqrt{\epsilon(1-\epsilon)}}{2} \operatorname{Adv}(f). \tag{40}$$

where

$$\operatorname{Adv}(f) := \max_{\Gamma} \frac{\|\Gamma\|}{\max_{i \in \{1,\ldots,n\}} \|\Gamma_i\|} \tag{41}$$

with the maximum taken over all adversary matrices Γ for the function f. (Often the notation $\operatorname{Adv}(f)$ is reserved for the maximization over nonnegative adversary matrices, with the notation $\operatorname{Adv}^{\pm}(f)$ for the generalized adversary method allowing negative weights.)

Example: Unstructured search

As a simple application of this method, we prove the optimality of Grover's algorithm. It suffices to consider the problem of distinguishing between the case of no marked items and the case of a unique marked item (in an unknown location). Thus, consider the partial function where S consists of the strings of Hamming weight 0 or 1, and f is the logical OR of the input bits. (Equivalently, we consider the total function OR but only consider adversary matrices with zero weight on strings of Hamming weight more than 1.)

For this problem, adversary matrices have the form

$$\Gamma = \begin{pmatrix} 0 & \gamma_1 & \cdots & \gamma_n \\ \gamma_1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_n & 0 & \cdots & 0 \end{pmatrix} \tag{42}$$

for some nonnegative coefficients $\gamma_1, \ldots, \gamma_n$. Symmetry suggests that we should take $\gamma_1 = \cdots = \gamma_n$. This can be formalized, but for the present purposes we can take this as an ansatz.

Setting $\gamma_1 = \cdots = \gamma_n = 1$ (since an overall scale factor does not affect the bound), we have

$$\Gamma^2 = \begin{pmatrix} n & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \cdots & 1 \end{pmatrix} \tag{43}$$

which has norm $\|\Gamma^2\| = n$, and hence $\|\Gamma\| = \sqrt{n}$. We also have

$$\Gamma_1 = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \tag{44}$$

and similarly for the other Γ_i, so $\|\Gamma_i\| = 1$. Thus we find $\mathrm{Adv}(\textsc{or}) \geq \sqrt{n}$, and it follows that $Q_\epsilon(\textsc{or}) \geq \frac{1-2\sqrt{\epsilon(1-\epsilon)}}{2}\sqrt{n}$. This shows that Grover's algorithm is optimal up to a constant factor (recall that Grover's algorithm finds a unique marked item with probability $1 - o(1)$ in $\frac{\pi}{4}\sqrt{n} + o(1)$ queries).

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 16: Span programs

Having discussed lower bounds on quantum query complexity, we now turn our attention back to upper bounds. The framework of *span programs* is a powerful tool for understanding quantum query complexity. Span programs are closely related to the quantum adversary method, and can be used to show that the (generalized) adversary method actually characterizes quantum query complexity up to constant factors.

For simplicity, we restrict our attention to the case of a (possibly partial) Boolean function $f\colon S \to \{0,1\}$ where $S \subseteq \{0,1\}^n$. Many (but not all) of the considerations for this case generalize to other kinds of functions.

The dual of the adversary method

Recall that the adversary method defines a quantity

$$\mathrm{Adv}^{\pm}(f) := \max_{\Gamma} \frac{\|\Gamma\|}{\max_{i\in\{1,\ldots,n\}} \|\Gamma_i\|} \tag{1}$$

such that $Q(f) = O(\mathrm{Adv}^{\pm}(f))$. Although not immediately obvious from the above expression, it can be shown that $\mathrm{Adv}^{\pm}(f)$ is the value of a *semidefinite program* (SDP), a kind of optimization problem in which a linear objective function is optimized subjected to linear and positive semidefiniteness constraints.

Unfortunately, the details of semidefinite programming are beyond the scope of this course. For a good introduction in the context of quantum information, see Lecture 7 of Watrous's lecture notes on Theory of Quantum Information.

A useful feature of SDPs is that they can be solved efficiently. Thus, we can use a computer program to find the optimal adversary lower bound for a fixed (finite-size) function. However, while this may be useful for getting intuition about a problem, in general this does not give a strategy for determining asymptotic quantum query complexity.

Another key feature of SDPs is the concept of semidefinite programming duality. To every primal SDP, phrased as a maximization problem, there is a dual SDP, which is a minimization problem. Whereas feasible solutions of the primal SDP give lower bounds, feasible solutions of the dual SDP give upper bounds. The dual problem can be constructed from the primal problem by a straightforward (but sometimes tedious) process. Semidefinite programs satisfy *weak duality*, which says that the value of the primal problem is at most the value of the dual problem. Furthermore, almost all SDPs actually satisfy *strong duality*, which says that the primal and dual values are equal. (In particular, this holds under the *Slater conditions*, which essentially say that the primal or dual constraints are strictly feasible.)

To understand any SDP, one should always construct its dual. Carrying this out for the adversary method would require some experience with semidefinite programs, so we simply state the result here. The variables of the dual problem can be viewed as a set of vectors $|v_{x,i}\rangle \in \mathbb{C}^d$ for all inputs $x \in S$ and all indices $i \in [n] := \{1,\ldots,n\}$, for some dimension d. For $b \in \{0,1\}$, we define the b-complexity $C_b := \max_{x\in f^{-1}(b)} \sum_{i\in[n]} \||v_{x,i}\rangle\|^2$. Since strong duality holds, we have the following.

Theorem. *For any function $f\colon S \to \{0,1\}$ with $S \subseteq \{0,1\}^n$, we have*

$$\mathrm{Adv}^{\pm}(f) = \min_{\{|v_{x,i}\rangle\}} \max\{C_0, C_1\} \tag{2}$$

where the minimization is over all positive integers d and all sets of vectors $\{|v_{x,i}\rangle \in \mathbb{C}^d \colon x \in S, i \in [n]\}$ satisfying the constraint

$$\sum_{i\colon x_i \neq y_i} \langle v_{x,i}|v_{y,i}\rangle = 1 - \delta_{f(x),f(y)} \quad \forall\, x \neq y. \tag{3}$$

By constructing solutions of the adversary dual, we place upper bounds on the best possible adversary lower bound. But more surprisingly, one can construct an algorithm from a solution of the adversary dual, giving an upper bound on the quantum query complexity itself.

Observe that if we replace $|v_{x,i}\rangle \to \alpha|v_{x,i}\rangle$ for all $x \in f^{-1}(0)$ and $|v_{y,i}\rangle \to |v_{y,i}\rangle/\alpha$ for all $y \in f^{-1}(1)$, we don't affect the constraints (3), but we map $C_0 \to \alpha^2 C_0$ and $C_1 \to C_1/\alpha^2$. Taking $\alpha = (C_1/C_0)^{1/4}$, we make the two complexities equal. Thus we have

$$\mathrm{Adv}^{\pm}(f) = \min_{\{|v_{x,i}\rangle\}} \sqrt{C_0 C_1}. \tag{4}$$

Note that the constraint (3) for $f(x) = f(y)$, where the right-hand side is zero, can be removed without changing the value of the optimization problem. (For functions with non-Boolean output, one loses a factor strictly between 1 and 2 in the analogous relaxation.) To see this, suppose we have a set of vectors $\{|v_{x,i}\rangle\}$ satisfying the constraint (3) for $f(x) \neq f(y)$ but not for $f(x) = f(y)$. Simply let $|v_{x,i}\rangle = |v'_{x,i}\rangle|x_i \oplus f(x)\rangle$ for all $x \in S$ and all $i \in [n]$. Then $\||v'_{x,i}\rangle\| = \||v_{x,i}\rangle\|$, and for the terms where $x_i \neq y_i$, we have $\langle v'_{x,i}|v'_{y,i}\rangle = \langle v_{x,i}|v_{y,i}\rangle$ if $f(x) \neq f(y)$ and $\langle v'_{x,i}|v'_{y,i}\rangle = 0$ if $f(x) = f(y)$.

Span programs

The dual of the adversary method is equivalent to a linear-algebraic model of computation known as *span programs*. This model was first studied in the context of classical computational complexity. It was connected to quantum algorithms for formula evaluation by Reichardt and Špalek, and was subsequently related to the adversary method by Reichardt.

A span program for a function $f\colon \{0,1\}^n \to \{0,1\}$ consists of a target vector $|t\rangle \in \mathbb{C}^D$, sets of input vectors $I_{i,b} \subset \mathbb{C}^D$ for all $i \in [n]$ and $b \in \{0,1\}$, and a set of free input vectors $I_{\mathrm{free}} \subset \mathbb{C}^D$. The set of available input vectors for input x is $I(x) := I_{\mathrm{free}} \cup \bigcup_{i\in[n]} I_{i,x_i}$. We say that a span program computes f if $|t\rangle \in \operatorname{span} I(x)$ if and only if $f(x) = 1$.

The complexity of a span program is measured by its *witness size*. If $f(x) = 1$, then there is a linear combination of vectors from $I(x)$ that gives $|t\rangle$; the witness size of x is the smallest squared length of the coefficients for any such linear combination. If $f(x) = 0$, then there is a vector that has inner product 1 with $|t\rangle$ that is orthogonal to all available input vectors; the witness size of x is the smallest squared length of the vector of inner products of this vector with *all* input vectors (of course, these inner products are zero for the available input vectors). The witness size of f is the largest witness size of any $x \in S$, or equivalently, the geometric mean of the largest witness sizes of 0- and 1-inputs.

The smallest witness size of any span program computing f is precisely $\mathrm{Adv}^{\pm}(f)$, and there is a close relationship between span programs and dual adversary solutions. Given a dual adversary

solution with vectors $|v_{x,i}\rangle$, one can construct a matrix whose rows are the vectors $\bigoplus_{i\in[n]}\langle\bar{x}_i|\langle v_{x,i}|$. Take the columns of this matrix in block i and subblock b to be the vectors in $I_{i,b}$, let the target vector be the all ones vector, and let there be no free input vectors. It can be shown that this gives a span program for f whose witness size is exactly the complexity of the dual adversary solution. Furthermore, every span program can be put into a canonical form for which this translation can be reversed to produce a dual adversary solution: taking the vectors of a canonical span program to be the columns of a matrix, the rows give dual adversary vectors for $x \in f^{-1}(0)$ and the witness vectors for $x \in f^{-1}(1)$ give the remaining dual adversary vectors. For more detail on this translation, see Lemma 6.5 of arXiv:0904.2759 (see the rest of that paper for more than you ever wanted to know about span programs).

We focus on dual adversary solutions here, as these are simpler to work with for the applications we consider. However, for other applications it may be useful to work directly with span programs instead; in particular, (non-canonical) span programs offer more freedom when trying to devise upper bounds.

Unstructured search

We now give a simple example of an optimal dual adversary solution, namely for unstructured search. Let $f\colon S \to \{0,1\}$ be defined by $f(x) = \textsc{or}(x)$ with $S = \{x \in \{0,1\}^n\colon |x| \leq 1\}$ the set of inputs with Hamming weight at most 1. Take dimension $d = 1$; let $|v_{0,i}\rangle = 1$ for all $i \in [n]$ and $|v_{x,i}\rangle = x_i$. The constraint (3) gives

$$\sum_{i\colon 0\neq(e_j)_i} \langle v_{0,i}|v_{e_j,i}\rangle = \langle v_{0,j}|v_{e_j,j}\rangle = 1 \tag{5}$$

for all $j \in [n]$ (where $e_j \in \mathbb{C}^n$ is the jth standard basis vector) and

$$\sum_{i\colon (e_j)_i\neq(e_k)_i} \langle v_{e_j,i}|v_{e_k,i}\rangle = \langle v_{e_j,j}|v_{e_k,j}\rangle + \langle v_{e_j,k}|v_{e_k,k}\rangle = 0 \tag{6}$$

for $j \neq k$, so the constraint is satisfied.

The 0- and 1-complexities of this solution are

$$C_0 = \sum_{i\in[n]} 1 = n \tag{7}$$

$$C_1 = \max_j \sum_{i\in[n]} \delta_{i,j} = 1. \tag{8}$$

Since $\sqrt{C_0C_1} = \sqrt{n}$, we see that $\mathrm{Adv}^{\pm}(f) \leq \sqrt{n}$, demonstrating that the previously discussed adversary lower bound is the best possible adversary lower bound.

It is easy to extend this dual adversary solution to one for the total OR function. For any $x \neq 0$, simply let $|v_{x,i}\rangle = \delta_{i,j}$, where j is the index of any particular bit for which $x_j = 1$ (e.g., the first such bit). Then the constraints are still satisfied, and the complexity is the same. As an exercise, you should work out an optimal dual adversary for AND.

Function composition

A nice property of the adversary method (in both dual and primal formulations) is its behavior under function composition. Given functions $f\colon \{0,1\}^n \to \{0,1\}$ and $g\colon \{0,1\}^m \to \{0,1\}$, we define

$f \circ g \colon \{0,1\}^{nm} \to \{0,1\}$ by $(f \circ g)(x) = f(g(x_1, \ldots, x_m), \ldots, g(x_{nm-m+1}, \ldots, x_{nm}))$. Here we focus on upper bounds, for which we have the following.

Theorem. $\mathrm{Adv}^{\pm}(f \circ g) \leq \mathrm{Adv}^{\pm}(f)\,\mathrm{Adv}^{\pm}(g)$.

Proof. Let $\{|v_{x,i}\rangle \colon x \in \{0,1\}^n, i \in [n]\}$ be an optimal dual adversary solution for f, and let $\{|u_{y,j}\rangle \colon y \in \{0,1\}^m, j \in [m]\rangle\}$ be an optimal dual adversary solution for g. Let $y = (y^1, \ldots, y^n)$ where each $y^i \in \{0,1\}^m$. Then define

$$|w_{y,(i,j)}\rangle = |v_{g(y),i}\rangle \otimes |u_{y^i,j}\rangle \tag{9}$$

where $g(y)$ denotes the vector with $g(y)_i = g(y^i)$.

We claim that this is a dual adversary solution for $f \circ g$. To see this, we compute

$$\sum_{(i,j) \colon y^i_j \neq z^i_j} \langle w_{y,(i,j)} | w_{z,(i,j)} \rangle = \sum_{i \in [n]} \langle v_{g(y),i} | v_{g(z),i} \rangle \sum_{j \colon y^i_j \neq z^i_j} \langle u_{y^i,j} | u_{z^i,j} \rangle \tag{10}$$

$$= \sum_{i \in [n]} \langle v_{g(y),i} | v_{g(z),i} \rangle (1 - \delta_{g(y^i),g(z^i)}) \tag{11}$$

$$= \sum_{i \colon g(y^i) \neq g(z^i)} \langle v_{g(y),i} | v_{g(z),i} \rangle \tag{12}$$

$$= 1 - \delta_{(f \circ g)(y),(f \circ g)(z)}. \tag{13}$$

Finally, since $\||w_{y,(i,j)}\rangle\| = \||v_{g(y),i}\rangle\| \cdot \||u_{y^i,j}\rangle\|$, using (2) gives

$$\mathrm{Adv}^{\pm}(f \circ g) \leq \max_y \sum_i \||v_{g(y),i}\rangle\|^2 \sum_j \||u_{y^i,j}\rangle\|^2 \tag{14}$$

$$\leq \mathrm{Adv}^{\pm}(f)\,\mathrm{Adv}^{\pm}(g) \tag{15}$$

as claimed. □

Note that here we needed the constraint (3) in the case where $f(x) = f(y)$.

In particular, combining this with the dual adversary for OR and a similar solution for AND, this shows that $\mathrm{Adv}^{\pm}(f) \leq \sqrt{n}$ for the n-input balanced binary AND-OR tree.

An algorithm from a dual adversary solution

The dual adversary is significant not just because it gives upper bounds on $\mathrm{Adv}^{\pm}(f)$, but because it directly gives a quantum algorithm for evaluating f with quantum query complexity $O(\mathrm{Adv}^{\pm}(f))$. (Note that the construction is not necessarily time-efficient—it may use many more elementary gates than queries—but many known algorithms developed using span programs have subsequently led to explicit, time-efficient algorithms.)

In particular, this shows that the quantum query complexity of the balanced binary AND-OR tree is $O(\sqrt{n})$. This was originally shown, up to some small overhead, using a continuous-time quantum walk algorithm based on scattering theory. The classical query complexity of this problem is $O(n^{\log_2(\frac{1+\sqrt{33}}{4})}) = O(n^{0.753})$, and no better quantum algorithm was known for many years. From the perspective of span programs, the formula evaluation algorithm can be seen a method of recursive evaluation with no need for error reduction.

Similarly to the quantum walk search algorithms we discussed previously, the algorithm for the adversary dual uses a product of two reflections. Let $A = \mathrm{Adv}^{\pm}(f)$, and let Δ be the projector onto $\mathrm{span}\{|\psi_x\rangle \colon f(x) = 1\}$ where

$$|\psi_x\rangle := \frac{1}{\sqrt{\nu_x}}\Big(|0\rangle + \tfrac{1}{\sqrt{2A}} \sum_{i\in[n]} |i\rangle|v_{x,i}\rangle|x_i\rangle\Big) \tag{16}$$

with $\{|v_{x,i}\rangle\}$ an optimal dual adversary solution. Here the normalization factor is

$$\nu_x = 1 + \frac{1}{2A} \sum_{i\in[n]} \||v_{x,i}\rangle\|^2 \leq \frac{3}{2}. \tag{17}$$

The reflection $2\Delta - I$ requires no queries to implement. Let $\Pi_x = |0\rangle\langle 0| + \sum_{i\in[n]} |i\rangle\langle i| \otimes I \otimes |x_i\rangle\langle x_i|$ be the projector onto $|0\rangle$ and states where the query and output registers are consistent. Then the reflection $2\Pi_x - I$ can be implemented using only two queries to the oracle O_x.

The algorithm runs phase estimation with precision $\Theta(1/A)$ on the unitary $U := (2\Pi_x - I)(2\Delta - I)$, with initial state $|0\rangle$. If the estimated phase is 1, then the algorithm reports that $f(x) = 1$; otherwise it reports that $f(x) = 0$. This procedure uses $O(A)$ queries. It remains to see why the algorithm is correct with bounded error.

First, we claim that if $f(x) = 1$, then $|0\rangle$ is close to the 1-eigenspace of U. We have $\Pi_x|\psi_x\rangle = |\psi_x\rangle$ for all x and $\Delta|\psi_x\rangle = |\psi_x\rangle$ for $f(x) = 1$, so clearly $U|\psi_x\rangle = |\psi_x\rangle$. Furthermore, $|\langle 0|\psi_x\rangle|^2 = 1/\nu_x \geq 2/3$ for all x, so surely $\|\Pi_x|0\rangle\|^2 \geq 2/3$. Thus the algorithm is correct with probability at least $2/3$ when $f(x) = 1$.

On the other hand, we claim that if $f(x) = 0$, then $|0\rangle$ has small projection onto the subspace of eigenvectors with eigenvalue $e^{i\theta}$ for $\theta \leq c/A$, for some constant A. To prove this, we use the following:

Lemma (Effective spectral gap lemma)**.** *Let $|\phi\rangle$ be a unit vector with $\Delta|\phi\rangle = 0$; let P_ω be the projector onto eigenvectors of $U = (2\Pi - I)(2\Delta - I)$ with eigenvalues $e^{i\theta}$ with $|\theta| < \omega$ for some $\omega \geq 0$. Then $\|P_\omega \Pi|\phi\rangle\| \leq \omega/2$.*

Let

$$|\phi_x\rangle := \frac{1}{\sqrt{\mu_x}}\Big(|0\rangle - \sqrt{2A} \sum_{i\in[n]} |i\rangle|v_{x,i}\rangle|\bar{x}_i\rangle\Big), \tag{18}$$

where the normalization factor is

$$\mu_x = 1 + 2A \sum_{i\in[n]} \||v_{x,i}\rangle\|^2 \leq 1 + 2A^2. \tag{19}$$

For any y with $f(y) = 1$, we have

$$\langle\psi_y|\phi_x\rangle = \frac{1}{\sqrt{\nu_y \mu_x}}\Big(1 - \sum_{i\colon y_i \neq x_i} \langle v_{y,i}|v_{x,i}\rangle\Big) = 0, \tag{20}$$

so $\Delta|\phi_x\rangle = 0$. Also, observe that $\Pi_x|\phi_x\rangle = |0\rangle/\sqrt{\mu_x}$. By the effective spectral gap lemma, $\|P_\omega|0\rangle\| \leq \sqrt{\mu_x}\omega \leq \sqrt{1 + 2A^2}\,\omega \approx \sqrt{2}A\omega$. Thus, choosing $\omega = \sqrt{\frac{2}{3}} \cdot \frac{1}{A}$ gives a projection of at most $1/\sqrt{3}$, so the algorithm fails with probability at most $1/3$ (plus the error of phase estimation, which can be made negligible, and the small error from approximating $1 + 2A^2 \approx 2A^2$, which is negligible if $A \gg 1$).

It remains to prove the lemma.

Proof. We apply Jordan's lemma, which says that for any two projections acting on the same finite-dimensional space, there is a decomposition of the space into a direct sum of one- and two-dimensional subspaces that are invariant under both projections. (We something closely related on the second assignment when computing the spectrum of a product of reflections.)

We can assume without loss of generality that $|\phi\rangle$ only has support on 2×2 blocks of the Jordan decomposition in which Δ and Π both have rank one. If the block is 1×1, or if either projection has rank 0 or 2 within the block, then U acts as either $\pm I$ on the block; components with eigenvalue -1 are annihilated by P_ω, and components with eigenvalue $+1$ are annihilated by Π.

Now, by an appropriate choice of basis, restricting Δ and Π to any particular 2×2 block gives

$$\bar{\Delta} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{21}$$

$$\bar{\Pi} = \begin{pmatrix} \cos\frac{\theta}{2} \\ \sin\frac{\theta}{2} \end{pmatrix} \begin{pmatrix} \cos\frac{\theta}{2} & \sin\frac{\theta}{2} \end{pmatrix} \tag{22}$$

where $\frac{\theta}{2}$ is the angle between the vectors projected onto within the two subspaces. A simple calculation shows that $(2\bar{\Pi} - I)(2\bar{\Delta} - I)$ is a rotation by an angle θ, so its eigenvalues are $e^{\pm i\theta}$. Since $\Delta|\phi\rangle = 0$, the component of $|\phi\rangle$ in the relevant subspace is proportional to $\binom{0}{1}$, and

$$\left\| \bar{\Pi} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\| = \left\| \sin\frac{\theta}{2} \begin{pmatrix} \cos\frac{\theta}{2} \\ \sin\frac{\theta}{2} \end{pmatrix} \right\| = \left|\sin\frac{\theta}{2}\right| \leq \frac{\theta}{2} \tag{23}$$

as claimed. □

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 17: Learning graphs

While span programs provide a powerful tool for proving upper bounds on quantum query complexity, they can be difficult to design. The model of learning graphs, introduced by Belovs, is a restricted class of span programs that are more intuitive to design and understand. This model has led to improved upper bounds for various problems, such as subgraph finding and k-distinctness.

Learning graphs and their complexity

A learning graph for an n-bit oracle is a directed, acyclic graph whose vertices are subsets of $[n]$, the set of indices of input bits. Edges of the learning graph can only connect vertices $\sigma \subset [n]$ and $\sigma \cup \{i\}$ for some $i \in [n] \setminus \sigma$. We interpret such an edge as querying index i, and we sometimes say that the edge $(\sigma, \sigma \cup \{i\})$ *loads* index i. Each edge e has an associated weight $w_e > 0$. We say that a learning graph computes $f\colon S \to \{0,1\}$ (where $S \subseteq \{0,1\}^n$) if, for all x with $f(x) = 1$, there is a path from $\varnothing$ to a 1-certificate for x (a subset of indices σ such that $f(y) = f(x)$ for all y such that $x_\sigma = y_\sigma$, where x_σ denotes the restriction of x to the indices in σ).

Associated to any learning graph for f is a complexity measure $\mathcal{C} = \sqrt{\mathcal{C}_0 \mathcal{C}_1}$, the geometric mean of the 0-complexity $\mathcal{C}_0$ and the 1-complexity $\mathcal{C}_1$. The 0-complexity is simply $\mathcal{C}_0 := \sum_e w_e$, where the sum is over all edges in the learning graph.

The definition of the 1-complexity is somewhat more involved. Associated to any x with $f(x) = 1$, we consider a *flow* in the learning graph, which assigns a value p_e to each edge so that for any vertex, the sum of all incoming flows equals the sum of all incoming flows. There is a unit flow coming from vertex $\varnothing$; this is the only source. A vertex can be a sink if and only if it contains a 1-certificate for x. The complexity of any such flow is $\sum_e p_e^2/w_e$. (Note that $w_e > 0$ for any edge in a learning graph, although we also have the possibility of omitting edges.) The complexity $\mathcal{C}_1(x)$ is the smallest complexity of any valid flow for x. Finally, we have $\mathcal{C}_1 := \max_{x \in f^{-1}(1)} \mathcal{C}_1(x)$, the largest complexity of any 1-input.

Unstructured search

Perhaps the simplest example of a learning graph is for the case of unstructured search. The learning graph simply loads an index. In other words, there is an edge of weight 1 from $\varnothing$ to $\{i\}$ for each $i \in [n]$. Clearly, we have $\mathcal{C}_0 = n$. To compute the 1-complexity, consider the input $x = e_i$ for some $i \in [n]$. For this input there is a unique 1-certificate, namely $\{i\}$. The only possible flow is the one with unit weight on the edge from $\varnothing$ to $\{i\}$. This gives $\mathcal{C}_1(e_i) = 1$ for all i, so $\mathcal{C}_1 = 1$. Therefore the complexity of this learning graph is $\mathcal{C} = \sqrt{\mathcal{C}_0 \mathcal{C}_1} = \sqrt{n}$.

It is not hard to see that the same learning graph works for the total function OR: for each x with $f(x) = 1$, we can send all flow to any particular i for which $x_i = 1$.

From learning graphs to span programs

We now show that every learning graph implies a dual adversary solution of the same complexity, so that the learning graph complexity is an upper bound on quantum query complexity, up to constant factors.

We construct vectors $|v_{x,i}\rangle$ for all $x \in S$. These vectors consist of a block for each vertex σ of the learning graph, with the coordinates within each block labeled by possible assignments of the bits in σ. Since we fix a particular index i, we can think of the blocks as labeling edges $e_{\sigma,i} := (\sigma, \sigma \cup \{i\})$. We define

$$|v_{x,i}\rangle = \begin{cases} \sum_\sigma \sqrt{w_{e_{\sigma,i}}} |\sigma, x_\sigma\rangle & \text{if } f(x) = 0 \\ \sum_\sigma \frac{p_{e_{\sigma,i}}}{\sqrt{w_{e_{\sigma,i}}}} |\sigma, x_\sigma\rangle & \text{if } f(x) = 1 \end{cases} \tag{1}$$

where the sums only run over those $\sigma \subset [n]$ for which $e_{\sigma,i}$ is an edge of the learning graph.

It is easy to check that this definition satisfies the dual adversary constraints. For any $x, y \in S$ with $f(x) = 0$ and $f(y) = 1$, we have

$$\sum_{i\colon x_i \neq y_i} \langle v_{x,i} | v_{y,i} \rangle = \sum_{i\colon x_i \neq y_i} \sum_\sigma \sqrt{w_{e_\sigma,i}} \frac{p_{e_{\sigma,i}}}{\sqrt{w_{e_{\sigma,i}}}} \langle x_\sigma | y_\sigma \rangle \tag{2}$$

$$= \sum_{i\colon x_i \neq y_i} \sum_{\sigma\colon x_\sigma = y_\sigma} p_{e_{\sigma,i}}. \tag{3}$$

Now observe that the set of edges $\{e_{\sigma,i}\colon x_\sigma = y_\sigma, x_i \neq y_i\}$ forms a cut in the graph between the vertex sets $\{\sigma\colon x_\sigma = y_\sigma\}$ and $\{\sigma\colon x_\sigma \neq y_\sigma\}$ Since $\varnothing$ is in the former set and all sinks are in the latter set, the total flow through the cut must be 1.

Recall that we do not have to satisfy the constraint for $f(x) = f(y)$ since there is a construction that enforces this condition without changing the complexity provided the condition for $f(x) \neq f(y)$ is satisfied.

It remains to see that the complexity of this dual adversary solution equals the original learning graph complexity. For $b \in \{0, 1\}$, we have

$$C_b = \max_{x \in f^{-1}(b)} \sum_{i \in [n]} \| |v_{x,i}\rangle \|^2 \tag{4}$$

$$= \max_{x \in f^{-1}(b)} \sum_{i \in [n]} \sum_\sigma \begin{cases} w_{e_{\sigma,i}} & \text{if } b = 0 \\ \frac{p^2_{e_{\sigma,i}}}{w_{e_{\sigma,i}}} & \text{if } b = 1 \end{cases} \tag{5}$$

$$= \begin{cases} \mathcal{C}_0 & \text{if } b = 0 \\ \max_{x \in f^{-1}(1)} \mathcal{C}_1(x) & \text{if } b = 1 \end{cases} \tag{6}$$

$$= \mathcal{C}_b. \tag{7}$$

Therefore $\sqrt{C_0 C_1} = \sqrt{\mathcal{C}_0 \mathcal{C}_1} = \mathcal{C}$ as claimed. In particular, $\mathrm{Adv}^{\pm}(f) \leq \mathcal{C}$, so $Q(f) = O(\mathcal{C})$.

Learning graphs are simpler to design than span programs: the constraints are automatically satisfied, so one can focus on optimizing the objective value. In contrast, span programs have exponentially many constraints (in n, if f is a total function), and in general it is not obvious how to even write down a solution satisfying the constraints.

Note, however, that learning graphs are not equivalent to general span programs. For example, learning graphs (as defined above) only depend on the 1-certificates of a function, so two functions with the same 1-certificates have the same learning graph complexity. The 2-threshold function (the symmetric Boolean function that is 1 iff two or more input bits are 1) has the same certificates as element distinctness, so its learning graph complexity is $\Omega(n^{2/3})$, whereas its query complexity is $O(\sqrt{n})$. This barrier can be circumvented by modifying the learning graph model, but even such variants are apparently less powerful than general span programs.

Element distinctness

We conclude by giving another simple example of a learning graph, one for element distinctness. (This requires generalizing learning graphs to non-Boolean input alphabet, but this generalization is straightforward.) We assume for simplicity that there is a unique collision—in fact, the analysis of the learning graph works for the general case by simply fixing one particular collision when designing a flow.

A convenient simplification is to break up the learning graph into k *stages*, which are simply subsets of the edges. To compute the complexity of a stage, we sum over only the edges in that stage. It is easy to see that there is a learning graph whose complexity is at most the sum of the complexities of the stages times the square root of the number of stages (which we will take to be constant). Let $\mathcal{C}_b^j$ denote the b-complexity of stage j. By dividing the weight of every edge in stage j by $\mathcal{C}_0^j$, we send $\mathcal{C}_0^j \to 1$ and $\mathcal{C}_1^j \to \mathcal{C}_0^j \mathcal{C}_1^j$. Then the total 0-complexity becomes $\mathcal{C}_0 = k$ and the total 1-complexity becomes

$$\mathcal{C}_1 = \sum_{j=1}^{k} \mathcal{C}_0^j \mathcal{C}_1^j \leq \left(\sum_{j=1}^{k} \sqrt{\mathcal{C}_0^j \mathcal{C}_1^j} \right)^2 \tag{8}$$

(since the 1-norm upper bounds the 2-norm), so $\mathcal{C} \leq \sqrt{k} \sum_{j=1}^{k} \sqrt{\mathcal{C}_0^j \mathcal{C}_1^j}$.

Another useful modification is to allow multiple vertices corresponding to the same subset of indices. It is straightforward to show that such learning graphs can be converted to span programs at the same cost, or to construct a new learning graph with no multiple vertices and the same or better complexity.

The learning graph for element distinctness has three stages. For the first stage, we load subsets of size $r-2$. We do this by first adding edges from $\varnothing$ to $\binom{n-i}{r-3}$ copies of vertex $\{i\}$, so that there are $\sum_{i=1}^{n} \binom{n-i}{r-3} = \binom{n}{r-2}$ singleton vertices. Then, from each of these singleton vertices, we load the remaining indices of each possible subset of size $r-2$, one index at a time. Every edge has weight 1. Then the 0-complexity of the first stage is $(r-2)\binom{n}{r-2}$.

To upper bound the 1-complexity of the first stage, we route flow only through vertices that do not contain the indices of a collision, sending equal flow of $\binom{n-2}{r-2}^{-1}$ to all subsets of size $r-2$. This gives 1-complexity of at most $(r-2)\binom{n-2}{r-2}\binom{n-2}{r-2}^{-2} = (r-2)\binom{n-2}{r-2}^{-1}$.

Overall, the complexity of the first stage is at most

$$\sqrt{(r-2)^2 \binom{n}{r-2} \binom{n-2}{r-2}^{-1}} = (r-2)\sqrt{\frac{n(n-1)}{(n-r+2)(n-r+1)}} = O(r). \tag{9}$$

The second and third stages each include all possible edges that load one additional index from the terminal vertices of the previous stage. Again every edge has unit weight. Thus, the 0-complexity is $(n-r+2)\binom{n}{r-2}$ for the second stage and $(n-r+1)\binom{n}{r-1}$ for the third stage. We send the flow through vertices that contain the collision pair (namely, that contain the first index of a collision in the second stage and the second index of a collision in the third stage). Thus, the 1-complexity is $\binom{n-2}{r-2}\binom{n-2}{r-2}^{-2} = \binom{n-2}{r-2}^{-1}$ in both the second and the third stages. This gives total complexity

$$\sqrt{(n-r+2) \binom{n}{r-2} \binom{n-2}{r-2}^{-1}} = O(\sqrt{n}) \tag{10}$$

for the second stage and

$$\sqrt{(n-r+1)\binom{n}{r-1}\binom{n-2}{r-2}^{-1}} = \sqrt{\frac{n(n-1)}{(r-1)}} = O(n/\sqrt{r}) \tag{11}$$

for the third stage.

Overall, the complexity is $O(r + \sqrt{n} + n/\sqrt{r})$. This optimized by choosing r to equate the first and last terms, giving $r = n^{2/3}$. The overall complexity is $O(n^{2/3})$, matching the optimal quantum walk search algorithm.

Other applications

The simple examples discussed above only involve problems for which the optimal query complexity was previously known using other techniques. However, several new quantum query upper bounds have been given using learning graphs. These include improved algorithms for the triangle problem (and more generally, subgraph finding, with an application to associativity testing) and the k-distinctness problem. (Note that the algorithm for k-distinctness uses a subtle modification of the learning graph framework.) Unfortunately, the details of these algorithms are beyond the scope of the course.

Quantum algorithms (CO 781/CS 867/QIC 823, Winter 2013)
Andrew Childs, University of Waterloo

LECTURE 18: Approximating the Jones polynomial

In this final lecture, we discuss a very different class of quantum algorithms, ones that approximately solve various #P-complete problems. The best-known example of such a quantum algorithm is for approximating the value of a link invariant called the Jones polynomial.

The Hadamard test

The quantum algorithm for approximating the Jones polynomial uses a simple primitive called the *Hadamard test*. This is equivalent to phase estimation with a single bit of precision. Given a unitary operation U and a state $|\psi\rangle$, the Hadamard test provides a means of estimating $\langle\psi|U|\psi\rangle$. The test applies a controlled-U operation to the state $|+\rangle\otimes|\psi\rangle$ and measures the first qubit in the basis $|\pm\rangle := \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$. The state before the measurement is

$$\frac{1}{\sqrt{2}}(|0\rangle|\psi\rangle + |1\rangle U|\psi\rangle) = \frac{1}{2}\big(|+\rangle(|\psi\rangle + U|\psi\rangle) + |-\rangle(|\psi\rangle - U|\psi\rangle)\big), \tag{1}$$

so

$$\Pr(\pm) = \frac{1}{4}\||\psi\rangle \pm U|\psi\rangle\|^2 \tag{2}$$

$$= \frac{1}{2}(1 \pm \mathrm{Re}\langle\psi|U|\psi\rangle). \tag{3}$$

In other words, the expected value of the outcome is precisely $\mathrm{Re}\langle\psi|U|\psi\rangle$. Replacing the states $|\pm\rangle$ by the states $|\pm i\rangle := \frac{1}{\sqrt{2}}(|0\rangle \pm i|1\rangle)$, a simple calculation shows that we can approximate $\mathrm{Im}\langle\psi|U|\psi\rangle$.

The Jones polynomial

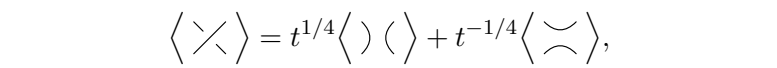
The Jones polynomial is a central object in low-dimensional topology with surprising connections to physics. Witten showed that the Jones polynomial is closely related to topological quantum field theory (TQFT). Friedman, Kitaev, Larsen, and Wang investigated the relationship between TQFT and topological quantum computing, showing that quantum computers can efficiently simulate TQFTs (thereby approximating the Jones polynomial), and that in fact TQFTs essentially capture the power of quantum computation. Here we describe the quantum algorithm for approximating the Jones polynomial in a way that does not explicitly refer to TQFT, following the treatment of Aharonov, Jones, and Landau.

To define the Jones polynomial, we must first introduce the concepts of knots and links. A *knot* is an embedding of the circle in $\mathbb{R}^3$, i.e., a closed loop of string that may wrap around itself in any way. More generally, a *link* is a collection of any number of knots that may be intertwined. In an *oriented link*, each loop of string is directed. It is natural to identify links that are *isotopic*, i.e., that can be transformed into one another by continuous deformation of the strings.

The Jones polynomial of an oriented link L is a Laurent polynomial $V_L(t)$ in the variable $\sqrt{t}$, i.e., a polynomial in $\sqrt{t}$ and $1/\sqrt{t}$. It is a *link invariant*, meaning that $V_L(t) = V_{L'}(t)$ if the oriented links L and L' are isotopic. While it is possible for the Jones polynomial to take the same value

on two non-isotopic links, it can often distinguish links; for example, the Jones polynomials of the two orientations of the trefoil knot are different.

An oriented link L can be specified by a *link diagram*, a drawing of the link in the plane with over- and under-crossings indicated. One way to define the Jones polynomial of a link diagram is as follows. First, let us define the *Kauffman bracket* $\langle L \rangle$, which does not depend on the orientation of L. Each crossing in the link diagram can be opened in one of two ways, and for any given crossing we have



$$= t^{1/4} \left\langle \;) \; (\; \right\rangle + t^{-1/4} \left\langle \; \smile \; \right\rangle, \tag{4}$$

where the rest of the link remains unchanged. Repeatedly applying this rule, we eventually arrive at a link consisting of disjoint unknots. The Kauffman bracket of a single unknot is $\langle \bigcirc \rangle := 1$, and more generally, the Kauffman bracket of n unknots is $(-t^{1/2} - t^{-1/2})^{n-1}$. By itself, the Kauffman bracket is not a link invariant, but it can be turned into one by taking into account the orientation of the link, giving the Jones polynomial. For any oriented link diagram L, we define its *writhe* $w(L)$ as the number of crossings of the form  minus the number of crossings of the form . Then the Jones polynomial is defined as

$$V_L(t) := (-t^{-1/4})^{3w(L)} \langle L \rangle. \tag{5}$$

Computing the Jones polynomial of a link diagram is quite difficult. A brute-force calculation using the definition in terms of the Kauffman bracket takes time exponential in the number of crossings. Indeed, exactly computing the Jones polynomial is #P-hard (except for a few special values of t), as shown by Jaeger, Vertigan, and Welsh. Here #P is the class of counting problems associated to problems in NP (e.g., computing the number of satisfying assignments of a Boolean formula). Of course, approximate counting can be easier than exact counting, and sometimes #P-hard problems have surprisingly good approximation algorithms.

Links from braids

It is useful to view links as arising from *braids*. A braid is a collection of n parallel strands, with adjacent strands allowed to cross over or under one another. Two braids on the same number of strands can be composed by placing them end to end. The *braid group* on n strands is an infinite group with generators $\{\sigma_1, \ldots, \sigma_{n-1}\}$, where σ_i denotes a twist in which strand i passes over strand $i+1$, interchanging the two strands. More formally, the braid group is defined by the relations $\sigma_i \sigma_{i+1} \sigma_i = \sigma_{i+1} \sigma_i \sigma_{i+1}$ and $\sigma_i \sigma_j = \sigma_j \sigma_i$ for $|i - j| > 1$.

Braids and links differ in that the ends of a braid are open, whereas a link consists of closed strands. We can obtain a link from a braid by connecting the ends of the strands in some way. One simple way to close a braid is via the *trace closure*, in which the ith strand of one end is connected to the ith strand of the other end for each $i = 1, \ldots, n$, without crossing the strands. A theorem of Alexander states that any link can be obtained as the trace closure of some braid. Another natural closure (for braids with an even number of strands) is the *plat* closure, which connects the first and second strands, the third and fourth strands, etc., at each end of the braid.

Representing braids in the Temperley-Lieb algebra

The Jones polynomial of the plat or trace closure of a braid can be expressed in terms of a representation of the braid group defined over an algebra called the Temperley-Lieb algebra. While the

definition of this algebra is fairly straightforward, the description of its representations is somewhat technical, and we will not give the details here; instead we only mention some general features.

We consider the case where $t = e^{2\pi i/k}$ is a kth root of unity. For such values, the relevant representation of the braid group is unitary. The dimension of this representation is exponential in n (specifically, it is the number of paths of length n that start from one end of a path with $k-1$ vertices), so it corresponds to a unitary operation on poly(n) qubits. The Jones polynomial of the plat closure of a braid is proportional to the expectation $\langle\psi|U|\psi\rangle$ of the associated representation matrix U in a fixed quantum state $|\psi\rangle$.

A quantum algorithm

The description of the Jones polynomial in terms of a representation of the Temperley-Lieb algebra naturally suggests a quantum algorithm for approximating the Jones polynomial. Suppose that we can efficiently implement unitary operations corresponding to twists of adjacent strands on a quantum computer. By composing such operations, we can implement a unitary operation corresponding to the entire braid. Then we can approximate the desired expectation value using the Hadamard test.

With a suitable choice for an encoding of the basis states of the representation of the braid group using qubits, one can show that the braid group representation operators corresponding to elementary twists can indeed be performed efficiently on a quantum computer. Given an explicit description of the braid group representation, the details of this implementation are fairly straightforward.

Applying this approach to the relevant unitary representation of the braid group, one obtains a quantum algorithm for approximating the Jones polynomial of the plat closure of a braid at a root of unity. In particular, for a braid on n strands, with m crossings, and with $t = e^{2\pi i/k}$, there is an algorithm running in time poly(n, m, k) that outputs an approximation differing from the actual value $V_L(t)$ of the Jones polynomial by at most $(2\cos\frac{\pi}{k})^{3n/2}/(N \cdot \mathrm{poly}(n, k, m))$, with only exponentially small probability of failure. Here N is an exponentially larger factor derived from the representation of the braid group.

The Jones polynomial of the trace closure of a braid can be similarly approximated by noting that this quantity is given by the *Markov trace* of the representation of the braid. The Markov trace is simply a weighted version of the usual trace, so it can be approximated by sampling $\langle\psi_p|U|\psi_p\rangle$ from an appropriate distribution over states $|\psi_p\rangle$. Performing such a procedure, one obtains an approximation of the Jones polynomial with additive error at most $(2\cos\frac{\pi}{k})^{n-1}/\mathrm{poly}(n, k, m)$, again in polynomial time and with exponentially small failure probability.

Quality of approximation

Without knowing more about the possible values of the Jones polynomial, it is hard to say whether the approximations described above are good. Notice that the algorithms only provide additive approximations, meaning that the error incurred by the algorithm is independent of the value being approximated, which is undesirable when that value is small. Indeed, the additive error increases exponentially with n, the number of strands in the braid. For some braids, the error might be larger than the value being approximated. It would be preferable to obtain a multiplicative approximation, but no such algorithm is known.

However, it can be shown that obtaining the additive approximation described above for the

Jones polynomial of the plat closure of a braid is as hard as any quantum computation. In other words, this quality of Jones polynomial approximation is BQP-complete. This can be shown by demonstrating that, with an appropriate encoding of qubits, the representations of the braid group can be used to implement a universal set of quantum gates. Thus, in principle, any quantum algorithm can be described in terms of some braid whose plat closure has a Jones polynomial encoding the result of the computation, with exponentially differing values corresponding to yes and no outcomes. Therefore, it is unlikely that a classical computer can obtain the same approximation, since this would give a classical algorithm for simulating a general quantum computation.

Approximating the Jones polynomial of the trace closure of a braid to the level described above turns out to be substantially easier: such a computation can be performed using a quantum computer whose initial state has only one pure qubit and many maximally mixed qubits. Such a device can approximate $\operatorname{tr} U$ by supplying the maximally mixed state in place of the pure state $|\psi\rangle$ in the Hadamard test. This does not immediately show how to approximate the Jones polynomial of the trace closure, since the Markov trace is a weighted trace. However, by using a different representation of the braid group to describe the Jones polynomial, Jordan and Shor showed that a single pure qubit indeed suffices. Furthermore, they showed that this problem is complete for the one clean qubit model, and hence apparently unlikely to be solvable by classical computers.

Other algorithms

The results described above can be generalized to many other related problems. Wocjan and Yard showed how to evaluate the Jones polynomial of a generalized closure of a braid, and how to evaluate a generalization of the Jones polynomial called the HOMFLYPT polynomial. Work of Aharonov, Arad, Eban, and Landau shows how to approximate the Tutte polynomial of a planar graph, which in particular gives an approximation of the partition function of the Potts model on a planar graph; this problem also characterizes the power of quantum computation, albeit only for unphysical choices of parameters. More generally, there are efficient quantum algorithms to compute additive approximations of tensor networks, as shown by Arad and Landau. There are also related quantum algorithms for approximating invariants of manifolds.