Lecture 23, Thurs April 13: BBBV, Applications of Grover

It's great that we can get a quadratic speedup with Grover's algorithm, but we were able to get an *exponential* speedup with Shor's algorithm…

So why can't we get a bigger speedup for unordered search?

By now, you should have some intuition for the differences between Shor's algorithm and Grover's algorithm.

Shor's algorithm provided an exponential speedup by orchestrating a very "global" phenomenon: an interference effect that revealed the period of a black-box periodic function.

Grover's algorithm let us turn a little amplitude into a bigger amplitude by adding $1/\sqrt{N}$ with each query. It's faster than classical brute-force search, but still laborious ($\sim\sqrt{N}$ time).

We're *still* hand-waving the issue though. We haven't ruled out the possibility of a quantum algorithm that beats Grover, solving unordered search in $\sqrt[3]{N}$ or $\log(N)$ queries or whatever. For that we need…

The BBBV Theorem (Bennett, Bernstein, Brassard, Vazirani 1994)

which proved that Grover's algorithm is indeed asymptotically optimal for the black-box unordered search problem.

Note that the BBBV Theorem was published in 1994, so it actually predates Grover. Grover's algorithm thus has the rare distinction of being an algorithm that was proved to be optimal *before* it was discovered.

Amusingly, BBBV were trying to prove that there's no magic way to search faster using a quantum computer. They were able to get a lower bound of $\sim\sqrt{N}$, and figured that tightening the bound to $\sim N$ was a technical issue that they could leave for the future—until Grover came along and showed why such a tightening is impossible.

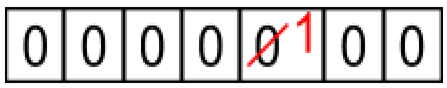
While by now we know many proofs of the BBBV Theorem, the original (and still most self-contained) proof uses what's called a Hybrid Argument.

Imagine we're using an arbitrary quantum algorithm to search for a single marked item in a list of size N. Without loss of generality, we can say that the algorithm makes T queries and looks like this:

$$U_0 \to Q \to U_1 \to Q \to U_2 \to Q \to \ldots$$

where each Q is a query, and each U_t is a unitary transformation that doesn't depend on the list. Now we do a test run of the algorithm on the all-zero list (without a marked item), and see what happens.

We'll argue that at least one item in the list was queried with a small amplitude, because there's only so much amplitude to go around. But now, if we change the value of *that item* from 0 to 1, then we can show that the algorithm



wouldn't much notice the change, by exploiting the fact that unitary transformations are linear and norm-preserving.

More concretely, we'll show that the final state of the algorithm is at most O($\frac{t}{\sqrt{N}}$) away from what it would have been, had we kept the list all-zero.

Note: while we'll only consider algorithms that apply unitary transformations, exactly the same proof carries over to algorithms that have intermediate measurements---because we can always model a measurement by a unitary transformation on a larger set of qubits, just like in the Many-Worlds Interpretation.

The argument is called "hybrid" because we'll create hybrid oracles, which answer the first queries as if they're the all-zero oracle, but then switch partway through the algorithm to having a single "1" entry.

<u>What is the state of the algorithm immediately before the *t*th query?</u>

$$|\Psi_t\rangle = \sum_{x,\, w} \alpha_{x,w,t}\, |x,\, w\rangle$$

Assuming the all-zero input.

x is whichever list item that we're querying next.

w is what's known in quantum algorithms as "the workspace," any qubits that the algorithm might use for internal purposes but that don't participate in the query.

In Grover's algorithm there's hardly any "workspace" to speak of: we do use auxiliary qubits to implement the diffusion operator, but those qubits are reset to their original state by the time the diffusion operator is finished. But the BBBV Theorem, to be fully general, will allow for unlimited workspace. We'll show that regardless of the workspace, the algorithm would require ~ $\sqrt{N}$ queries.

$\alpha_{x,w,t}$ is the amplitude of basis state $|x,\, w\rangle$ at time *t*.

Now we define the "query magnitude" of an element $x \in \{1, \dots, N\}$ to be

$$M_x = \sum_{t=1}^{T} \sum_{w} |\alpha_{x,w,t}|^2$$

I.e., the query magnitude is the sum, over all time steps *t*, of the probability that we would find the algorithm querying item *x* if we measured at time *t*.

Observe that by doing some rearranging, we find that the sum of all the query magnitudes is

$$\sum_{x=1}^{N} M_x = \sum_{t=1}^{T} \Big(\sum_{x=1}^{N} \sum_{w} |\alpha_{x,w,t}|^2 \Big) = \sum_{t=1}^{T} 1 = T$$

Since the sum is *T*, the average query magnitude is $\frac{T}{N}$. Now, any list of numbers has at least one number that's at most the average.

This is sometimes referred to as the "Lake Wobegon Principle," after the fictional town where everyone was above average.

So let x^* be a list element with query magnitude $M_{x^*} \leq \frac{T}{N}$.

The idea here is that the algorithm only has so much amplitude to spread around, and thus most database items must not get "monitored" too closely. So if we pick one such item, x^*, and make it the marked item, then the algorithm will mostly fail to notice the change.

More formally, we have

$$\sum_{t=1}^{T} \sum_{w} |\alpha_{x^*,w,t}|^2 \leq \frac{T}{N}.$$

But it would be more useful to us to have an upper bound on

$$\sum_{t=1}^{T} \sqrt{\sum_{w} |\alpha_{x^*,w,t}|^2}.$$

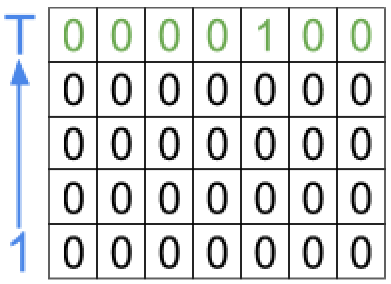
To get from one to the other we use the Cauchy-Schwarz Inequality, which is super useful in quantum information (and many other fields).

Given a unit vector $\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix}$, what is the maximal sum of the absolute values of its entries, $\sum_{n=1}^{N} |\alpha_i|$?

The Cauchy-Schwarz Inequality says that we can maximize the sum by making all entries equal, with the vector $\begin{bmatrix} \frac{1}{\sqrt{N}} \\ \vdots \\ \frac{1}{\sqrt{N}} \end{bmatrix}$, so that the sum is $\frac{N}{\sqrt{N}} = \sqrt{N}$.

<u>So what does the Cauchy-Schwarz Inequality say about our case?</u>

Well, if each of the T terms of the form $\sum_{w} |\alpha_{x^*,w,t}|^2$ were set equal to $\frac{1}{N}$, then the sum of their square roots would be $\frac{T}{\sqrt{N}}$. So this is the maximum.



T	0	0	0	0	1	0	0
	0	0	0	0	0	0	0
	0	0	0	0	0	0	0
	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

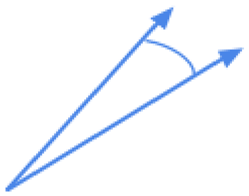
<u>Why is this relevant?</u>

Now comes the hybrid part of the argument.

Picture a table where each row is our database at a particular point in time, with time increasing upwards. Initially the table is filled with zeros, meaning that the oracle answers all queries with zero.

Now we're going to change the table so that the oracle answers $f(x^*) = 1$ for the last query (and *only* for the last query). This means that the state of the algorithm after the final query $|\Psi_T\rangle$, is going to change from what it was before---but we know it can change only in branches that were lucky enough to query x^*.

So how much can things change (in Euclidean distance)? Well, since the query flips the amplitude, the change is equal to the total norm with which we made the query, times 2:



$$\big|\, |\Psi_T\rangle - |\Psi_T'\rangle \,\big| = 2\sqrt{\sum_w |\alpha_{x,w,T}|^2}$$

Here we're using the fact that, by assumption, the states immediately before the T^{th} query are identical in the two situations.

<u>So what happens if the oracle treats x^* as marked only for the last *two* queries?</u>

The total amplitude devoted to querying x^* before the final amplitude is $2\sqrt{\sum_w |\alpha_{x,w,T-1}|^2}$, so we get

$$\big|\, |\Psi_{T-1}'\rangle - |\Psi_{T-1}''\rangle \,\big| \le 2\sqrt{\sum_w |\alpha_{x,w,T-1}|^2}\,.$$

<u>But couldn't the last query push these further apart?</u>

Here we come to a crucial point: we claim that it can't. For in both cases, the last query is applying the same unitary transformation, which means that the inner product between the states can't change.

So we get $\big|\, |\Psi_T'\rangle - |\Psi_T''\rangle \,\big| \le 2\sqrt{\sum_w |\alpha_{x,w,T-1}|^2}$ and hence

$\big|\, |\Psi_T\rangle - |\Psi_T''\rangle \,\big| \le 2\sqrt{\sum_w |\alpha_{x,w,T-1}|^2} + 2\sqrt{\sum_w |\alpha_{x,w,T}|^2}$ by the triangle inequality.

Continuing in the same way for all T of the queries, we find that $\big|\, |\Psi_T\rangle - |\Psi_T''''''\cdots'\rangle \,\big|$ is upper-bounded by $2\frac{T}{\sqrt{N}}$. This means, in particular, that after we measure at the end of the algorithm, we can have found the marked item x^* with probability at most $O(\frac{T^2}{N})$, precisely the success probability that Grover's algorithm achieves.

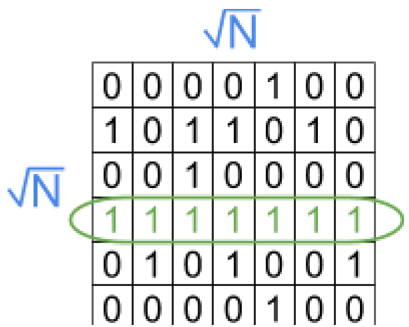
This "BBBV Theorem" has since been enormously generalized, to a whole theory about lower bounds on quantum query complexity, which unfortunately we won't really enter into in this course--but see Prof. Aaronson's graduate course for more!

Now that we understand Grover's algorithm, we can apply it to solve many, many problems that are not quite as simple as unordered search. Our first example of this…

The OR's of AND's

N input bits are arranged into a square table of size $\sqrt{N}$ by $\sqrt{N}$.

The problem is to determine: are there any rows with all 1's?



0	0	0	0	1	0	0
1	0	1	1	0	1	0
0	0	1	0	0	0	0
1	1	1	1	1	1	1
0	1	0	1	0	0	1
0	0	0	0	1	0	0

This problem lets us encode many other problems that involve multiple quantifiers. For example, in chess, we may want to know "Is there a move I can make such that my opponent has no possible response that checkmates me?"

Classically, it's clear that you have to look through almost the entire table, searching each row until you've either found a 0 or found that the row is all 1's.

Quantumly, we could speed this up by searching each row for 0's using Grover's algorithm. The running time for each row would be $\sqrt{\sqrt{N}} = N^{1/4}$, or technically $N^{1/4} \log N$, if we repeat the Grover search on each row enough times to have (say) a $1/N$ probability of error. This means that searching the whole table would take time

$$\sqrt{N}\ N^{1/4} \log N = N^{3/4} \log N.$$

Alternatively we could do Grover's algorithm over all the rows, such that each row is counted as a "marked item" if and only if a classical algorithm (which we run as an inner loop) finds a zero in that row. This also has an $\sim N^{3/4}$ runtime.

Naturally, the next idea is to run Grover's algorithm recursively, *inside of itself*, where the outer Grover (over the rows) will count a given row as being marked, if and only if the inner Grover failed to find a zero in that row. Again, because Grover's algorithm has some probability of error, at least naïvely we have to repeat the inner runs about $\log N$ times to push the error probability per row down to about $\frac{1}{N}$.

So our final runtime is $O(\underbrace{\sqrt{\sqrt{N}}}_{\text{outer G.A.}}\ \underbrace{\sqrt{\sqrt{N}}}_{\text{inner G.A.}}\ \underbrace{\log N}_{\text{Error Avoidance}}) = \underbrace{\mathbf{O(\sqrt{N}\log N)}}_{\textbf{the Grover speedup!}}$

With some cleverness, people have since been able to remove the log N factor.

Why couldn't we just do Grover's algorithm once, over the whole table?

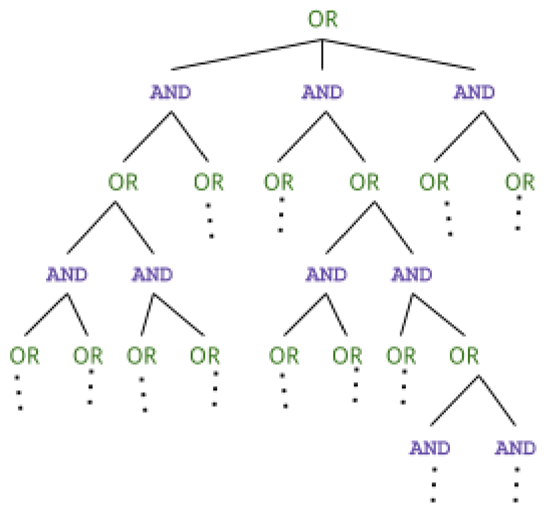
Well, just because there's a 0 *somewhere* in the table, doesn't mean that there couldn't be a row of all 1's somewhere else.

The first problem in quantum computing that Professor Aaronson worked on was trying to generalize the BBBV Theorem to show that "recursive Grover" is optimal: in other words, that any possible quantum algorithm for the OR-of-ANDs requires at least $\sim\sqrt{N}$ queries. (The obvious lower bound is only $\sim N^{1/4}$.) He spent a whole summer trying to solve this problem using the "polynomial method," but couldn't crack it. Later, Andris Ambainis, then a PhD student at Berkeley, invented a totally new method for proving lower bounds on quantum query complexity, and applied it to solve this problem.

Which was why Prof. Aaronson decided to go to Berkeley for grad school.

We could easily generalize this to evaluate (e.g.) an OR of ANDs of ORs by doing *three* recursive layers of Grover search, and so forth.

If we allow an arbitrary number of layers, then we get the A.I. concept of game trees, for two-player games of alternation such as chess and Go. Here the goal is to find a move you can make (represented by an OR over various options), which given any move that your opponent makes (represented by ANDs), allows for a move that you can make, that for any move your opponent makes, etc. … eventually wins you the game.



The problem is that, as the game tree gets deeper and deeper, the advantage of Grover's algorithm over classical search *seems* to get weaker and weaker, for two reasons: first, the amplification that's needed at each layer to prevent error buildup, and second, the constant factors, which multiply across the layers. Note that each layer actually needs to run Grover's algorithm on the layer below it *twice*:

- Once to do $|x\rangle \rightarrow |x\rangle|f(x)\rangle$
- And once to uncompute garbage.

For this reason, the constant factor $\pi/4$, in the running time of Grover's algorithm, actually becomes $\pi/2$, and $\pi/2 > 1$.

In short, none of this answers the natural question: *"Can a quantum computer help you play chess?"* For game-tree search with a deep enough tree, Prof. Aaronson and some others conjectured that the diminishing returns from Grover's algorithm would end up negating any asymptotic advantage over a classical computer.

In 2007, however, Farhi, Goldstone, and Gutmann, and others who built on their work, dramatically refuted that conjecture. The upshot of their work is that we now know how to evaluate *any* game tree with N leaves, no matter how deep, in $O(\sqrt{N})$ time on a quantum computer. (This is also known to be asymptotically optimal.)

So, yes, quantum computers probably *would* help you play chess!

To put some numbers on this: Claude Shannon famously estimated the number of possible board positions in chess as $\sim 10^{43}$, which is certainly out of range for any existing computer on earth. But if quantum computers brought that down to $\sim 10^{21.5}$, solving chess might *just* be doable.

Though it raises a philosophical question: Have you actually "solved" chess if you don't have a solution table that anyone can examine, but only a quantum computer that always wins?

In the next lecture, we'll see some additional applications of Grover's algorithm, to the so-called *collision* and *element distinctness* problems.