Constraint Satisfiability Problem

CSC 591/ECE 592 Final Report

Obblivignes KanchanadeviVenkataraman
Computer Science Department
North Carolina State University
Raleigh, NC, USA
okancha@ncsu.edu

Shubdeep Mohapatra
Electrical and Computer Engineering Department
North Carolina State University
Raleigh, NC, USA
smohapa@ncsu.edu

***Abstract*—We solved the problem of Constraint Satisfiability–specifically the Boolean Satisfiability Test–using both classical and, more importantly, quantum systems using Grover's algorithm. We present related work describing both algorithms, explain how both methods work in detail and our own design and implementation of the same. We conclude by stating the results of both methods and determine that the quantum algorithm can perform similarly to the classical algorithm. However, there are limitations that need to be addressed with the quantum algorithm, such as the substantial increase in error when using a real quantum machine as well as the critical ratio limitation within the 3-SAT problem.**

***Keywords*—Boolean Satisfiability Test, Conjunctive Normal Form, 3-SAT, Grover's Algorithm, Quantum Computing, Critical Ratio**

I. Introduction

One of the most fundamental problems in computer science and mathematics is the Constraint Satisfiability Problem, specifically the Boolean Satisfiability Problem. In order to solve this problem, we will explain the problem in detail, find relevant research papers to solve this problem via classical (Schoning's algorithm) and quantum (Grover's algorithm) methods. Finally, we created our own implementation of both algorithms, tested them against various Boolean expressions, and analyzed our results comparatively to generate conclusions about our findings, including identifying the best-performing algorithm.

II. Background

First, let us understand the terminologies behind the Boolean Satisfiability Problem. A boolean expression is *satisfiable* when any set of variable values will make the expression True, and vice versa for False. If the expression cannot be proven True for any set of inputs, then the expression will be False. For example, given expression (a ⋀ b) ⋁ c, this will be satisfiable whenever c is True or if a AND b are both True. However, an expression like (a ⋀ ¬a) ⋁ (b ⋀ ¬b) would be unsatisfiable because it will be impossible for a to be both True and False, making the expression unsatisfiable.

However, not all expressions are suitable to be tested for satisfiability. They must all be in a specific form, which we will call the Conjunctive Normal Form (CNF), which is

defined in Figure 1. To summarize the Figure, a boolean expression is considered in CNF form when it consists of a conjunction of n *clauses*, meaning that each clause is separated by AND symbols. Each clause, in turn, is a disjunction of k *literals*, meaning that each literal is separated by OR symbols. A literal is simply a variable that can either be negated using the NOT symbol or not (i.e. both a and ¬a would be considered a literal). From this point onwards, we will refer to CNF form boolean expressions as *sentences*. With the use of DeMorgan's law, any boolean expression can be converted to CNF form [1, 3].

Given n clauses and k literals, the Conjunctive Normal Form (CNF) is defined as

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

where

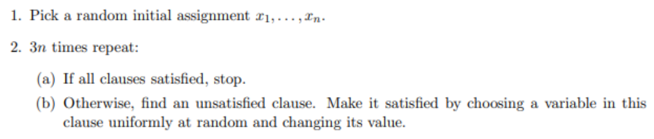
$$C_1 = a_1^1 \vee a_2^1 \vee a_3^1 \vee \ldots \vee a_k^1, C_2 = a_1^2 \vee a_2^2 \vee a_3^2 \vee \ldots \vee a_k^2, \ldots C_n = a_1^n \vee a_2^n \vee a_3^n \vee \ldots \vee a_k^n$$

Fig. 1. Definition of CNF-form boolean expression

There are many forms of satisfiability (more commonly abbreviated as SAT) problems, which are of type n-SAT, where n is the maximum number of literals for each clause. A few of the most well-known examples are 2-SAT and 3-SAT, where the former has been proven to be solvable in polynomial time. However, the latter has been proven to be NP-complete, meaning it is a non-deterministic polynomial time problem. This was first proven by Stephen A. Cook in 1971 through the Cook-Levin Theorem. In this paper, we discuss a classical solution, which was originally designed by Schoning, and a quantum solution, where we used Grover's algorithm to solve the 3-SAT problem [2, 3].

III. Classical Design/Methodology

In order to form a baseline model of the performance of this problem, we identified a classical algorithm that we can use to compare its performance to the quantum algorithm. One of the most optimized classical algorithms we discovered was a variation of Schoning's algorithm, which is stated in Figure 2 below. It states to pick a random assignment (i.e. either 0 or 1) for each of the input variables $\{x_1, x_2, x_3,..., x_n\}$ where n is the number of variables given. Next, we will need to iterate up to 3n times, where for each iteration we check whether the current set of inputs will make the sentence True. If it is True, we stop iterating; however, if it is not True, we find an unsatisfied clause and modify one of the input variables uniformly and at random and repeat the loop. According to the paper that presented it, the time complexity of this algorithm is about $O(1.329..^{n})$ [2].

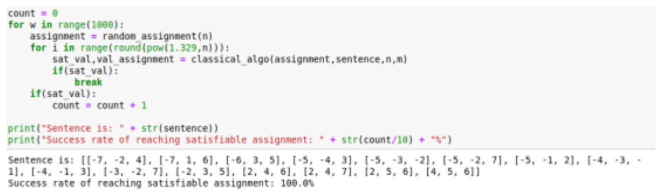


1. Pick a random initial assignment $x_1, \ldots, x_n$.
2. $3n$ times repeat:
   (a) If all clauses satisfied, stop.
   (b) Otherwise, find an unsatisfied clause. Make it satisfied by choosing a variable in this clause uniformly at random and changing its value.

Fig. 2. Classical Algorithm [2]

IV. Classical Results

After implementing this using the Python language, we were able to determine that this implementation worked fairly well, and we were able to determine a valid set of inputs for each variable to make any given sentence satisfiable, and if none were found, unsatisfiable, as you can see from Figure 3 below. As you can see, for a given set of input variables $\{x_1, x_2, x_3,..., x_n\}$, each clause is entered as an array of literals (which are either negated or not) where each variable is given its corresponding subscript numbers, and the sentence itself is an array of clauses. From this implementation, we were able to determine a sentence's success rate, meaning whether a literal assignment would be achieved to make the sentence True.



```
count = 0
for w in range(1000):
    assignment = random_assignment(n)
    for i in range(round(pow(1.329,n))):
        sat_val,val_assignment = classical_algo(assignment,sentence,n,m)
        if(sat_val):
            break
    if(sat_val):
        count = count + 1

print("Sentence is: " + str(sentence))
print("Success rate of reaching satisfiable assignment: " + str(count/10) + "%")
```

```
Sentence is: [[-7, -2, 4], [-7, 1, 6], [-6, 3, 5], [-5, -4, 3], [-5, -3, -2], [-5, -2, 7], [-5, -1, 2], [-4, -3, -
1], [-4, -1, 3], [-3, -2, 7], [-2, 3, 5], [2, 4, 6], [2, 4, 7], [2, 5, 6], [4, 5, 6]]
Success rate of reaching satisfiable assignment: 100.0%
```

Fig. 3. Classical Algorithm Results

V. Quantum Design/Methodology

One of the best quantum algorithms that we found could work for our particular problem is an altered form of the Grover's algorithm, which is ideal for "finding a solution to a disordered array" [3]. As an overview, for an array of n indices, Grover's algorithm enables us to have a time complexity of O($\sqrt{n}$), whereas the traditional classical algorithm has a time complexity of O(n), thereby giving us the opportunity to speed-up the computational efficiency by a square-root factor. According to [2], the time complexity of the quantum implementation is about $O(1.153..^{n})$, representing an equivalent square-root speedup over the best classical method discussed in section III.

This particular solution, provided to us by Runkai Zhang, Jing Chen, and Huiling Zhao, modifies the original Grover's algorithm to fit within the scope of the Boolean Satisfiability Test. As the paper mentions, we have constructed our entire model using the Qiskit library [4]. We will describe each aspect of our Grover's algorithm implementation below.

A. *Grover's Algorithm Input Preprocessing*

First, we will transform each of the clauses within the sentence into a binary string in order to make each literal true (i.e. making literals that have the ¬ symbol be 0, and those without them be 1). We will also need to define a circuit, which given an input of n variables, will consist of (1) n input qubits, n ancillary qubits, 1 output qubit, and 1 classical output bit. First, we will need to put each of the classical bits into quantum bits (or *qubits)* via quantum superposition using the Hadamard (H) gate. Next, we will perform the X gate to the ancillary qubit so that they are in the |1> qubit state, assuming that our chosen input is correct. Now, we are ready to discuss the next part of Grover's algorithm: the oracle.

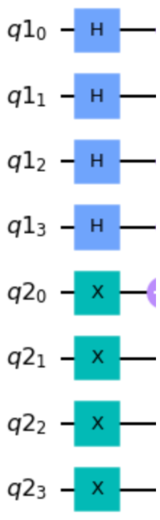


Fig. 4. Quantum Circuit Input Preprocessing for 3 Inputs

B. *Grover's Algorithm Oracle*

Similar to Grover's algorithm, we will need to implement the oracle which aims to "flip the sign of states we want" [3], and store those results into each corresponding clause's qubit. This can be done using a multi-controlled X gate, otherwise known as a Toffoli gate, wherein it will only flip the qubit when all of the control qubits are of state |1> (see Figure 5). The Toffoli gate's control bits will be each of the input qubits for each clause and the target qubit will be each clause's corresponding output qubit. In this problem, since we are trying to remove all the cases where the solution is False in order to find when the solution is True, we will flip the qubit when its equivalent binary bit is 1 (and not flip it otherwise), before inputting it in the Toffoli gate. Since all qubits are initializes to be |1>, this will store qubit states of |0> for the ancillary qubits whose corresponding clause is True and vice versa.

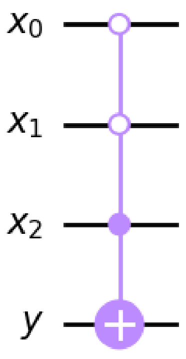


Fig. 5. Multi-controlled Toffoli Gate for clause $x_0 \lor x_1 \lor \neg x_2$

After proceeding to do this for each individual qubit, we will store the value of this particular input sequence using an output qubit, wherein we will use an additional multi-control Toffoli gate that uses all of the ancillary qubits as the control qubits, and the output qubit as the target qubit. Since the qubit will then only be |1> when the sentence is satisfied and |0> otherwise, we will need to perform an X gate to reverse this behavior for the diffusion portion of the circuit, and then perform a H gate again to convert this qubit into a superposition state of its own: either |+> if the sentence is True or |-> if the sentence is False.

Finally, we will need to uncompute the circuit by performing the same multi-control Toffoli gates to the ancillary qubits but in the reverse order in order to reinstate the original state of the ancillary qubits. An example of this is provided in Figure 6.

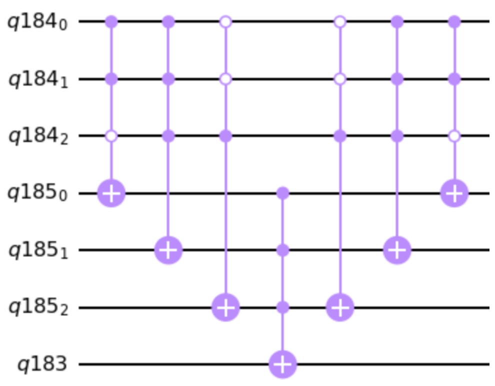


Fig. 6. Oracle for the sentence [[-1,-2,3],[-1,-2,-3],[1,2,-3]]

C. *Grover's Algorithm Diffuser*

In Grover's algorithm, the diffuser's main goal is to amplify the input strings that satisfy whatever you are trying to search for; in this case, what would make the Binary Satisfiability Test True. In order to do so, we will need to apply the H and X gates, respectively, to all input qubits.

Next, we will need to apply a final multi-control Toffoli gate that takes the output qubit as the target qubit, and the input qubits as the control qubit. This amplifies the output of the target state if the qubit state is |->, whilst suppressing output of the target state if the qubit state is |+>, hence identifying whether this particular input sequence has met the Boolean Satisfiability Test. Finally, we uncompute the operations that we have performed to input qubits by performing the H and X gates. An example of this is shown in Figure 7 below.

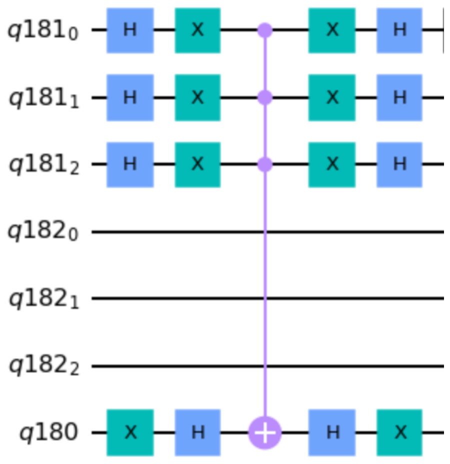


Fig. 7. Grover's Algorithm Diffuser Circuit

After applying each of these aspects of Grover's algorithm, we perform measurement on each of the input qubits, which will convert the quantum state to a classical bit, which will allow us to interpret the answer that the circuit has found. The full quantum circuit that we constructed can be viewed below.

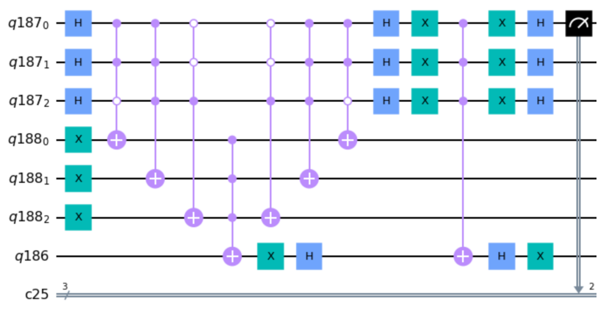


Fig. 8. Full circuit for the sentence [[-1,-2,3],[-1,-2,-3],[1,2,-3]]

VI. Quantum Results

From the results that we obtained over several experiments, we are able to successfully prove that our quantum circuit was able to identify the binary strings that would meet the given sentence that we provided. We will share two examples that we ran in a simulated circuit using the Qiskit library (as mentioned earlier) and a total number of 1024 shots, or repetitions of each circuit.

A. Representation of Sentences

To represent a sentence in Qiskit, we have renamed each positive literal x_i to an integer i. And for each negative literal ~x_i to -i. So, for example x_1 V x_2 V ~x_3 can be represented as [1,2,-3].

B. Simulation results

Our first example was with the sentence [[1, -3, -4], [2, -3, -4], [1, 2, 3], [-1, 2, -3]] (1), where we were able to generate a histogram of our results shown in Figure 4. Likewise, we also considered a sentence [[-1, -2, 3], [-1, -2, -3], [1, 2, 3]] (2), where we generated the histogram shown in Figure 5.

These figures show all possible classical bit inputs ("input strings") that, after running the quantum circuit, were found to satisfy the sentence, where input strings that satisfied the sentence more often had higher occurrences than those that did not as frequently. For example, in Figure 4, we found that input strings "0001," "0100," "0101," "0110," "1000," "1001," "1100," "1101," "1110," and "1111" all satisfied sentence (1), whereas all other input string combinations were unable to do so. Likewise in Figure 5, we determined that input strings "001," "010," "011," "100," and "101" were able to satisfy sentence (2), which are indeed correct. Moreover, these results show that the circuit was able to clearly distinguish between satisfiable and unsatisfiable inputs without any significant noise, due to such a huge contrast between the corresponding types in the terms of the number of occurrences within each histogram.

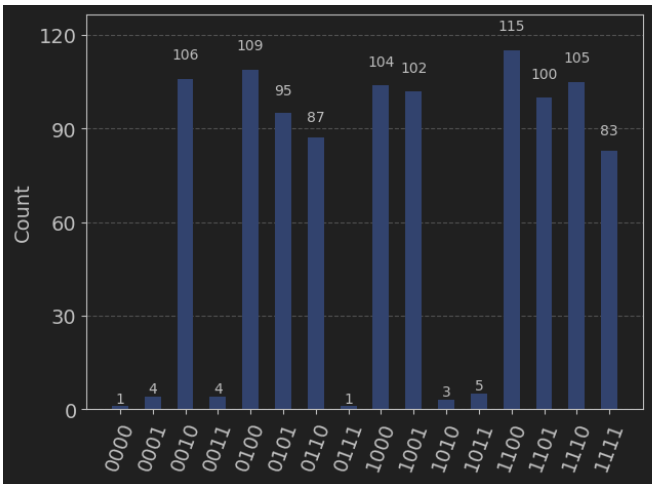


Fig. 9. Quantum Algorithm Simulation Results for Sentence (1)

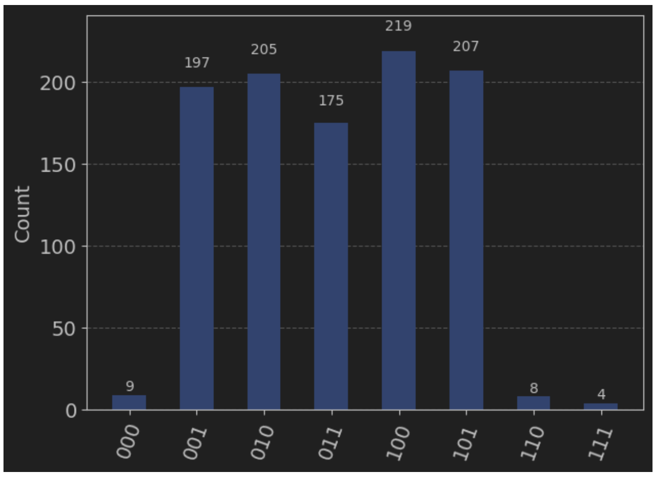


Fig. 10. Quantum Algorithm Simulation Results for Sentence (2)

Due to the insignificant amount of noise in our results, we also considered running these experiments on a noisy quantum machine to compare any potential differences that they might have. For our experiment, we used IBM Q's quantum computers, specifically "IBM_nacza," which is a 127-qubit system [5]. Using the same number of shots, we generated a histogram of our results for sentence (1) in Figure 11 and sentence (2) in Figure 12.

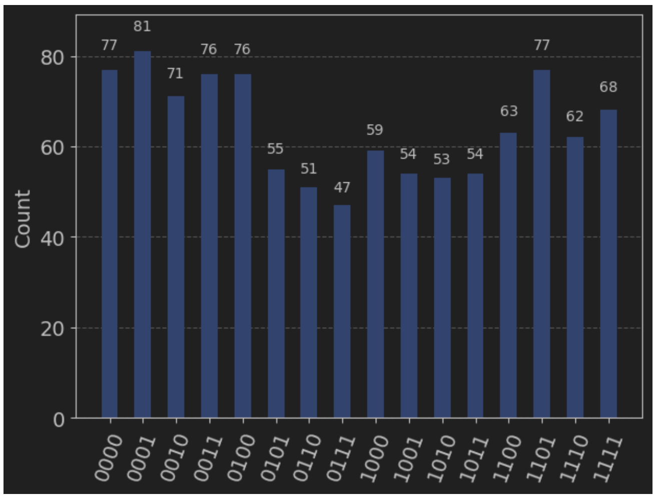


Fig. 11. Quantum Algorithm Noisy Results for Sentence (1)

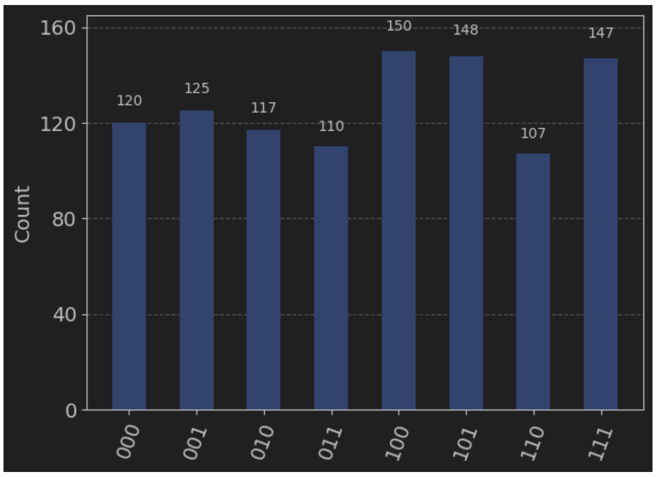


Fig. 12. Quantum Algorithm Noisy Results for Sentence (2)

From both of these figures, we can easily determine that the circuit is unable to find any solution that would satisfy the sentence due to the increase in noise. We can attribute this to the several multi-controlled Toffoli gates that we used in the circuit as they can prove to be quite noisy in a real-world context. Unfortunately, with our current implementation, this problem will only become more prevalent with more input variables present in the sentence. In order to efficiently implement such an algorithm in a real-world context, we will need to have a way to (a) reduce the number of multi-controlled Toffoli gates within our circuit or (b) improve quantum hardware performance on multi-controlled Toffoli gates.

VII. Quantum Analysis: Critical Ratio

A very common concept in the context of the 3-SAT problem is the *critical ratio* m/n , ie , the ratio between the number of clauses (m) and the number of variables (n). We tested our circuit for a varying number of clauses for a sentence with 4 variables.

Case 1 :- n = 4, m = 4
Sentence :- [[-4, -3, -2], [-4, -2, 1], [-4, 1, 2], [2, 3, 4]]

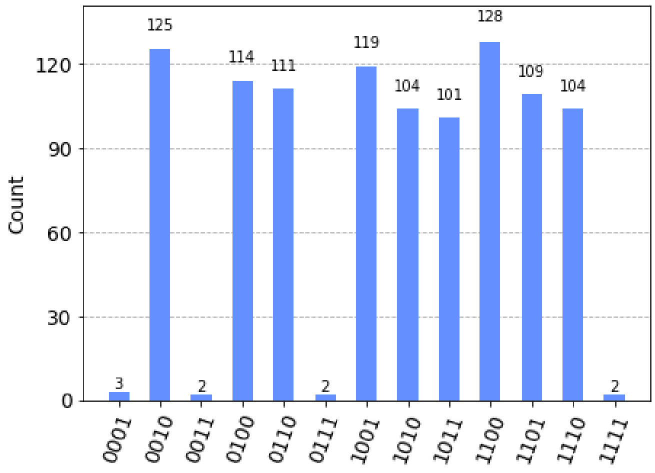


Fig. 13. Quantum Algorithm Simulation Results for m/n = 1

Case 2:- n = 4, m= 8
Sentence :- [[-4, -1, 3], [-4, 1, 3], [-3, -1, 4], [-2, -1, 3], [-2, 1, 3], [-2, 3, 4], [-1, 2, 3], [-1, 3, 4]]

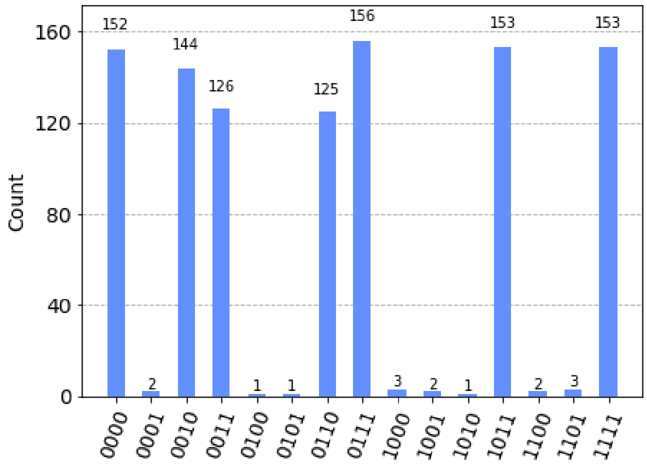


Fig. 14. Quantum Algorithm Simulation Results for m/n = 2

For Fig 10 and Fig 11, the quantum circuit is able to come to a suitable answer majority of the time. The bit strings measured the least amount of times are the assignments which make the sentences false.

Case 3:- n = 4,m = 12
Sentence :- [[-4, -3, -2], [-4, -3, -1], [-4, -2, -1], [-4, -2, 3], [-3, -2, 4], [-3, -1, 2], [-3, -1, 4], [-3, 1, 4], [-3, 2, 4], [-2, -1, 3], [-2, -1, 4], [2, 3, 4]]

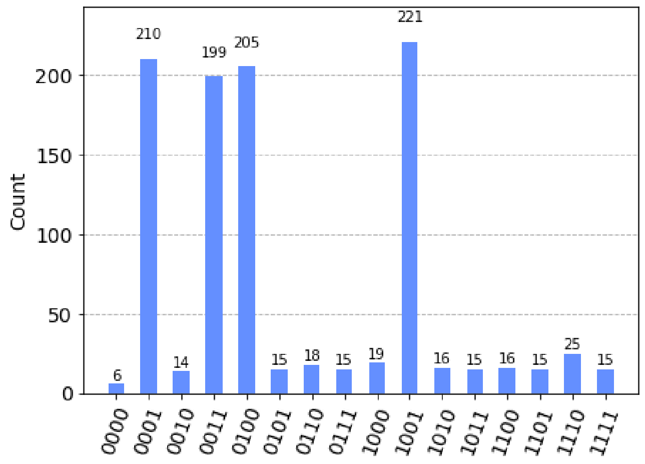


Fig. 15. Quantum Algorithm Simulation Results for m/n = 3

Case 4:- n = 4,m =16
Sentence :- [[-4, -3, -1], [-4, -3, 2], [-4, -2, 3], [-4, -1, 2], [-4, -1, 3], [-4, 1, 2], [-4, 1, 3], [-4, 2, 3], [-3, -2, 1], [-3, -1, 2], [-3, -1, 4], [-3, 2, 4], [-2, -1, 3], [-2, 1, 3], [-2, 3, 4], [1, 3, 4]]

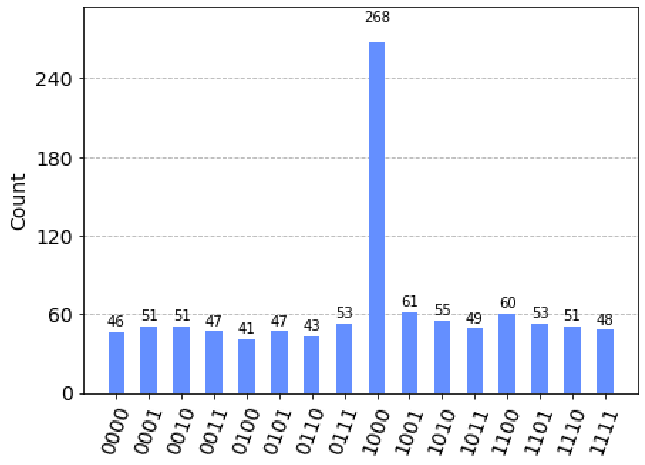


Fig. 16. Quantum Algorithm Simulation Results for m/n = 4

For Fig 12 and Fig 13, the quantum circuit is able to come to a suitable answer majority of the time. The bit strings measured the least amount of times are the assignments which make the sentences false. However, if you look at Fig 13, the probability of measuring state '1000' is distinctly the highest as it is the only solution of the sentences. As the m/n ratio approaches the critical ratio, finding multiple satisfying assignments becomes difficult as the number of clauses with positive and negative literals of the same variable increases.

Case 5:- n =4,m =18
Sentence:- [[-4, -3, -2], [-4, -3, 1], [-4, -2, 1], [-4, -2, 3], [-4, -1, 2], [-4, 1, 2], [-4, 1, 3], [-4, 2, 3], [-3, -1, 4], [-3, 1, 4], [-2, -1, 3], [-2, -1, 4], [-2, 1, 3], [-2, 1, 4], [-1, 3, 4], [1, 2, 3], [1, 3, 4], [2, 3, 4]]

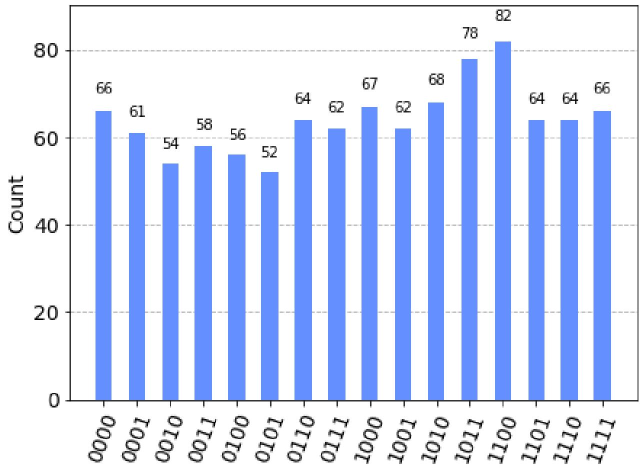


Fig. 17. Quantum Algorithm Simulation Results for m/n = 4.25

In Fig 14, the quantum circuit is unable to come to a discriminating solution . This is because we have reached the critical ratio. The critical ratio for a 3 sat problem is ~4.25. At this point of inflection , coming to a discriminating solution for the sentence becomes improbable. Our quantum circuit confirms that this is indeed the critical ratio for a 3-SAT problem.

VIII. Conclusion

This report summarizes the methodology of using Grovers's algorithm to solve the 3-SAT problem. The essential component of the resulting quantum circuit is the oracle which provides us a way to check the satisfiability of the sentence given different variable assignments. The oracle takes advantage of the custom control Toffoli gates provided in Qiskit. We further analyze the correctness of the quantum circuit by verifying the value of the critical ratio for 3-SAT sentences by checking the assignment values for different sentences by varying the number of clauses.

References

[1] S. Datta, "SAT and 3-sat - cook-levin theorem," Baeldung on Computer Science, https://www.baeldung.com/cs/cook-levin-theorem-3sat (accessed

Nov. 29, 2023).

[2] A. Ambainis, “Quantum Search Algorithms”, Apr. 2005. arXiv:quant-ph/0504012v1

[3] R. Zhang, J. Chen, H. Zhao. “Procedure of Solving 3-SAT Problem by Combining Quantum Search Algorithm and DPLL Algorithm” in Computing, Performance and Communication Systems, vol. 4, p. 14-24, 2020. doi: 10.23977/cpcs.2020.41003

[4] Qiskit, “Qiskit/Qiskit: Qiskit is an open-source SDK for working with quantum computers at the level of extended quantum circuits, operators, and primitives.,” GitHub, https://github.com/Qiskit/qiskit (accessed Dec. 5, 2023).

[5] IBM, “IBM Quantum Platform,” IBM Quantum, https://www.ibm.com/quantum (accessed Dec. 5, 2023).