CMSC 858L: Quantum Complexity

Instructor: Daniel Gottesman

Spring 2023

10 More oracle algorithms

10.1 References

Again, Simon's and Grover's algorithms are both standard quantum textbook material. Nielsen and Chuang, *Quantum Computation and Quantum Information*, has a chapter on Grover's algorithm and mentions Simon's algorithm as an example of the broader framework of abelian hidden subgroup algorithms. The hybrid argument showing a lower bound on Grover's algorithm and the separation of BQP from NP is from Bennett, Bernstein, Brassard, and Vazirani, "Strengths and Weaknesses of Quantum Computing," quant-ph/9701001 (often known as "BBBV"), although I have modified the argument slightly. (The BBBV paper also proved $\mathrm{BQP}^{\mathrm{BQP}} = \mathrm{BQP}$.)

10.2 Correction

To prove $\mathrm{BQP}^{\mathrm{BQP}} = \mathrm{BQP}$, we need to do a bit more than I said last time. When you call a quantum oracle, all you get is the answer, with no scratch bits with extra information lying around. This is important to make quantum oracle algorithms work, because otherwise the scratch bits might have information about the query that would effectively decohere the different branches of the computation and prevent interference later on. Thus, if you use an explicit BQP subroutine in place of an oracle call, you need to make sure that the subroutine leaves no scratch bits behind. In the case where the BQP subroutine gives the answer with amplitude 1, this is easy to ensure: Run the unitary U, use a CNOT to copy the answer onto an extra qubit (which will be the answer qubit), then run U^{-1} to reverse the original computation. When the amplitude of the answer is 1, copying it with a CNOT doesn't change the state, so U^{-1} brings us back to the initial state except for the answer qubit. If the probability of success is less than 1, this reversal might not work exactly, but if it is very close to 1, then the CNOT will only make a slight change to the state and reversing will give us something very close to the initial state. The subroutine will then function almost the same as the idealized oracle. Thus, there is a second reason to amplify the success probability of the subroutines in order to prove $\mathrm{BQP}^{\mathrm{BQP}} = \mathrm{BQP}$.

10.3 Simon's algorithm

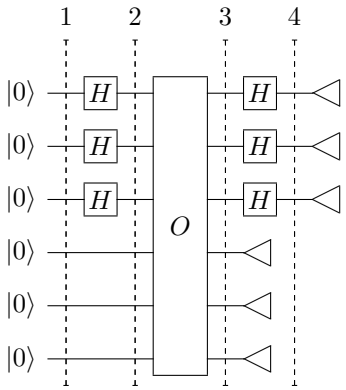
Simon's algorithm is historically important because it is the first exponential separation between BPP and BQP, and because it served as inspiration for Shor's algorithm. It is also an oracle problem, but it uses a functional oracle $O : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$. To make this quantum (or reversible), we therefore need n ancilla qubits: $O|x\rangle|c\rangle = |x\rangle|c \oplus O(x)\rangle$, with c being an n-bit string and the XOR being a bitwise sum.

The function in Simon's problem has the promise that for distinct x and y, $O(x) = O(y)$ iff $x \oplus y = s$ for some secret s. Our goal is to learn s. This is a property of O; it could be determined given a full list of values of O.

So what is the classical randomized query complexity of $s(O)$? We can find an upper bound through the following algorithm: Choose random x and query $O(x)$, keeping a list of all queries and outcomes. After each query, we check if the value $O(x)$ is equal to $O(y)$ for some y that we previously queried. If so, we halt

and return $s = x \oplus y$. Otherwise, we add $(x, O(x))$ to the list and keep going. Birthday paradox arguments show that we are likely to find a collision of two values $O(x) = O(y)$ after about $2^{n/2}$ queries. Similarly, since the values for unequal values of O are arbitrary, we learn little by having queries that don't match, and birthday paradox arguments show that we need $\Omega(2^{n/2})$ queries. Thus, $R(s) = \Theta(2^{n/2})$.

The quantum algorithm for Simon's problem looks superficially very similar to that for the Deutsch-Jozsa algorithm, but there are some crucial differences. The analysis is (unsurprisingly) more similar to Shor's algorithm.



At step 3, we have the state $\sum_x |x\rangle|O(x)\rangle$. The measurement of the ancilla qubits at this stage is not needed but it makes the analysis simpler: We get a value w and collapse to just two values x and y such that $O(x) = O(y) = w$, so the state (pre-Hadamards) is $(|x\rangle + |y\rangle)|w\rangle$.

Now,

$$H^{\otimes n}|x\rangle + H^{\otimes n}|y\rangle = \sum_z [(-1)^{x\cdot z} + (-1)^{y\cdot z}]|z\rangle \tag{1}$$

$$= \sum_z (-1)^{x\cdot z}[1 + (-1)^{(x\oplus y)\cdot z}]|z\rangle \tag{2}$$

$$= \sum_{z|s\cdot z=0} (-1)^{x\cdot z}|z\rangle \tag{3}$$

since $x \oplus y = s$. Thus, at step 4, we get a random z such that $s \cdot z = 0$. If we run this algorithm multiple times, we get a set of linear equations for s. After $O(n)$ trials, we are very likely to have n linearly independent such equations: we get another new equation whenever the new z is not in the linear span of the previous ones, which happens with probability at least $1/2$ if the existing z's don't already span the binary vector space. Once we have n independent equations, we can solve for s in polynomial time and not additional queries. This tells us that $Q(s) = O(n)$. In fact, $Q(s) = \Theta(n)$, although we don't yet have the techniques required to prove this.

Simon's algorithm shows an exponential separation between quantum and randomized classical query complexity. It can also be used to create an oracle O (albeit a functional one rather than decisional one) such that $\mathrm{BPP}^O \neq \mathrm{BQP}^O$.

10.4 Grover's algorithm

Grover's algorithm is an algorithm that performs unstructured search. This could be a database that is specially designed to allow retrieving superpositions of different entries, but it is more likely to be searching a function to find inputs that give outputs with a particular property (for instance, equal to a specific value). Grover's algorithm is phrased as solving the following oracle problem:

Definition 1. *The* unstructured search problem *is as follows; Given an oracle $O : \mathbb{Z}_N \to \mathbb{Z}_2$, find input x_0 such that $O(x_0) = 1$. Any such x_0 is known as a* marked *state.*

There are a number of equivalent variations: The oracle O could be structured in a number of different ways; for instance, it could have any range, not just a single bit, and there is some particular value we want to find. Sometimes we are given a promise that there is a unique solution or a known number t of solutions. Sometimes the only goal is to determine if $O(x) = 0 \forall x$ or if $\exists x_0$ such that $O(x_0) = 1$; this is equivalent to finding the OR of all the bits that make up the bit string representation of the oracle O. This last decision version of the problem is equivalent up to logarithms using binary search as before: search half the domain to determine which half the solution is and then repeat with the narrowed-down part of the domain.

Unstructured search can act as a model for NP-complete problems. Any algorithm that works for unstructured search will solve any problem in NP: Let $O(w)$ be the checking circuit applied to witness w for some particular instance x of a language L. Then determining whether or not there exists w such that $O(w) = 1$ is the same as determining if there is a witness for instance x, which in turn is the same as determining if $x \in L$. Of course, many NP problems will have a lot of structure in the checking circuit, so while an oracle algorithm will work to find a witness, it may be that there are other faster algorithms which take advantage of structure in the real problem that is not present in the oracle. However, the general belief is that the reason NP-complete problems are so hard is that they don't have any *usable* structure. That belief could be wrong, of course, but if so, that would suggest that for NP-complete problems, the best algorithms are derived from solutions to the unstructured search problem.

Grover's algorithm differs from the previous examples we have done in two aspects: First, it is only a polynomial speedup instead of an exponential speedup. We will need to develop techniques to prove a lower bound on the quantum query complexity to prove this. Second, the unstructured search problem is not a promise problem, whereas the previous examples were both promise problems. A lot of complexity theorists seem to consider this important, although I don't really understand the reasons why. One significance it has is that we know that there cannot be an exponential separation between classical and quantum query complexity for total functions (ones without a promise), at least for decision problems. Whereas in contrast, we have just seen an exponential separation between them when we do have a promise. However, it turns out that for sampling problems, Yamakawa and Zhandry have shown it is possible to find an exponential separation. (I don't think that we will cover their work in this class, unless someone wants to pick it for their project.)

In any case, Grover's algorithm provides a square root speedup. It works by repeatedly performing the following operation, starting on the state $\sum_x |x\rangle$:

$$G = H^{\otimes n} F_0 H^{\otimes n} (-1)^O. \tag{4}$$

Here, H is the Hadamard transform, F_0 is an operation that does a phase of -1 on the $|00\dots0\rangle$ state and $+1$ on all other basis states, and $(-1)^O$ is the version of the oracle that does a phase $-1^{O(x)}$ on a basis state $|x\rangle$. This phase oracle can be implemented using one standard oracle call by applying it with an ancilla qubit in the state $|-\rangle$, as for the Deutsch-Jozsa algorithm.

The Grover iteration rotates the state in the 2-dimensional subspace spanned by $\sum_{x|O(x)=1} |x\rangle$ and $\sum_{x|O(x)=0} |x\rangle$, the superpositions of marked and unmarked states. If there are t marked states and N possible inputs, each application of G rotates the state by an angle θ, with $\sin\theta = \sqrt{\frac{t}{N}}$. Therefore, after about $\pi/4\sqrt{N/t}$ iterations (each of which requires one oracle call), the state will be close to a superposition of marked states, and measuring produces a random marked state with high probability. The quantum query complexity is thus $Q = O(\sqrt{N/t})$, which for all $t > 0$ is $O(\sqrt{N})$.

When t is not known, there are various solutions, one of the most elegant being to note that Grover iterations produce a periodic oscillation with a period depending on t, and that therefore we can use Shor's period-finding algorithm to approximate t without dramatically more oracle calls. Once we have a rough approximation of t, we can measure after the appropriate amount of time; since we don't know t exactly, the probability of getting a marked element will be $O(1)$ but not necessarily extremely close to 1, but by trying a few times and checking the answers using the oracle, we can rapidly find a marked element. When $t = 0$, if we try this same procedure and fail to find a marked element, we can be fairly confident that there are no marked elements. Thus, the query complexity for the decision version is also $Q = O(\sqrt{N})$.

We can compare this with the classical randomized query complexity for this problem. The hardest case is likely to be cases where there is only 1 marked element. Let O_{x_0} be the oracle $O_{x_0}(x) = 0$ unless $x = x_0$ and $O_{x_0}(x_0) = 1$. We want to determine if the oracle is O_{x_0} for some x_0 or if it is O_0, defined by $O_0(x) = 0$ for all x. An arbitrary classical algorithm will make a sequence of queries $y_1, y_2, \ldots, y_r$. If the oracle is O_{x_0} and any $y_i = x_0$, then the algorithm has succeeded. If we fix the randomness used in the classical algorithm (which we can do by assuming it comes from a pre-determined list of random bits), then the sequence $y_1, y_2, \ldots, y_r$ doesn't depend on the actual oracle we have, only on the values $O(y_i)$. If the algorithm has not succeeded by query j, then $O(y_i) = 0$ for all $i \leq j$, which means that the sequence $y_1, \ldots, y_j$ is that same for all oracles such that $y_i \neq x_0$ for any $i < j$. Considering all such oracles, there are $N - j$ oracles O_{x_0} that are still possible as well as the oracle O_0. The next query y_{j+1} is the same for all these oracles, so the probability of succeeding with query $j + 1$ averaged over x_0 is at most $1/N - j$. The total probability of success with r queries is therefore at most

$$\sum_{j=1}^{r} 1/N - j \leq r/(N - r). \tag{5}$$

For the success probability to exceed $2/3$, we therefore need

$$r \geq 2/3(N - r), \tag{6}$$

or $r \geq 3/5N$. Since the classical query complexity $R = O(N)$ (we can try all inputs if needed), $R = \Theta(N)$. The quantum algorithm offers a quadratic improvement in the query complexity.