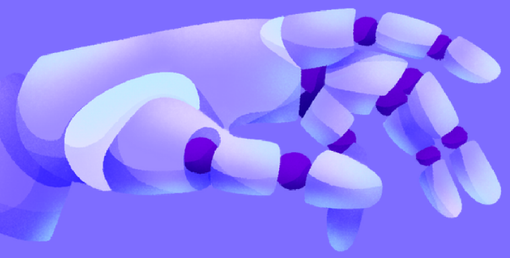


QUANTUM ALGORITHM FOR SOLVING SATISFIABILITY PROBLEMS USING GROVER'S SEARCH

INTRODUCTION

Grover search is one of the most popular algorithms used for searching a solution among many possible candidates using Quantum Computers. If there are N possible solutions among which there is exactly one solution (that can be verified by some function evaluation), then Grover search can be used to find the solution with O(sqrt(N)) function evaluations. This is in contrast to classical computers that require Ω(N) function evaluations: the Grover search is a quantum algorithm that provably can be used search the correct solutions quadratically faster than its classical counterparts.

Here, we are going to illustrate the use of Grover search to solve a combinatorial problem called Exactly-1 3-SAT problem. The Exactly-1 3-SAT problem is a NP-complete problem, namely, it is one of the most difficult problems that are interconnected (meaning that if we solve any one of them, we essentially can solve all of them). Unfortunately, there are many natural problems that are NP-complete, such as, the Traveling Salesman Problem (TSP), the Maximum Cut (MaxCut) and so on. Up to now, there is no classical and quantum algorithm that can efficiently solve such NP-hard problems.

We begin with an example of the Exactly-1 3-SAT problem. Then, we show how to design an evaluation function which is also known as the oracle (or, blackbox) which is essential to Grover search. Finally, we show the circuit of Grover search using the oracle and present their results on simulator and real-device backends.

x_1	x_2	x_3	f	Comment
0	0	0	1	Not a solution because there are three True literals in the second clause
0	0	1	0	Not a solution because f is False
0	1	0	1	Not a solution because there are two True literals in the first clause
0	1	1	1	Not a solution because there are three True literals in the third clause
1	0	0	0	Not a solution because f is False
1	0	1	1	**Solution**. BINGO!!
1	1	0	1	Not a soluton because there are three True literals in the first clause
1	1	1	0	Not a solution because f is False

From the table above, we can see that the assignment $x_1x_2x_3 = 101$ is the solution fo the Exactly-1 3-SAT problem to f. In general, the Boolean function f can have many clauses and more Boolean variables.

Grover Search: putting all together

The complete steps of Grover search is as follow.

1. Create the superposition of all possible solutions as the initial state (with working qubits initialized to zero)

$$\sum_{j=0}^{2^n-1} \frac{1}{2^n} |j\rangle |0\rangle$$

2. Repeat for T times:

   Apply the blackbox function

   Apply the inversion-about-the-average function

3. Measure to obtain the solution

A blackbox function to check the assignment of Exactly-1 3-SAT problem

Here, we describe a method to construct a circuit to check the assignment of Exactly-1 3-SAT problem. The circuit can then be used as a blackbox (or, oracle) in Grover search. To design the blackbox, we do not need to know the solution to the problem in advance: it suffices to design a blackbox that checks if the assignment results in f evaluates to True or False. It turns out that we can design such a blackbox efficiently (in fact, any NP-complete problem has the property that although finding the solution is difficult, checking the solution is easy).

For each clause of f, we design a sub-circuit that outputs True if and only if there is exactly one True literal in the clause. Combining all sub-circuits for all clauses, we can then obtain the blackbox that outputs True if and only if all clauses are satisfied with exactly one True literal each.

For example, let us consider the clause $(x_1 \vee \neg x_2 \vee x_3)$. It is easy to see that y defined as

$$y = x_1 \oplus \neg x_2 \oplus x_3 \oplus (x_1 \wedge \neg x_2 \wedge x_3),$$

is True if and only if exactly one of x_1, $\neg x_2$, and x_3 is True. Using two working qubits, y can be computed by the following sub-circuit. Below, $x_1x_2x_3$ is renamed as $q_1q_2q_3$, q_4 is used as a working qubit, and q_5 is used to store the value of y.

Another important procedure in Grover search is to have an operation that perfom the *inversion-about-the-average* step, namely, it performs the following transformation:

$$\sum_{j=0}^{2^n-1} \alpha_j |j\rangle \rightarrow \sum_{j=0}^{2^n-1} \left(2 \left(\sum_{k=0}^{k=2^n-1} \frac{\alpha_k}{2^n} \right) - \alpha_j \right) |j\rangle$$

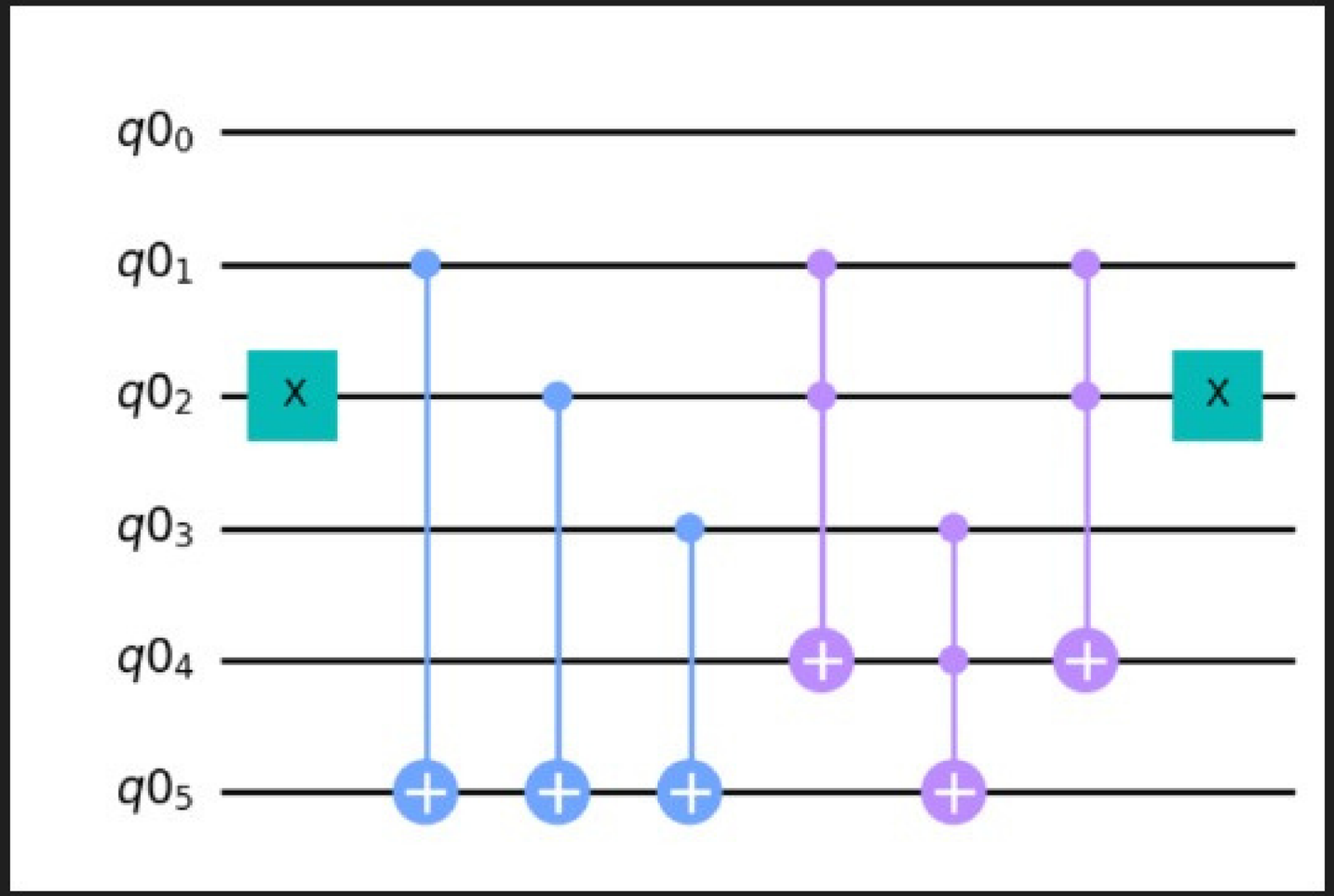
The above transformation can be used to amplify the probability amplitude α_s when s is the solution and α_s is negative (and small), while α_j for $j \neq s$ is positive. Roughly speaking, the value of α_s increases by twice the average of the amplitudes, while others are reduced. The inversion-about-the-average can be realized with the sequence of unitary matrices as below:

$$H^{\otimes n} (2|0\rangle\langle 0| - I) H^{\otimes n}$$

The first and last H are just Hadamard gates applied to each qubit. The operation in the middle requires us to design a sub-circuit that flips the probability amplitude of the component of the quantum state corresponding to the all-zero binary string. The sub-circuit can be realized by the following function, which is a multi-qubit controlled-Z which flips the probability amplitude of the component of the quantum state corresponding to the all-one binary string. Applying X gates to all qubits before and after the function realizes the sub-circuit.

```
style, def_font_ratio = load_style(self._style)
```

...



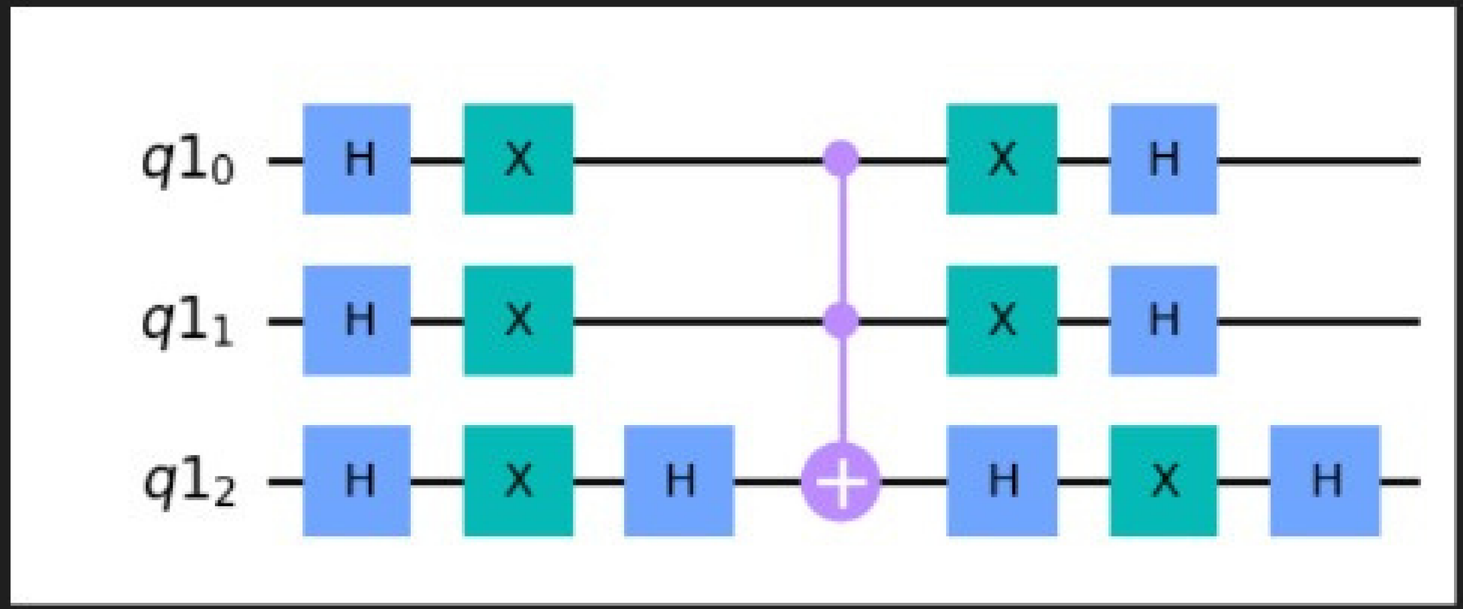

```python
def black_box_u_f(circuit, f_in, f_out, aux, n, exactly_1_3_sat_formula):
    """Circuit that computes the black-box function from f_in to f_out.

    Create a circuit that verifies whether a given exactly-1 3-SAT
    formula is satisfied by the input. The exactly-1 version
```

```
15  # -- end function
```

[6] ✓ 0.0s

```
1  qr = QuantumRegister(3)
2  qInvAvg = QuantumCircuit(qr)
3  inversion_about_average(qInvAvg, qr, 3)
4  qInvAvg.draw(output='mpl')
```

[7] ✓ 0.2s




```
"""
Grover search implemented in Qiskit.

This module contains the code necessary to run Grover search on 3
```

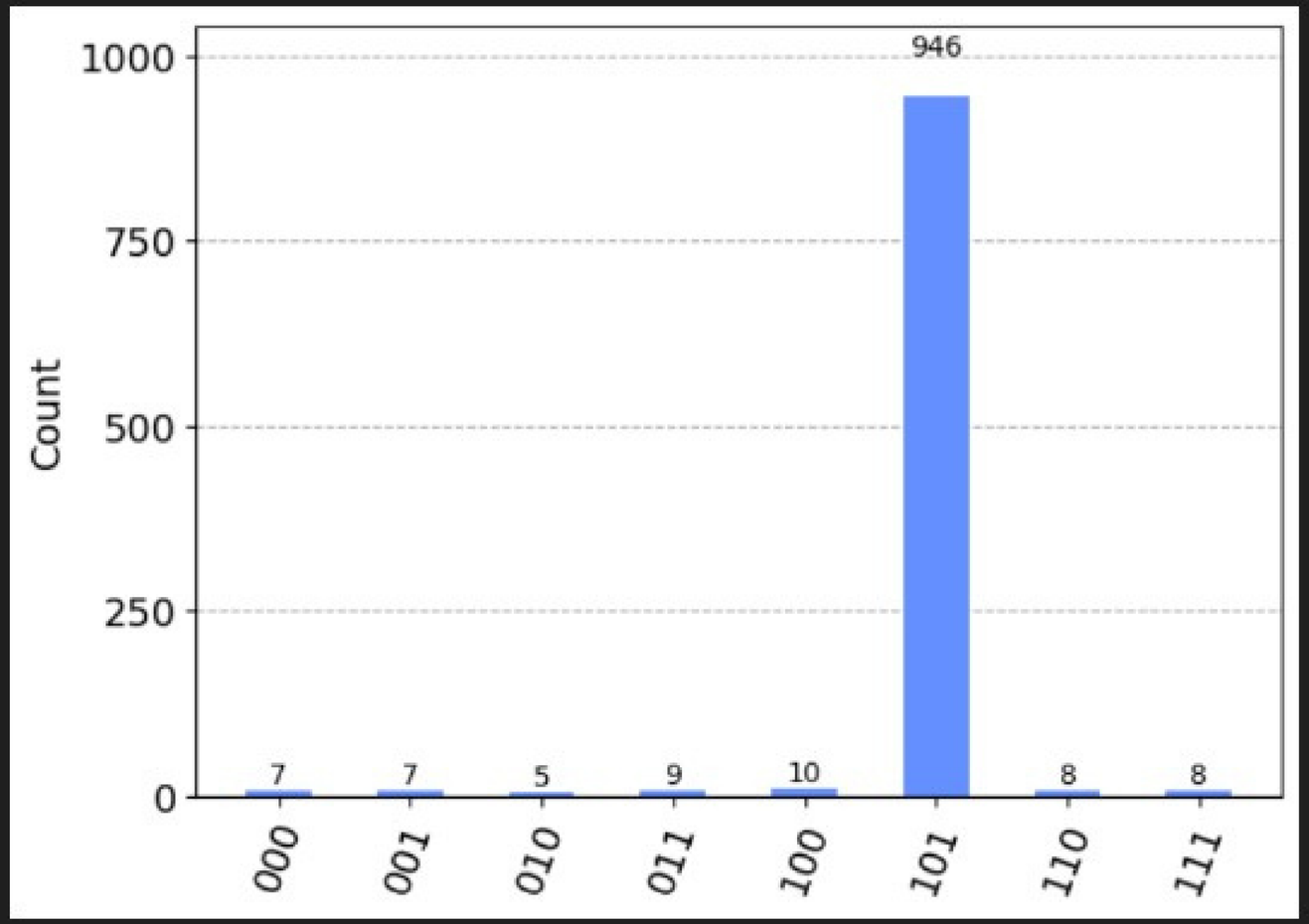
project.ipynb

college works > Qc > project.ipynb > grover.draw(output='mpl', scale=0.5)

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...

64

[8] ✓ 0.4s




```
1  from qiskit import IBMQ
2
3  API_KEY = '85cb8a7e6f047877bc60dffc222a7f0bf7bc8d2a2f230b25bd8e873eea0f8a6bee8607430d8789e05bdd26011ff0d179b6dfa58ee529b8963
```

Go Run Terminal Help

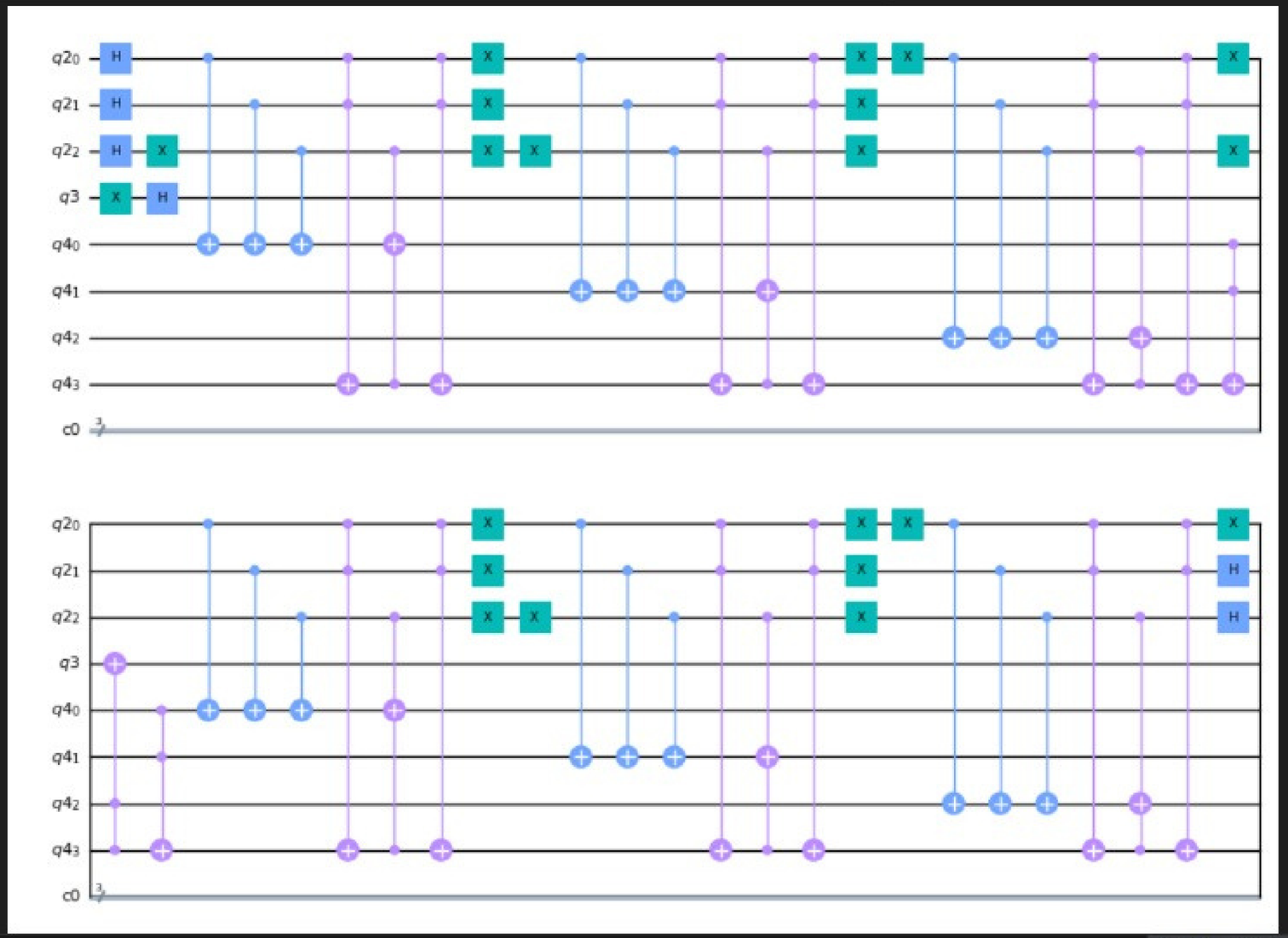
Untitled (Workspace)

project.ipynb

college works > Qc > project.ipynb > grover.draw(output='mpl', scale=0.5)

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...

Python 3.11.3



27 0

Screen Reader Optimized Cell 12 of 12 Spell tabnine starter

Go Run Terminal Help

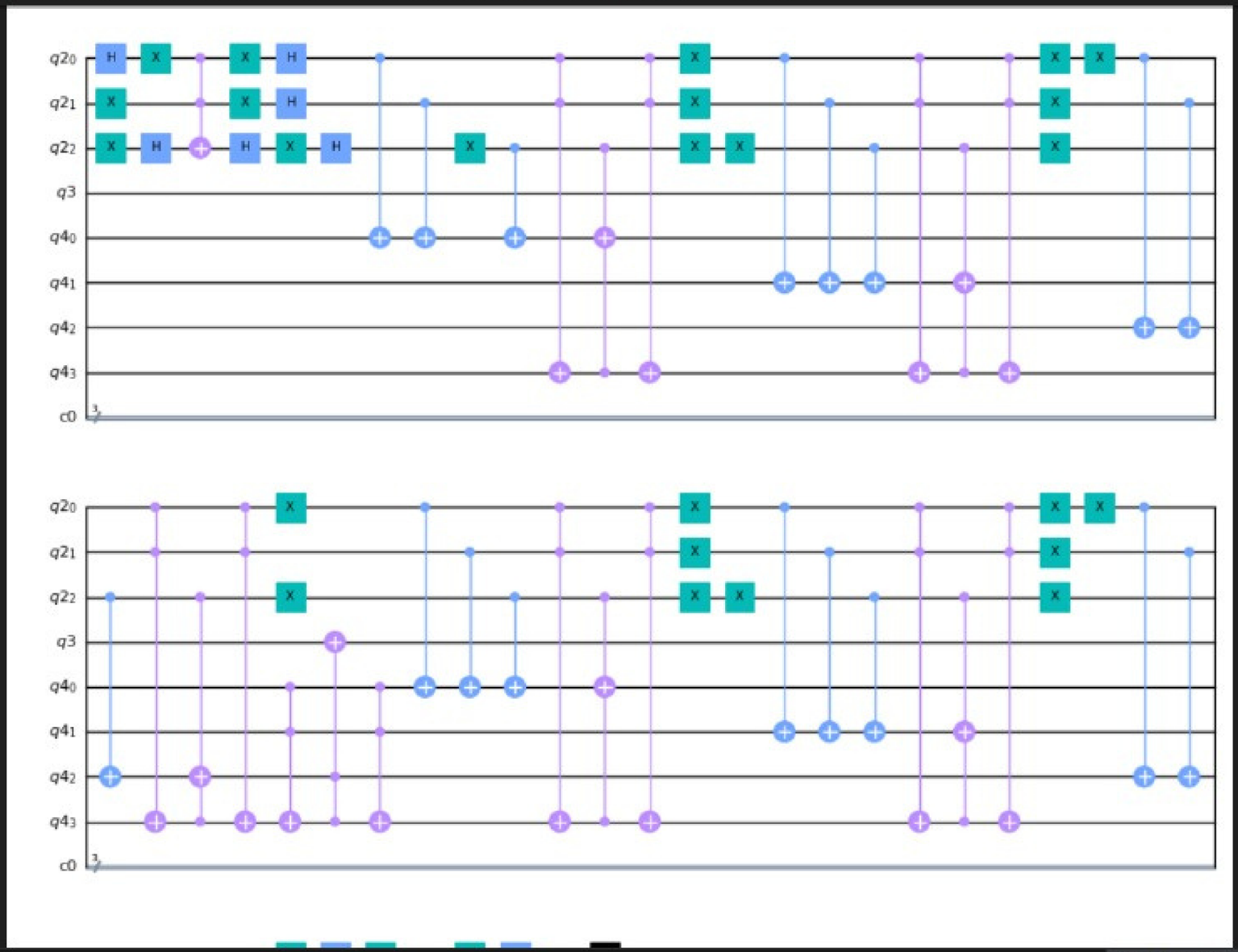
Untitled (Workspace)

project.ipynb

college works > Qc > project.ipynb > grover.draw(output='mpl', scale=0.5)

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ···

Python 3.



27 0

Screen Reader Optimized Cell 12 of 12 ✓ Spell tabnine starter

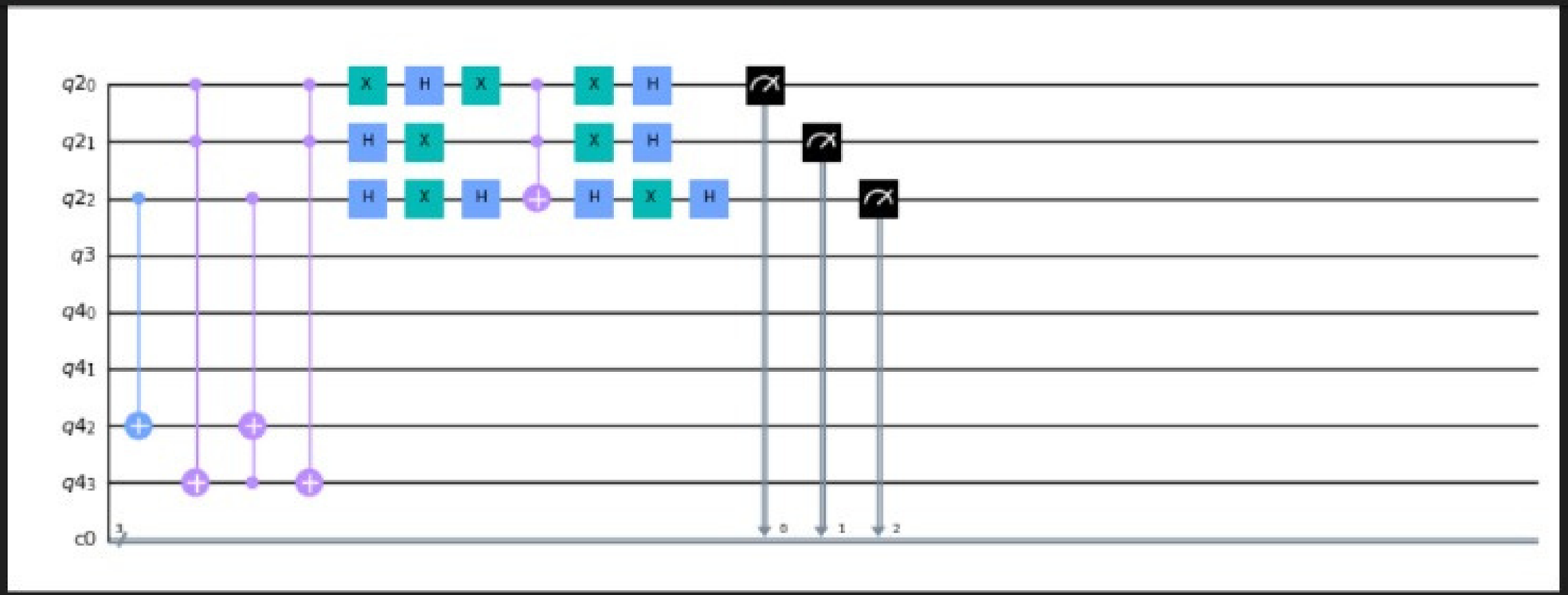
Go Run Terminal Help

Untitled (Workspace)

project.ipynb

college works > Qc > project.ipynb > grover.draw(output='mpl', scale=0.5)

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...



THANK YOU