Quantum Algorithms 4/4/2023

Lecture 19: Finding Hidden Subgroups

Instructor: Dieter van Melkebeek

In this lecture we examine a problem about finding a hidden XOR-shift, often referred to as Simon's problem. We develop a quantum algorithm that only needs a linear number of blackbox queries, whereas any classical algorithm needs exponentially many. The problem is an instantiation of the hidden subgroup problem, which we introduce in its generality and present other interesting instantiations of.

1 Finding a Hidden XOR-Shift

1.1 Problem statement

Suppose we are given a black box function

$$f : \{0,1\}^n \to \{0,1\}^n$$

for which there exists a nonzero $s \in \{0,1\}^n$ such that for any two inputs $x_1, x_2 \in \{0,1\}^n$ with $x_1 \neq x_2$

$$f(x_1) = f(x_2)$$

if and only if

$$x_1 = x_2 \oplus s.$$

Put colloquially, f is a 2-to-1 function where each pair of inputs differ by the XOR-shift s. Simon's problem is to find the value of s. Specifically, we want to do this with a small number of queries to the black-box function f.

1.2 Deterministic algorithm

Naively, since we know our function is 2-to-1 over all N possible inputs, we can guarantee a collision with $\frac{N}{2} + 1$ queries by the pigeonhole principle.

A better way to solve this problem is known the "baby-step giant-step" approach or sometimes the "meet in the middle" approach. This approach works by dividing the bitstring s into two parts. Let s_{left} represent the left $\lfloor \frac{n}{2} \rfloor$ bits of s and s_{right} represent the right $\lceil \frac{n}{2} \rceil$ bits. We have that $f(s_{left}0^{\lceil \frac{n}{2} \rceil}) = f(0^{\lfloor \frac{n}{2} \rfloor}s_{right})$. More generally, for any $x_1 \in \{0,1\}^{\lfloor \frac{n}{2} \rfloor}$ and $x_2 \in \{0,1\}^{\lceil \frac{n}{2} \rceil}$:

$$f(x_1 0^{\lceil \frac{n}{2} \rceil}) = f(0^{\lfloor \frac{n}{2} \rfloor}x_2) \Leftrightarrow x_1 x_2 = 0^n \vee x_1 x_2 = s.$$

This means that we can obtain s by finding the nontrivial collision between the list of values $f(x_1 0^{\lceil \frac{n}{2} \rceil})$ for $x_1 \in \{0,1\}^{\lfloor \frac{n}{2} \rfloor}$ and the list of values $f(0^{\lfloor \frac{n}{2} \rfloor}x_2)$ for $x_2 \in \{0,1\}^{\lceil \frac{n}{2} \rceil}$. This requires only $O(\sqrt{N})$ queries to the blackbox and $O(\sqrt{N} \log N)$ other operations.

Lower bound. The query bound of $\Theta(\sqrt{N})$ is tight up to constant factors. We can see this with an adversarial argument such that every query to f returns a distinct value (until no longer mathematically possible). In this scenario, the only values of s that can be ruled out are the results of pairwise XOR operations between query inputs. After making the k-th query to f, we have $\binom{k}{2}$ pairs of inputs that we know are not related by s. If we have chosen our inputs wisely, then each pair of inputs will correspond to unique values of s so that we have eliminated $\binom{k}{2}$ distinct possibilities. In any case, there can be no more than $\binom{k}{2}$ for s that are ruled out. In order to know the answer, we need to eliminate $N-2$ values for s. The number 2 comes from the knowledge that (1) $s=0$ is not possible and (2) if we have eliminated all but one remaining possibility then we have found s. Therefore, we require k^* queries where $\binom{k^*}{2} = N-2$.

$$\binom{k^*}{2} = \frac{k^*!}{2!(k^*-2)!} = \frac{k^*(k^*-1)}{2} = N-2$$

Thus, any deterministic classical algorithm requires $k^* = \Omega(\sqrt{N})$ queries.

1.3 Probabilistic algorithm

In a probabilistic classical setting, we can solve this problem with a simple algorithm based on the birthday paradox. By making $O(\sqrt{N})$ queries with random inputs, we are likely to find a collision. This is conceptually simpler than our deterministic algorithm but does not improve the query complexity beyond a constant factor.

Furthermore, by Yao's principle, it can be proven that a probabilistic algorithm cannot do better than $O(\sqrt{N})$ query complexity.

1.4 Quantum algorithm

Our algorithm consists of running a simple quantum subroutine multiple times.

Quantum subroutine. Below is a circuit that makes a single query and allows us to solve our problem with $O(n) = O(\log N)$ queries. The circuit looks very similar to the one from our first elementary quantum algorithms: the one for distinguishing between constant and balanced Boolean functions, and for learning linear Boolean functions. The only difference is that the second register in that circuit consists of a single qubit rather than the n qubits shown here. Below we prove that this circuit lets us find a hidden XOR shift.

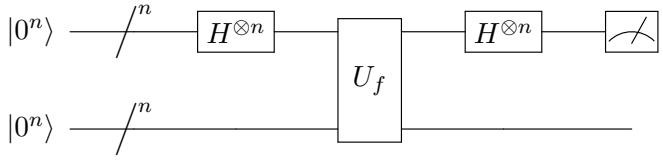

Figure 1: Quantum circuit for finding a hidden XOR shift. The $\not\big/^n$ represents a collection of n wires.

Now we analyze the behavior of this circuit. The first n-fold Hadamard gate takes us from $|0^n\rangle\,|0^n\rangle$ to

$$\frac{1}{\sqrt{N}} \sum_{x \in \{0,1\}^n} |x\rangle |0^n\rangle .$$

Then, the $2n$-qubit unitary gate U_f, which performs $|x\rangle |y\rangle \to |x\rangle |y \oplus f(x)\rangle$, further transforms the state to

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

Finally, the last n-fold Hadamard gate transforms the state to

$$\frac{1}{N} \sum_{x,y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle |f(x)\rangle .$$

At this point in the circuit, we measure the first register which is in a superposition, so we are sampling from some probability distribution. To understand that probability distribution, we look at the amplitudes of the states in our superposition. Writing

$$\sum_{z \in \text{Range}(f)} \sum_{y \in \{0,1\}^n} \alpha_{y,z} |y\rangle |z\rangle ,$$

the probability of measuring a particular $y \in \{0,1\}^n$ is $\sum_{z \in \text{Range}(f)} |\alpha_{y,z}|^2$.

Remember that f is a 2-to-1 function so its range is of size $\frac{N}{2}$. States in our superposition can only interfere with each other if they share the same $|f(x)\rangle$ in the second register. Therefore, each value of x gives exactly 2 states that interfere with each other. For a fixed $y \in \{0,1\}^n$ and $z \in \text{Range}(f)$, let x^* be one of the inputs such that $z = f(x^*) = f(x^* \oplus s)$. (The choice of x^* from the pair is arbitrary.) Using this, we can write

$$\alpha_{y,z} = \frac{1}{N} \left((-1)^{x^* \cdot y} + (-1)^{(x^* \oplus s) \cdot y} \right) .$$

Now, using the property that

$$(x^* \oplus s) \cdot y = (x^* \cdot y) \oplus (s \cdot y),$$

we can rewrite $\alpha_{y,z}$ as

$$\alpha_{y,z} = \frac{1}{N} (-1)^{x^* \cdot y} (1 + (-1)^{s \cdot y}) = \begin{cases} \frac{2}{N} (-1)^{x^* \cdot y} & \text{if } s \cdot y = 0 \bmod 2 \\ 0 & \text{if } s \cdot y = 1 \bmod 2. \end{cases}$$

Since $|\alpha_{y,z}|$ does not depend on z, our probability of observing y can be written as

$$\Pr[y] = \sum_{z \in \text{Range}(f)} |\alpha_{y,z}|^2 = \begin{cases} \frac{2}{N} & \text{if } s \cdot y = 0 \bmod 2 \\ 0 & \text{if } s \cdot y = 1 \bmod 2 \end{cases} .$$

So our observed y is always satisfies $s \cdot y = 0 \bmod 2$ and is chosen uniformly at random among all such strings. The set of all possible outcomes forms the vector space

$$s^\perp \doteq \{y \in \{0,1\}^n : s \cdot y = 0 \bmod 2\}.$$

The cardinality of $s^\perp$ is 2^{n-1}.

Overall algorithm. We run the quantum circuit in Figure 1 multiple times. The i-th run yields $y_i \in s^\perp$ chosen uniformly at random and independent from the earlier runs. Subsequent runs of the circuit only yield new information if we measure y_i that is linearly independent from the previously measured $y_1, \ldots, y_{i-1}$ values. The probability that our new vector is linearly independent from previous measurements can be written as

$$\Pr[y_i \notin \text{span}(y_1, \ldots, y_{i-1})] = 1 - \frac{|\text{span}(y_1, \ldots, y_{i-1})|}{|s^\perp|} = 1 - \frac{2^d}{2^{n-1}}$$

where

$$d = \dim(\text{span}(y_1, \ldots, y_{i-1})).$$

As long as $d < n-1$ then the probability $p = 1 - \frac{2^d}{2^{n-1}} \geq \frac{1}{2}$. Treating this as a Bernoulli experiment, the expected number of trials to achieve a new linearly independent measurement is $\frac{1}{p} \leq 2$. Thus, the expected number of runs until we reach $d = n-1$ is $O(n)$. At that point, we have $n-1$ linearly independent equations of the form $y_i \cdot s = 0 \bmod 2$. Since we have n unknown variables (i.e., the bits of s) and $n-1$ linearly independent equations, the system has two solutions: the zero vector and the XOR shift s.

It would be even better if we did not need to rely on probabilistic success to find linearly independent values of y_i. Since we know the exact probability of success (which is $1 - \frac{d}{2^{n-1}}$), we can use amplitude amplification to guarantee that we measure a linearly independent y_i. Our success criterion is $s \notin \text{span}(y_1, \ldots, y_{i-1})$ and our unitary circuit A is the circuit in Figure 1 (without the final measurement) which is also its own inverse. Since our success probability is always at least $\frac{1}{2}$ and known exactly, we only require one iteration of amplitude amplification. This results in 3 applications of our circuit: one application to generate the initial state $A|0^{2n}\rangle$, one application of A^{-1} to get to the $|0^{2n}\rangle$ state initially, and two to effectuate the single reflection around $A|0^{2n}\rangle$. Therefore we make 3 queries to the black box function per y_i. In order to achieve $n-1$ linearly independent y_i's, we require exactly $3(n-1)$ queries to f, at which point we can solve for s.

Thus the query complexity of this quantum algorithm is $O(n) = O(\log N)$, representing an exponential improvement over the best possible classical algorithms.

Exercise #11. Given two one-to-one functions $f_0, f_1 : \{0,1\}^n \to \{0,1\}^n$ where $f_1(x) = f_0(x \oplus s)$ for some $s \in \{0,1\}^n$. Find s, with certainty, using $O(n)$ queries. Note that s may be 0^n. You can make use of a unitary multiplexer $V : |b\rangle|\psi\rangle \mapsto |b\rangle U_{f_b}|\psi\rangle$.

2 Hidden Subgroup Problem

The Hidden Subgroup Problem (HSP) is a generalization of finding a hidden XOR-shift as we have discussed before. We start with formalizing the computational problem.

Input A blackbox $f : G \to R$ for some group G and set R such that for some subgroup H of G:

$$f(x_1) = f(x_2) \Leftrightarrow Hx_1 = Hx_2,$$

where $Hx \doteq \{h \cdot x : h \in H\}$ is the right coset of x modulo H, and $\cdot$ denotes the group operation in multiplicative notation. An equivalent statement to the one above is:

$$f(x_1) = f(x_2) \Leftrightarrow x_1 \cdot x_2^{-1} \in H.$$

In summary, the value that our blackbox takes on an element x of the group only depends on which right coset x belongs to, and it takes distinct values on distinct cosets.

Output A set S of generators for H, i.e., $S \subseteq G$ such that $\langle S \rangle \doteq \{s_1 \cdot s_2 \ldots s_k : s_1, s_2 \ldots, s_k \in S \cup S^{-1}, k \in \mathbb{N}\}$ equals H. There always exists a set of generators of a size at most $\log_2(|H|)$. Since H is a subset of G and $|G|$ can be at most exponential in the number n of qubits, there is hope for a quantum algorithm running in time polynomial in n.

2.1 Instantiations

We start by casting problems we have seen before as instantiations of HSP.

Finding a hidden XOR-shift. Recall that for the hidden XOR-shift problem, we are given some function $f : \{0,1\}^n \to \{0,1\}^n$ such that for some nonzero shift $s \in \{0,1\}^n$:

$$f(x_1) = f(x_2) \Leftrightarrow x_1 = x_2 \vee x_1 = x_2 \oplus s,$$

and the goal is to find s. To frame it as an instantiation of the HSP, we write:

$$f(x_1) = f(x_2) \Leftrightarrow x_1 - x_2 \in \{0, s\}.$$

The group for this problem is $G = \mathbb{Z}_2^n$ under component-wise addition, with $H = \{0, s\}$.

Constant vs. balanced for $n = 1$. We are given a function f from 1 bit to 1 bit, and we want to know whether f is constant or balanced.

$$f : \{0,1\} \to \{0,1\}$$

We claim that this can be viewed as an instantiation of the Hidden Subgroup Problem. Indeed, if we consider $G = \mathbb{Z}_2$ under addition, then $H = G$ in the constant case, and $H = \{0\}$ in the balanced case.

Learning linear functions. We are given a Boolean function that maps n bits to 1 bit of the form:

$$f : \{0,1\}^n \to \{0,1\} : x \mapsto a \cdot x \text{ for some } a \in \{0,1\}^n.$$

Our output is simply a. To cast this as an instantiation of the HSP we write:

$$f(x_1) = f(x_2) \Leftrightarrow a \cdot x_1 = a \cdot x_2 \Leftrightarrow a \cdot (x_1 - x_2) = 0 \Leftrightarrow x_1 - x_2 \in a^{\perp}.$$

Thus, the underlying group is $G = \mathbb{Z}_2^n$ under component-wise addition and the hidden subgroup $H = a^{\perp}$. The solution to the HSP gives us a set of generators for $a^{\perp}$, but we want a, not a set of generators for $a^{\perp}$. To get a, we solve the homogeneous system of linear equations of the form $a \cdot y = 0$ where y are the generators for $a^{\perp}$; the nontrivial solution to these equations is a.

Note that the above instantiations all used as the underlying groups $\mathbb{Z}_2^n, +$. Here are a few interesting instantiations that involve other groups.

Discrete log. This application involves a finite Abelian group. It is a fact that for prime p, the multiplicative group:

$$\mathbb{Z}_p^\times \doteq \{x \in \mathbb{Z}_p : \gcd(x, p) = 1\} = \{1, 2, \ldots, p-1\}$$

is cyclic. Therefore, there exists an element that can generate every element in the group. We call such element a generator. Given a prime p, a generator g for $\mathbb{Z}_p^\times$, and an arbitrary element $a \in \mathbb{Z}_p^\times$, our goal is to output the unique $\ell \in \mathbb{Z}_{p-1}$ such that $g^\ell = a$. We know this is possible because a belongs to $\mathbb{Z}_p^\times$ and that g is a generator for the cyclic group $\mathbb{Z}_p^\times$. To cast this problem as a HSP requires some ingenuity. We make use of the following function:

$$f : \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1} \to \mathbb{Z}_p : (x, y) \mapsto a^x g^y \bmod p.$$

We want to know when the pair (x_1, y_1) and (x_2, y_2) map to the same value under f:

$$f(x_1, y_1) = f(x_2, y_2) \Leftrightarrow a^{x_1} g^{y_1} = a^{x_2} g^{y_2} \bmod p.$$

We know that a can be written as g^ℓ so we have:

$$\Leftrightarrow g^{\ell x_1 + y_1} = g^\ell x_2 + y_2 \bmod p.$$

Since g is a generator, the two powers of g are the same if and only if the exponents are the same modulo $p - 1$

$$\Leftrightarrow \ell x_1 + y_1 = \ell x_2 + y_2 \bmod (p-1).$$

Now rearranging terms:

$$\Leftrightarrow \ell(x_1 - x_2) = y_2 - y_1 \bmod (p-1).$$

Simply writing in a different format:

$$\Leftrightarrow (x_1 - x_2, y_1 - y_2) \in \langle (1, -\ell) \rangle \text{ in } (\mathbb{Z}_{p-1}^2, +).$$

Thus, the function f defines a HSP over additive group $\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$ with $H = \langle (1, -\ell) \rangle$. A solution to this HSP gives us some set of generators for H. The set will need to contain an element of the form $(a, -a\ell)$, where $\gcd(a, p-1) = 1$. From any such elements, we can retrieve ℓ by multiplying the second component with the multiplicative inverse of a modulo $p - 1$ (which exists because $\gcd(a, p-1) = 1$ and can be found efficiently).

Period finding. This application involves an infinite Abelian group, namely $\mathbb{Z}$ under addition. We have a function $f : \mathbb{Z} \to \mathbb{Z}$ such that for some nonzero $p \in \mathbb{N}$:

$$f(x_1) = f(x_2) \Leftrightarrow x_1 - x_2 \text{ is a multiple of } p, \text{ or equivalently, equals 0 modulo } p.$$

The goal is to output p. Specifically, for this problem we require that there are no identical values within the period p. All values must be unique. We can cast this as an instantiation of the HSP for group $G = \mathbb{Z}$ under addition, with $H = \langle p \rangle$. Integer factorization reduces to this problem of period finding, and even to the special case of finding the order of an integer a modulo another integer μ (that is relatively prime to a). So, if we can efficiently solve period finding or order finding, we know that we can efficiently factor integers. In fact, this is the key ingredient in Shor's algorithm for integer factorization.

Graph automorphism and isomorphism. This application involves a non-Abelian finite group. An automorphism is an isomorphism of an object to itself. It is a well-known fact that a permutation $\sigma \in S_n$ is an automorphism of a (simple) graph A iff the adjacency matrix of A is invariant under permuting the rows and columns by σ, where n is the number of vertices of the graph. The adjacency matrix is a square matrix (n rows, n columns) where the (i, j)-entry is 1 if the vertex i and j are adjacent, and 0 otherwise. The set of automorphisms, $\mathrm{Aut}(A)$, is a group under composition. We now cast the question of determining $\mathrm{Aut}(A)$ into HSP. Consider the function:

$$\begin{aligned} f : S_n &\to \{0,1\}^{n \times n} \\ \sigma &\mapsto \sigma(A) \end{aligned}$$

where we identify the graph A with its adjecency matrix, which is fixed. We can then see when permutations map to the same value:

$$f(\sigma_1) = f(\sigma_2) \Leftrightarrow \sigma_1(A) = \sigma_2(A) \Leftrightarrow \sigma_1^{-1} \circ \sigma_2 \in \mathrm{Aut}(A).$$

Thus, finding generators for $\mathrm{Aut}(A)$ is an instantiation of the HSP over $(S_n, \circ)$. Observe that two connected graphs A_1 and A_2 are isomorphic iff the disjoint union $A \doteq A_1 \sqcup A_2$ has an automorphism that maps a vertex from A_1 to A_2, which is the case iff at least one element in any generating set of $\mathrm{Aut}(A)$ maps A_1 to A_2. Furthermore, isomorphism of arbitrary graphs reduces to isomorphism of connected graphs by adding to each of the two graphs a special vertex that is connected to all the vertices of the graph and is made unique such that any isomorphism between A_1 and A_2 needs to map the special vertex of A_1 to the special vertex of A_2. The latter can be done by adding a second new vertex to each graph that is only connected to the first special vertex.

3 Quantum algorithms

We now discuss the instantiations of HSP for which efficient quantum algorithms are know. The situation depends on the underlying group G.

Finite Abelian groups. If the underlying group G is a finite Abelian group, then we can efficiently solve the HSP. The algorithm is essentially a generalization of the algorithm that we have seen for finding a hidden XOR-shift, where the role $H^{\otimes n}$ is taken over by quantum Fourier transform for group G. The approach is referred to as Fourier sampling. We will discuss the Fourier transform and Fourier sampling in the next lecture. The approach assumes an efficient quantum Fourier transform over G, which we will develop explicitly for the group $\mathbb{Z}_N, +$. We can handle to HSP instantiations for the discrete log problem with this case, which allows us to break crypto-systems like Diffie-Hellman key exchange (as we will see later).

Integers under addition. Integers under addition is a non-finite Abelian group. The HSP over this group is equivalent to period finding. With an efficient algorithm for this HSP, we can efficiently perform integer factorization, which allows us to break crypto-systems like RSA (as we will see later).

Symmetric group. Graph isomorphism reduces to the HSP over the symmetric group. Even though we can efficiently compute the quantum Fourier transform over the symmetric group, we do not know how to efficiently solve the HSP over the symmetric group, and an efficient quantum algorithm for graph isomorphism remains open. The Fourier sampling approach provably does not work in this setting; the resulting distribution does not contain enough information to distinguish between the isomorphic and non-isomorphic cases in general.

Dihedral group. The dihedral group captures the symmetries of regular N-gon (rotations and reflections). It is a non-Abelian group with composition as the group operation. Efficiently solving the HSP in this group is also still open. If it can be done, it would break cryptographic systems based on the shortest lattice vector problem.