# CMSC 858L: Quantum Complexity
# Unit 3: Oracles and Query Complexity

Instructor: Daniel Gottesman

Fall 2026

## 1 Oracles

### 1.1 Generalities about oracles

An oracle is complexity-speak for a black box that computes a function but you have no idea what is going on inside. Oracles have two primary applications in quantum complexity. One of them is the study of oracles in their own right, through the study of *query complexity*, which is the number of times you need to query the oracle to determine some property of the oracle. Query complexity is a nice thing to study because it is possible to actually prove bounds on query complexity in many cases, letting us determine the actual complexity of many problems. A second application of oracles is to study regular complexity by looking at relationships between regular complexity classes which have access to oracles. Finally, the language of oracles can also be helpful in understanding other complexity-theoretic results. We'll discuss all three kinds of applications of oracles to some degree in this class, but we will start off with query complexity.

First, we need to note that there is a distinction between classical and quantum oracles.

**Definition 1.** *A* classical oracle $O(x)$ *takes a* query $x$ *and gives an output, often 1 bit. There is a corresponding* quantum oracle $O$ *which acts on a query state* $|\psi\rangle = \sum_x \alpha_x |x\rangle$ *and an ancilla qubit* $|c\rangle$ *and does a controlled-*$O$ *operation:*

$$O|\psi\rangle|c\rangle = \sum_x \alpha_x |x\rangle|c + O(x)\rangle. \tag{1}$$

A classical oracle is a function from $n$ bits to 1 bit (or more), whereas a quantum oracle is a unitary operation on $n+1$ qubits, but one that corresponds to the reversible version of the classical oracle applied to superpositions. Sometimes people also talk about more general versions of quantum oracles, *unitary oracles*, which are general black-box unitary operations not necessarily of the form above, so there is no corresponding classical oracle.

**Definition 2.** *We can think of a classical oracle and the corresponding quantum oracle as a long bit string, with the* $i$*th bit of the string being the value* $O(i)$*. Let* $f(O)$ *be some function of the oracle (thought of as a bit string). The* classical query complexity $D(f)$ *is the minimum number of classical oracle calls needed by a deterministic classical algorithm with access to the oracle* $O$ *to compute* $f(O)$*. The* randomized query complexity $R(f)$ *is the minimum number of classical oracle calls needed by a randomized classical algorithm with access to the oracle* $O$ *to compute* $f(O)$ *with probability at least* $2/3$*. The* quantum query complexity $Q(f)$ *is the minimum number of quantum oracle calls needed by a quantum algorithm with access to the oracle* $O$ *to compute* $f(O)$ *with probability at least* $2/3$*. In some cases, we have a* promise *on the oracle, which means the oracles are drawn from a subset of possible bit strings.*

The point here is that $f(O)$ is some property of the oracle that we wish to understand. For instance, you might want to know if $O(i)$ always has the same output or if it has different outputs depending on the

input $i$. The query complexities $D(f)$, $R(f)$, and $Q(f)$ in some sense capture the difficulty of computing the function $f(O)$, but note that the query complexity makes *no* restriction on the number of qubits or gates used other than oracle calls. In many cases, there are algorithms known for which the non-oracle resources used are comparable or less than the number of queries needed, but not always. In particular, for the non-abelian hidden subgroup problem (which includes graph isomorphism as a special case), the quantum query complexity is polynomial, but the only known algorithms still use exponentially many gates.

Nevertheless, in many cases, oracle algorithms can be translated into algorithms of similar efficiency when the black box is replaced by an explicitly computable function. Thus, study of query complexity can help us find algorithms. Lower bounds on query complexity can also give us insight into potential non-black-box algorithms, in that they tell us that if there is an algorithm that does better than the query complexity, it must use some structure in the explicit function that is not present or not accessible in the oracle.

A study of quantum query complexity generally involves 4 pieces: First, an *upper bound on the classical deterministic and/or randomized query complexity*, generally in the form of a classical algorithm to determine the function. Whether we are primarily interested in the deterministic or randomized complexity depends on the problem. Second, a *lower bound on the classical query complexity* showing that this algorithm is the best achievable. Third, an *upper bound on the quantum query complexity*, in the form of a quantum algorithm to compute the function. We are usually (but not always) interested in problems with a quantum speedup, in which case we want the quantum *upper* bound to be lower than the classical *lower* bound. Finally, we also would like a *lower bound on the quantum query complexity*, to show that the quantum algorithm is the best achievable.

In many cases, one or more of these will be trivial. However, if some are unknown, then that means we don't fully understand the problem. For instance, if a classical lower bound is known but the upper bound is missing or does not match, that may mean that the classical lower bound should actually be higher; whereas if the classical upper bound is known, but the lower bound is missing, that means there could potentially be a better classical algorithm, which in turn means we can't be sure that the problem does offer a quantum speedup. Not infrequently, though, we can actually prove (with no complexity assumptions) that all four of these bounds have specific values and with the upper and lower bounds matching.

Note that $Q(f) \leq R(f) \leq D(f)$ because any deterministic algorithm can be performed as a "randomized" algorithm that doesn't make use of any random bits, and any randomized algorithm can be simulated on a quantum system using $H|0\rangle = |+\rangle$ states as random coins. If there are $N$ possible query values (i.e., $x$ is a $\log_2 N$ bit number), then $D(f) \leq N$ since we can learn the whole oracle by making $N$ queries and then compute the function $f$ from those values.

## 1.2 Deutsch-Jozsa Algorithm

The first example of quantum query complexity is always the Deutsch-Jozsa problem. In this problem, we are promised that the oracle $O : \mathbb{Z}_2^n \to \mathbb{Z}_2$ is either *constant* (i.e., $O(x) = O(y)$ for all $x, y$) or *balanced* ($|\{x \text{ s.t. } O(x) = 0\}| = |\{x \text{ s.t. } O(x) = 1\}|$). Our goal is to determine which with 100% probability.

First we analyze the classical upper and lower bounds. We are going to look first at $D(f)$, the deterministic classical query complexity. Certainly there is an algorithm that uses $2^{n-1} + 1$ queries: Perform queries for that many values of $x$, say $x = 0, 1, 2, \ldots, 2^{n-1}$. If they are all the same, return "constant;" otherwise return "balanced." If the oracle is balanced, at most $2^{n-1}$ entries can can have the same value, so the algorithm will return the correct answer, and if it is constant, all entries will indeed be the same and therefore the algorithm is again correct.

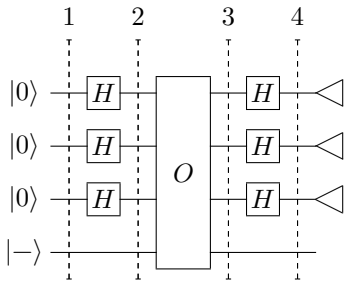
**Claim 1.** *To get the right answer, a deterministic classical algorithm needs at least $2^{n-1} + 1$ queries in the worst case.*

*Proof.* Suppose we have an algorithm that makes only $2^{n-1}$ queries. Let the output of the first query $x_0$ be $a$. Based on that, the algorithm decides to make a second query $x_1$, and suppose that the output of that query is also $a$. There are certainly oracles which have $O(x_0) = O(x_1) = a$, so this is a possibility. The algorithm continues deciding on queries $x_i$ for $i < 2^{n-1}$, and suppose that all of the queries return $a$.

There exists a constant oracle $O_1$ for which $O(x) = a$ for all $x$ that is consistent with all of these queries. There is also a balanced oracle $O_2$ consistent with all of these queries: $O(x_i) = a$ for $i = 0, \ldots, 2^{n-1} - 1$ and $O(y) = a \oplus 1$ for $y \neq x_i$ for any $i$. This algorithm must pick one of $O_1$ and $O_2$ and therefore will fail if presented with the other oracle. □

These two arguments together show that $D(f) = 2^{n-1} + 1$ for the Deutsch-Jozsa problem.

Now let us look at the quantum complexity. First, the upper bound: There is a quantum algorithm to solve this problem with just 1 quantum query:




The last qubit starts in the state $|-\rangle = |0\rangle - |1\rangle$. After the measurement, we do classical post-processing to determine the output: If the measurement result is all 0, output "constant". Otherwise, output "balanced".

To see that this works, let us follow the state of the system at each step:

1. $|00\ldots0\rangle|-\rangle$
2. $(\sum_x |x\rangle)|-\rangle$
3. $\sum_x (-1)^{O(x)}|x\rangle|-\rangle$

At this point, what happens next depends on if it is constant or balanced.

- **Constant:** The state pre-Hadamards is $\pm\sum_x |x\rangle|-\rangle$, so the Hadamards give $\pm|00\ldots0\rangle|-\rangle$.
- **Balanced:** Let $|\psi\rangle = \sum_x |x\rangle$ and $|\phi\rangle = \sum_x (-1)^{O(x)}|x\rangle$. Then $\langle\psi|\phi\rangle = \sum_x (-1)^{O(x)} = 0$ since there are as many $+1$ terms as $-1$ terms. But $H^{\otimes n}$ is unitary and $H^{\otimes n}|\psi\rangle = |00\ldots0\rangle$. Thus, $H^{\otimes n}|\phi\rangle$ is orthogonal to $|00\ldots0\rangle$, which means the measurement outcome will always have at least one 1 in it.

Therefore, the quantum query complexity $Q(f) \leq 1$. But we know that $Q(f) \neq 0$, since it is not possible to determine if the oracle is constant or balanced without looking at it at all, so we find $Q(f) = 1$.

So this is pretty good: We have a nice separation between $Q(f) = 1$ and $D(f) = 2^{n-1} + 1$. Unfortunately, the separation is not so impressive if we compare to $R(f)$, the classical randomized query complexity: If we choose random queries, if the function is constant, the answer will always be the same, but if the function is balanced, the odds are good we will rapidly get a different answer. So here is a randomized algorithm: Make $r$ queries to random values of $x$. If all answers are the same, return "constant." Otherwise, return "balanced." When the function is balanced, the probability that each additional query is different from the first one is $1/2$, and all are independent. (We can increase this slightly by making sure we don't repeat queries; then the probability drifts slightly above $1/2$ because we have more unused values that are different to the first query than ones that are the same.) Therefore, with $r$ queries, the chance that we see a different response is $1 - 2^{-(r-1)}$. To get the success probability above $2/3$, we only need $r = 3$ queries. Thus, $R(f) \leq 3$. We can't do it with 1 query (since any response we could get is consistent with either type of function), and with 2 queries, there is still at least a probability $2^{n-1}/(2^n - 1)$ that the two responses are the same even with a balanced function, in which case we cannot distinguish between constant and balanced. Thus, $R(f) = 3$. This means that there is only. a constant separation between $Q(f)$ and $R(f)$.

## 1.3 Complexity classes relative to an oracle

Nevertheless, this result lets us say something meaningful about complexity classes. We do this by switching to the question of oracle separations, so let us first discuss what this means.

**Definition 3.** *A language $O$ can be interpreted as an oracle, or really a famiy of oracles, which outputs $O(x) = 1$ if $x \in O$ and $O(x) = 0$ if $x \notin O$. We can define a classical or quantum circuit or Turing machine (deterministic, probabilistic, or non-deterministic) with oracle access to $O$ as one which has the standard capabilities plus the extra capability to make oracle queries to $O$ as a single computational step. A complexity class $C$* relative to an oracle *$O$ is written $C^O$ and indicates a class defined as usual but using circuits or machines with oracle access to $O$.*

Thus, $\mathrm{P}^O$ consists of problems solved by uniform polynomial size circuits that can query $O$ and get an answer in 1 step, and $\mathrm{BQP}^O$ consists of problems solved with bounded error by uniform polynomial size quantum circuits that can query the quantum oracle $O$ and get an answer in 1 step. Essentially we can think of classes relative to oracle $O$ as living in some parallel universe in which there is some fundamental gate that solves $O$.

For instance, imagine that some extinct alien civilization left behind lots of black boxes with advanced technology that we don't understand, but we do know how to send information $x$ into them and get a single bit output $O(x)$. What can we do with computers that incorporate these alien black boxes? If we are using deterministic classical computers, we get $\mathrm{P}^O$; if quantum computers, we get $\mathrm{BQP}^O$.

The oracle $O$ can in principle be anything. If it is something weak, then working relative to $O$ may not change anything. For instance, if $O \in \mathrm{P}$, then $\mathrm{P}^O = \mathrm{P}$. Why? Because $O \in \mathrm{P}$, there exists an algorithm $A$ with running time $f(|x|)$, $f = O(\mathrm{poly}(|x|))$, that decides $O$. If we have a circuit of size $g(|x|)$ using $O$, we can substitute $A$ for $O$ and get the same behavior. The size of the circuit has increased, since the 1-step $O$ is replaced by the $f(|x|)$ steps for $A$, but the total size of the modified circuit is at most $f(|x|)g(|x|)$, and if $g(|x|) = O(\mathrm{poly}(|x|))$, then so is this product. Therefore, any language decided in polynomial time by a circuit with oracle access to $O$ can also be decided in polynomial time by a circuit with no access to $O$. This result is usually written as $\mathrm{P}^{\mathrm{P}} = \mathrm{P}$ since it holds for any language in P.

**Theorem 1.** *$BQP^{BQP} = BQP$*

*Proof.* We can apply a similar argument, but there are a few things we need to be careful about. First, the modified algorithm can potentially fail both because the original oracle algorithm can fail (with probability at most 1/3) but because each of the algorithms subsituting for $O$ can fail as well. We can solve this problem by amplifying the success probability of all the subroutines through repetition to a sufficiently high value that the overall success probability of the modified algorithm remains at least 2/3.

Second, 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 $\mathrm{BQP}^{\mathrm{BQP}} = \mathrm{BQP}$.

□

We can also have crazy, powerful oracles that can't exist in reality, like an oracle for the halting problem. That's OK; oracles are not necessarily supposed to represent the real world. They just provide us an avenue

for studying the relationships between complexity classes. Many complexity-theoretic techniques *relativize*, meaning whatever relationship they prove between complexity classes still holds when both classes are taken relative to some oracle. As a very simple example, we know that $\mathrm{P} \subseteq \mathrm{BQP}$ because any classical circuit can be implemented as a quantum circuit. This same argument shows that $\mathrm{P}^O \subseteq \mathrm{BQP}^O$.

One advantage of thinking about classes relative to oracles is that not infrequently, we can find an oracle $O$ such that we can *prove* a separation between complexity classes relative to $O$ with no additional assumptions. In other cases, we can find an oracle $O'$ such that we can prove that two complexity classes are equal relative to $O'$. In fact, it is fairly common that we can do both: prove that the classes are different relative to some $O$ and that they are the same relative to $O'$. For instance, there exist oracles $O$ and $O'$ such that $\mathrm{P}^O \neq \mathrm{NP}^O$ and $\mathrm{P}^{O'} = \mathrm{NP}^{O'}$. This might make it seem this was all a futile exercise, but we have actually learned something important: If we want to prove a definitive relationship between these two classes, it will need to use a technique that does not relativize. It's just that …we don't know too many such techniques. But we do know some, and many of the most famous complexity theory results (including ones we'll discuss in this class) don't relativize.

With those caveats in mind, let's see how the Deutsch-Jozsa algorithm implies an oracle separation between P and BQP. Consider generating a language $O$ (which will be our oracle) using the following procedure: For each value of $n$, choose a random bit $b(n)$. If $b(n) = 0$, then choose a random constant function $f(x)$ for $|x| = n$; if $b(n) = 1$, choose a random balanced function $f(x)$ for $|x| = n$. Then $O$ includes those $x$ with $|x| = n$ and $f(x) = 1$. Now consider the language $B$ which consists of $1^n$ for any $n$ such that $b(n) = 1$.

**Claim 2.** *$B \notin P^O$ but $B \in BQP^O$.*

*Proof.* Given instance $1^n$ of $B$, to decide it, a circuit (classical or quantum) needs to determine if $O$ is constant or balanced on inputs of size $n$. A call to $O$ does exactly what an oracle query does in the Deutsch-Jozsa problem. A quantum circuit can do it with one call to $O$ and a linear number of standard quantum gates. A classical circuit requires at least $2^{n-1} + 1$ calls to $O$, and in particular, no polynomial size classical deterministic circuit can decide $B$. □

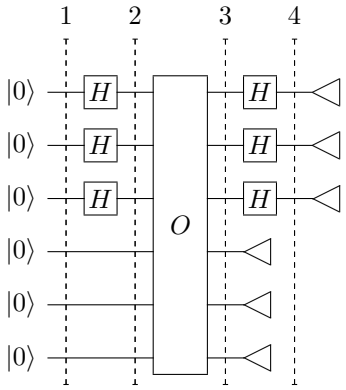
## 1.4 Simon's algorithm

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,

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

$$= \sum_z (-1)^{x\cdot z}[1 + (-1)^{(x\oplus y)\cdot z}]|z\rangle \tag{3}$$

$$= \sum_{z|s\cdot z=0} (-1)^{x\cdot z}|z\rangle \tag{4}$$

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 $\mathrm{BPP}^O \neq \mathrm{BQP}^O$.

## 1.5 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 4.** *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 in 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 it is a correct belief, 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 without a promise.

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

$$G = H^{\otimes n} F_0 H^{\otimes n} (-1)^O. \tag{5}$$

Here, $H$ is the Hadamard transform, $F_0$ is an operation that does a phase of $-1$ on the $|00\ldots0\rangle$ 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 $\theta$, 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

$$\sum_{j=1}^{r} 1/N - j \leq r/(N-r). \tag{6}$$

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

$$r \geq 2/3(N-r), \tag{7}$$

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.

# 2 Quantum lower bounds

We now embark on studying methods of lower bounding the quantum query complexity. We'll apply each method to unstructured search, but we will look at some other applications as well.

## 2.1 Hybrid argument

But so far we have just an upper bound on the quantum query complexity. Can we set a lower bound on the quantum query complexity? Yes, in fact, we can, but it requires some work. The first argument I will show you is known as a *hybrid* argument, but then we will cover some other techniques for proving quantum lower bounds and we'll try them out on this same problem.

**Theorem 2.** *The quantum query complexity for unstructured search is $Q = \Omega(\sqrt{N})$, and thus in fact $Q = \Theta(\sqrt{N})$.*

*Proof.* Suppose we have an algorithm that makes $T$ queries. That algorithm can be written as an alternating sequence of unitaries and oracle calls $U_0, O, U_1, O, \ldots, U_{T-1}, O, U_T$. The unitaries don't have to all be the same and they can be many-qubit very complex unitaries — we only care about the query complexity, not the circuit complexity. Importantly, the sequence $U_0, U_1, \ldots, U_T$ of unitaries is the same regardless of the oracle used, since the algorithm knows nothing about the oracle beyond what it learns in oracle calls, and its reaction to those oracle calls is subsumed into the unitaries $U_i$.

Let

$$V_0 = U_T \prod_{i=T-1}^{0} O_0 U_i \tag{8}$$

and

$$V_{x_0} = U_T \prod_{i=T-1}^{0} O_{x_0} U_i. \tag{9}$$

We are using the same notation for oracles as before: $O_0$ is the oracle with no marked elements and $O_{x_0}$ is the oracle with only $x_0$ marked. We wish to show that $V_0|\psi\rangle$ is close to $V_{x_0}|\psi\rangle$ for any $|\psi\rangle$ unless $T = \Omega(\sqrt{N})$.

To do this, we will proceed through a sequence of hybrid circuits (thus the name of the method)

$$V_{x_0}^{(j)} = U_T \prod_{i=T-1}^{j} O_{x_0} U_i \prod_{i=j-1}^{0} O_0 U_i. \tag{10}$$

Thus $V_{x_0}^{(0)} = V_{x_0}$ and $V_{x_0}^{(T)} = V_0$. Each successive $V_{x_0}^{(j)}$ has one less call to $O_{x_0}$ and one more call to $O_0$ instead. We will actually show that $V_{x_0}^{(j)}$ and $V_{x_0}^{(j+1)}$ are very close for almost all $x_0$, and then the total distance between $V_0$ and $V_{x_0}$ can be at most $T$ times the maximum distance between adjacent $V_{x_0}^{(j)}$'s.

Since the only difference between $V_{x_0}^{(j)}$ and $V_{x_0}^{(j+1)}$ is a single oracle call, we can simplify the comparison into one between $V_{x_0}^f O_0 V^i$ and $V_{x_0}^f O_{x_0} V^i$, where $V^i$ is the product of unitaries (including oracles) performed before the $(j+1)$th oracle call and $V_{x_0}^f$ is the product of unitaries (including oracles) performed after the $(j+1)$th oracle call. Note in particular that while $V_{x_0}^f$ can depend on $x_0$, $V^i$ does not. Let $|\psi\rangle$ be the initial state of the algorithm and let

$$|\phi\rangle = V^i|\psi\rangle = \sum_x \alpha_x |x\rangle \otimes |\phi_x\rangle. \tag{11}$$

Here the first register is the query that will be given to the oracle at step $j+1$ and we can assume that the states $|\phi_x\rangle$ are normalized.

We can simplify this by noting that by permuting the inputs to the oracle, we can exchange any $O_{x_0}$ with any other $O_{x_0'}$. In particular, the algorithm has no reason to prefer any $x_0$'s over any others. Indeed, any algorithm can be symmetrized by choosing random permutation of inputs and applying it everywhere without altering the average success probability, averaged over $x_0$. We can purify this procedure (so that we can use unitaries everywhere), and the conclusion is that since $|\phi\rangle$ doesn't depend at all on the actual $x_0$ in the oracle, the amplitudes $|\alpha_x|$ must have the same magnitude $1/\sqrt{N}$. Then

$$O_0|\phi\rangle = \sum_x \alpha_x |x\rangle \otimes |\phi_x\rangle \tag{12}$$

$$O_{x_0}|\phi\rangle = \sum_{x \neq x_0} \alpha_x |x\rangle \otimes |\phi_x\rangle - \alpha_{x_0}|x_0\rangle \otimes |\phi_{x_0}\rangle. \tag{13}$$

(Assume we use the oracle in the phase form, but the standard form doesn't materially alter this argument.) Thus,

$$\langle\phi|O_0^\dagger O_{x_0}|\phi\rangle = \sum_{x \neq x_0} |\alpha_x|^2 - |\alpha_{x_0}|^2 = (N-2)/N. \tag{14}$$

This is indeed the lowest fidelity we can achieve between these two states for any ancilla state for the oracle output. Because the operation $V_{x_0}^f$ is unitary, even though it depends on the marked value $x_0$, the fidelity between the two possible final states for $V_{x_0}^{(j)}$ and $V_{x_0}^{(j+1)}$ has to remain the same as the fidelity between $O_0|\phi\rangle$ and $O_{x_0}|\phi\rangle$, that is to say at least $1 - 2/N$.

The next step is to add up all these distances to get the distance between the $V_{x_0}^{(j)}$'s in order to find the distance between $V_0$ and $V_{x_0}$. However, infidelity is not a metric and doesn't need to satisfy the triangle inequality, so we will switch to trace distance. The trace distance is at most

$$\sqrt{1-F^2} \leq \sqrt{1-(1-2/N)^2} = \sqrt{4/N - 4/N^2} \leq 2/\sqrt{N}. \tag{15}$$

The trace distance does satisfy the triangle inequality, so

$$D(V_0|\psi\rangle, V_{x_0}|\psi\rangle) \leq \sum_{j=0}^{T-1} D(V_{x_0}^{(j+1)}|\psi\rangle, V_{x_0}^{(j)}|\psi\rangle) \leq 2T/\sqrt{N}. \tag{16}$$

(Trace distance is actually between density matrices, so consider the above pure states as shorthand for the corresponding density matrices.) Thus, if $T = o(\sqrt{N})$, the outcome distributions of the two states must be very close for large $N$.

Because of the symmetrization, we have made all the trace distances $D(V_0|\psi\rangle, V_{x_0}|\psi\rangle)$ the same, but before symmetrization, some could be higher than that while others would be lower. However, by the Markov inequality, the number of $x_0$ such that $D(V_0|\psi\rangle, V_{x_0}|\psi\rangle) > 2kT/\sqrt{N}$ is at most $N/k$. In particular, when $T = o(\sqrt{N})$, the fraction of values of $x_0$ for which the trace distance is not small is itself vanishing in the limit of large $N$.

Let $P_b(O)$ be the probability that this algorithm outputs $b$ to the decision problem when given oracle $O$. Since for almost all $x_0$, $D(V_0|\psi\rangle, V_{x_0}|\psi\rangle)$ is small when $T = o(\sqrt{N})$, $P_b(O_0)$ must be close $P_b(O_{x_0})$ and therefore, the algorithm fails to be correct with probability 2/3. In order to achieve this level of accuracy, we must have $T = \Omega(\sqrt{N})$.

□

## 2.2 Separating BQP from NP relative to an oracle

We then get

**Theorem 3.** *Relative to a random oracle $O$, $NP^O \not\subseteq BQP^O$ with probability* 1.

This gives some evidence, for what it's worth, that BQP can't solve NP-complete problems.

*Proof.* Given $O$, we can define another oracle $\tilde{O} : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ (for all $n$) as follows: the $i$th bit of $\tilde{O}(x)$ is $O(x, i)$, where the input $(x, i)$ is the concatenation of $x$ and $i$, where $i$ is written using exactly $\lceil \log_2 n \rceil$ bits. $\tilde{O}$ can be computed using $O(\log n)$ calls to $O$ and a particular value of $O$ for a given input of size $n + \lceil \log_2 n \rceil$ can be computed using one call to $\tilde{O}$. Thus, querying $O$ and querying $\tilde{O}$ are essentially equivalent up to possible $O(n)$ factors. Furthermore, every function $\tilde{O}$ can be defined in this way from an $O$, and all are equally likely for random $O$, so $\tilde{O}$ is a random functional oracle.

Define the language $L$ as follows: $y \in L$ iff $\exists\, x$ such that $\tilde{O}(x) = y$. This language can be decided in $\mathrm{NP}^O$ using just one oracle call to $\tilde{O}$ and therefore just $n$ to $O$.

However, it cannot be decided in $\mathrm{BQP}^O$: Any algorithm to decide it using $T$ queries to $O$ can be coverted to an algorithm using at most $T$ queries to $\tilde{O}$. Queries to $\tilde{O}$ are not going to be any more effective for this problem than queries to a decision version which returns 0 if $\tilde{O} \neq y$ and returns 1 if $\tilde{O} = y$. Since $\tilde{O}$ is random, for most values of $y$, there is at most one $x_0$ such that $\tilde{O}(x_0) = y$, and we know from the previous proof that for any quantum query algorithm using $o(\sqrt{N})$ queries, for almost all $x_0$, the algorithm does not succeed with probability at least 2/3. That is, for most values of $\tilde{O}$ for this particular value of $n$, the algorithm fails. By letting $n \to \infty$, we get that $L \notin \mathrm{BQP}^O$ with probability 1.

□

## 2.3 Bounds from polynomials

Another approach to setting lower bounds to query complexity is to approximate the functions we need to compute by polynomials. In fact, if we think of the oracle as an $N$-bit string with $i$th bit $X_i$, we can think of the $X_i$'s as variables that take on the value 0 or 1 as we vary the oracle. Then the function $f(O)$ we are supposed to compute in the query complexity problem is a polynomial in the $X_i$'s of degree at most $N$: Any function on a finite number of points (in this case $2^N$ points) can be written as a polynomial. A priori, this polynomial would have very high degree, as high as $2^N - 1$. However, $f$ is only evaluated for $X_i = 0$ or $X_i = 1$, so any factor $X_i^a = X_i$ for $a > 1$. Thus, the highest degree term possible is $\prod_i X_i$, which is degree $N$.

The function $f$ itself is only defined at points where $X_i = 0, 1$, but the polynomial can be sensibly interpolated to arbitrary real values for $X_i$ between 0 and 1. There might be a much lower degree polynomial than the obvious one that exactly matches $f$ or gets close to it at all points where all $X_i = 0, 1$ but behaves very differently elsewhere.

**Definition 5.** *The* degree $\deg f$ *of a function $f$ on $N$ variables is the smallest value $d$ such that there exists polynomial $p(X_0, \ldots, X_{N-1})$ of degree $d$ (with real coefficients) with $f(x_0, \ldots, x_{N-1}) = p(x_0, \ldots, x_{N-1})$ for all $(x_0, \ldots, x_{N-1}) \in \{0, 1\}^N$. The* approximate degree $\widetilde{\deg} f$ *of $f$ is the smallest degree $d$ such that there exists polynomial $p$ of degree $d$ (with real coefficients) with*

$$|f(x_0, \ldots, x_{N-1}) - p(x_0, \ldots, x_{N-1})| \leq 1/3 \tag{17}$$

*for all $(x_0, \ldots, x_{N-1}) \in \{0, 1\}^N$.*

It turns out that the approximate degree will give us a lower bound on the quantum query complexity:

**Theorem 4.** $Q(f) \geq (\widetilde{\deg} f)/2$

*Proof.* The main observation is that if we have a quantum query algorithm with $T$ queries, the amplitudes are given by polynomials of degree at most $T$. To prove this claim, consider again the algorithm as an alternating sequence of unitaries and oracle calls. Before the first oracle call, the state is $\sum_{i,j} \alpha_{i,j}|i\rangle|j\rangle$, with the first register being the one used to query the oracle. After the first oracle call, the state is now

$$\sum_{i,j}(-1)^{O(i)}\alpha_{i,j}|i\rangle|j\rangle = \sum_{i,j}(1-2X_i)\alpha_{i,j}|i\rangle|j\rangle. \tag{18}$$

Here we are substituting $O(i) = X_i$ and using the fact that the oracle takes values only at 0 and 1. At this point, therefore, the amplitudes are linear functions of the variables $X_i$ as the oracle changes.

By the same argument, if just before the $j$th oracle call, the amplitudes are degree $j-1$ polynomials of the $X_i$ variables, then after the $j$th oracle call, the amplitudes are degree $j$ polynomials. We therefore can prove the claim that the final amplitudes are degree $T$ polynomials by induction, provided that the unitaries between the oracle calls don't mess up this property. That is easy to check: If before $U$, the state is $\sum_a q_a(X_0,\ldots,X_{N-1})|a\rangle$, with $q_a$ polynomials of degree $j$, then after $U$, the state is

$$\sum_b \left[\sum_a q_a(X_0,\ldots,X_{N-1})U_{ba}\right]|b\rangle, \tag{19}$$

and $\sum_a q_a(X_0,\ldots,X_{N-1})U_{ba}$ is also a polynomial of degree $j$.

So now we know the final state has the form $\sum_a q_a(X_0,\ldots,X_{N-1})|a\rangle$, where the $q_a$'s are polynomials of degree at most $T$. Note, however, that $q_a$ could have complex coefficients even though for $\widetilde{d}(f)$ we care about polynomials over the real numbers; but this will get taken care of soon.

At the final stage, we make a measurement of the first qubit to determine the outcome of the algorithm. The probability that the outcome is 1 is

$$\text{Prob}(1) = \sum_{a=1a'} |q_{1a'}(X_0,\ldots,X_{N-1})|^2. \tag{20}$$

When the $q_a$'s are complex polynomials of degree at most $T$, Prob(1) is therefore a real polynomial $p(X_0,\ldots,X_{N-1})$ of degree at most $2T$.

Now, since the algorithm decides $f$ correctly, that means that $\text{Prob}(1) \geq 2/3$ if $f(O) = 1$ and $\text{Prob}(1) \leq 1/3$ if $f(O) = 0$. In particular, in either case,

$$|f(O) - p(X_0,\ldots,X_{N-1})| \leq 1/3. \tag{21}$$

The algorithm must work for all oracles, so this equation holds for all $(X_0,\ldots,X_{N-1}) \in \{0,1\}^N$.

Since $\widetilde{\deg}(f)$ is the smallest degree of a polynomial satisfying this condition, and $p$ is a polynomial of degree $2T$ satisfying the condition, it follows that $\widetilde{\deg}(f) \leq 2T$. Thus, $Q(f) \geq \widetilde{\deg}(f)/2$. □

## 2.4 Examples of polynomial bounds

Let us look at a few example functions and how to bound their query complexity with polynomials. The first example is the unstructured search problem, where you are supposed to compute the OR of all the bits in the bit string to determine if there is a marked element or not. The OR function is the polynomial $p(X_0,\ldots,X_{N-1}) = 1-\prod_{i=0}^{N-1}(1-X_i)$. This is a degree $N$ polynomial. However, note that Grover's algorithm tells us that there is a degree $O(\sqrt{N})$ polynomial that approximates $p$. But is this the best we can do?

Another example is the PARITY function, which is 0 if there are an even number of 1s in the input and 1 if there are an odd number of 1s. That is, given an oracle $O$, are there an even or odd number of inputs that give output 1? Let us figure out a polynomial for PARITY: Let $p_n$ be the polynomial for $n$ variables. Then

$$p_n = X_{n-1}(1-p_{n-1}) + (1-X_{n-1})p_{n-1} = X_{n-1} + (1-2X_{n-1})p_{n-1}. \tag{22}$$

We get

$$p_n = \sum_{i=0}^{n-1} X_i \prod_{j=i+1}^{n-1} (1 - 2X_j) = \sum_{s=1}^{n} \sum_{i_0<i_1<\ldots<i_s} (-2)^{s-1} \prod_{j=1}^{s} X_{i_j}. \tag{23}$$

With $N$ variables, this is a degree $N$ polynomial. But is there a lower degree polynomial that provides a good approximation?

It is not necessarily straightforward to find the approximate degree. Luckily, a lot is known about polynomials and approximating functions by polynomials, since a similar technique also works to set bounds on classical query complexities. One important strategy is to simplify to polynomials of a single variable. One way to do this is to symmetrize by averaging over permutations of the input bits. This gives us a polynomial of the same (or lower) degree that is a function only of the Hamming weight (number of 1s) of the input, since any input with a given Hamming weight can be permutated to any other input of the same Hamming weight, so those two inputs have the same value for the symmetrized polynomial.

If the function $f$ is already symmetric, the polynomial can be directly written in terms of the Hamming weight. In this case, let $f_k$ be the $f(X_0, \ldots, X_{n-1})$ for any $(X_0, \ldots, X_{N-1})$ of Hamming weight $k$ (all give the same value for $f_k$), and let

$$\Gamma(f) = \min\{|2k - N + 1| \text{ such that } f_k \neq f_{k+1}\}. \tag{24}$$

The classical study of these polynomials produced the following result:

**Theorem 5.** *If $f$ is a non-constant symmetric Boolean function on $\{0,1\}^N$, then $\widetilde{\deg}(f) = \Theta(\sqrt{N(N - \Gamma(f))})$.*

We can immediately apply this to give a lower bound on unstructured search: The function $f$ is OR, and is symmetric, so the theorem applies. $f$ is 0 for the all-0 input and 1 otherwise; that is, $f_0 = 0$, $f_i = 1$ for $i > 0$. Thus, $\Gamma(f) = N - 1$ and the theorem implies $\widetilde{\deg}(f) = \Theta(\sqrt{N})$. Theorem 4 then implies that the quantum query complexity is $\Omega(\sqrt{N})$.

The method of polynomials is quite powerful. As another example: the PARITY function, which is also symmetric. It has $f_k = 0$ for even $k$ and $f_k = 1$ for odd $k$. Thus adjacent $f_k$ are always unequal, and the minimum value of $|2k - N + 1|$ is when $k = \lfloor N/2 \rfloor$, and $\Gamma(f)$ is 0 or 1, depending on if $N$ is odd or even. In any case, the theorem then implies that $\widetilde{\deg}(f) = \Theta(N)$ and so we have $Q(\text{PARITY}) = \Theta(N)$ (since $N$ is also an upper bound. This is thus a problem for which quantum computing offers no speedup at all (or rather at most a constant factor speedup; in fact, it offers a factor of 2 speedup). The polynomial method can also be applied to problems with promises or which are not symmetric, but it requires more work and more results about properties of polynomials.

Another application is to limit the possible quantum speedup. The relationship between classical query complexity and approximate degree is not as straightforward as the relationship with quantum query complexity. However, using some more careful analysis of the structure of polynomials and some additional techniques, one can show the following:

**Theorem 6.** *For any total function $f$ (i.e., one with no promise on the input oracle), $D(f) = O(Q(f)^4)$. In addition, there exists an $f$ such that $D(f) = \tilde{\Omega}(Q(f)^4)$. (The tilde means "up to log factors".)*

Thus, without a promise, any decision problem can be sped up by at most the 4th power by a quantum algorithm. Up to logarithmic factors, this is actually achievable — there are quartic speedups.

Note that the polynomial method doesn't always give a tight bound. There are problems for which there is a lower degree approximate polynomial than the actual query complexity.

## 2.5 Adversary method

Another method to bound query complexity is known as the *adversary method* or *quantum adversary method*. In this technique, along with the quantum computer running the query algorithm, there is a second quantum

register which governs which oracle we have. This quantum register is in a state which represents a superposition of all possible oracles. In the initial state, the quantum computer has no information about the oracle and so the two registers start out in a tensor product. The oracle calls create entanglement between the two registers, representing the quantum computer learning some information about the oracle. In a successful algorithm, the final state must have some degree of entanglement, and by bounding the amount of entanglement generated by each oracle call, we can set a lower bound on the number of oracle calls needed.

Let us apply this technique to unstructured search. We will restrict attention to oracles that have exactly one marked element; a lower bound on the number of queries needed to solve this problem will also provide a lower bound on the harder problem where the number of marked elements is unknown. In the initial state, the computer is in some standard state, such as $|\psi_0\rangle = |00\ldots0\rangle \otimes \sum_i |O_i\rangle$, where $|O_i\rangle$ indicates the oracle with element $i$ marked. In particular, the second ("oracle") register is in a pure state $\rho_0 = \sum_{ij} \rho_{0,ij}|i\rangle\langle j|$, $\rho_{ij} = 1/N$ when normalized properly.

The final state if the algorithm succeeds with 100% chance is $|\psi_{T-1}\rangle = \sum_i |i\rangle|\alpha_i\rangle|O_i\rangle$ (with some scratch space left in the state $|\alpha_i\rangle$, since the computer has learned the value of the marked element in the oracle. The state is now maximally entangled and the density matrix of the oracle register is now the maximally mixed state: $\rho_{T-1} = \sum_{ij} \rho_{T-1,ij}|i\rangle\langle j|$, $\rho_{T-1,ij} = \delta_{ij}/N$.

If there is not 100% chance of success, the final state can be written as

$$|\psi_{T-1}\rangle = \frac{1}{\sqrt{N}} \sum_i |\xi_i\rangle \otimes |O_i\rangle \tag{25}$$

with proper normalization. (The algorithm doesn't change the oracle register except for a phase during an oracle call, so the absolute value of the amplitude of each oracle option must remain $1/\sqrt{N}$.) When the algorithm succeeds with probability $1 - \epsilon > 1/2$, then for all $i$,

$$|\langle i|\xi_i\rangle| \geq \sqrt{1-\epsilon}. \tag{26}$$

We will need some information about the final density matrix of the oracle register, so let us compute it:

$$\rho_{T-1} = \frac{1}{N} \sum_i |O_i\rangle\langle O_i| + \frac{1}{N} \sum_{i \neq j} \langle \xi_j|\xi_i\rangle |O_i\rangle\langle O_j|. \tag{27}$$

We are particularly interested in the off-diagonal terms, so let us focus on $\rho_{T-1,ij} = \frac{1}{N}\langle \xi_j|\xi_i\rangle$. Let

$$|\xi_i\rangle = a|i\rangle|\omega_{ii}\rangle + b|j\rangle|\omega_{ij}\rangle + c|\eta\rangle \tag{28}$$

$$|\xi_j\rangle = a'|j\rangle|\omega_{jj}\rangle + b'|i\rangle|\omega_{ji}\rangle + c'|\eta'\rangle \tag{29}$$

with $|a|^2 \geq 1-\epsilon$, $|a'|^2 \geq 1-\epsilon$, $|b|^2 + |c|^2 \leq \epsilon$, $|b'|^2 + |c'|^2 \leq \epsilon$, and $|\eta\rangle$ and $|\eta'\rangle$ both orthogonal to both $|i\rangle$ and $|j\rangle$. Then

$$\langle \xi_j|\xi_i\rangle = ab'\langle \omega_{ji}|\omega_{ii}\rangle + a'b\langle \omega jj|\omega_{ij}\rangle + cc'\langle \eta'|\eta\rangle \tag{30}$$

$$|\langle \xi_j|\xi_i\rangle| \leq |ab'| + |a'b| + |cc'| \tag{31}$$

$$\tag{32}$$

If we fix $a$, $a'$, $b$, and $c$, we can maximize $|ab'| + |a'b| + |cc'|$ by letting the vector $(b', c')$ be parallel to $(a, c)$, so $c'/b' = c/a$. Then if $|b'|^2 + |c'|^2 = \epsilon_j$, $|b'|^2(1 + |c|^2/|a|^2) = \epsilon_j$ and $|a'| = \sqrt{1-\epsilon_j}$, so

$$|ab'| + |a'b| + |cc'| = |ab'| + |a'b| + |c^2b'/a| \tag{33}$$

$$= (|a| + |c^2/a|)\sqrt{\epsilon_j}|a|/\sqrt{|a|^2 + |c|^2} + |b|\sqrt{1-\epsilon_j} \tag{34}$$

$$= \sqrt{|a|^2 + |c|^2}\sqrt{\epsilon_j} + |b|\sqrt{1-\epsilon_j} \tag{35}$$

$$= \sqrt{1-|b|^2}\sqrt{\epsilon_j} + |b|\sqrt{1-\epsilon_j}. \tag{36}$$

Since $\epsilon_j$ is small, this is maximized by letting $|b|$ be its maximum value $\sqrt{\epsilon}$. This means the maximum occurs when $c = 0$, which also means $c' = 0$. Thus,

$$\langle \xi_j | \xi_i \rangle \leq 2\sqrt{\epsilon(1-\epsilon)}. \tag{37}$$

We next must compute how much the entanglement between the two registers can change with a single oracle call. Since the off-diagonal terms start initially at $1/N$ and end at $2\sqrt{\epsilon(1-\epsilon)}/N$ (at most), we will how much those off-diagonal terms decrease with each call and use this to quantify the entanglement at each stage. In particular, let

$$S = \sum_{i \neq j} |\rho_{ij}|, \tag{38}$$

the sum of the absolute values of the $N(N-1)$ off-diagonal terms. For the initial state, $S_0 = N - 1$. In the final state,

$$S_{T-1} \leq \frac{2N(N-1)}{N}\sqrt{\epsilon(1-\epsilon)} = 2(N-1)\sqrt{\epsilon(1-\epsilon)}. \tag{39}$$

Since $\epsilon < 1/2$, $2\sqrt{\epsilon(1-\epsilon)} < 1$ and $S_0 - S_{T-1} = c(N-1)$ for a constant $c = 1 - 2\sqrt{\epsilon(1-\epsilon)}$.

Let $|\psi^-_{t-1}\rangle = \sum_{ij} \alpha_{ij} |i\rangle|\phi_{ij}\rangle|O_j\rangle$ be the state of the two registers just before the $t$-th oracle call. Then the state after the $t$-th oracle call is

$$|\psi^+_{t-1}\rangle = \sum_{i \neq j} \alpha_{ij} |i\rangle|\phi_{ij}\rangle|O_j\rangle - \sum_i \alpha_{ii} |i\rangle|\phi_{ii}\rangle|O_i\rangle. \tag{40}$$

Let us calculate the corresponding density matrices for the oracle register:

$$\rho^-_{t-1} = \mathrm{Tr}_{\text{computer}} \sum_{ij} \sum_{kk'} \alpha_{ki} \alpha^*_{k'j} |k\rangle|\phi_{ki}\rangle|O_i\rangle\langle k'|\langle\phi_{k'j}|\langle O_j| \tag{41}$$

$$= \sum_{ij} \sum_{k} \alpha_{ki} \alpha^*_{kj} \langle \phi_{kj}|\phi_{ki}\rangle |O_i\rangle\langle O_j| \tag{42}$$

$$\rho^+_{t-1} = \mathrm{Tr}_{\text{computer}} \sum_{i \neq j} \left[ \sum_{k \neq i, k' \neq j} \alpha_{ki} \alpha^*_{k'j} |k\rangle|\phi_{ki}\rangle\langle k'|\langle\phi_{k'j}| - \sum_{k \neq i} \alpha_{ki} \alpha^*_{jj} |k\rangle|\phi_{ki}\rangle\langle j|\langle\phi_{jj}| \right.$$

$$\left. - \sum_{k' \neq j} \alpha_{ii} \alpha^*_{k'j} |i\rangle|\phi_{ii}\rangle\langle k'|\langle\phi_{k'j}| + \alpha_{ii} \alpha^*_{jj} |i\rangle|\phi_{ii}\rangle\langle j|\langle\phi_{jj}| \right] |O_i\rangle\langle O_j| + \sum_i \rho_{ii} |O_i\rangle\langle O_i| \tag{43}$$

$$= \sum_{i \neq j} \left[ \sum_{k \neq i,j} \alpha_{ki} \alpha^*_{kj} \langle \phi_{kj}|\phi_{ki}\rangle - \alpha_{ji} \alpha^*_{jj} \langle \phi_{jj}|\phi_{ji}\rangle - \alpha_{ii} \alpha^*_{ij} \langle \phi_{ij}|\phi_{ii}\rangle \right] |O_i\rangle\langle O_j| + \sum_i \rho_{ii} |O_i\rangle\langle O_i|. \tag{44}$$

Since our quantification of entanglement doesn't involve the diagonal terms, we don't bother to calculate them. In particular, we find that for $i \neq j$,

$$\rho^+_{t,ij} = \rho^-_{t,ij} - \alpha_{ji} \alpha^*_{jj} \langle \phi_{jj}|\phi_{ji}\rangle - \alpha_{ii} \alpha^*_{ij} \langle \phi_{ij}|\phi_{ii}\rangle. \tag{45}$$

Then the change in $S$ across the oracle step is

$$|S^- - S^+| \leq \sum_{i \neq j} |\alpha_{ji} \alpha^*_{jj} \langle \phi_{jj}|\phi_{ji}\rangle + \alpha_{ii} \alpha^*_{ij} \langle \phi_{ij}|\phi_{ii}\rangle| \tag{46}$$

$$\leq 2 \sum_{i \neq j} |\alpha_{ji}||\alpha_{jj}| \tag{47}$$

$$= 2 \sum_j |\alpha_{jj}| (\sum_{i \neq j} |\alpha_{ji}|). \tag{48}$$

Let $\gamma_j = \sum_i |\alpha_{ji}|^2$. Then, by the Cauchy-Schwartz inequality,

$$\sum_{i \neq j} |\alpha_{ji}| \leq \sqrt{\sum_{i \neq j} 1} \sqrt{\sum_{i \neq j} |\alpha_{ji}|^2} = \sqrt{N-1} \sqrt{\gamma_j - |\alpha_{jj}|^2}. \tag{49}$$

Thus,

$$|S^- - S^+| \leq 2\sqrt{N-1} \sum_j |\alpha_{jj}| \sqrt{\gamma_j - |\alpha_{jj}|^2} \tag{50}$$

$$\leq 2\sqrt{N-1} \sqrt{\sum_j |\alpha_{jj}|^2} \sqrt{\sum_j (\gamma_j - |\alpha_{jj}|^2)} \tag{51}$$

$$\leq 2\sqrt{N-1} \sqrt{A(1-A)} \tag{52}$$

$$\leq \sqrt{N-1}. \tag{53}$$

again using Cauchy-Schwarz to get the second line and using the normalization condition $\sum_j \gamma_j = 1$ and letting $A = \sum_j |\alpha_{jj}|^2$ to get the third line. Each oracle call drops $S$ by $O(\sqrt{N})$.

The oracle algorithm consists of alternating unitaries on the computer and oracle calls. The unitaries don't affect the oracle density matrix at all and thus don't change $S$. Therefore, to get a change of $c(N-1)$ in $S$ between the initial and final states, we need $c(N-1)/(2\sqrt{N-1}) = O(\sqrt{N})$ oracle calls.

The adversary method as presented here is sometimes stronger than the polynomial method and sometimes weaker. That is, there are some cases where the polynomial method can prove a good lower bound but the adversary method can't and vice-versa. However, the adversary method can be considerably generalized and in principle can give fairly tight bounds in most cases. In practice, though, it may be more tractable to prove things using the polynomial method.

## 2.6 More applications of the adversary method

The same technique that we saw for the unstructured search problem can be used to prove the following theorem:

**Theorem 7.** *Let $f$ be a function of $N$ binary variables $X_0, \ldots, X_{N-1}$ and $X$ and $Y$ be two subsets of possible input values such that $\forall\, x \in X, y \in Y, f(x) \neq f(y)$. Suppose there exists a relation $R \subseteq X \times Y$ with the following properties:*

1. *$\forall\, x \in X$, there are* at least *$m$ different values of $y \in Y$ such that $(x, y) \in R$*
2. *$\forall\, y \in Y$, there are* at least *$m'$ different values of $x \in X$ such that $(x, y) \in R$*
3. *$\forall\, x = (x_0, \ldots, x_{N-1}) \in X, i \in \{0, \ldots, N-1\}$, there are* at most *$l$ different values of $y = (y_0, \ldots, y_{N-1}) \in Y$ such that $(x, y) \in R$ and $x_i \neq y_i$.*
4. *$\forall\, y = (y_0, \ldots, y_{N-1}) \in Y, i \in \{0, \ldots, N-1\}$, there are* at most *$l'$ different values of $x = (x_0, \ldots, x_{N-1}) \in X$ such that $(x, y) \in R$ and $x_i \neq y_i$.*

*Then any quantum algorithm for $f$ uses at least $\Omega(\sqrt{mm'/(ll')})$ queries.*

The inputs again represent different oracles and the function is the oracle property we are supposed to compute. The individual variables $X_i$ are the values $O(i)$. The sets $X$ and $Y$ can be thought of as separating the inputs into sets with different function values. $R$ sets up connections between elements with different function values. The first two conditions on the relation say that there are lots of things related to every $x \in X$ and every $y \in Y$. The last two conditions say that there are not too many things related to $x \in X$ that are different in a particular coordinate value and similarly for $y \in Y$.

So, to see how this works in practice, consider the data search/ OR problem. We will let $X$ be the all-0 input $(0, \ldots, 0)$, so $f(x) = 0$ for $x \in X$. We will let $Y$ be the set of all Hamming weight 1 inputs $O_i$. $f(y) = 1$ for $y \in Y$. We will just let the relation $R$ be complete, so all $(x, y) \in X \times Y$ are related.

Then $m = N = |Y|$ and $m' = 1 = |X|$. For $x \in X$, each coordinate has exactly 1 $y \in Y$ that differs from it in that coordinate ($O_i$ differs from the all-0s input on coordinate $i$), and each $y \in Y$ has at most 1 $x \in X$ that differs from it on coordinate $i$. Thus $l = l' = 1$. The theorem then says we need $\Omega(\sqrt{N})$ queries.

Another example is an OR/AND tree. Let $N$ be a perfect square, let $g(X_0, \ldots, X_k)$ be the OR function of the $k+1$ inputs, and let $f(X_0, \ldots, X_{N-1})$ be the AND of $g(X_a, \ldots, X_{a+\sqrt{N}-1})$ for $a = 0, \ldots, \sqrt{N} - 1$. That is, $f = 1$ iff there is at least one marked element in each block of $\sqrt{N}$ inputs to the oracle.

To apply the theorem, we let $X$ be the set of oracles that have exactly 1 marked element for each block of $\sqrt{N}$ oracle inputs and let $Y$ be the set of oracles that have exactly one block of $\sqrt{N}$ oracle inputs that has no marked elements, and all other blocks have exactly 1 marked element. Then $f(x) = 1$ for $x \in X$ and $f(y) = 0$ for any $y \in Y$. These two sets are basically as close as we can get while still having different function values. You can see that changing just one oracle value for $Y$ in the block that has no marked elements will result in an element of $X$. The relation $R$ is defined to be all such pairs $(x, y)$ which differ on exactly one variable.

For this $R$, take any $x \in X$. It has $\sqrt{N}$ 1 values, one in each block of $\sqrt{N}$. By changing any one of them, we get an element of $Y$, so $m = \sqrt{N}$. However, on any particular coordinate, there is at most one related element of $Y$ that differs on that coordinate, so $l = 1$. Similarly, for each $y \in Y$, it is related to exactly $\sqrt{N}$ values of $x$, that differ from it on one of the coordinates in the unmarked block, but there is no more than one that differs on any particular coordinate. Thus, $m' = \sqrt{N}$ and $l' = 1$.

Then applying the theorem tells us that the quantum query complexity is at least $\Omega(\sqrt{N})$.

## 2.7 Quantum Communication Complexity

Now for a digression. Let's talk about a different kind of complexity. We've already seen circuit complexity, the size of a circuit to solve some computational problem, and query complexity, the number of oracle queries needed to learn some property of the oracle. *Communication complexity* of a problem is the amount of communication needed to compute some function whose inputs are distributed. Again, we don't care about the gate complexity, only the amount of communication. As with query complexity, it is actually possible to prove things about communication complexity.

There are bewildering number of different variants of communication complexity. Let's focus on 2-party communication complexity, where Alice and Bob have inputs $x$ and $y$ and are trying to compute $f(x, y)$. Even in this context, there are many different choices to make. Obviously, one choice is quantum vs. classical. But there are actually a number of natural ways to make communication complexity quantum. You can say Alice and Bob can send qubits and we want to count the number of qubits. Or you say Alice and Bob have lots of entangled states but they only communicate classically and we count the number of classical bits sent. Or we could count both entangled states and classical bits. Or we could have entangled states but also quantum communication. Luckily, some of these variants are equivalent: In particular, using 1 maximally entangled state and two classical bits, we can use quantum teleportation to send one qubit. We can also use one transmitted qubit plus one maximally entangled state to send 2 classical bits using superdense coding. So when there is plenty of entanglement, the transmission of 1 qubit is equivalent to the transmission of 2 classical bits, and therefore these two types of quantum communication complexity are equivalent up to a factor of 2. But when entanglement is limited, there is a tradeoff between entanglement and communication.

You can also play around with the transmission rules. For instance, maybe both Alice and Bob can send qubits to each other, or alternatively Alice can send to Bob but not vice-versa. Or Alice and Bob don't communicate directly but instead both send a message to a third party who is supposed to process their transmissions and compute the function. And if Alice and Bob can both send messages, can they go back and forth, or just send once and that's it? Basically all of these different variants can be *separated*: i.e., there exist problems for which the communication complexity of the variants is not the same. And, finally, you can consider deterministic or zero-error communication complexity or bounded-error communication complexity,

or even communication complexity with one-sided error (0 error for a yes instance, for instance, but not for a no instance); you can also do this with query or gate complexity.

The most common, default model of quantum communication complexity is where Alice and Bob send qubits and they can both send to each other back and forth in as many rounds as necessary. The measure of quantum communication complexity is how many qubits need to be transmitted in total between Alice and Bob to compute $f(x, y)$ with high probability (greater than 2/3).

The reason I am sticking this topic in here is that, in this model, there is a way to convert a quantum query complexity algorithm into a quantum communication complexity protocol with communication equal to the number of queries times $2(\log N+1)$ (for $N$ possible inputs to the query algorithm). Specifically, given a quantum algorithm to compute $f(O)$ for an oracle $O(z)$, we get a quantum communication complexity protocol to compute $g(x, y)$, where $g(x, y) = f(O)$ where $x$ and $y$ are $N$-bit vectors and the oracle $O$ is defined by the $N$-bit vector $x \wedge y$ (the bitwise AND of $x$ and $y$). (You can use any function $\mathbb{Z}_2 \times \mathbb{Z}_2 \to \mathbb{Z}_2$, but the AND function seems to work particularly well.)

As a concrete example, suppose we start with the unstructured search problem, find a marked element $z_0$ such that $O(z_0) = 1$. The corresponding communication complexity problem is to find $i$ such that $x_i = y_i = 1$. (Or more simply, to learn if there is such an $i$.) This is quite a natural problem, known as the *disjointness* problem: For instance, suppose you wish to set up a meeting with your advisor. Here the $N$ bits of $x$ and $y$ represent time slots in your calendar and that of your advisor. The $i$th bit is 0 if that time slot is unavailable and 1 if it is free. You are looking for a time slot where both are free.

How many bits do you need to send? One of you could send their whole schedule, indicating when they are free and when they are busy, but is that the best thing to do? Classically, this is basically the best you can do — the classical communication complexity is $\Theta(N)$, but if you can send qubits, you can do better.

The idea is to use Grover's algorithm, essentially treating your advisor's schedule as the oracle. Suppose at some point, you have the state $\sum_i \alpha_i |i\rangle$. Then you (quantumly) consult your schedule and compute that in an ancilla register, so the state is $\sum_i \alpha_i |i\rangle|x_i\rangle$. Now you send this state to your advisor, and they perform the unitary $U|i\rangle|c\rangle = (-1)^{cz_i}|i\rangle|c\rangle$ and send the system back. You then erase the ancilla to get $\sum_i \alpha_i (-1)^{x_i z_i}|i\rangle$. You've just performed the oracle $O$ given by the vector $x \wedge y$, as desired. You can thus run Grover's algorithm, using this subroutine in place of the oracle. Each "oracle" call uses $\log N + 1$ qubits transmitted in each direction. At the end, you measure, and with high probability you find a marked element for $O$, namely a location $i$ such that $x_i y_i = 1$, as desired.

The $\log N$ overhead can be removed, so the actual quantum communication complexity for this problem is $O(\sqrt{N})$, which turns out to be optimal. To prove a lower bound on the quantum communication complexity, however, we can't rely on the oracle bounds (since there might be communication complexity problems that are not converted query algorithms), and different techniques are needed to lower bound communication complexity. I won't go into those, however.

Disjointness provides a quadratic improvement in communication between quantum vs. classical. An exponential separation is known as well, for a more artifical promise problem. Exponential separations are also known between most (maybe all) pairs of variant quantum communication complexity problems mentioned before. As far as I can see, it looks like it is still an open question whether it is possible to get an exponential improvement in communication complexity between randomized classical communication and quantum communication for a total function, at least for the standard model.

# 3 References

The discussion of (classical) oracles and results relative to oracles are in most standard references on classical complexity theory. The Deutsch-Jozsa algorithm, Simon's algorithm, Grover's algorithm and a basic discussion of quantum oracles are both standard quantum textbook material, for instance in Nielsen and Chuang, *Quantum Computation and Quantum Information*. (Simon's algorithm is mentioned 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 $\mathrm{BQP}^{\mathrm{BQP}} = \mathrm{BQP}$.) Lower bounds using polynomials were developed by Beals, Buhrman, Cleve, Mosca, and de Wolf, "Quantum Lower Bounds by Polynomials," quant-ph/9802049.

Lower bounds using polynomials were developed by Beals, Buhrman, Cleve, Mosca, and de Wolf, "Quantum Lower Bounds by Polynomials," quant-ph/9802049. The adversary method was introduced by Ambainis, "Quantum lower bounds by quantum arguments," quant-ph/0002066.

The optimal separation between classical and quantum query complexity is from Aaronson, Ben-David, Kothari, Rao, and Tal, "Degree vs. Approximate Degree and Quantum Implications of Huang's Sensitivity Theorem," 2010.12629 and Ambainis, Balodis, Belovs, Lee, Santha, and Smotrovs, "Separations in Query Complexity Based on Pointer Functions," 1506.04719.

The adversary method, including the theorem presented in these lecture notes, was introduced by Ambainis, "Quantum lower bounds by quantum arguments," quant-ph/0002066. The relationship between quantum communication complexity and oracles was shown by Buhrman, Cleve, and Wigderson, "Quantum vs. classical communication and computation," quant-ph/9802040, and also gave a $O(\sqrt{N} \log N)$ algorithm for the OR/AND tree problem given as an example.