QIP Note: On the Quantum Fourier Transform and Applications

Ivan Damgård

1 Introduction

This note introduces Fourier transforms over finite Abelian groups, and shows how a quantum implementation of these transforms can be used to solve the hidden subgroup problem in any Abelian group. Efficient quantum algorithms for discrete log and factoring follow as special cases.

2 Simon's Problem

We first look at a simple problem introduced by Simon. Suppose we are given a quantum circuit U_f computing a function $f : \{0,1\}^n \rightarrow \{0,1\}^m$. We are promised that f is a two-to-one function, and furthermore that there exists some $s \neq 0$ with the property that for any $x \in \{0,1\}^n$, $f(x) = f(x \oplus s)$. Stated differently: every value in $Im(f)$ has exactly 2 preimages, and these are of the form $x, x \oplus s$ for some x. Our task is to find s.

It is easy to see that the classical version of this problem requires exponentially many evaluations of f, even if probabilistic computation is allowed. As we shall see, there is a quantum solution that requires only a linear number of evaluations to find a solution with probability arbitrarily close to 1. The algorithm uses the following subrutine:

1. Starting in state $|0^n\rangle\ |0^m\rangle$, apply $H^{\otimes n}$ to the first n-bit register.
2. Apply U_f.
3. Measure the second m-bit register (as we will never touch this register again during the algorithm, we might as well wait until the end with measuring, but doing it here makes the analysis simpler).
4. Apply $H^{\otimes n}$ to the first n-bit register.
5. Measure the first n-bit register.

Let us compute how the state of the computer evolves through the algorithm, using as usual the convention that $N = 2^n$: After step 1, we have

$$\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle\ |0^m\rangle$$

After applying U_f, we have

$$\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle\ |f(x)\rangle$$

Since the superposition is equally weighted over all x and f is two-to-one, measuring the second register produces a random $y \in Im(f)$, and the first register collapses to an equally weighted superposition over the two preimages of y. More formally speaking, by the measurement, we are splitting the entire Hilbert space in 2^{n-1} subspaces. There is one subspace for each $y \in Im(f)$, and each subspace consists of all vectors of form $|z\rangle\,|y\rangle$. The projection of the state above on such a subspace is of form $\frac{1}{\sqrt{N}}(\,|x\rangle + \,|x \oplus s\rangle)\,|y\rangle$, where $f(x) = f(x \oplus s) = y$. This projection has the same length for any y. The state after measurement is obtained by normalizing, so we have

$$\frac{1}{\sqrt{2}}(\,|x\rangle + \,|x \oplus s\rangle)\,|y\rangle$$

After application of $H^{\otimes n}$, we get

$$H^{\otimes n}\frac{1}{\sqrt{2}}(\,|x\rangle + \,|x \oplus s\rangle)\,|y\rangle = \frac{1}{\sqrt{2}}(H^{\otimes n}\,|x\rangle + H^{\otimes n}\,|x \oplus s\rangle)\,|y\rangle$$

Using the general formula for the action of $H^{\otimes n}$ on basis vectors, we get

$$\frac{1}{\sqrt{2}}\frac{1}{\sqrt{N}}\sum_{z=0}^{N-1}((-1)^{x\cdot z} + (-1)^{z\cdot(x\oplus s)})\,|z\rangle$$

The dot-product on bitstrings distributes over $\oplus$, so the coefficient in front of $|z\rangle$ can be rewritten as

$$(-1)^{x\cdot z} + (-1)^{z\cdot(x\oplus s)} = (-1)^{x\cdot z}(1 + (-1)^{z\cdot s})$$

Since the probability of measuring z in the final step is the square of the above number times $1/(2N)$, it is clear that the probability is 0 if $z \cdot s = 1$ and $2/N$ otherwise.

In other words, this subrutine produces a randomly chosen n-bit string z, such that $z \cdot s = 0$. If we think of s as a string of n unknown bits, this gives us one linear equation in these unknowns. It is a well-known fact that if one chooses for instance $2n$ random n-bit vectors, they will span the entire vector space of n-bit vectors except with probability exponentially small in n. So if we run the subrutine $2n$ times, we will get enough equations to determine s, except with exponentially small probability.

3 Fourier Transforms over Abelian groups

Simon's problem is not terribly interesting in itself because no really interesting applications of it are known. However, it is a special case of a very general technique, and will therefore serve nicely as a running example in the following.

3.1 Characters

Let G be a finite Abelian group. By the well known structure theorem for Abelian groups, G is isomorphic to a direct product of cyclic groups, say, of orders $t_1, t_2, ..., t_n$. In other words,

$$G \simeq Z_{t_1} \times ... \times Z_{t_n}$$

For simplicity, we will assume throughout that G is simply equal to this direct product. As an example, the set of bit strings of length n is an Abelian group with bit wise XOR as group operation, and as such it is equal to $Z_2 \times ... \times Z_2$ (n factors).

Every Abelian group G of order N has associated to it precisely N so called *characters*. These are homomorphisms from G into the complex numbers. To describe what characters are, suppose first that $G = Z_t$ for some t. Then for each element $g \in Z_t$, we define the character $\chi_g() : G \to C$ by

$$\chi_g(h) = \omega_t^{gh}$$

where ω_t is the principal complex t'th root of unity, $\omega_t = e^{i2\pi/t}$. Note that this expression makes sense because g, h can be thought of as integers between 0 and $t-1$. For the more general case of $G = Z_{t_1} \times ... \times Z_{t_n}$, we can think of elements g, h as n-tuples of numbers, $g = (g_1, ..., g_n), h = (h_1, ...h_n)$. In this case, we define

$$\chi_g(h) = \prod_{i=1}^{n} \omega_{t_i}^{g_i h_i}$$

Some basic facts about characters are then easy to show:

Lemma 1. *For any $g, h \in G$, $\chi_g(h) = \chi_h(g)$*

Lemma 2. *Let N be the order of G, $N = \prod_i t_i$. Consider the N complex-valued vectors*

$$|v_g\rangle = \frac{1}{\sqrt{N}} \begin{pmatrix} \chi_g(h_1) \\ \chi_g(h_2) \\ . \\ . \\ \chi_g(h_N) \end{pmatrix}$$

one vector for each $g \in G$, where the $h_1, ..., h_N$ is a complete list of elements in G. These vectors are unit vectors and are pairwise orthogonal. In particular, we have for any $g \neq 0$ that $\sum_i \chi_g(h_i) = 0$.

Proof. That the vectors are unit vectors follows trivially from the fact that each entry is a complex number of norm 1. For pairwise orthogonality, assume first $G = Z_t$. Then we have, for $g \neq h$ that

$$\langle v_g | v_h \rangle = \frac{1}{t} \sum_{i=0}^{t-1} \chi_g(i)^* \chi_h(i) = \frac{1}{t} \sum_{i=0}^{t-1} (\omega_t^{gi})^* \omega_t^{hi} = \frac{1}{t} \sum_{i=0}^{t-1} ((\omega_t^*)^g \omega_t^h)^i$$

Since $g \neq h$, $(\omega_t^*)^g \omega_t^h$ is a non-trivial t'th root of unity (i.e., it is not 1, but becomes 1 when raised to the t'th power). It is a well-known fact that when adding all powers of such a root of unity, we get 0.

For general G, orthogonality follows by using the above and the generalized definition of characters. The last claim follows from the fact that $v_0 = 1/\sqrt{N}(1, 1, ..., 1)$ and v_0 is orthogonal to v_g.

The set of characters is a group with multiplication as group operation and the trivial caharacter (which sends everything to 1) as neutral element. It is easily seen that $\chi_{g+h} = \chi_g \chi_h$ and hence the mapping that sends g to χ_g is a group homomorphism, and even an isomorphism since the neutral element 0 is the only element whoose character is the trival character. So if we let $Im(G)$ be the group of characters of G, we have

Lemma 3. *For any Abelian group G, we have $Im(G) \simeq G$.*

3.2 The Fourier Transform

Suppose we set up a quantum computer acting on a space where there is one basis vector for each element $g \in G$, called $|g\rangle$. When implementing this physically, we can use n' q-bits where $2^{n'} \geq N = |G|$. This means that the space is really $2^{n'}$ dimensional, but we will only be operating in the N-dimensional subspace spanned by the $|v_g\rangle$'s so we will ignore the rest of the space in what follows.

We now build an N by N matrix whose columns are exactly the vectors $|v_g\rangle$ we defined above. It now follows from the orthogonality relations that this matrix is unitary. The operator defined by this matrix is called F_G. It can also be defined by what it does to each basis vector:

$$F_G |g\rangle = \frac{1}{\sqrt{N}} \sum_{h \in G} \chi_g(h) |h\rangle,$$

namely the output will be the quantum state corresponding to the vector $|v_g\rangle$, which takes above form when written in the usual notation for a state. Since the operator is unitary, it can (at least in principle) be implemented. It is also clear from the above that the N by N matrix defining F_G is symmetric and hence $F_G^{-1} = F_G^{\dagger} = F_G^{*}$.

If we return to the example where $G = Z_2 \times ... \times Z_2$ is the group of n-bit strings with bit-wise XOR (i.e., addition mod 2) as group operation, we have that $t_1 = ... = t_n = 2$ and $\omega_{t_1} = ... = \omega_{t_n} = -1$. Then we can think of elements $g, h \in G$ as n-tuples $g = (g_1, ..., g_n), h = (h_1, ..., h_n)$ where all g_i, h_i's are 0/1 values. Consequently, we have $\chi_g(h) = \prod_{i=1}^{n} (-1)^{g_i h_i}$. Plugging this into the definition of F_G above, we obtain:

$$F_G |g\rangle = \frac{1}{\sqrt{N}} \sum_{h \in G} (\prod_{i=1}^{n} (-1)^{g_i h_i}) |h\rangle = \frac{1}{\sqrt{N}} \sum_{h \in G} (-1)^{g \cdot h} |h\rangle,$$

where $g \cdot h$ is the usual inner product on bit strings. In other words, in this case $F_G = H^{\otimes n}$.

3.3 Fourier Sampling

This section deals with the problem of implementing a Fourier transform F_G in cases where either direct implementation of F_G is difficult or some information about G is missing.

We first consider the case where G is cyclic of order N, and we write $F_G = F_N$ in this case. If N is a 2-power, there is a simple circuit implementing F_N, described in Nielsen and Chuang. In other cases, in particular if N has large prime factors, no such simple circuit is known. It turns out, however, that one can approximate the result of applying

F_N by instead applying F_M where $M > N$ is chosen sufficiently large. This was first shown by Hales and Halgren, and later a stronger version of the result was shown by Høyer. We concentrate here on the main way in which Fourier transforms are applied in algorithms, namely where the transform is applied to some input state, after which the result is measured.

So we assume we are given some algorithm $\mathcal{A}$ that ends by applying F_N and a measurement, and the measurement result gives a solution to some computational problem with probability ϵ. This is known as a *Fourier sampling algorithm.* The idea is now to replace the final part of the algorithm by applying F_M, a measurement and some (classical) post processing. This produces as output either "failure" or a result in the same range as the output from the original algorithm. By the Fourier sampling theorem described below, if we choose M appropriately, the probability of non-failure is large, and given we did not fail, the distribution of the outcome is close to what the original algorithm would produce. In particular, we will be able to solve the same problem $\mathcal{A}$ was designed for, with good probability.

To describe the Fourier sampling result in more detail, we need some notation: fix some state $|u\rangle = \sum_{i=0}^{N-1} u_i |i\rangle$ to be used as input to the experiment where we apply F_N to get $|v\rangle = F_N |u\rangle$ and measure $|v\rangle$. This experiment induces a probability distribution $\mathcal{D}_v$ over the integers from 0 to $N-1$.

Suppose we apply instead F_M to $|u\rangle$, where $M > N$. Here, we think of $|u\rangle$ as a vector in an M-dimensional space, where only the first N coordinates may be different from 0. The result $|w\rangle = F_M |u\rangle$ when measured yields a distribution $\mathcal{D}_w$ over the interval from 0 to $M-1$. Since we want to relate this to the original distribution, we need some way to relate results in $[0..M-1]$ to results in $[0..N-1]$. To this end, for $i \in [0..N-1]$, we let $i' = \lfloor iM/N + 1/2 \rfloor$ denote the closest integer to iM/N.

The idea is now to do the following experiment: first we measure $|w\rangle$. If this yields a result j such that $j = i'$ for some $i \in [0..N-1]$ we will consider this to be "equivalent" to having measured i in the original experiment, and we output i as the result. For any other value of j we output "failure"[1]. We let $\mathcal{D}_{w'}$ be the distribution of outputs from this experiment conditioned on the event that we did not fail. Thus both distributions $\mathcal{D}_v, \mathcal{D}_{w'}$ are defined over $[0..N-1]$. We let $|\mathcal{D}_v - \mathcal{D}_{w'}|$ be the statistical distance between $\mathcal{D}_v, \mathcal{D}_{w'}$, i.e., $|\mathcal{D}_v - \mathcal{D}_{w'}| = \sum_i |\mathcal{D}_v(i) - \mathcal{D}_{w'}(i)|$. We now have

Theorem 1. *Given any $N > 16$, $s \geq 1$, $M \geq s \cdot 12N \log_2 N$. Let $|u\rangle$ be any input state and let $\mathcal{D}_v, \mathcal{D}_{w'}$ be the distributions of results of applying F_N, F_M to $|u\rangle$ as described above. Then we have*

$$|\mathcal{D}_v - \mathcal{D}_{w'}| \leq 4/s$$

$$Pr(Success) \geq \frac{N}{M}$$

[1] In the following, M will be at least $12N$, in which case it is easy to identify failures and compute i in case of success: for success, jN/M should be an integer plus or minus at most $1/24$ and we round jN/M to an integer to get i

We now return to the original scenario, where we were given a Fourier sampling algorithm $\mathcal{A}$. Recall that $\mathcal{A}$ could solve the problem it was designed for while using F_N with probability ϵ. Now, if we choose $s = 8/\epsilon$, $M = s \cdot 12N\log_2 N$, then the above result says that, replacing F_N by F_M, we will get a successful sampling with probability at least $\epsilon \cdot 1/(96\log_2 N)$ and given we sample successfully, the result we get will solve the problem with probability at least $\epsilon/2$.

This gives an efficient Fourier sampling algorithm for any $G = Z_N$ (as long as N is known). For non-cyclic G, we can exploit that if for instance $G = Z_N \times Z_U$, then $F_G = F_N \otimes F_U$, and so using the above theorem we can approximate F_G by $F_M \otimes F_V$. Where $M > N, U > V$ are appropriately chosen. This can also be generalized to a larger number of cyclic components.

3.4 The Orthogonal Subgroup

For any subgroup $H \leq G$, we can define the orthogonal subgroup $H^\perp$, as

$$H^\perp = \{g \in G |\ \chi_g(h) = 1 \text{ for all } h \in H\}$$

Since we have $\chi_{g'+g} = \chi_{g'}\chi_g$, this is clearly a subgroup of G. In the case of $G = Z_2 \times ... \times Z_2$, we are saying that $z \in H^\perp$ iff $(-1)^{z \cdot x} = 1$ for all $x \in H$. In other words $z \cdot x = 0$. So this just means that if we think of z, x as vectors in an n dimensional vector space of the field with 2 elements, then H is a subspace and $H^\perp$ is the orthogonal complement to H. This is the motivation for the terminology $H^\perp$.

Lemma 4. *We have $H^\perp \simeq G/H$, in particular $|H^\perp| = |G|/|H|$.*

Proof. We give only a sketch here: let $Im(H^\perp) = \{\chi_g |\ g \in H^\perp\}$. Then by Lemma 3, $Im(H^\perp) \simeq H^\perp$. Furthermore, it can be shown that there is a one-to-one correspondence between characters in $Im(H^\perp)$ and characters of the factor group G/H, in other words, we have $Im(H^\perp) \simeq Im(G/H)$, and finally by lemma 3, we have $Im(G/H) \simeq G/H$.

Unfortunately, the simple interpretation of $H^\perp$ as the orthogonal complement of a vector space does not carry over to arbitrary groups. As an example of this, consider the case where $G = Z_9$ and H is the subgroup of order 3, i.e., $H = \{0, 3, 6\}$. In order for an element i to be in $H^\perp$, it has to satisfy $\chi_3(i) = \omega_9^{3i} = 1$ and $\chi_6(i) = \omega_9^{6i} = 1$, in other words $3i = 6i = 0 \bmod 9$. It is easy to see that this is satisfied iff $i = 0, 3, 6$, so in this case we have $H = H^\perp$. This generalizes: if $G = Z_{N^2}$ for some N, and H is the order N subgroup, then $H^\perp = H$.

From $\chi_g(h) = \chi_h(g)$ it follows that $(H^\perp)^\perp = H$. In other words, that $H^\perp$ determines H uniquely. In fact, it is always possible to go from efficiently from $H^\perp$ to H, as we shall see. We will need the following basic facts:

Lemma 5. *For any natural number t, we have that $\phi(t)/t$ is in $\Omega(1/\log\log t)$, where $\phi()$ is Euler's ϕ-function. This implies that if we consider the experiment of choosing repeatedly and uniformly elements in an Abelian group G, the expected number of elements one needs to choose to obtain a set of elements that generate G is logarithmic in $|G|$.*

Proof. The first fact is simply basic well known number theory, we will not prove it here. If G is cyclic of order t, there are $\phi(t)$ generators in G, so the first fact implies that the expected number of elements to choose to get a generator is $O(\log \log t)$. The general case now follows from the fact that the number of cyclic components in G is logarithmic in the order of G.

We then have:

Lemma 6. *Given a set of elements that generate $H^\perp$, one can compute a random element in H in polynomial time (polynomial in the bit length of the numbers on the input). In particular, one can compute a set of generating elements in H in expected polynomial time.*

Proof. The technique behind this is slightly out of scope for the course, so we give only a sketch here: in general, $G = Z_{t_1} \times ... \times Z_{t_n}$. Let e be the least common multiple of all the t_i's. It is easy to see that then any root of unity ω_{t_i} can be written as a power of ω_e, indeed $\omega_{t_i} = \omega_e^{e/t_i}$. It follows that for any elements $g = (g_1, ..., g_n), h = (h_1, ..., h_i)$, we have for the character value $\chi_g(h) = \omega_e^{\alpha_1^g h_1 + ... + \alpha_n^g h_n}$ where the coefficients α_i^g satisfy that $\alpha_i^g = eg_i/t_i$. Therefore given a set of generators $g^{(1)}, ..., g^{(d)}$ of $H^\perp$, we know that h is in H if and only if it satisfies the linear equations

$$\alpha_1^{g^{(i)}} h_1 + ... + \alpha_n^{g^{(i)}} h_n = 0 \bmod e$$

for $i = 1, ..., d$. Hence to select a random element in H, it suffices to select a random solution to this linear equation system. This can always be done efficiently, provided soluitons exist, but this is guaranteed in this case. The final claim now follows from Lemma 5.

The proof of the above lemma can be seen as a direct generalization of the linear system of equations we had to solve in Simon's algorithm to find the hidden s: The elements $g^{(1)}, ..., g^{(d)}$ in $H^\perp$ correspond to the z-values with $z \cdot s = 0$ from Simon's algorithm, while the generators in H we produce correspond to s.

3.5 A Quantum Algorithm Producing Elements in the Orthogonal Subgroup

Let $H \leq G$, and let $H_1, ..., H_T$ be the cosets of H in G, $T = |G|/|H|$. If we are given a state that is an equally weighted superposition over all elements in some coset H_i, then we shall see that applying F_G to this state produces an equally weighted superposition over all elements in $H^\perp$.

For a concrete example, suppose $G = Z_2 \times ... \times Z_2$, and let H be the order 2 subgroup consisting of 0^n and a single bitstring s. Then cosets of this subgroup in G are of the form $x, x \oplus s$ for some x. Therfore, the state we have after step 3 in Simon's algorithm is precisely a superposition over a coset. Then we applied $H^{\otimes n}$ which, as we have seen, is F_G in this case, and indeed this produces a superposition over all z with $z \cdot s = 0$, i.e. z's in $H^\perp$. In general, we have

Lemma 7. *Notation as above. For any coset H_i, we have*

$$F_G(\frac{1}{\sqrt{|H|}}\sum_{g\in H_i}|g\rangle) = \frac{1}{\sqrt{|H^\perp|}}\sum_{h\in H^\perp}\chi_h(g_i)\,|h\rangle$$

where g_i is any fixed element (representative) in the coset H_i.

Proof. Inserting the definition of F_G and reversing the order of summation, we get

$$F_G(\frac{1}{\sqrt{|H|}}\sum_{g\in H_i}|g\rangle) = \frac{1}{\sqrt{|G||H|}}\sum_{h\in G}\sum_{g\in H_i}\chi_g(h)\,|h\rangle$$

Let g_i be a representative for the coset H_i, so that $g \in H_i$ can be written as $g = g_i\tau$ for some $\tau \in H$. For the inner sum above, we then have:

$$\sum_{g\in H_i}\chi_g(h) = \sum_{g\in H_i}\chi_h(g) = \chi_h(g_i)\sum_{\tau\in H}\chi_h(\tau)$$

If $h \in H^\perp$, then $\chi_h(\tau) = 1$. Then the inner sum is $\chi_h(g_i)|H|$ and the total amplitude on $|h\rangle$ in the output from F_G is $\frac{\sqrt{|H|}}{\sqrt{|G|}}\chi_h(g_i)$. Therefore, if we were to measure the output state, the probability of measuring any such h is $|H|/|G|$, and since there are $|G|/|H|$ elements in $H^\perp$, any other element in G must have amplitude 0.

4 The Hidden Subgroup Problem

Suppose we are given a group G, a quantum circuit U_f computing a function $f : G \rightarrow \{0,1\}^m$, and a quantum circuit computing F_G. We are promised that there is a subgroup H of G such that the function f is constant and distinct on all cosets of H in G. More precisely, if g is in coset H_i and h is in coset H_j, then $f(g) = f(h)$ precisely if $i = j$. The problem is to find a set of elements that generate H.

It should be obvious that Simon's problem is a special case of the Hidden subgroup problem, with $G = Z_2\times ...\times Z_2$, $H = \{0, s\}$ and $F_G = H^{\otimes n}$. Indeed, the following algorithm that solves the hidden subgroup problem is a generalization of Simon's algorithm. As before when we introduced F_G we assume we have on register with n' qbits large enough to hold (superpositions of) elements of G and one that can hold m qbits.

1. Starting in state $|0_G\rangle\ |0^m\rangle$, where $|0_G\rangle$ is the basis state corresponding to the neutral element in G, apply F_G to the first register.
2. Apply U_f.
3. Measure the second m-bit register (as we will never touch this register again during the algorithm, we might as well wait until the end with measuring, but doing it here makes the analysis simpler).
4. Apply F_G to the first register.
5. Measure the first register.

Let us analyze what the result is of running this subrutine. The state after the first application of F_G is

$$\frac{1}{\sqrt{|G|}}\sum_{g\in G}|g\rangle\,|0^m\rangle$$

by definition of F_G. Applying U_f and measuring the second register produces

$$\frac{1}{|H|}\sum_{g\in H_i}|g\rangle\,|y\rangle$$

where H_i is some coset of H and y is the value f takes on all elements in H_i. This can be argued by a straightforward generalization of the corresponding arguments for Simon's algorithm. We now have precisely the situation we analyzed in the previous section, so it follows that applying F_G results in a superposition over $H^\perp$, and hence measuring produces a random element in $H^\perp$. Repeating this a number of times logarithmic in $|G|$ (and hence polynomial in n', m) implies we will have a generating set of $H^\perp$ except with negligible probability. We may then classically (and in polynomial time) compute a generating set for H, as described earlier, provided of course, that we know the group structure of G, that is, the number of cyclic components and their order. Otherwise, we do not have information enough to solve the linear equations involved. Summarizing, we have:

Theorem 2. *The Hidden Subgroup Problem can be solved for any Abelian group G and function f using an expected number of applications of F_G and U_f that is polynomial in the size of the input and classical polynomial time computation, provided the number of cyclic components of G and their order is known.*

As we shall see, this result has a number of very interesting applications. We describe the most important ones here. It is usually clear in such applications that U_f can be implemented efficiently, for instance because f is already easy to compute classically. It is less clear how to do F_G in general, but in most cases the Fourier sampling theorem will allow us to solve this problem.

4.1 Discrete Logarithms

Let a prime p be given, and elements $g, h \in Z_p^*$, where $h \in < g >$. The discrete log problem is to find x such that $g^x = h \bmod p$. This problem is extremely important in modern cryptography and is believed to have no efficient classical solution.

We can assume without loss of generality that the order of g is $p-1$. If this is not the case, we can choose a random element $\alpha \in Z_p^*$, which with significant probability will have order $p-1$. Then we find y, z such that $\alpha^y = g \bmod p, \alpha^z = h \bmod p$. We then have $yx = z \bmod p-1$ and can find x (the equation may not determine x uniquely modulo $p-1$, but even in this case it is easy to find *some* solution).

We define $G = Z_{p-1}\times Z_{p-1}$. Define the function $f : G \to Z_p^*$ by $f(a,b) = g^a h^b \bmod p$. Define H to be the subgroup consisting of pairs a, b such that $a + bx = 0 \bmod p-1$. Note that this is precisely the set of elements that are mapped to 1 by f. It is easy to check that in fact f is constant and distinct on each coset of H, simply because the value of $g^a h^b = g^{a+bx} \bmod p$ determines $a + bx$ uniquely modulo $p - 1$.

The general algorithm for the hidden subgroup problem now allows us to find random elements in H. Clearly, knowing a, b such that $a + bx = 0 \bmod p - 1$ allows us to find x if b is invertible modulo $p - 1$. For any value of $p - 1$, there is significant probability that this happens for a random b.

The remaining question is how we implement the two applications of F_G that the quantum part of the algorithm calls for.

For the first application, note that this is only there to set up an uniform superposition over all elements in the group. For this, it is enough to be able to make a uniform superposition over all numbers in Z_{p-1}, since then we just do this twice in two different sets of qbits. This can always be done:

Lemma 8. *For any number r, an equal superposition of numbers in Z_r can be produced efficiently.*

Proof. Let m be minimal such that $2^m \leq p - 1$, and let A be a circuit on $m + 1$ qbits which on input $|x\rangle\, |b\rangle$ where $x \in \{0, 1\}^m$ and b is one bit, outputs $|x\rangle\, |b \oplus g(x)\rangle$, where $g(x) = 1$ if and only if $0 \leq x < p - 1$. Since A is the standard quantum circuit of an easy classical function, A can be efficiently implemented. We now start with $|0^m\rangle\, |0\rangle$, we apply $H^{\otimes m}$ to the first m bits, apply A and measure the last bit. If the result is 1, we continue, else we give up and try again. It is easy to see that if the result is 1, we have an equal superposition of numbers in Z_{p-1}, and that the result is 1 with probability at least 1/2.

The second application of F_G is just Fourier sampling which we showed how to do earlier. If $p - 1$ has only small prime factors, we can do exactly the transform needed over $Z_{p-1} \times Z_{p-1}$, otherwise the Fourier sampling theorem can be used.

4.2 Period finding and Factoring

Let a composite integer N be given. The factoring problem is to find some factor v in N, such that $v \neq 1, N$.

To factor N, choose a random $x \in Z_N \setminus \{0\}$ and compute $gcd(x, N)$. If this is not 1, we are done. Otherwise x is randomly chosen in Z_N^*. Define the function $f : Z \rightarrow Z_N^*$ by $f(i) = x^i \bmod N$. This function is periodic, its period being the order of x in Z_N^*. That is, the order of x by definition is the minimal positive number r such that $x^r \bmod N = 1$. And so f satisfies that $f(i) = f(i + r)$ for any i. Imagine that we have an algorithm for finding a multiple of the period of any such f, assuming that f easy to compute. If this was the case, we could factor easily by the following result from basic computational number theory:

Lemma 9. *Given N and a black box that computes a multiple of the order of any $x \in Z_N^*$, N can be efficiently factored.*

Proof. We give only a sketch here: given rt, a multiple of the order of x, write rt as $rt = 2^v w$, where w is odd. Compute the sequence of numbers

$$x^w \bmod N, x^{2w} \bmod N, ..., x^{2^v w} \bmod N$$

Note that the next number in the sequence is obtained by squaring, and that it ends with a 1. If the first 1 in the sequence is not x^w, define y to be the number before the first 1, else define y to be 1. Clearly we have $y^2 = 1 \bmod N$, and hence $(y-1)(y+1) = 0 \bmod N$. It follows that if $y \neq \pm 1 \bmod N$, $gcd(y-1, N)$ is a non-trivial factor in N. It can be shown that for any N, the probability that a random x yields a factor in this way is at least $1/2$.

We are left with the problem of finding the period of a function f. We will give a quantum algorithm for this, so we assume we have a quantum circuit U_f of the usual form, computing f. The problem can be rephrased as a hidden subgroup problem, where however some technical dificulties arise.

Consider some multiple tr of the period r. Define $G = Z_{tr}$, and H to be the subgroup consisting of all multiples of r. It follows immediately that f restricted to numbers in G is constant and distinct on cosets of H.

Now, if we try to apply the Hidden subgroup algorithm, the first problem is to set up a random superposition of numbers from 0 to $rt-1$ for some t. How can we do this when we don't know r or any multiple of it? Note first that we can at least assume that we have an upper bound r' on r, such that $r < r' < 2r$. If this were not the case, we could initially set $r' = 1$ and repeatedly try to solve the problem, each time doubling our previous guess at r'. Note that since we know f is periodic, we can always test easily if a given number is a multiple of the period. Now let m be minimal such that $r'^2 \leq 2^m$ and simply apply $H^{\otimes m}$ on $|0^m\rangle$. By integer divison, there exist s, t such that $2^m = rt + s$, where $0 \leq s < r$, so 2^m is a good approximation to rt in the sense that $(2^m - rt)/2^m$ is exponentially small in m. This implies that the distance from $1/\sqrt{2^{m/2}} \sum_{x=0}^{2^m-1} |x\rangle$ to $1/\sqrt{rt} \sum_{x=0}^{rt-1} |x\rangle$ (the state we wanted) is exponentially small. This means that any amount of computation we do afterwards will lead to almost the same probability distribution over the final measurement results. In other words, we can proceed, assuming that we have an exact superposition over the integers from 0 to rt, where however, we do not know rt exactly. We will set $N = rt$ in the following

The next problem in applying the algorithm is to perform the transform F_N. This is not possible directly, and even if we could do it, we would only be able to find elements of $H^\perp$. In fact, $H^\perp$ will consist of all multiples of t, and a generator is an element jt with j prime to r. The probability that j is prime to r is $\Omega(1/\log\log r)$ (see Lemma 5) and so is certainly polynomial in n, the number of bits we are working on. However, even given such a generator, finding the period from it seems to require that we know rt, which is what we were looking for in the first place! We now describe how to get around this without chasing our own tail.

First define p_r to the the probability that a randomly chosen j modulo r is relatively prime to r. Since $N = rt$ we have $p_r \in \Omega(1/\log\log N)$, by Lemma 5. Then we let $s = 8/p_r$, we clearly have $s \in O(\log\log N)$.

Now note that we can always assume that we have M such that $s \cdot 12N \log_2 N \leq M \leq s \cdot 24N \log_2 N$. From the phase where the initial superposition is created, we have an upper bound 2^m on N such that $2^m - N$ is small, i.e., less that $\sqrt{N}$. So we can start by setting $M = 2^m$, and repeat our algorithm, each time doubling our previous guess at M, until we get a solution.

Once we get to a good M and apply F_M, the Fourier sampling theorem gives us that with probability at least $1/(24s \log_2 N) \in \Omega(1/\log N \log\log N)$, we produce a successful sample. Such a sample is an integer a which equals jtM/N rounded to the nearest integer for some $0 \leq j < N$. If we had we used the transform F_N to generate j, it would have been uniformly chosen and so would be prime to r with probability p_r. So by the sampling theorem, the j we implicitly generate here is prime to r with probability at least $p_r - 4/s = p_r/2 \in \Omega(1/\log\log N)$. In such a case, we have

$$|a - \frac{jtM}{N}| = |a - \frac{jM}{r}| \leq 1/2$$

and hence

$$|\frac{a}{M} - \frac{j}{r}| \leq \frac{1}{2M}$$

where $r \leq \sqrt{M}$. Using the continued fraction algorithm (see Appendix 4.4 in Nielsen and Chuang), it is now possible to recover j and r efficiently. To see why recovering them is *possible*, note that if we have two rational numbers $j/r, j'/r'$, with $r, r' \leq \sqrt{M}$, we have that

$$|\frac{j}{r} - \frac{j'}{r'}| \geq \frac{1}{r'r} \geq \frac{1}{M}$$

Therefore, in the interval from $a/M - 1/2M$ to $a/M + 1/2M$, there can be at most one rational number with denominator at most $\sqrt{M}$. Furthermore, when we find that number (i.e., its nominator and denominator), we will indeed know both j and r since if j is prime to r, the fraction j/r cannot be reduced.

Since all involved success probabilities are $\Omega(1/\text{polylog}N)$, we can recover the period in expected polynomial time, i.e. in time polynomial in the number of bits we are working with.

4.3 Other potential applications

It is generally believed that quantum computation will not be sufficient to do all of NP efficiently, so therefore people have been looking for computational problems that are notoriously hard classical, but are not believed to be NP hard. The discrete log and factoring problems are of this kind, and so is the graph isomorphism problem. This problem is also interesting because it can be rephrased as a hidden subgroup problem, albeit in a non-Abelian group:

Deciding if two given graphs are isomorphic is equivalent to deciding if a given graph $\mathcal{G}$ has any non-trivial automorphisms. Suppose the graph has n nodes, let G be the group of all permutations on n nodes, and let H be the subgroup that stabilizes the graph

$\mathcal{G}$. The original problem now translates into deciding if H is non-trivial. Let f be the mapping that sends a permutation $\phi \in G$ to the graph $\phi(G)$. Then clearly, f is constant and distinct on cosets of H, and finding a set of generators for H would certainly allow us to decide if H is a non-trivial subgroup.

One can define Fourier transforms over any group, also non-Abelian ones, and algorithms for computing them also exist. But unfortunately, it does not seem to be the case that ability to do the Fourier transform over G implies you can solve the hidden subgroup problem for non-Abelian G. Therefore, no efficient quantum algorithm for the the graph isomorphism problem is known so far.

5 An Exercise

We give here an exercise designed to illustrate why the Fourier sampling result is true. Let F_N, F_M be the Fourier transforms over cyclic groups of order N, M, and let $|v\rangle = \sum_{x=0}^{N-1} \alpha_x |x\rangle$ be some input state. Assume that N divides M.

1. Assume we compute $F_N |v\rangle$ and measure the resulting state. Find an expression in terms of the α_x's for the probability of measuring some given $0 \leq z < N$.
2. Assume we measure instead $F_M |v\rangle$, so we get some $0 \leq y < M$ as result. Define y to be good, if $y = zM/N$ for some $0 \leq z < N$. Show that the probability of producing a good y is N/M.
3. Suppose we do the experiemnt of measuring $F_M |v\rangle$, and output "fail" if the result y is not good, and otherwise output $z = yN/M$. Show that the distribution of the output, given non-failure, equals the one produced in question 1.

You have now seen that an idealized version of the Fourier sampling result is true if M is a multiple of N. The idea behind the real Fourier sampling result may then be seen as saying the following: any M can be written as $M = Nq + r$ where $0 \leq r < N$ by integer division. Thus, the distance from M to a multiple of N, relatively speaking, is $(M - Nq)/M \leq 1/q$. In other words, any large enough M is a good approximation to some multiple of N, and so the conclusions we have drawn should hold approximately for any large enough M.

6 Notes and References

Simon's problem and the algorithm for solving it was presented in [4]. Shor published his algorithms for factoring and discrete log in [3]. The Fourier sampling result by Hales and Hallgren appeared first in [1], and then a stronger version with a simplified proof by Høyer appeared in [2].

References

1. L.Hales and S. Hallgren: *Quantum Fourier Sampling Simplified*, Proceedings of STOC 99.
2. Peter Høyer: *Simplified Proof of the Fourier Sampling Theorem*, Manuscript, available from www.brics.dk.

3. P. Shor: *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Computing 26 (1997), pp.1484-1509.
4. D.Simon: *On the Power of Quantum Computation*, Proceedings of FOCS 93.