Grover’s Quantum Algorithm CPSC 4110

By Ryan Baluyut and Jenna McDonnell

**Introduction**

This project is based on the use of Grover’s quantum algorithm in solving the satisfiability problem. To talk about the algorithm, we first need to discuss the satisfiability problem itself, specifically, Boolean satisfiability.

**Boolean Satisfiability**

The Boolean satisfiability problem “asks whether the variables of a given Boolean formula can be consistently replaced by the values TRUE or FALSE in such a way that the formula evaluates to TRUE” [1]. The input in this problem is a Boolean expression that connects multiple clauses (made up of k variables connected by the Boolean OR operator) using the Boolean AND operator. When k = 3, this looks like:

where n is the number of clauses, and each clause looks like some variation of:

the variation being that each of the k variables can be as written v or NOT v (, using the Boolean NOT operator [1]. This is important because this sets each of the variables to TRUE or FALSE, giving us a string of Boolean values combined using OR. Let’s look at an example of the Boolean satisfiability along with one solution and one non-solution to the problem:

In this case, we see that is a solution because:

A non-solution would be because:

**Theory Behind Grover’s Quantum Algorithm**

To explain Grover’s algorithm, we first need to define the “hit vector” and the “solution-smoothness property” [2]. To define the “hit vector” we say that if is a solution and if is not a solution [2]. The solution-smoothness property states two facts:

1. *All entries of that are indexed by solutions have the same value.*
2. *The entries corresponding to non-solutions also agree on their value.*

[2]

Now that we have defined and the solution-smoothness property, we can state another fact:

*Every vector in the two-dimensional subspace spanned by and*

*has the solution-smoothness property.*

[2]

Using this fact, we can now use Grover’s oracle to compute the reflection of about [2].

**The Algorithm**

To state Grover’s algorithm, we assume that , the number of solutions, is known. Then, the steps in the algorithm are as follows:

1. Initialize the vector to be the start vector .
2. Compute and .
3. Repeat the following Grover iteration times:
4. Apply the reflection of to via the Grover oracle , obtaining the vector .

(Note: here refers to the “miss vector” which returns the opposite values of the hit vector).

1. Apply the reflection of to , obtaining the new value of .
2. Measure the final state , giving a string .
3. If , we stop because we have found a solution. Otherwise, we repeat the process.

**Quantum Algorithm vs. Classical Algorithm**

Now that we have defined Grover’s algorithm, the question becomes: “Why is this better than a classical algorithm that does the same thing?” The answer, as with many quantum algorithms, is that the Quantum algorithm is faster, and in this section, we will explain why this is true.

We want to show that “Grover’s algorithm finds a member of in an expected number iterations, where , and in overall time ” [2].

Let be the angle between and before the inner loop of the algorithm has been run [2]. We will assume and starts at [2]. is the x-axis so the Grover reflection about m puts at angle [2]. The distance from to is now , so reflecting about gives us:

[2]

because doubles the distance between and and adds it to , which is [2]. This means that the reflection about and the reflection about combined are the same as rotating , our original measure of , by [2]. The resulting angle of is within of [2]. We know for an iteration of the inner loop to occur, so the success probability of is [2]. This means that , so which means that [2]. With this, we have proven that “Grover’s algorithm finds a member of in an expected number iterations, where , and in overall time ” [2].

**Using Grover’s Quantum Algorithm to Solve the Boolean Satisfiability Problem**

Now that we know what the Boolean satisfiability problem is, and why Grover’s quantum algorithm is the more efficient algorithm to solve the problem, we can use Grover’s to solve this problem on a quantum computer. For this, we used IBM’s Quantum to run our code [3]. All code snippets used below are from the IBM Quantum Lab. The code itself was taken from the Quskit textbook [1]. The Boolean satisfiability problem we are trying to solve is as follows:

[1].

The DIMACS representation of this problem is:

C example DIMACS-CNF 3-SAT

P cnf 3 3

1 2 -3 0

-1 -2 -3 0

-1 2 3 0

1. In this first block of code, we import all the necessary packages to be able to use Grover’s oracle. We then adjusted the DIMACS file to represent the above problem.

Graphical user interface, text, application

Description automatically generatedThe circuit is shown in the picture above. We studied and understood the original DIMACS file first given by qiskit textbook by creating a local file “3sat.DIMACS” on our computers. The file contained the same format and Boolean expression. Afterwards, we tried to make our own version of Boolean expression by following the Boolean expression given by one of the problems in the qiskit textbook. In the photo above, we uploaded the DIMACS file that contained the Boolean expression into IBM Quantum Lab in our local folder, this allows the program to locate the file easily. Furthermore, we also needed the DIMACS file to initiate “PhaseOracle” so that we can draw an output.

A picture containing text

Description automatically generated

In the photo shown above, the “class verifier” is one of Qiskit’s built-in tool to check if Grover’s algorithm is working correctly by producing “true” or “false”. How this works is that we initialize an input of strings such as “001” (in our example) to the line “v.is\_correect(‘string’)” to find the expression we are looking for. The verifier class checks the DIMACS file if the string we initialized was matched in the DIMACS file we used. The output produces “true” if it finds the string in the DIMACS file, else it outputs “false” if it does not match any.

Chart, bar chart

Description automatically generated

As seen from the previous section that we obtained a solution whether the string was found or not (true or false). On top of that, we also used a backend simulator from Qiskit textbook to output the results in the plot histogram. This output produces a graph of probabilities where we can see that the strings 001, 100, and 111 all have high probabilities because we provided this in the DIMACS file being associated.

Graphical user interface, text

Description automatically generated

The previous photos above showed only a simulator by providing a DIMACS file of we know what the high probabilities will be when simulated. This code above, is a prediction when we use real quantum devices that have an imperfect state.

References

[1] The Jupiter Book Community, *Solving Satisfiability Problems using Grover’s Algorithm.* Quskit. Accessed on: Nov. 30, 2021. [Online]. Available: <https://qiskit.org/textbook/ch-applications/satisfiability-grover.html>

[2] R. J. Lipton and K. W. Regan, *Introduction to Quantum Algorithms via Linear Algebra*, 2nd ed. Cambridge, MA: The MIT Press, 2021.

[3] IBM, *IBM Quantum.* Accessed on: Nov. 30, 2021. [Online]. Available: https://quantum-computing.ibm.com/