Sevag Gharibian

Lecture 8: Simon's algorithm and applications to cryptography

"Great things are not done by impulse, but by a series of small things brought together."
— Vincent van Gogh.

In the previous lecture, we began our foray into quantum algorithms with the Deutsch, Deutsch-Josza, and Berstein-Vazirani algorithms (the latter two were actually the same algorithm). Although an important proof of concept, the Deutsch-Josza algorithm did not give us a strong separation between classical and quantum query complexity, as recall a classical *randomized* algorithm can solve the Deutsch-Josza problem with only $O(2^{-K})$ probability of error, where K is the number of oracle queries.

In this lecture, we shall study Simon's algorithm, which was the first to yield an exponential separation in query complexity between the quantum and classical randomized settings. *A priori*, Simon's problem will again appear artificial. First appearances, however, can be deceiving: Not only has recent research shown that Simon's algorithm can be used to break certain classical cryptosystems, but the algorithm was a crucial step towards inspiring Peter Shor in his development of the celebrated quantum factoring algorithm.

1 Simon's algorithm

Similar to the Deutsch-Josza and Bernstein-Vazirani problems, Simon's problem is interested in computing some property of a given function $f : \{0,1\}^n \mapsto \{0,1\}^n$, where f is specified via a black-box oracle U_f. As before, we work in the query model, meaning we only count the number of queries made to U_f, and all other unitary gates are "free". The specific property Simon's problem is interested in is as follows: Given the promise that there exists a string $s \in \{0,1\}^n$, such that $f(x) = f(y)$ if and only if $x = y$ or $x = y \oplus s$ (where $\oplus$ denotes bitwise XOR), find s.

Exercise. Suppose $f : \{0,1\}^2 \mapsto \{0,1\}^2$ is such that $f(00) = 01$, $f(01) = 10$, $f(10) = 01$, and $f(11) = 10$. What is s in this case?

1.1 Birthdays and a naive classical algorithm

Before giving a quantum algorithm, let us first brainstorm how one might naively try to solve Simon's algorithm classically. Intuitively, assuming $s \neq 0^n$, "all that is required" to find s is to compute $x \neq y \in \{0,1\}$ such that $f(x) = f(y)$. For then we know $x = y \oplus s$, and so

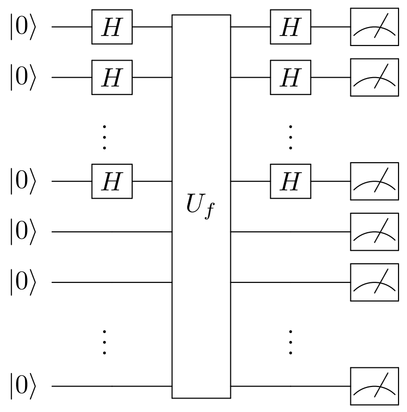
$$0^n = x \oplus x = (x \oplus y) \oplus s,$$

implying $s = x \oplus y$ due to our use of XOR. Thus, we are tasked with finding a *collision*, i.e. a pair of inputs yielding the same output. Let us be exceedingly naive — let us randomly pick inputs x uniformly at random and compute $f(x)$ until we obtain a collision. What is the expected number of queries this requires if we wish to succeed with probability at least, say, $1/2$? For this, think back to the last birthday party you attended — if there were N attendees present, how large would N have to be before the odds that there existed two attendees with the same birthday was at least $1/2$? The answer is not ≈ 365, but rather $N = 23$, by the Birthday Paradox. More generally, the latter says if there are D days in the year, a collision occurs with probability at least $1/2$ if $N \in O(\sqrt{D})$. Thus, if we equate birthdays with query results $f(x)$, we expect to make $O(\sqrt{2^n})$ queries before we find two inputs with the same output with probability at least $1/2$.

Exercise. The naive algorithm above assumed $s \neq 0^n$; how can we "complete" the algorithm so that with $O(\sqrt{2^n})$ queries, with probability at least 1/2, we output the correct s, even if s can equal 0^n?

1.2 Simon's algorithm

Interestingly, the "quantum" portion of Simon's algorithm is almost identical to the Bernstein-Vazirani algorithm. It is given by the following circuit, denoted C_n, acting on $2n$ qubits:



Exercise. What are the differences between the Deutsch-Josza circuit and the one above?

Exercise. Does the circuit above use phase kickback? Why or why not, intuitively?

For clarity, when a query has n output bits in C_n, we define for $f : \{0,1\}^n \mapsto \{0,1\}^n$:

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$$

for bit-wise XOR $\oplus$.

Despite its similarity to Deutsch-Josza and Berstein-Vazirani, however, this circuit is *not* the entire algorithm for Simon's problem — rather, it will be crucial to repeat C_n multiple times to obtain a set of n bit strings $\{y_i\}$, and classically postprocess the $\{y_i\}$ to find the desired string s. Thus, Simon's algorithm requires not 1 query, but $O(n)$ queries. This is revealing of a general view of quantum algorithms which is worth noting — a quantum circuit (such as C_n) generally allows one to *sample* from an output probability distribution, which might otherwise be difficult to sample from classically. In Simon's algorithm, as we shall now see, each run of C_n allows us to sample from the uniform distribution over strings

$$Y = \{y \in \{0,1\}^n \mid s \cdot y = 0\}. \tag{1}$$

The ability to sample from this distribution, coupled with classical polynomial-time processing, will suffice to solve Simon's problem.

Exercise. Suppose you are given the ability to sample from the uniform distribution over Y in Equation (1). Why might this suffice to efficiently compute s classically?

Analysis of C_n. Let us now trace through the execution of each run of C_n. There are four timesteps before the final measurement, whose states we denote $|\psi_1\rangle, |\psi_2\rangle, |\psi_3\rangle, |\psi_4\rangle \in (\mathbb{C}^2)^{\otimes 2n}$, respectively. Clearly, $|\psi_1\rangle = |0\rangle^{2n}$ and $|\psi_2\rangle = |+\rangle^{\otimes n}|0\rangle^{\otimes n}$. Expanding this in the standard basis, we have that

$$|\psi_3\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle|f(x)\rangle.$$

Normally, we would now apply the last set of Hadamards; however, it will be enlightening to play a technical trick first. Observe that in the circuit diagram above, the Hadamards on the first n qubits and the measurements on the last n qubits act on disjoint sets of qubits; thus, these operations commute, and the order in which we apply them is irrelevant. Thus, let us first measure the second register, obtaining some outcome $y \in \{0,1\}^n$. The key observation is that the postmeasurement state on the first register must be *consistent* with y, i.e. the first register can now only contain $x \in \{0,1\}^n$ such that $f(x) = y$. But by our promise on f, we know something about this — assuming $s \neq 0^n$, there are precisely two preimages x of y, namely x and $x \oplus s$! Our postmeasurement state, which we denote $|\psi_3'\rangle$, is hence:

$$|\psi_3'\rangle = \frac{1}{\sqrt{2}}(|x\rangle + |x \oplus s\rangle)|f(x)\rangle.$$

This demonstrates one of the strengths of quantum computation — by *postselecting* on a measurement outcome in the second register, we can collapse a superposition over the first register onto some postmeasurement state which reveals some desired structure.

Exercise. We claimed above that assuming $s \neq 0^n$, there are precisely two preimages x of y, namely x and $x \oplus s$. In other words, f is a 2-to-1 function. Convince yourself that this claim is correct.

Exercise. Assume $s = 0^n$. Is f still 2-to-1? If not, then what type of function is f?

We are now ready to apply the final set of Hadamards on the first register of $|\psi_3'\rangle$. Recall from Lecture 7 that for any $x \in \{0,1\}^n$, $H^{\otimes n}|x\rangle = (1/\sqrt{2^n})\sum_{y\in\{0,1\}^n}(-1)^{x\cdot y}|y\rangle$ for $\cdot$ the inner product modulo 2. Thus,

$$\begin{aligned}
|\psi_4\rangle &= \frac{1}{\sqrt{2^{n+1}}}\left(\sum_{y\in\{0,1\}^n}(-1)^{x\cdot y}|y\rangle + \sum_{y\in\{0,1\}^n}(-1)^{(x\oplus s)\cdot y}|y\rangle\right)|f(x)\rangle \\
&= \frac{1}{\sqrt{2^{n+1}}}\left(\sum_{y\in\{0,1\}^n}(-1)^{x\cdot y}(1+(-1)^{s\cdot y})|y\rangle\right)|f(x)\rangle,
\end{aligned}$$

where we have used the fact that $(x \oplus s)\cdot y = x \cdot y \oplus s \cdot y$. Observe now that in the exponents, $s \cdot y \in \{0,1\}$ (since we are working modulo 2), and $s \cdot y = 1$ leads to an amplitude of 0. Thus, we really have state

$$|\psi_4\rangle = \frac{1}{\sqrt{2^{n-1}}}\sum_{s\cdot y=0}(-1)^{x\cdot y}|y\rangle|f(x)\rangle,$$

i.e. an equal superposition (up to relative phase) of y in set Y (Equation (1)). Thus, measuring the first register now allows us to sample uniformly at random from Y, as claimed.

Exercise. The prefactor of $|\psi_4\rangle$ of $\sqrt{2^{-n+1}}$ suggests that $|Y| = 2^{n-1}$. Prove the latter statement holds.

Classical postprocessing. We now show how to use the ability to sample from Y to solve Simon's problem. Repeating the circuit C $n-1$ times yields strings $y_1, \ldots, y_{n-1}$, each of which satisfies $s \cdot y_i = 0$. Thus we have a linear system (where $s(i)$ denotes the ith bit of string s)

$$\begin{aligned}
s(1)y_1(1) + s(2)y_1(2) + \cdots s(n)y_1(n) &= 0 \\
s(1)y_2(1) + s(2)y_2(2) + \cdots s(n)y_2(n) &= 0 \\
&\vdots \qquad (2)\\
s(1)y_{n-1}(1) + s(2)y_{n-1}(2) + \cdots s(n)y_{n-1}(n) &= 0, \qquad (3)
\end{aligned}$$

where all arithmetic is done over the finite field $\mathbb{F}_2$ with elements $\{0,1\}$. By Gaussian elimination, which works over any field, we can solve this system in cubic *arithmetic* operations, i.e. $O(n^3)$ time, to obtain the possible solutions s. Here, an "arithmetic" operation means we are counting each field operation (i.e. an addition, subtraction, multiplication, or division) as costing one "unit of time". (One can also consider the *bit complexity* of an algorithm, which further includes how many operations over individual bits each field operation requires.) Note that $s = 0^n$ is *always* a solution to the system in Equation (3), since the system is homogeneous (i.e. includes no constant terms). By the promise on f, we know that either $s = 0^n$ is the only solution, or there is precisely one other $s \neq 0^n$, which is what we are interested in.

Probability of failure. Thus far, the algorithm we have described appears to succeed with certainty. However, there is a catch to Gaussian elimination which we have neglected to mention — to obtain a unique non-zero solution s (if it exists), we require the strings y_i to be linearly independent over $\mathbb{F}_2$. Luckily, since we are sampling *uniformly* over Y, it can be shown that the probability of $y_1, \ldots y_{n-1}$ being independent is at least $1/4$. While this may seem a poor success probability *a priori*, it is easy to boost this to something exponentially close to 1, as you will now show.

Exercise. Show that if the probability of failure in each run of the algorithm is at most $0 < c < 1$, then with K runs of the algorithm, we can guarantee success with probability at least $1 - c^{-K}$.

2 Application to cryptography

As mentioned at the outset of this lecture, Simon's algorithm inspired Shor's factoring algorithm, which in turn breaks important cryptosystems such as RSA. However, as we now discuss, even Simon's algorithm can be used to break a cryptographic primitive, *assuming* an appropriate query access model. Our particular exposition will be based on [SS17] (see also [KM10, KLLNP16]).

Feistel networks. A common idea in theoretical cryptography is to use the simplest assumptions possible to build secure cryptographic primitives. One such primitive is the computation of a *random permutation*, i.e. given an input $x \in \{0,1\}^n$, output $\pi(x)$ for π a permutation on n bits chosen uniformly at random. What do we need to implement such a primitive? Intuitively, one might guess that if we assume the ability to at least perform *random functions* $f : \{0,1\}^n \mapsto \{0,1\}^n$ (not necessarily permutations), then this could be bootstrapped to simulate random permutations.

Indeed, this is the aim of a *Feistel network*. Let us focus on the case of 3-round Feistel networks. Such a network takes inputs of the form $(l, r) \in \{0,1\}^n \times \{0,1\}^n$, and each round i produces string $(l_i, r_i) \in \{0,1\}^n \times \{0,1\}^n$, defined recursively as follows for $i \in \{1, 2, 3\}$:

$$l_i = r_{i-1} \qquad r_i = l_{i-1} \oplus f_i(r_{i-1}). \tag{4}$$

Here, the $f_i : \{0,1\}^n \mapsto \{0,1\}^n$ denote the random functions, which we assume the ability to perform. Classically, Feistel networks are provably cryptographically secure, which roughly means given a black box $U : \{0,1\}^{2n} \mapsto \{0,1\}^{2n}$ performing either a Feistel network or a truly random permutation, one cannot distinguish in polynomial-time which of the two cases holds (formal definitions are beyond the scope of this course). However, it turns out that if we assume one can quantumly query U in *superposition*, then one can break the security of the 3-round Feistel network efficiently. Specifically, in this setup, one can use Simon's algorithm to efficiently distinguish whether a given U is a Feistel network or a random permutation.

How to break a Feistel network. Let $U : \{0,1\}^{2n} \mapsto \{0,1\}^{2n}$ be a black box which either computes a 3-round Feistel network with internal random functions $f_1, f_2, f_3 : \{0,1\}^n \mapsto \{0,1\}^n$, or a truly random permutation on $2n$ bits. Our task is to distinguish which of the two cases holds.

To apply Simon's algorithm, our goal is to ideally define a function f satisfying

$$f(x) = f(y) \text{ if and only if } x = y \oplus s \tag{5}$$

when U is a Feistel network. Let us begin by looking at just the first n bits of the output of $U(x,y)$, which define some function $V:\{0,1\}^{2n}\mapsto\{0,1\}^n$. In the case of a Feistel network, note that Equation (4) implies

$$V(x,y)=y\oplus f_2(x\oplus f_1(y)).$$

Exercise. Verify the equation above.

To define f, we fix arbitrary distinct strings $\alpha,\beta\in\{0,1\}^n$. The function $f:\{0,1\}\times\{0,1\}^n\mapsto\{0,1\}^n$ will take in two inputs, a *control* bit a, and an *argument* b, such that

$$f(a,b)=\begin{cases}V(b,\alpha)\oplus\alpha & \text{if } a=0,\\ V(b,\beta)\oplus\beta & \text{if } a=1.\end{cases}$$

In other words, depending on the control bit a, we feed either α or β as the second input to black box U, and subsequently bitwise XOR on α or β.

Let us now see why f almost satisfies the promise of Simon's problem (i.e. Equation (5) when U is a Feistel network. For this, we require the following exercise.

Exercise. Suppose U is a Feistel network. Show that

$$f(a,b)=\begin{cases}f_2(b\oplus f_1(\alpha)) & \text{if } a=0,\\ f_2(b\oplus f_1(\beta)) & \text{if } a=1.\end{cases}$$

With this exercise in hand, we claim that the reverse implication of Equation (5) holds, i.e. if $x=y\oplus s$, then $f(x)=f(y)$. To see this, define mask $s=(1,f_1(\alpha)\oplus f_1(\beta))$. Then, for any input $(0,b)\in\{0,1\}\times\{0,1\}^n$ (the case of input $(1,b)$ is analogous),

$$f((0,b)\oplus s)=f(1,b\oplus f_1(\alpha)\oplus f_1(\beta))=f_2(b\oplus f_1(\alpha))=f(0,b).$$

We conclude that if U is a Feistel network, then

$$\exists s\in\{0,1\}^{n+1}\text{ such that, for all inputs }(a,b),\ f(a,b)=f((a,b)\oplus s).\qquad(6)$$

Thus, the reverse implication of Equation (5) holds. Unfortunately, the *forward* implication of Equation (5) does not in general hold.

Exercise. Why is it not necessarily true that if U is a Feistel network and if $f(a,b)=f(c,d)$, then $(a,b)=(c,d)\oplus s$? (Hint: Recall the fact that in Simon's problem, the function f was 2-to-1 assuming $s\neq 0^n$. Is f above 2-to-1 in general?)

Luckily, it turns out Simon's algorithm does not need this forward direction of Equation (5) to hold.

We can now outline the algorithm for breaking the security of the Feistel network. For this, note that if U is *not* a Feistel network, but rather a random permutation, then the odds of there existing an s satisfying Equation (6) can be shown to be negligible. Thus, intuitively, given black box access to U, we can run Simon's algorithm to see if we can compute a mask s satisfying Equation (6); if we succeed, we conclude U is a Feistel network, and if we fail, we conclude U is a random permutation.

This outline is more formally stated below; its analysis is left as an exercise. As a first step in the analysis, one needs that given black-box access to U, one can simulate black-box access to f, which you will now show.

Exercise. Given black-box access to U, i.e. the ability to map input $|x, y\rangle$ to $|x, y \oplus U(x)\rangle$, show how to simulate black box access to f, i.e. how to simulate mapping $|u, y\rangle \mapsto |u, y \oplus f(u)\rangle$.

The algorithm of [SS17] (see also [KM10]) is as follows. It returns either FEISTEL or RANDOM to denote its guess as to the identity of U.

1. Initialize empty set S, whose purpose will be to store samples of strings y obtained via Simon's algorithm.
2. Repeat until S contains n linearly independent strings $y \in \{0,1\}^{n+1}$:
   (a) If $|S| \geq 2n$, return FEISTEL.
   (b) Run Simon's algorithm on the black box for f to sample string y, and add y to S.
3. (Compute s) Solve the system of equations $\{y \cdot s = 0\}$ to obtain $s \in \{0,1\}^{n+1}$.
4. (Check if s is correct) Pick input $(a, b) \in \{0,1\} \times \{0,1\}^n$ uniformly at random, and check if Equation (6) holds. If yes, output FEISTEL.
5. Output RANDOM.

In summary, the algorithm essentially exploits the following observation: If U is a random permutation, it will have no structure. On the other hand, if U is a Feistel network, an appropriately defined f has the structure induced by Equation (6). As we know from the first half of this lecture, Simon's algorithm can uncover structure of the type of Equation (6), hence breaking the security of Feistel networks. (Again, this assumes one is allowed to query U in superposition, which is a non-trivial assumption.)

References

[KLLNP16] M. Kaplan, Gaëtan L., A. Leverrier, and M. Naya-Plascencia. Breaking symmetric cryptosystems using quantum period finding. In *Proceedings of Advances in Cryptology - CRYPTO 2016.*, volume 9815 of *Lecture Notes in Computer Science*, 2016.

[KM10] H. Kuwakado and M. Morii. Quantum distinguisher between the 3-round feistel cipher and the random permutation. In *2010 IEEE International Symposium on Information Theory*, pages 2682–2685, June 2010.

[SS17] T. Santoli and C. Schaffner. Using Simon's algorithm to attack symmetric-key cryptographic primitives. *Quantum Information & Computation*, 17(1&2):65–78, 2017.