Bogus Quantum Algorithms

Sam Jaques

May 20, 2023

Studying quantum algorithms I have come up with ideas that made me think "Wait, doesn't this break *everything*?". And then of course it turns out there is some subtle flaw to my idea.

I thought I'd share these as a teaching tool. Just as debugging a classical program can really teach you about the computer, so quantum "debugging" should hopefully teach about quantum algorithms.

If you have your own bogus algorithm, please send it to me and I'll attribute you and add it to the list!

1 Grover With a Simple Oracle

Grover's algorithm is provably optimal in query complexity. But *what* function do we need to query? That is, why can't I initialize a superposition of inputs and call the oracle to get the state:

$$\frac{1}{\sqrt{n}}\sum_{x=0}^{n}|x\rangle\,|f(x)\rangle \tag{1}$$

and then make an extra bit $b(x)$ representing whether $f(x)$ was the "right" value:

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

the uncompute $f(x)$:

$$\frac{1}{\sqrt{n}}\sum_{x=0}^{n}|x\rangle\,|0\rangle\,|b(x)\rangle \tag{3}$$

and then for my "oracle" I just check the bit $b(x)$. So sure, I'll have to check that bit $O(\sqrt{n})$ times, but I've saved all my actual oracle computations.

2 Bit-by-Bit Grover

Suppose we are looking for a unique bitstring $a = a_1 \cdots a_n$ among all n-bit strings, defined such that $f(a) = 0$ and $f(b) = 1$ for all $b \neq 0$, for an easily computable function f. The idea is to search one bit at a time. We need the following routine, that searches for a single bit.

First, prepare a uniform superposition of 2 qubits: $|00\rangle + |10\rangle + |01\rangle + |11\rangle$. Then, apply the oracle to the "and" of the output and apply it to a phase gate (that is: if 1 is the right bit, only $|11\rangle$ will get its

phase flipped, and if 0 is the right bit, only $|00\rangle$). Then apply a Hadamard to each qubit. The resulting state will be either:

$$\begin{cases} |00\rangle + |11\rangle\,, & f(0) = 1 \\ |10\rangle - |01\rangle\,, & f(1) = 1 \end{cases} \tag{4}$$

So: Apply a CNOT from the first to the second qubit, then a Hadamard to the first qubit, to get $|0\rangle\,|0\rangle$ for $f(0) = 1$ and $-|1\rangle\,|1\rangle$ for $f(1) = 1$. Apply a CNOT from the second back to the first, and a controlled-phase from the second, and get $|00\rangle$ vs. $|01\rangle$. Discard the first register, keep the second.

If neither one is successful (i.e., $f(0) = f(1) = 0$, then there will be no phase flips, and you will get $|+0\rangle$ as a final state. However: We can then save $f(x)$, for the value x in the second qubit, into an ancilla qubit, then use that to control a Hadamard on the first qubit. Thus we get $|00\rangle$ if $f(0) = f(1) = 0$, and still what we want otherwise.

Thus: Suppose we have a state of the form

$$\frac{1}{2^{j/2}} \sum_{\vec{b}_j=0}^{2^j} |b_1 b_2 \cdots b_j\rangle\,|\psi_{n-j}(b_1 \cdots b_j)\rangle \tag{5}$$

where $|\psi_{n-j}(b_1 \cdots b_j)\rangle$ is 0 if $a_1 \cdots a_j \neq b_1 \cdots b_j$, and otherwise, $\psi_{n-j}(b_1 \cdots b_j) = a_{j+1} \cdots a_n$.

We can decompose this into two states:

$$\frac{1}{2^{(j-1)/2}} \sum_{\vec{b}_{j-1} \neq \vec{a}_{j-1}} |b_1 \cdots b_{j-1}\rangle\,\tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) + \frac{1}{2^{(j-1)/2}}\,|\vec{a}_{j-1}\rangle\,\tfrac{1}{\sqrt{2}}(|\overline{a_{j-1}}\rangle\,|0\rangle + |a_{j-1}\rangle\,|a_{j+1} \cdots a_n\rangle). \tag{6}$$

and introduce an extra superposition next to b_j:

$$\frac{1}{2^{(j-1)/2}} \sum_{\vec{b}_{j-1} \neq \vec{a}_{j-1}} |b_1 \cdots b_{j-1}\rangle\,\tfrac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \tag{7}$$

$$+ \frac{1}{2^{(j-1)/2}}\,|\vec{a}_{j-1}\rangle\,\tfrac{1}{2}(|\overline{a_{j-1}}\rangle\,(|0\rangle + |1\rangle)\,|0\rangle + |a_{j-1}\rangle\,(|0\rangle + |1\rangle)\,|a_{j+1} \cdots a_n\rangle). \tag{8}$$

Then for the 2 qubits representing b_j, we can use the function I described at the start, using the rest of the bits as input to the oracle, and end up with the state

$$\frac{1}{2^{(j-1)/2}} \sum_{\vec{b}_{j-1} \neq \vec{a}_{j-1}} |b_1 \cdots b_{j-1}\rangle\,|0\rangle + \frac{1}{2^{j/2}}\,|\vec{a}_{j-1}\rangle\,\tfrac{1}{2}\,|\overline{a_{j-1}}\rangle\,|a_j\rangle\,|a_{j+1} \cdots a_n\rangle\,. \tag{9}$$

This is precisely the same as the state we started with, but with j decreased by 1. Since this clearly the matches the required state for $j = n$, we can repeat this procedure n times and get a final state of $|a_1 \cdots a_n\rangle$.

Each iteration only called the oracle twice, plus a few Hadamards and CNOTs, so this solves generic search of size 2^n in $O(n)$ gates.

3 Hidden Shift with Cayley Graph Path-Finding

Given a group G with a small set S of generators, with G acting on some set X, suppose we are given that $g * x = y$ for some unknown g and some known x and y. Our goal is to find the string $g = s_1 \cdots s_r$, where $s_i \in S$.

Consider the Cayley graph generated by S. What we can do is construct a superposition of bitstrings $b_0 \cdots b_s$, where b_i indexes an element of S, and we can iterate through them. Start with the identity, then multiply by s_{b_0}, then multiply that by s_{b_1}, etc., where if $s_{b_i} = s_{b_{i-1}}^{-1}$, then don't multiply by anything (it acts as the identity). This will create a uniform superposition of all group elements that are distance up to s from the identity, in the Cayley graph. For each state in the superposition, apply that group element to x. This gives a superposition of elements in X. We can also make the same superposition and apply it all to y.

The idea is that we control which superposition to create with a single qubit initialized to $|+\rangle$. Suppose $|x\rangle$ and $|y\rangle$ are the two states, where $|x\rangle$ is all points in X reachable from a path starting at x and $|y\rangle$ is all points reachable from y. Then we have $\frac{1}{\sqrt{2}}(|0\rangle|x\rangle + |-\rangle|y\rangle$. Apply a Hadamard to the first qubit:

$$(H \otimes I)\tfrac{1}{\sqrt{2}}(|0\rangle|x\rangle + |-\rangle|y\rangle = \tfrac{1}{2}(|0\rangle|x\rangle + |1\rangle|x\rangle + |0\rangle|y\rangle - |1\rangle|y\rangle) \tag{10}$$

$$= \tfrac{1}{2}(|0\rangle(|x\rangle + |y\rangle) + |1\rangle(|x\rangle - |y\rangle)) \tag{11}$$

The probability of measuring the left state, with the first qubit 0, is $\frac{\||x\rangle+|y\rangle\|}{2}$, and the probability of measuring 1 is $\frac{\||x\rangle-|y\rangle\|}{2}$. Thus, by repeating this experiment, we can approximate (with precision linear in the number of repetitions) the distance between $|x\rangle$ and $|y\rangle$.

Suppose we have two equal-sized sets A_x and A_y in X, and $|x\rangle$ is a uniform superposition of elements in A_x and $|y\rangle$ is the same for A_y. Then $|x\rangle - |y\rangle$ will cancel out all the elements that are the same. Thus, the distance between them will be proportional to the symmetric difference $(A_x \cup A_y) \setminus (A_x \cap A_y)$.

To use this: We modify the construction of the superpositions slightly: We pick one particular element s_x of S, and use that to start the walk that we apply to x, and another element s_y and use that the start the walk that we apply to y. If we picked the correct ones (i.e., $s_x = s_1$ and $s_y = s_r^{-1}$), then the walks will have much more overlap than if we picked the wrong ones. Since we can measure the overlap (i.e., the symmetric difference), we can determine when we pick the right element.

There is likely some work to iron out the details: The difference in overlap might be small at certain distances. For example, if we pick s, the distance of the walk in the Cayley graph, to be less than $r/2$ (r is the minimum length of a word of generators in S to construct g such that $g * x = y$), the overlap should be 0. One thing we could do is repeat the process for $s = 1, 2, \cdots$ until the overlap is nearly 1. At some point, the difference should be noticeable between paths that start with the right edge and paths that do not, and noticeable with (likely) a polynomial number of tests.

Suppose that we need $f(\log n)$ tests for a sufficiently accurate measure of distance (where $|G| = n$ and f is a polynomial). Then the run-time should be $O(r^2|S|f(\log n))$; presumably, $|S| = O(\log n)$ and $r = O(\log n)$, so we get $O((\log n)^3 f(\log n))$.

Thus, we have a polynomial algorithm for the hidden shift problem. Further, we made no assumptions that the group was abelian!