CS294-4: Fourier Transforms and Theoretical Computer Science		**Spring 1999**
	Lecture 11: Quantum Algorithms	
Lecturer: Umesh Vazirani	*April 16, 1999*	*Scribe: Ziv Bar-Yossef*

This lecture continues the review of the major quantum algorithms. We will discuss the next algorithms:

- Computation of Fourier Transform
- Shor's Factoring Algorithm
- Shor's Discrete Log Algorithm
- Grover's Search Algorithm

1 Computation of Fourier Transform

We consider two kinds of Fourier Transforms:

- **Fourier Transform over $\mathbb{Z}_N$**

$$|k\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \omega_N^{jk} |j\rangle$$

  where the basis vectors are interpreted as values in $\mathbb{Z}_N$, and ω_N is the N'th primitive root of unity ($\omega_N = e^{\frac{2\pi i}{N}}$).

- **Fourier Transform over $\mathbb{Z}_2^n$**

$$|k\rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{j \in \mathbb{Z}_2^n} (-1)^{\langle j,k \rangle} |j\rangle$$

  where the basis vectors are interpreted as binary strings in $\mathbb{Z}_2^n$.

1.1 Computation of Fourier Transform over $\mathbb{Z}_N$

Straightforward classical computation of Fourier Transform over $\mathbb{Z}_N$ takes $O(N^2)$ steps. For $N = 2^l$ the Fast Fourier Transform (FFT) algorithm (discussed in Lecture 3) runs in $O(N \log N)$ steps. In the quantum world the functions to which we apply the Fourier Transform are compactly represented as superpositions of $O(\log N)$ qubits. This enables exponentially faster algorithms—in some cases only $O(\log N)$ steps are required to compute the Fourier Transform of a quantum superposition. Nevertheless, measurement of the Fourier coefficients would still take an exponential number of steps ($O(N)$), simply because there are N of those.

We will consider only two easy cases of computing Fourier Transform over $\mathbb{Z}_N$:

- N is a product of small factors: $N = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_r^{e_r}$, where $p_i^{e_i} = O(\log N)$ for all $i = 1, \ldots, r$.

- N is a power of 2: $N = 2^l$.

There is an efficient algorithm due to Kitaev [5], which we will not review here, for handling the general case.

We begin with the first case. Note that $FT_N = FT_{p_1^{e_1}} \otimes FT_{p_2^{e_2}} \otimes \cdots \otimes FT_{p_r^{e_r}}$ (FT_N is the matrix of the Fourier Transform over $\mathbb{Z}_N$). This identity follows from the Chinese Remaindering representation of each element $a \in \mathbb{Z}_N$: $a = (a \bmod p_1^{e_1}, a \bmod p_2^{e_2}, \ldots, a \bmod p_r^{e_r})$.

$FT_{p_i^{e_i}}$ is a unitary transformation of dimension $p_i^{e_i} = O(\log N)$. Therefore, brute force is sufficient to compute it efficiently, as implied by the next fundamental property of quantum computation:

Theorem 1 *Any $m \times m$ unitary transformation can be ϵ-approximated by a product of poly$(m, \frac{1}{\epsilon})$ elementary unitary transformations (unitary transformations that affect a constant number of qubits).*

This fact has many proofs. The best approximations were achieved by Kitaev [4] and Solovay and Yao [7].

Returning to the Fourier Transform algorithm, we can compute each of $FT_{p_i^{e_i}}$ in poly $\log(N)$ steps. Since $p_i^{e_i} \geq 2$ for all i, then r is at most $O(\log N)$. Hence, the total number of steps required to compute FT_N is poly $\log(N)$. In other words, brute force provides a polynomial time quantum algorithm to compute the Fourier Transform in this case.

We turn now to the second case, in which $N = 2^l$. The quantum algorithm in this case is simply an adaptation of the classical FFT algorithm to the quantum setting. Naturally, it is called QFFT.

Let us first recall how FFT works. The standard formulation of the Fourier Transform matrix over $\mathbb{Z}_N$ contains ω_N^{jk} in its (j,k) entry. We reorder the columns of this matrix as follows: we put first all the columns that correspond to even k's and then the columns that correspond to odd k's. A function to which we wish to apply the Fourier Transform is specified by a vector in C^N. We need to reorder this vector's entries in the same way: first all the entries that correspond to even k's (we denote them by k_{even}), and then the entries that correspond to odd k's (k_{odd}). We get the next formulation:

$$
FT_N = \frac{1}{\sqrt{N}} \begin{array}{c c} & \begin{array}{cc} 2k & 2k+1 \end{array} \\ \begin{array}{c} j \\ j+\frac{N}{2} \end{array} & \left(\begin{array}{c|c} \omega_N^{j \cdot 2k} & \omega_N^{j(2k+1)} \\ \hline \omega_N^{(j+\frac{N}{2})2k} & \omega_N^{(j+\frac{N}{2})(2k+1)} \end{array} \right) \end{array} \left(\begin{array}{c} k_{even} \\ \hline k_{odd} \end{array} \right)
$$

$$
= \frac{1}{\sqrt{N}} \left(\begin{array}{c|c} \omega_{\frac{N}{2}}^{jk} & \omega_N^{j} \omega_{\frac{N}{2}}^{jk} \\ \hline \omega_{\frac{N}{2}}^{jk} & -\omega_N^{j} \omega_{\frac{N}{2}}^{jk} \end{array} \right) \left(\begin{array}{c} k_{even} \\ \hline k_{odd} \end{array} \right)
$$

Note that the product of each of the four submatrices with the corresponding half of the vector (either k_{even} or k_{odd}) is essentially a Fourier Transform over $\mathbb{Z}_{\frac{N}{2}}$. To compute the product of the two right submatrices with k_{odd}, we first have to apply the Fourier Transform on k_{odd}, and then multiply the j'th row of the result by either ω_N^j or $-\omega_N^j$. This observation results in the FFT recursive circuit for computing the Fourier Transform over $\mathbb{Z}_N$ depicted in Figure 1.

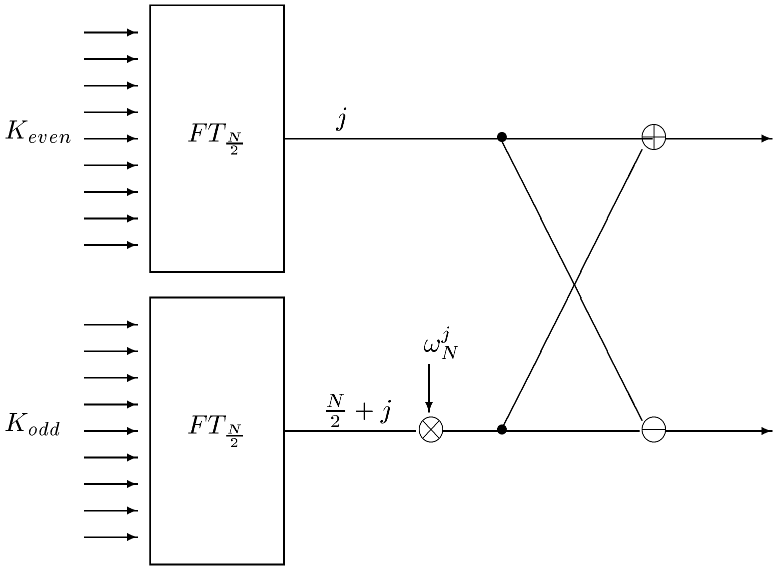

Figure 1: The FFT Circuit

This circuit basically applies the Hadamard transformation on the least significant bit of the vector values. In addition, it multiplies the odd values by ω_N^j. Therefore, the explicit formulation of the FFT circuit works on the input bits in reversed order: first it applies the Hadamard transformation and the unity root multiplication on the most significant bit of all values, then it works on the next to the most significant bit, and so on.

This circuit has $\log N$ levels—one level per bit in the representation of the vector values. Each level contains $O(N)$ gates. Hence, the total circuit size is $O(N \log N)$.

The translation of this classical FFT circuit to a quantum QFFT circuit is straightforward from the above description. Quantum states can hold superpositions of all vector entries. Hence, each level in the FFT circuit can be replaced in the quantum circuit by a constant number of gates that work on the corresponding qubit. Like the FFT circuit, the quantum circuit works on the input qubits in reversed order. The recursive description of the algorithm is specified in Figure 2.

Note that the size of this quantum circuit is $O(\log N)$: we have $\log N$ input wires (corresponding to the $\log N$ qubits specifying each basis vector $|k\rangle$), and we apply two gates for each qubit. Therefore, we managed to shrink the size of the circuit exponentially relative to the classical case.

1.2 Computation of Fourier Transform over $\mathbb{Z}_2^n$

This case is very easy, since FT_2^n, the Fourier Transform over $\mathbb{Z}_2^n$, equals to $FT_2^{\otimes n}$. F_2 is simply the 2×2 Hadamard transformation. Hence, n steps are sufficient to compute the Fourier Transform.

Note that this case is very similar to the computation of FT_N for $N = 2^l$, apart from the multiplication by ω_N^j.

The $QFFT_N$ Algorithm

Input: $\sum_{k\in\mathbb{Z}_N} a_k|k\rangle$

(1) Apply $QFFT_{\frac{N}{2}}$ on the first $l-1$ qubits

$$\Downarrow$$

$\sum_{j\in\{0,1\}^{l-1}, b\in\{0,1\}} c_{j,b}|j,b\rangle$

(2) Conditional phase shift: multiply by ω_N^j whenever $b=1$

$$\Downarrow$$

$\sum_{j\in\{0,1\}^{l-1}, b\in\{0,1\}} c_{j,b}\ \omega_N^{jb}|j,b\rangle$

(3) Apply Hadamard transformation on b

$$\Downarrow$$

The result: Fourier Transform on $\sum_{k\in\mathbb{Z}_N} a_k|k\rangle$

Figure 2: The QFFT Algorithm

2 Shor's Factoring Algorithm

The Factoring problem is a very fundamental one in computer science. It is of special importance to cryptography, where its hardness assumption constitutes the major security factor in many cryptosystems (most notably, the RSA public key cryptosystem).

In Factoring we are given an integer N, and we want to find one of its non-trivial (neither 1 or N) factors. Classically, the best factoring algorithm known runs at time $2^{O(\sqrt[3]{N})}$. In an amazing breakthrough Shor [6] showed a polynomial time quantum algorithm that solves the problem. The new algorithm is based on Simon's algorithm (discussed in the previous lecture) for the Hidden Subgroup problem.

Shor actually presented a quantum algorithm for a different problem: finding the order of a random x in $\mathbb{Z}_N$. Solving this problem enables factoring N too, as we will see shortly.

Definition 2 *The group of integers that are relatively prime to N is $\mathbb{Z}_N^* = \{x\in\mathbb{Z}_N\setminus\{0\} \ :\ gcd(x,N)=1\}$. The operation of this group is $x\cdot y = (xy) \bmod N$. The size of $\mathbb{Z}_N^*$, denoted by $\Phi(N)$, is called the Euler Phi function.*

Definition 3 *Let $x\in\mathbb{Z}_N^*$. The* **order** *of x (denoted by $ord_N(x)$) is $\min_{r\geq 1} x^r \equiv 1 \pmod N$.*

The next lemma shows that if we know to compute the order of a random $x\in\mathbb{Z}_N^*$, then with high probability we can find a non-trivial square root of 1 modulo N (i.e., a square root which is not 1 or -1).

Lemma 4 *If $N=PQ$ is odd (P and Q are primes), then for a random $x\in\mathbb{Z}_N^*$ with probability $\geq \frac{1}{2}$ $r = ord_N(x)$ is even and $x^{\frac{r}{2}} \not\equiv \pm 1 \pmod N$.*

We leave the proof of this lemma as an exercise.

Now, if we take $y = x^{\frac{r}{2}}$ for such an x, we get that $y^2 \equiv 1 \pmod N$ but $y \not\equiv \pm 1 \pmod N$. In other words, $N \mid y^2 - 1$ but $N \not\mid (y \pm 1)$. Therefore, both $gcd(y+1, N)$ and $gcd(y-1, N)$ are neither 1 nor N. Hence, they will give a non-trivial factor of N. Note that computing the gcd can be done efficiently, using Euclid's algorithm.

Lemma 4 works for odd N's that are products of two primes P and Q. This is sufficient for our needs, because even N's can be factored very easily: 2 is one of their factors. Moreover, it turns out that the hardest integers to factor are ones of the form $N = PQ$, where P and Q are evenly sized primes. Factoring such integers efficiently will enable to factor any integer efficiently.

Thus, the Factoring problem is reduced to the problem of computing the order of an element $x \in \mathbb{Z}_N^*$. From now on we fix some $x \in \mathbb{Z}_N^*$ and denote by r its order.

We show in the sequel a simplified version of Shor's algorithm. We will use the unrealistic assumption that we can work over the group $\mathbb{Z}_{\Phi(N)}$, although we don't know what $\Phi(N)$ is (actually, if we knew $\Phi(N)$ we could factor N). There is a recent work due to Lisa Hales and Sean Hallgren [3] that shows how a proof that makes Fourier Transforms over $\mathbb{Z}_K$ for some unknown K can be transformed into a legal proof that makes Fourier Transforms over $\mathbb{Z}_{K'}$ for some known $K' \geq K$. We will see this result in the next lecture.

Note that by Euler's theorem $x^{\Phi(N)} \equiv 1 \pmod N$. Hence $r \mid \Phi(N)$. We will need this property later in the proof.

To shorten notations we will denote $\Phi(N)$ by M.

Define $f : \mathbb{Z}_M \to \mathbb{Z}_N, f(a) = x^a \bmod N$. We will show that f satisfies the conditions of the Hidden Subgroup problem (see Lecture 10).

For what values of a is $f(a) = 1$? Only for $\{0, r, 2r, \ldots, M-r\} = \langle r \rangle$. (Recall that $r \mid M$)

For what values of a is $f(a) = x$? Only for $\{1, r+1, 2r+1, \ldots, M-r+1\} = \langle r \rangle + 1$.

Similarly, $f(a) = x^k$ for $\langle r \rangle + k$. Hence, f is constant on cosets of $\langle r \rangle$. Now we can use the Hidden Subgroup algorithm. We get the algorithm specified in Figure 3.

After the measurement of the second register at step (3) we get a uniform superposition of the elements of some coset of $\langle r \rangle$. Applying Fourier Transform on such a coset gives a uniform superposition on the cosets of $\langle r \rangle$— the elements of the quotient group $\mathbb{Z}_M / r$. But this quotient group is isomorphic to the group $\langle \frac{M}{r} \rangle$. Therefore, we get at step (4) a uniform superposition on the elements of this group.

Since l is measured uniformly at random at step (5), then with high probability $gcd(l, M) = 1$. Therefore, $gcd(\frac{lM}{r}, M) = \frac{M}{r}$. Now, the algorithm can recover r by outputting $\frac{M}{\frac{M}{r}}$.

3 Shor's Discrete Log Algorithm

Discrete Log is another important problem in computer science. Similar to the Factoring problem, it has no known classical polynomial time algorithm. Its hardness assumption underlies some cryptographic protocols, such as the Diffie-Helman key exchange protocol.

A generalization of Shor's quantum algorithm for Factoring to two dimensions yields a polynomial time quantum algorithm for Discrete Log. This algorithm was also first introduced in [6].

In the Discrete Log problem we are given a prime p, a generator g of $\mathbb{Z}_p^*$ ($\mathbb{Z}_p^*$ is cyclic if p is a prime), and an element $x \in \mathbb{Z}_p^*$. We need to find the single r for which $g^r \equiv x \pmod p$.

Note that the group $\mathbb{Z}_p^*$ is of size $p-1$ and isomorphic to $\mathbb{Z}_{p-1}$. We will work over the group $\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$.

Shor's Factoring Algorithm

Initial superposition: $|0\rangle \otimes |0\rangle$

(1) Apply Fourier Transform over $\mathbb{Z}_M$ on the first register

$\Downarrow$

$\frac{1}{\sqrt{M}} \sum_{a \in \mathbb{Z}_M} |a\rangle \otimes |0\rangle$

(2) Compute $f(a) = x^a \bmod N$

$\Downarrow$

$\frac{1}{\sqrt{M}} \sum_{a \in \mathbb{Z}_M} |a\rangle \otimes |x^a\rangle$

(3) Measure the second register; the result is some value x^k

$\Downarrow$

$\frac{1}{\sqrt{\frac{M}{r}}} \sum_{j=0}^{\frac{M}{r}} |jr+k\rangle \otimes |x^k\rangle$

(4) Apply Fourier Transform on the first register

$\Downarrow$

$\frac{1}{\sqrt{r}} \sum_{l=0}^{r-1} |\frac{lM}{r}\rangle \otimes |x^k\rangle$

(5) Measure the first register; the result is some $\frac{lM}{r}$ for $l \in \{0, 1, \ldots, r-1\}$

(6) Compute $\frac{M}{r} = gcd(\frac{lM}{r}, M)$ and output $r = \frac{M}{\frac{M}{r}}$

Figure 3: Shor's Factoring Algorithm

Kitaev's methods [5] work also for general Abelian groups. In particular, we know how to efficiently compute Fourier Transforms over $\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$ in the quantum setting.

We use again Simon's algorithm for the Hidden Subgroup problem. We define a function $f : \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$ as follows: $f(a,b) = g^a x^{-b} \pmod{p}$. We show now that f satisfies the conditions of the Hidden Subgroup problem.

For what values of (a,b) is $f(a,b) = 1$? $f(a,b) = g^a x^{-b} = g^a g^{-br} = g^{a-br}$. Hence, $f(a,b) = 1 = g^0$ only when $a \equiv br \pmod{p-1}$. This means that $f(a,b) = 1$ for $\{(r,1), (2r,2), \ldots, ((p-2)r, p-2)\}$. This is exactly the subgroup $\langle (r,1) \rangle$ in $\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$.

Similarly, $f(a,b) = g^k$ for $\{(r+k,1), (2r+k,2), \ldots, ((p-2)r+k, p-2)\} = \langle (r,1) \rangle + (k,0)$. Therefore, f is constant on cosets of $\langle (r,1) \rangle$. Using the Hidden Subgroup algorithm we get the algorithm for Discrete Log specified in Figure 4.

After the measurement of the second register at step (3) we get a uniform superposition of the elements

Shor's Discrete Log Algorithm

Initial superposition: $|0\rangle \otimes |0\rangle$

(1) Apply Fourier Transform over $\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$ on the first register

$\Downarrow$

$\frac{1}{p-1} \sum_{(a,b) \in \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}} |a, b\rangle \otimes |0\rangle$

(2) Compute $f(a, b) = g^a x^{-b} \pmod{p}$

$\Downarrow$

$\frac{1}{p-1} \sum_{(a,b) \in \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}} |a, b\rangle \otimes |g^{a-br}\rangle$

(3) Measure the second register; the result is some value g^k

$\Downarrow$

$\frac{1}{\sqrt{p-1}} \sum_{l=0}^{p-1} |lr + k, l\rangle \otimes |g^k\rangle$

(4) Apply Fourier Transform on the first register

$\Downarrow$

$\frac{1}{\sqrt{p-1}} \sum_{(c,d) \in S} |c, d\rangle \otimes |g^k\rangle$

(5) Measure the first register; the result is some (c, d)

(6) If $gcd(c, p-1) = 1$ output $-dc^{-1} \pmod{p-1}$

Figure 4: Shor's Discrete Log Algorithm

of some coset of $\langle (r, 1) \rangle$. Applying Fourier Transform on such a coset gives a uniform superposition on the elements of the quotient group $\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1} / \langle (r, 1) \rangle$. This quotient group is isomorphic to the perpendicular subspace of $\langle (r, 1) \rangle$, which we denote by S (i.e., for all $(c, d) \in S$ and all $(lr, l) \in \langle (r, 1) \rangle$ the inner product is zero: $lrc + ld \equiv 0 \pmod{p-1}$).

Since (c, d) is measured from a uniform superposition at step (5), then with high probability $gcd(c, p-1) = 1$. Hence, w.h.p. c has an inverse modulo $p - 1$. Now $lrc + ld \equiv 0 \pmod{p-1}$ for all $l \in \mathbb{Z}_{p-1}$ implies $rc \equiv -d \pmod{p-1}$, which in turn derives $r \equiv -dc^{-1} \pmod{p-1}$.

4 Grover's Search Algorithm

In the Database Search problem we are given a function $f : \{1, 2, \ldots, N\} \to \{0, 1\}$, where f is 1 on exactly one of the inputs x. Our goal is to find this x. We can think of $\{1, 2, \ldots, N\}$ as an exponentially large database. f is a function that specifies an element of the database we wish to retrieve.

Classical deterministic algorithms cannot solve the problem in less than N steps (search the whole database). Probabilistic algorithms can reduce the expected search time to $\frac{N}{2}$. Nevertheless, any classical algorithm would require an exponential number of steps ($\Omega(N)$) to solve the problem.

Ideally, a quantum algorithm would succeed to find x in polynomial time (poly $\log(N)$ steps). However, Bennet, Bernstein, Brassard, and Vazirani prove in [1] that $\Omega(\sqrt{N})$ queries to the database are required by any quantum algorithm.

On the other hand, Grover showed in [2] a quantum algorithm for the problem that makes $O(\sqrt{N}))$ queries. Hence, this algorithm is optimal up to a constant. We describe this algorithm in the sequel.

The Database Search problem has major complexity theoretic implications. Searching for an element in a huge database can be thought as searching for a satisfying assignment to a formula among an exponential number of possible assignments. Grover's algorithm implies that quadratic speed-up is possible for NP complete problems: they can be solved in $2^{\frac{n}{2}}$ steps by quantum algorithms. On the other hand, this is the best that can be done by "black-box" methods. The lower bound of Bennett et al. implies that relative to a random oracle $NP \not\subseteq BQP$. Hence, non-relativizing techniques would be required for devising a better quantum algorithm for NP complete problems. Finding such an algorithm is as hard as tackling the classical "P vs. NP" problem. It follows that unless $P = NP$, it is not likely that quantum computers can solve NP complete problems in polynomial time.

Next, we outline Grover's algorithm. Fix some database f and assume $j \in \{1, 2, \ldots, N\}$ is the element we are searching for (i.e., $f(j) = 1$ and $f(i) = 0$ for $i \neq j$).

We start with a uniform superposition over the database elements: $\sum_{i=1}^{N} |i\rangle$. Our goal is to arrive at a superposition, in which the amplitude of $|j\rangle$ would be constant. In that case a measurement will give the right result ($|j\rangle$) with high (constant) probability.

The algorithm makes T iterations. The superposition before the k'th iteration is denoted by $\sum_{i=1}^{N} \alpha_i^k |i\rangle$. At each iteration we perform one query to the database and apply some "magical" unitary transformation we will describe later. The application of this transformation builds up constructive interference that increases the amplitude of $|j\rangle$ by $\frac{1}{\sqrt{N}}$ at each step, while keeping the amplitudes of the other basis vectors about the same. After $T = O(\sqrt{N})$ iterations the amplitude of $|j\rangle$ is a constant, as wanted.

Assume, for simplicity, $N = 2^n$ (the proof can be generalized to any N). The unitary transformation we apply at each step is the following:

$$U = \begin{pmatrix} -1+\frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \\ \frac{2}{N} & -1+\frac{2}{N} & \cdots & \frac{2}{N} \\ . & . & \cdots & . \\ . & . & \cdots & . \\ . & . & \cdots & . \\ \frac{2}{N} & \frac{2}{N} & \cdots & -1+\frac{2}{N} \end{pmatrix}$$

To see U is indeed a unitary transformation and that it can be computed efficiently, one should note that it equals to the following:

$$U = H \times \begin{pmatrix} 1 & & & & & & 0 \\ & -1 & & & & & \\ & & -1 & & & & \\ & & & \cdot & & & \\ & & & & \cdot & & \\ & 0 & & & & \cdot & \\ & & & & & & -1 \end{pmatrix} \times H$$

where H is the $N \times N$ Hadamard matrix (here we use the fact $N = 2^n$). This equality can be easily verified by multiplying each of the sides by a vector

$$\begin{pmatrix} \alpha_1 \\ \cdot \\ \cdot \\ \cdot \\ \alpha_N \end{pmatrix}$$

The i'th entry of the result in both cases is $-\alpha_i + \frac{2}{N}\sum_{l=1}^{N}\alpha_l = -\alpha_i + 2\mu$, where μ is the mean of the α_i's: $\mu = \frac{1}{N}\sum_{l=1}^{N}\alpha_l$.

What U does is reflecting $\alpha_1, \ldots, \alpha_N$ about their mean. Denote by $\beta_1, \ldots, \beta_N$ the result of the application of U on $\alpha_1, \ldots, \alpha_N$. Note the two next properties of U:

1. U is invariant under the mean: the mean of $\beta_1, \ldots, \beta_N$ is exactly the mean of $\alpha_1, \ldots, \alpha_N$.
2. If α_i is at distance d from the mean μ, then β_i is at distance $-d$ from μ.

Figure 5 illustrates graphically the operation of U.

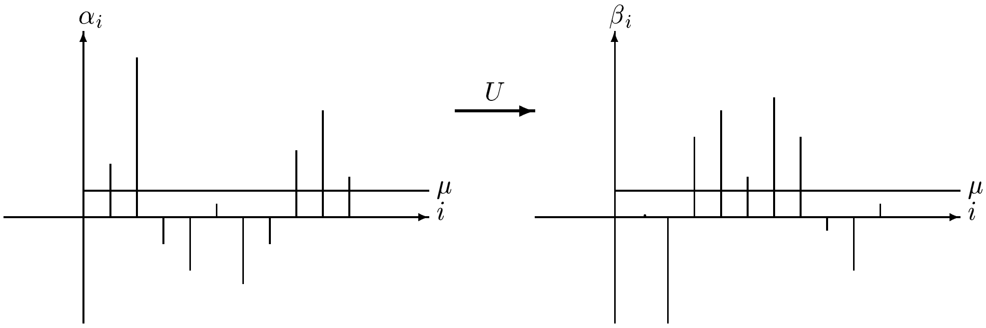

Figure 5: Reflection about the Mean

Grover's algorithm begins with a uniform superposition on the database elements: $\frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle$. The graphical illustration of the superposition's amplitudes is presented in Figure 6.

The algorithm makes a phase inversion of $|j\rangle$ and leaves all the other basis vectors as is. This is achieved by querying f and applying the unitary transformation that maps $|i\rangle$ to $(-1)^{f(i)}|i\rangle$ on the current superposition. The new superposition obtained after this step is $\frac{1}{\sqrt{N}}\sum_{i\neq j}|i\rangle - \frac{1}{\sqrt{N}}|j\rangle$ (see Figure 7).

Next, the algorithm applies the transformation U, which reflects the amplitudes about their mean. Since the amplitudes of the basis vectors $|i\rangle$ for $i \neq j$ are very close to the mean, they do not change significantly.

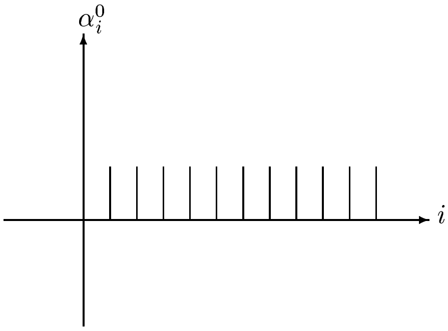

Figure 6: Initial Superposition

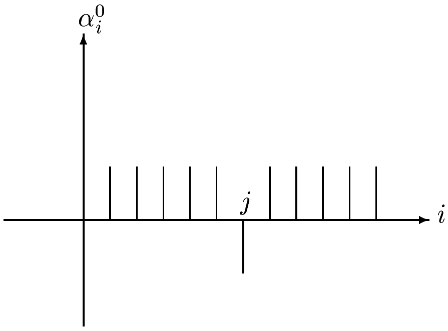

Figure 7: Superposition after Phase Inversion of j

However, the amplitude of $|j\rangle$, which is very far from the mean almost doubles in its absolute value. It follows that α_j^1 increases by $\frac{1}{\sqrt{N}}$ relative to α_j^0. Figure 8 illustrates the superposition obtained after the inversion about the mean.

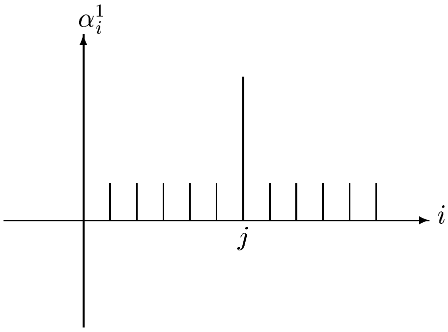

Figure 8: Superposition after Inversion about the Mean

This concludes one iteration of Grover's algorithm. The full algorithm, specified in Figure 9, makes $O(\sqrt{N})$ such iterations. At each iteration the amplitude of $|j\rangle$ increases by $\frac{1}{\sqrt{N}}$, while the amplitudes of the other basis vectors decrease moderately. After the loop ends, $|j\rangle$ has a constant amplitude, hence it is measured with high probability at the last step.

Grover's Search Algorithm

Initial superposition: $|0\rangle$

(1) Apply Fourier Transform over $\mathbb{Z}_N$

$$\Downarrow$$

$$\frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle$$

(2) Repeat for $k = 0, 1, \ldots, T \quad (T = O(\sqrt{N}))$:

(3) Apply phase inversion of $|j\rangle$

$$\Downarrow$$

$$\sum_{i\neq j}\alpha_i^k|i\rangle - \alpha_j^k|j\rangle$$

(4) Apply inversion about the mean (transformation U)

$$\Downarrow$$

$$\sum_{i=1}^{N}\alpha_i^{k+1}|i\rangle$$

(5) Measure $|i\rangle$

Figure 9: Grover's Search Algorithm

References

[1] C.H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. on Computing*, 26(5):1510–1523, 1997.

[2] L.K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of* 28^{th} *Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996.

[3] L. Hales and S. Hallgren. Sampling fourier transforms on different domains. In *Proceedings of* 31^{st} *Annual ACM Symposium on Theory of Computing*, pages 330–338, 1999.

[4] A.Y. Kitaev. *Russian Math. Surveys*, 52(6):1191, 1995.

[5] A.Y. Kitaev. Quantum measurements and the abelian stabilizer problem. quant-ph/9511026, 1995.

[6] P.W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. on Computing*, 26(5):1484–1509, 1997.

[7] R. Solovay and A.C.C. Yao. preprint, 1996.