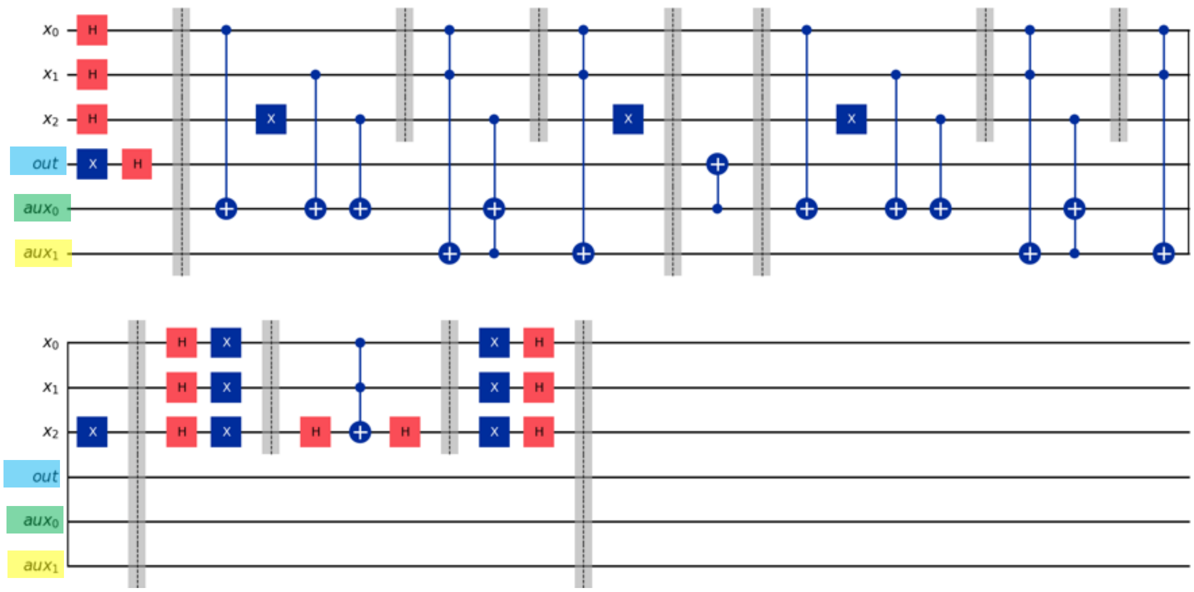
Grover’s Algorithm V-Gate Explanation

Ashwin Kaliyaperumal



This is an explanation of the Grover’s algorithm segment, specifically the implementation of the V gate, in Notebook 5 that implements expression [[1, 2, -3]].

I’ll be splitting up the sections by barrier gates. Each new line will correspond to a “slice” of the circuit – you can think of each new equation as the computations after each layer of the circuit.

1. Initialization

We first start with initializing our quantum circuit. We start off with each qubit in the |0> state. We then put all of our input qubits in an equal superposition and transform our output qubit into the |-> state. This output qubit set as the |-> will be crucial to understanding how the reflection is implemented.

|aux$_1$>|aux$_0$>|out>|x$_2$ x$_1$ x$_0$> = |0>|0>|0>|000> -> qubits start at all zeros

= |0>|0>|1> (|000> + |001> + |010> + |011> + |100> + |101> + |110> + |111>) -> apply the first layer of gates

= |0>|0>(|0>-|1>)(|000> + |001> + |010> + |011> + |100> + |101> +|110> + |111>) -> apply the second layer of gates

2. Oracle Application

Now, we need to add the oracle to phase flip (reflect) the target states. In theory, all we are doing is adding a few control-control-control-Z gates using our input and output qubits – we are reflecting the states that satisfy our 1-in-3 SAT expression above. In practice, we are doing this by implementing our 1-in-3 SAT expression into our quantum circuit using the quantum equivalents of our basic gate operations (AND's, OR's, NOT's, and XOR's).

We start off with adding all of our OR gates. Just like our first circuit in this notebook, we can't add XOR gates directly to qubits – we need to use an ancillary qubit to connect the results of these XOR's together. Additionally, we need to realize that we are working with a NOT x_2, so we add an X (bit flip) gate to encode this in our quantum circuit.

As you'll notice before, $|aux_0>$has been moved inside the parenthesis. This is to show the effect of the different input states on$|aux_0>$ – this will be important later on.

XOR Gates:

= |0>(|0>-|1>)(|0>|000> + |1>|001> + |0>|010> + |1>|011> + |0>|100> + |1>|101> +|0>|110> + |1>|111>) -> XOR Gate w/ x_0

= |0>(|0>-|1>)(|0>|100> + |1>|101> + |0>|110> + |1>|111> + |0>|000> + |1>|001> +|0>|010> + |1>|011>) -> NOT Gate w/ x_2

= |0>(|0>-|1>)(|0>|100> + |1>|101> + |1>|110> + |0>|111> + |0>|000> + |1>|001> +|1>|010> + |0>|011>) -> XOR Gate w/ x_1

= |0>(|0>-|1>)(|1>|100> + |0>|101> + |0>|110> + |1>|111> + |0>|000> + |1>|001> +|1>|010> + |0>|011>) -> XOR Gate w/ x_2

Now, we add in our AND gate. We use $|aux_1>$ as an intermediate to store the result from the first AND gate, and then we add in the second AND gate's result to$|aux_0>$.

AND Gates:

= (|0>-|1>)(|0>|1>|100> + |0>|0>|101> + |0>|0>|110> + |1>|1>|111> + |0>|0>|000> + |0>|1>|001> + |0>|1>|010> + |1>|0>|011>) -> AND Gate w/ x_0 and x_1

= (|0>-|1>)(|0>|1>|100> + |0>|0>|101> + |0>|0>|110> + |1>|0>|111> + |0>|0>|000> + |0>|1>|001> + |0>|1>|010> + |1>|0>|011>) -> AND Gate w/ aux_1 (x_0 and x_1) and x_2

Notice how we only have three states that induced our green aux qubit to the |1> state. These are the same exact states that satisfy our Boolean expression! However, we have a couple of steps that we need to take before we measure our qubits and reap the benefits from our computations. First, we need to reset out yellow aux qubit.

Undo |aux$_1$>:

= |0> (|0>-|1>)(|1>|100> + |0>|101> + |0>|110> + |0>|111> + |0>|000> + |1>|001> + |1>|010> + |0>|011>) -> AND Gate w/ x_0 and x_1

We also need to undo the NOT gate we applied on x_2 earlier.

Undo NOT gate on x_2:

= |0> (|0>-|1>)(|1>|000> + |0>|001> + |0>|010> + |0>|011> + |0>|100> + |1>|101> + |1>|110> + |0>|111>) -> NOT Gate w/ x_2

Now, here is the most important part – applying a controlled phase flip to our output qubit. We want our output qubit to phase flip if, and only if, our green aux qubit is one. This is where the usefulness of initializing our output qubit in the |-> state comes in. Since we apply a controlled not gate onto our output qubit, if our aux qubit is |1>, our output qubit turns into X|-> = -|-> for a given state.

Apply |aux$_0$> to output qubit:

= |0> (|0>-|1>) (-|1>|000> + |0>|001> + |0>|010> + |0>|011> + |0>|100> + -|1>|101> + -|1>|110> + |0>|111>) -> Phase flip if |aux$_0$> = |1>

We're at the final step. We just need to reset out green aux qubit. We do this by re-applying our oracle again. This should make intuitive sense – since we are applying the same operations again, and since all of our operations are their own inverses, we should end up right back to where we started. Below is a more thorough breakdown of that process.

Apply oracle again:

= |0> (|0>-|1>) (-|1>|000> + |1>|001> + |0>|010> + |1>|011> + |0>|100> + -|0>|101> + -|1>|110> + |1>|111>) -> XOR Gate w/ x_0

= |0> (|0>-|1>) (-|1>|100> + |1>|101> + |0>|110> + |1>|111> + |0>|000> + -|0>|001> + -|1>|010> + |1>|011>) -> NOT Gate w/ x_2

= |0> (|0>-|1>) (-|1>|100> + |1>|101> + |1>|110> + |0>|111> + |0>|000> + -|0>|001> + -|0>|010> + |0>|011>) -> XOR Gate w/ x_1

= |0> (|0>-|1>) (-|0>|100> + |0>|101> + |0>|110> + |1>|111> + |0>|000> + -|0>|001> + -|0>|010> + |0>|011>) -> XOR Gate w/ x_2

= (|0>-|1>) (-|0>|0>|100> + |0>|0>|101> + |0>|0>|110> + |1>|1>|111> + |0>|0>|000> + -|0>|0>|001> + -|0>|0>|010> + |1>|0>|011>) -> AND Gate w/ x_0 and x_1

= (|0>-|1>) (-|0>|0>|100> + |0>|0>|101> + |0>|0>|110> + |1>|0>|111> + |0>|0>|000> + -|0>|0>|001> + -|0>|0>|010> + |1>|0>|011>) -> AND Gate w/ aux_1 (x_0 and x_1) and x_1

= |0>|0> (|0>-|1>) (-|100> + |101> + |110> + |111> + |000> + -|001> + -|010> + |011>) -> AND Gate w/ x_0 and x_1

|0>|0> (|0>-|1>) (-|000> + |001> + |010> + |011> + |100> + -|101> + -|110> + |111>) -> NOT Gate w/ x_2

We've reached the end of our reflection gate (V gate)! Notice how the circuit is almost exactly the same as our original circuit after initialization. The only difference – three phase shifts in front of the |000>, |101>, and |110> states. With a truth table (shown below), we can confirm that these are indeed the correct states.

Truth Table:

x_0	x_1	x_2	$\neg x_2$	$x_0 \oplus x_1$	$x_0 \oplus x_1 \oplus \neg x_2$	$x_0 \wedge x_1 \wedge \neg x_2$	$x_2\, x_1\, x_0$	$x_0 \oplus x_1 \oplus \neg x_2 \oplus (x_0 \wedge x_1 \wedge \neg x_2)$
0	0	0	1	0	1	0	000	1
0	0	1	0	0	0	0	100	0
0	1	0	1	1	0	0	010	0
0	1	1	0	1	1	0	110	1
1	0	0	1	1	0	0	001	0
1	0	1	0	1	1	0	101	1
1	1	0	1	0	1	1	011	0
1	1	1	0	0	0	0	111	0