Quantum Algorithm for Maximum Satisfiability

Abdirahman Alasow
Department of Electrical & Computer Engineering
Portland State University
Portland, OR, USA
alasow@pdx.edu

Marek Perkowski
Department of Electrical & Computer Engineering
Portland State University
Portland, OR, USA
mperkows@ee.pdx.edu

2022 IEEE 52nd International Symposium on Multiple-Valued Logic (ISMVL) | 978-1-6654-2395-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISMVL52857.2022.00012

Abstract— Satisfiability (SAT) problem in engineering and computer science is to find the set of assignment values of input variables for the given Boolean function that evaluate this function to TRUE or prove that such satisfying values do not exist. For POS SAT Problem, we propose a novel quantum algorithm for the maximum satisfiability (MAX-SAT) which returns the maximum number of OR terms that are satisfied for the SAT-unsatisfiable function, providing us with the information how far the given Boolean function is from the SAT satisfaction. We use Grover's algorithm with a new block called quantum counter in the oracle circuit.

Keywords— Satisfiability, maximum satisfiability, quantum counter, Grover search algorithm, quantum circuit.

I. Introduction

A. Boolean Satisfiability

Boolean or propositional-logic expressions are built from input variables using operators AND, OR, EXOR and NOT. Satisfiability (SAT) problem is the problem of determining if there exists a set of values for the variables which makes a given Boolean formula $f(x_1, x_2, \dots, x_n)$ evaluate to TRUE (1). For example, the Boolean function $f(a,b,c) = (a+b+\bar{c})(\bar{a}+\bar{b}+c)(b+c)$ is satisfiable because when $c = 1$ and either a or b is equal to 1, then $f(a,b,c)$ evaluates to 1. Satisfiability expression (circuit) is often expressed in the Product of Sum (POS) form, which is a logical AND of OR terms, each OR term being an inclusive sum of literals. As an example, the POS SAT function $f(a,b) = (a+b)(\bar{a}+\bar{b})(\bar{a}+b)(a+\bar{b})$ is not satisfiable, because for no binary assignment of values for variables *a* and *b,* $f(a,b)$ would evaluate to 1.

Satisfiability problems have a wide range of applications such as: model checking in electronic design automation (EDA) [1], automatic test pattern generation (ATPG) [2], software and hardware verification [3], and circuit design [4]. Satisfiability problems have also many applications in Artificial Intelligence [5], robotics and electronic design. Based on Cook's theorem [6], satisfiability is an NP-complete problem. Solving a satisfiability problem involving many variables and terms using traditional algorithms is computationally expensive.

B. Maximum Satisfiability

Maximum satisfiability (MAX-SAT) seeks to satisfy the maximum number of constraints of a given Boolean function. MAX-SAT is an optimization version of the SAT problem. In some real-life problems suppose a Boolean function in the POS form contains thousands of sum (OR) terms (called also clauses), the MAX-SAT problem is to examine maximum numbers of terms that are satisfied. For example, $f(a,b,c,\dots,N) = (a+b+\bar{c})(\bar{a}+\bar{b}+c)(b+c)\dots(\dots,\dots) = 1$. The function f is true for a binary assignment of values to variables $a, b, c, \dots, N$ for which all terms are true. This is the SAT satisfiability. In contrast, the goal of MAX-SAT is not only finding the decision satisfied/unsatisfied (yes/no), but also provides the maximum number of terms (clauses) that are satisfied with the actual satisfying assignment values for the variables in case that the formula is not SAT satisfiable. The MAX-SAT is considered to be an NP-hard problem [15].

There are several extensions and modifications to the MAX-SAT problem formulated as above. For instance, sometimes not all constraints of a problem can be satisfied, but some of them must be satisfied. In such a case MAX-SAT constraints can be divided into two set of clauses:

- ***Hard clauses***: The constraints that must be satisfied.
- ***Soft clauses***: The constraints that may or may not be satisfied, but we want to satisfy as many as possible.

There are three main variants of MAX-SATs [24, 28]:

1) ***Weighted MAX-SAT****:* Each clause has associated weight cost and the objective is to maximize the sum of the weights of the satisfied clauses.

2) ***Partial MAX-SAT****:* Finds the assignment values for the variables that must be satisfied for all hard clauses and must be maximized on the soft clauses.

3) ***Weighted partial MAX-SAT*** is a combination of the partial and weighted MAX-SAT.

II. Related Works

There are many optimization problems related to MAX-SAT, variants and applications of MAX-SAT. Some of the successful applications used for MAX-SAT are data analysis and machine learning, planning and scheduling, verification and security, bioinformatics, and combinatorial optimization [24].

A. Classical Algorithms for MAX-SAT

There are many classical algorithms for solving MAX-SAT problems: exact algorithms, stochastic local search algorithms [16, 17, 30], evolutionary algorithms [18, 19] and hybrids of local search and evolutionary algorithms [20, 21]. Exact algorithms are often used for small or medium size problems

2378-2226/22/$31.00 ©2022 IEEE
DOI 10.1109/ISMVL52857.2022.00012

which can be easily verified as satisfied or unsatisfied. The exact algorithms are based on Davis-Putnam-Logemann-Loveland algorithm (DPLL) [22]; example being the Branch-and-Bound algorithm [8, 29] which represents the search space of all possible value assignments to variables as a search tree. Branch-and-Bound explores the branch of the tree and creates new formulas with partial assignments in the internal nodes until the solution is found. The solution is stored in the leaf nodes which are bound to prevent unnecessary branches. For large size problems are used stochastic local search algorithms and evolutionary algorithms which can potentially provide a high quality solution [20, 25].

B. Quantum Algorithms for MAX-SAT

MAX-SAT is an NP-hard problem and is one of the most widely studied optimization problems in classically algorithms. These NP-hard problems can be potentially solved by quantum algorithms which would offer significant improvements over the classical algorithms, assuming existence of quantum computers with sufficiently many qubits.

There is some active research to solve the SAT and MAX-SAT problems using the currently available quantum computers, especially the D-wave quantum annealer (QA) systems [9]. The SAT and MAX-SAT are encoded into Quadratic Unconstrained Binary Optimization (QUBO) compatible with the quantum annealer architecture. QUBO is a mathematical class of problems expressed in binary variables as linear or pairwise quadratic terms which may include constraints.

Practical MAX-SAT problems contain hundreds of variables and terms/clauses, which cannot be handled by the current available quantum computers. Thus, due to the limited number of qubits available, some algorithms suggested to reduce the number of qubits. For instance, the quantum cooperative search algorithm for 3-SAT [10] proposed Grover's search algorithm combined with a classical algorithm that decreases the total number of variables by replacing some qubits with classical bits. However, still the number of needed ancilla qubits is equal to the number of terms when applied to POS 3-SAT problems.

We propose a new quantum circuit using Grover's search algorithm which can be applied to both SAT and MAX-SAT problems with the reduced quantum cost. The main idea is to avoid large Toffoli gates that have high quantum cost and lead to decoherence. Our novel quantum oracle circuit design requires less logical qubits to implement the maximum satisfiability problem. This is based on replacing large AND gate collecting results from clauses by a quantum counter that counts the number of satisfied clauses inside the SAT oracle upgraded MAX-SAT oracle. Because modern quantum computer and simulators have limited total number of qubits, our quantum algorithm allows to to solve larger MAX-SAT problems. However, because of limited number of qubits it is not competing with modern software MAX-SAT solvers.

III. Preliminaries

Quantum counter can be built from NOT, CNOT and multi-qubit Toffoli gates, our design uses Peres gates, because the design with Peres gates leads in many cases to substantial circuit cost reduction. Peres gates are built from truly quantum gates CV and CV$^+$ and other Controlled-Nth Root of NOT gates, which requires to explain these gates first.

A. Controlled-Nth Root of NOT

Mathematically, a quantum gate with n qubit input can be represented as a 2^n x 2^n unitary matrix. N-th root of NOT gate can be constructed from matrix representation as follows:

$$\sqrt[n]{NOT} = \frac{1}{2}\begin{vmatrix} 1+e^{\frac{i\pi}{n}} & 1-e^{\frac{i\pi}{n}} \\ 1-e^{\frac{i\pi}{n}} & 1+e^{\frac{i\pi}{n}} \end{vmatrix}.$$

There exists also the "Inverse of N-th root of NOT gate" in which the plus and minus signs are reversed. Below given are notations and properties that will be used in the paper to design larger Peres gates: V gate $= \sqrt{NOT}$ gate. $V^\dagger$ gate is inverse of V gate. Where $V^\dagger$ is called V dagger or conjugate of V. $W = \sqrt{V} = \sqrt[4]{NOT}$; $G = \sqrt{W} = \sqrt[8]{NOT}$; $VV = NOT$; $VV^\dagger = I$; $WW = V$; $GG = W$. The Controlled-Nth root of NOT gate is a 2-qubit gate, where the first qubit is the control, and the second qubit is the target. When the control is one ($|1\rangle$) then the target qubit calculates the Nth root of NOT gate applied to its input value. Otherwise, with control $|0\rangle$ the target qubit is not changed. The matrix representation of controlled-Nth root of NOT gate is:

$$\text{Controlled-}\sqrt[n]{NOT} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1+e^{\frac{i\pi}{n}}}{2} & \frac{1-e^{\frac{i\pi}{n}}}{2} \\ 0 & 0 & \frac{1-e^{\frac{i\pi}{n}}}{2} & \frac{1+e^{\frac{i\pi}{n}}}{2} \end{vmatrix}.$$

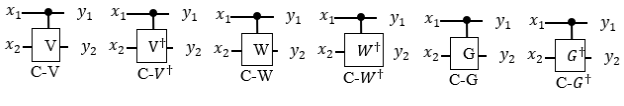

Fig. 1. Symbols of Controlled-nth root of NOT gate and their inverse.

Fig. 1 shows examples of various controlled-Nth root of NOT gates that we will use in our design of large Peres gates used in counters.

B. Quantum Cost

Quantum cost of a quantum circuit is the number of elementary quantum gates used to build the circuit. The elementary quantum gates are primitive gates which are 1*1 and 2*2 reversible gates. The cost of the primitive gates is equal to 1, therefore, the quantum cost is just the number of primitive gates. For illustration, these are three elementary quantum gates which are used to calculate the quantum cost: NOT, controlled-nth root of NOT, and CNOT gates where cost of each gate is equal to 1. (There are some more accurate characterizations of costs of primitive quantum gates [26] but for this paper we use the approximate costs defined as above).

Toffoli gate could be built using controlled-nth root of NOT gate [12]. A 3-bit Toffoli gate from Fig. 2 has two control qubits and one target qubit and is built from controlled V/V† gates and CNOT gates. The quantum cost of the 3-bit Toffoli gate is 5. The generalized formula for quantum cost of m-control Toffoli gate [11] is equal to $2^{m+1} - 3$.

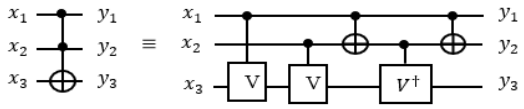

Fig. 2. 3-bit Toffoli gate represented as controlled -V/ $V^\dagger$ and CNOT gates.

C. Peres Gate

The Peres gate [7] can be characterized as a sequence of n-Toffoli followed by Feynman (CNOT) gates. For instance, a 3-bit Peres gate consists of a 3-bit Toffoli and a CNOT gates (Fig.3 I). When the 3-bit Toffoli and CNOT gates are implemented separately the cost would be 6 (Fig. 3II). However, the 3-bit Peres gate costs 4 because the adjacent CNOT gates cancel each other. Thus, the Peres gates are used for quantum cost reduction of quantum circuits, and for blocks of the iterative counter in this paper specifically.

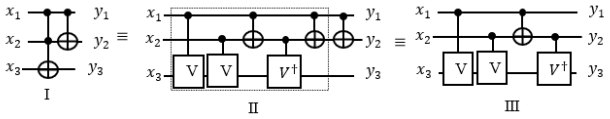

Fig. 3. (I) 3-bit Peres gate (II) decomposed Toffoli gate with CNOT. (III) 3-bit Peres gate and its representation using controlled -V/ $V^\dagger$ and CNOT gates

The Fig.3 (III):

- If x_1 is 1 and x_2 is equal to 0 or vice versa, then the transformation applied to x_3 and one of the V-gate will become active and the other one will be inactive which behaving as the identity. Also, CNOT will become active which produces 1 that will activate $V^\dagger$-gate, thus $VV^\dagger = I$.
- If both x_1 and x_2 are equal to 1, then the transformation applied to x_3 and two of the V-gate will become active. Also, CNOT will become inactive which produces 0 that will inactivate $V^\dagger$-gate, thus $VV = NOT$.
- If both x_1 and x_2 are equal to 0, then no transformation is applied on the gates.

In general, n-controlled Peres gate consist of n-1 Toffoli and one CNOT gate. Each n-qubit Peres gate can be built recursively using the n-1 Peres gate block and few additional controlled gates. The reader can appreciate this recursive way of building counter blocks of any size by analyzing Fig. 4 in which a 4-controlled gate at the right uses the 3-controlled Peres gate in four upper qubits.

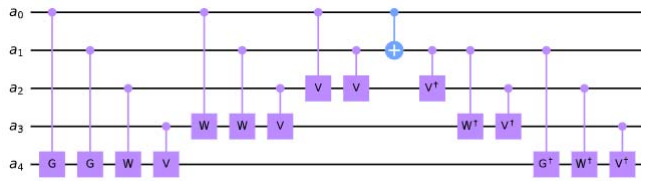

Fig. 4. A Peres gate realized on five qubits

As shown in Fig. 4, the 5-qubit Peres gate uses the 4-qubit Peres gate as its sub-circuit. Fig. 3 and 4 illustrate that the general formula for the quantum cost of m-controlled Peres gate [13] is equal to m^2. For larger design, the Peres gate can be designed as recursive blocks as shown in Fig. 5.

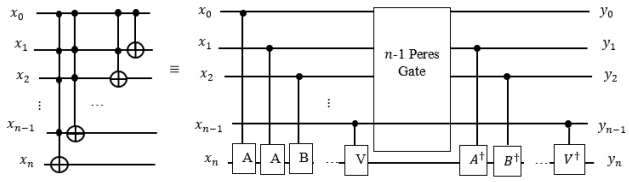

Fig. 5. Generalized Peres gate realized on n qubits

IV. Quantum Algorithm for Maximum Satisfiability

In traditional Grover's algorithm oracles are composed of Toffoli and NOT gates, one needs to keep results of all OR terms for the final AND gate being the decision output of the oracle. The answer to each OR term is stored in a separate ancilla qubit; thus, we need the number of ancilla qubits equal to the number of terms in the function. In Boolean functions involving thousands of terms, this would mean the Grover's oracle needing thousands of ancilla qubits. If there are T terms in a function, we would need T ancilla qubits. For large T, the number of required ancilla qubits becomes unrealistically large, even for future large quantum computers with thousands of logical qubits. Therefore we present here a novel quantum oracle circuit design that requires $\lceil \log_2 T \rceil + 1$ ancilla qubits when T is not power of 2 or $\lceil \log_2 T \rceil + 2$ ancilla qubits when T is power of 2 in order to keep the circuit from growing too large. Our design improves also the overall runtime. For example, in traditional oracles if there are 1,000,000 terms then we need same number as 1,000,000 ancilla qubits but for our design we need only 21 ancilla qubits. To eliminate the need of ancilla qubits, we make use of the concept of an iterative quantum counter built from blocks, each block built from controlled Peres gates. We connect one block of the iterative quantum counter after each Toffoli gate representing the OR term of the function POS formula. The satisfiability value of this term controls the block of the counter by activating this block or not. It then increments the count by 1 or 0, depending on the truth value of the OR term. Thus, our quantum counter counts the number of satisfied OR terms in the Boolean function implemented as a POS.

We assign a counter block for each OR term, where the result of the term is used as one of the control qubits of the counter. When the term evaluates to 0, nothing is registered in the counter. When it evaluates to 1, the counter outputs the binary number $value + 1$ to the previously accumulated count *value*. The use of quantum counter allows us to send the result from the Toffoli gate representing one OR term to the counter circuit, hence eliminates the need for an ancilla qubit. We can set the function qubit back to 1 by mirroring the Toffoli gate used to compute the result and set the input qubits back to the original by applying NOT gates when appropriate. Our design drastically reduces the number of qubits needed for a function at the cost of replicating Toffoli gates in the POS expression and the costs of the iterative counter.

A. Grover's Search Algorithm

Grover's Algorithm [14] searches an unordered array of N elements to find a particular element with a given property. In

classical computations, in the worst case, this search takes N queries (tests, evaluations of the classical oracle). On the average case, the particular element will be found in $N/2$ queries. Grover's algorithm can find the element in $\sqrt{N}$ queries. Thus, Grover's algorithm can be used to solve the decision maximum satisfiability k-SAT for every value of k. Grover's algorithm is a quantum search algorithm, which speeds up a classical search algorithm of complexity $O(N)$ to $O(\sqrt{N})$ in the space of N objects, hence Grover gives a quadratic speed up. To solve the optimization problem of finding MAX-SAT with maximum value of k Grover's Algorithm has to be repeated.

The MAX-SAT contains n variables from the given Boolean function which is used to represent the search space of $N = 2^n$ elements. To apply the MAX-SAT in Grover's algorithm, these N elements are applied in a superposition state which is the input to the oracle. If the oracle recognizes an element as the solution, then the phase of the desired state is inverted. This is called the Phase inversion of the marked element. Marked element is a true minterm of function *f* from the oracle. The true minterm is a product of all variables of function *f* that evaluates to *f* = 1. The Grover's search algorithm uses another trick called *inversion about the mean* (average), which is also known as the *diffusion operation* or *amplitude amplification*. Inversion about mean amplifies the amplitude of the marked states and shrinks the amplitudes of other items. The amplitude amplification increases the probability of marked states, so that measuring the final states will return the target solution with high probability near to 1.

An oracle is a black box operation which takes an input and gives an output which is a yes/no decision. Quantum oracle is a reversible circuit that is used in quantum algorithms for the estimation of the value of the Boolean function realized in it. Quantum oracle has also to replicate all input variables on the respective output qubits. If oracle uses ancilla qubits initialized to $|0\rangle$, it has to return also a $|0\rangle$ for every ancilla qubit. The classical oracle function is defined as a Boolean function $f(x)$ which takes a proposed solution x of the search problem. If x is the solution, then $f(x) = 1$; If x is not a solution, then $f(x) = 0$. The quantum oracle is a unitary operator O such that:

$$|x\rangle|q\rangle \overset{O}{\rightarrow} |x\rangle|q \oplus f(x)\rangle$$

where x is the value in search space, q is a single qubit; the oracle qubit, and $\oplus$ is the EXOR operator (called also the addition modulo 2). A simplified formula of the quantum oracle can be written as:

$$|x\rangle \overset{O}{\rightarrow} (-1)^{f(x)}|x\rangle$$

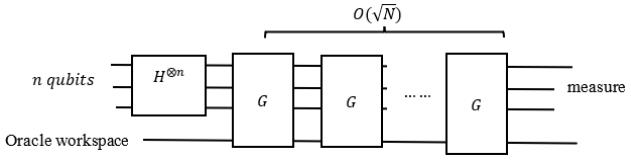

Fig. 6. Schematic circuit for Grover's algorithm [14].

As shown in Fig. 6, the n qubits in the superposition state result from applying a vector of Hadamard gates to initial state $|0\rangle^n$. Next applied is repeated operator G which is called the Grover Loop. After the iteration of the Grover Loop operator $O(\sqrt{N})$ times the output is measured for all input qubits. Oracle can use an arbitrary number of ancilla qubits, but all these qubits must be returned to value $|0\rangle$ inside the oracle. The Grover Loop G is a quantum subroutine which can be broken into four steps as shown in Fig. 7:

- Apply the Oracle *O*. This step is phase inversion.
- Apply the Hadamard transform $H^{\otimes n}$ ($H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$)
- Perform the condition phase shift also known as a Zero state phase shift, in which all states receive a phase shift of -1 except for the zero state $|0\rangle$. This step is also known as the diffusion operator.
- Apply the Hadamard transform $H^{\otimes n}$

The number of required iterations for Grover operator is: $R \leq \left\lceil \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rceil$ where M is number of solutions and N is number of all search space elements.

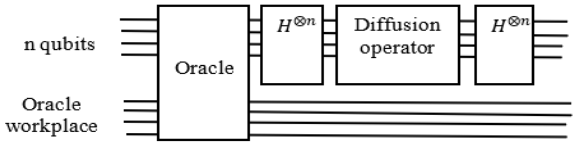

Fig. 7. Grover Loop Operator

B. *Quantum Counter*

The quantum counter block should be built from multi-controlled Peres gates as explained in Section III.C, where the first qubit of the Peres gate is applied a constant 1 with other variables together then the Peres gate converted as quantum control. (This qubit will be next taken from OR term of the satisfiability formula to activate the counter block realized from Peres gates). For simplicity of explanation, we assume that the counter block is built from Toffoli and CNOT gates as shown in Fig. 8.

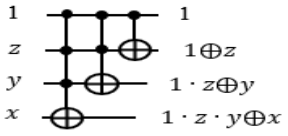

Fig. 8. Three-qubit quantum counter

Here z is the least significant qubit and x the most significant. The outputs of CNOT and two of the Toffoli gates are $1\oplus z$, $1 \cdot z\oplus y$, and $1 \cdot z \cdot y\oplus x$ respectively. When $xyz = 000$, the first Toffoli gate outputs $1 \cdot z \cdot y\oplus x = 1 \cdot 0 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$ and the second $1 \cdot z\oplus y = 1 \cdot 0 \cdot \oplus 0 = 0 \oplus 0 = 0$. The outputs of the qubits *y* and *x* are both zeros. The output of the qubit *z* is $1\oplus z = 1 \oplus 0 = 1$. Hence the circuit incremented 000 by 1 to 001. Quantum counter circuit indeed outputs the value *input*+1.

If we connect the first control input of the quantum counter block to a circuit, then the output of the connected circuit (a term of the POS) will either activate or deactivate the counter. When the output of the connected circuit is equal to 1, the output of the

counter block is incremented by 1. When the output of the circuit is equal to 0, the output of the counter block is unchanged.

C. *Traditional Oracle for Satisfiability Boolean function*

To build an OR term using a Toffoli gate, we use De Morgan's Law to convert the term into a product of the same variables $a + b + c = \overline{\overline{a + b + c}} = \overline{\bar{a} \cdot \bar{b} \cdot \bar{c}}$. With the XOR operation, $1 \oplus a = \bar{a}$. Hence $a + b + c = \overline{\bar{a} \cdot \bar{b} \cdot \bar{c}} = 1 \oplus \bar{a}\bar{b}\bar{c}$. The corresponding quantum circuit using a Toffoli gate is shown in Fig.9.

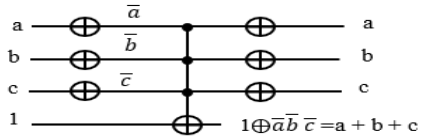

Fig. 9. Convert sum term to product term using De Morgan's law

Suppose we have a Boolean function $f(a,b,c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$ from Karnaugh map in Table I. As one can see in Table I, there are 4 ones which means the solution of the Boolean variables in binaries are (abc = 010, 011, 111, 101) which are satisfied for the Boolean function. Every true minterm in the Karnaugh map from Table I is a marked element and potential solution to Grover Algorithm. However, in one run of Grover's only one solution is found.

TABLE I. KARNAUGH MAP OF POS FOR THE BOOLEAN FUNCTION $f(a,b,c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$

ab\c	0	1
00	0	0
01	1	1
11	0	1
10	0	1

We build a quantum oracle for the Grover's Loop using Toffoli gates, in which the XOR gate is controlled by the product of variables. We need to first convert the Sum expressions into Products using De Morgan's Law. $a + b + \bar{c} = \overline{\overline{a + b + \bar{c}}} = \overline{\bar{a}\bar{b}\bar{\bar{c}}} = \overline{\bar{a}\bar{b}c}$; $\bar{a} + \bar{b} + c = \overline{\overline{\bar{a} + \bar{b} + c}} = \overline{\bar{\bar{a}}\bar{\bar{b}}\bar{c}} = \overline{ab\bar{c}}$; $b + c = \overline{\overline{b + c}} = \bar{\bar{b}}\bar{c}$. After building each term with the corresponding product expression, each with an assigned ancilla qubit for the output, we need to put the terms together as the product of the OR terms for the entire function $f(a,b,c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$. Since $xyz \oplus 0 = xyz$, we use another Toffoli gate controlled by the product of the OR terms XORed with 0. The schematic of the entire circuit for $f(a,b,c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$ is shown in Fig. 10:

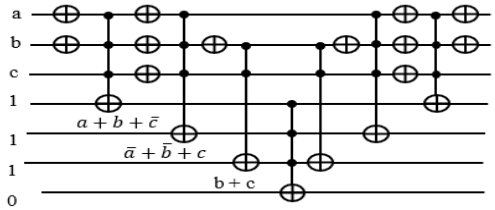

Fig. 10. Traditional oracle for Multiple input Toffoli gate used as global AND gate

To set the input qubits and ancilla qubits back to their original states, we mirror all the circuits up to the $f(a,b,c)$ on the right-hand side of the function gate.

Let define n number for variables and t number for terms then the number of qubits q needed for the oracle is: $q = n + t + 1$

where 1 is for the OR terms XORed with 0. Notice that we need three ancilla qubits, which is equal to the number of terms. For a function involving thousands of terms, we would need an equal number of ancilla qubits.

D. *Proposed construction of a quantum oracle for MAX-SAT*

Our proposed circuit does not require keeping the OR terms for the later calculation of the function. All we need to know is if each term is satisfied or not, and we pass the result to the counter block assigned to it. Thereafter, we put the ancilla qubit back to the original state 1 by mirroring. Depending on neighboring expressions, there are opportunities to cancel double NOT gates, yet saving the number of gates needed.

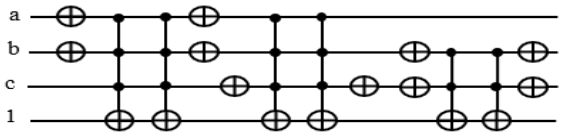

Fig. 11. Improved version of the part the oracle

The target output of each Toffoli gate realizing an OR term is used to activate the counter block corresponding to it. Notice that there are two NOT gates adjacent to each other, cancelling each other out. Hence we can remove those gates from our circuit. The resulting circuit is

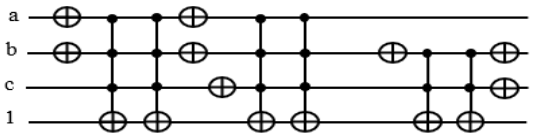

Fig. 12. Improved and optimized version of the part the oracle

There are 8 NOT and 6 Toffoli gates in this design in Fig. 12 as opposed to 12 NOT and 7 Toffoli gates in the traditional design in Fig 10. The reason why we need ancilla qubits in the traditional design is because we need the outputs from the Toffoli gates recorded in the ancilla qubits for counting the number of satisfied terms. By sending the satisfaction result for each term to quantum counter, we are able to reset the output line back to 1.

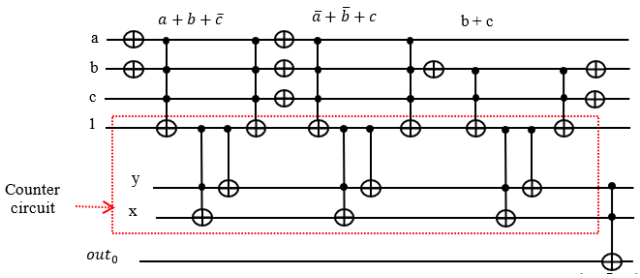

Fig. 13. Improved complete oracle using quantum counter

The count for the number of satisfied terms are output on the xy qubits. In this case we have 3 satisfied terms and want to have 3 as output expressed as 11 which expressed as $xy \oplus out_0 = xy \oplus 0$ on a Toffoli gate. If the Boolean function f is satisfied then the outcome out_0 should be 1. The entire oracle with the function and the iterative counter is shown in Fig. 13. We applied this oracle in the Grover search algorithms for R=2 iterations from this formula: $R \leq \left\lceil \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rceil$ where $M = 4$ is the number of solutions in our problem from Table II, and N=8 is the number of all search space elements (cells of the Karnaugh map from Table I). In general, the value of M is calculated using Quantum Counting algorithm [14], but an unsolved problem, the value of M is taken as 1 to run the Grover iterations R.

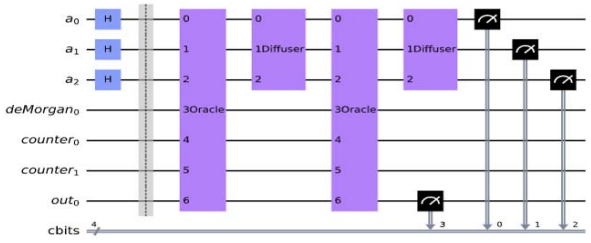

Fig. 14. MAX-SAT applied Grover's search algorithm. $f(a,b,c) = (a+b+\bar{c})(\bar{a}+\bar{b}+c)(b+c)$

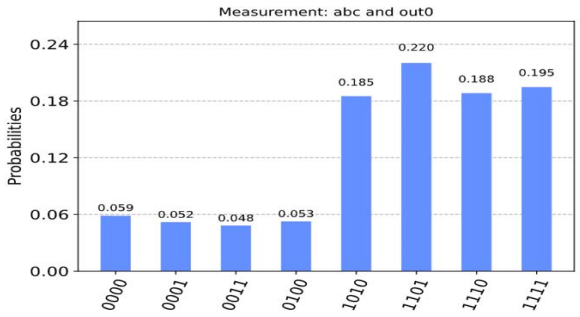

Fig. 15. Measurement of the Boolean variables and the outcome of function $f(a,b,c) = (a+b+\bar{c})(\bar{a}+\bar{b}+c)(b+c)$

In Fig. 14 we run the circuit on the 'qasm_simulator' from QISKIT for 1024 shots (independent runs to get high precision probability) which the circuit produces the correct answers. We measured a_0, a_1, a_2 and out_0 in Fig. 14 where a_0, a_1, a_2 corresponds the Boolean variables a, b, c respectively in Fig. 13. As can be see seen in Fig. 15, illustrates the QISKIT [23] output graphics for the simulated circuit. The measured values with high probability are 1010, 1101, 1110, 1111 where the most significant qubit is out_0 which is 1 and the least three significant qubits 010, 101, 110, 111 are all satisfied values for the Boolean function. These solutions correspond to the true minterms from Table I. For unsatisfied the measured value with low probability are 0000, 0001, 0011, 0100 where the most significant qubit is out_0 which is 0 and the least three significant qubits 000, 001, 011, 100 are all unsatisfied values for the Boolean function.

E. Verifying an unsatisfiable function

Suppose a function with four OR terms $f(a,b) = (a+b)(\bar{a}+b)(a+\bar{b})(\bar{a}+\bar{b})$ which no assignment of values a and b evaluates the function to 1. We need to first convert the OR terms into Products using De Morgan's Law and then build the oracle for the given Boolean function.

$$a+b = \overline{\overline{a+b}} = \overline{\bar{a}.\bar{b}};\ \bar{a}+b = \overline{\overline{\bar{a}+b}} = \overline{\bar{\bar{a}}.\bar{b}} = \overline{a.\bar{b}};\ a+\bar{b} = \overline{\overline{a+\bar{b}}} = \overline{\bar{a}.\bar{\bar{b}}} = \overline{\bar{a}.b};\ \bar{a}+\bar{b} = \overline{\overline{\bar{a}+\bar{b}}} = \overline{\bar{\bar{a}}.\bar{\bar{b}}} = \overline{a.b}$$

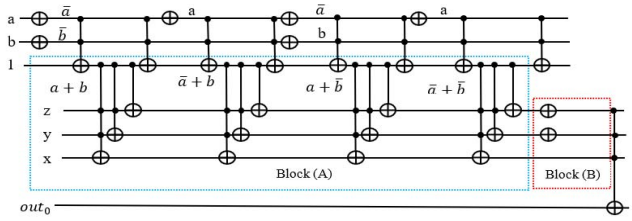

Fig. 16. Oracle with counter $f(a,b) = (a+b)(\bar{a}+b)(a+\bar{b})(\bar{a}+\bar{b})$

The 4 qubits $(1, z, y, x)$ in block (A) realize the counter, which can count from 0 to 7. We need the last qubit with out_0 ancilla bit to produce 1 when all terms are satisfied for Grover's algorithm. Since this function has 4 terms, to check satisfiability which is the last qubit should be 1, we need to add two NOT gates in the block (B) which makes the last qubit to produce 1 if the Boolean function is satisfied. The function $f(a,b)$ from Fig. 16 is not satisfiable, so comparing to value 4 in the last gate would not generate any correct solution. Grover's algorithm will give few random values that can be verified on the satisfiability formula outside the Grover's Algorithm using function $f(a,b)$. Therefore, we remove the two NOT gates in the block (B) to get the maximum satisfied terms of the function.

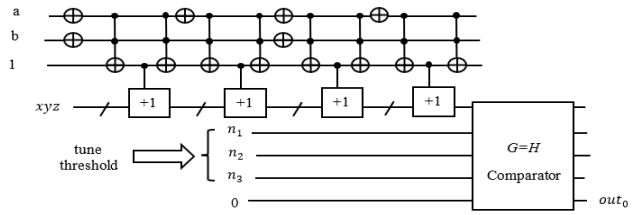

Fig. 17. Oracle with counter circuit and threshold with comparator

In a more general case in Fig. 17, we repeat the Grover Algorithm with tuning values of thresholds until equal to counter value xyz. The comparator $G = H$ compares the output from the counter with the threshold value given as constant values n_1, n_1, and n_3. For instance, $f(a,b) = (a+b)(\bar{a}+b)(a+\bar{b})(\bar{a}+\bar{b})$ has 4 terms, we tune the threshold value from 4, 3, 2, and 1 until the condition is met. The value of the counter where the condition is met is the MAX-SAT value. If the condition is met, ancilla qubit out_0 will be flipped. It changes the quantum phase of the solution, so that the elements that satisfy all constraints are marked. This method of threshold with comparator is useful to check when the exact number of terms (constraints) are known which can be checked whether the threshold is equal to counter value. For instance, if there are 10 constrains in given function but it should satisfy minimum 7 constrains, then set the threshold to 7 and check if counter equals to 7. There are applications based on method of threshold with comparator such as finding the minimum set of support [27].

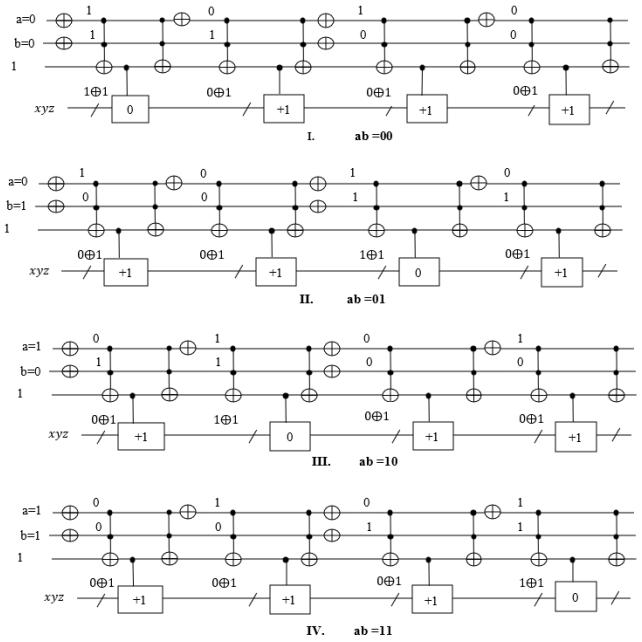

Fig. 18. MAX-SAT verification

Every binary vector $|a, b\rangle$ of a solution can be verified by running outside of the Grover Algorithm, as can be seen in Fig. 18 in which the maximum number of satisfied terms is 3 out of 4. We applied one Grover's Loop iteration for this the oracle to get the MAX-SAT. In Fig. 19 we run the circuit on the 'qasm_simulator' from QISKIT for 1024 shots.

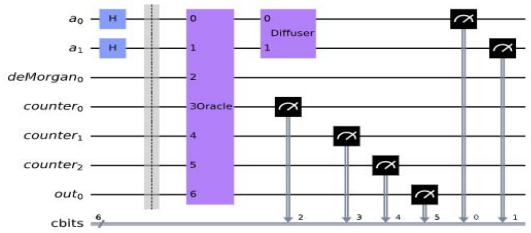

Fig. 19. $f(a,b) = (a+b)(\bar{a}+b)(a+\bar{b})(\bar{a}+\bar{b})$ applied Grover's algorithm

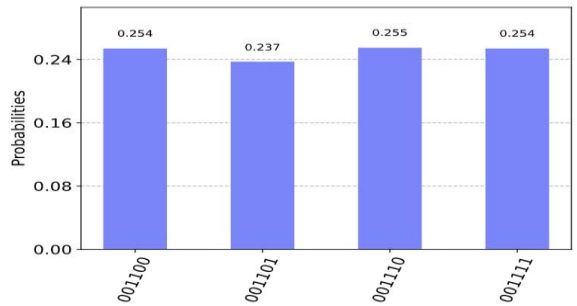

Fig. 20. Measurement of $f = (a,b,c)a+b)(\bar{a}+b)(a+\bar{b})(\bar{a}+\bar{b})$

In Fig. 19 we measured the Boolean variables, counter, and the output. In Fig. 20, the most significant qubit out_0 always is 0 which means the Boolean function is not satisfied because there are no such binary values for the least two significant qubits 00, 01, 10, 11 which would satisfy the Boolean function. However, the novelty of our design is that the counter qubits give the maximum numbers of satisfied terms in Boolean function. The counter qubits are second, third, and fourth qubits from the most significant qubit which in this case is 011.

V. Calculation of Quantum Cost

A. Calculation of Quantum Counter Size

In general, If there are T terms in given Boolean function then the total number of qubit that needs for quantum counter is:

- $\lceil \log_2 T \rceil + 1$ ancilla qubits when T is not a power of 2
- $\lceil \log_2 T \rceil + 2$ ancilla qubits when T is power of 2

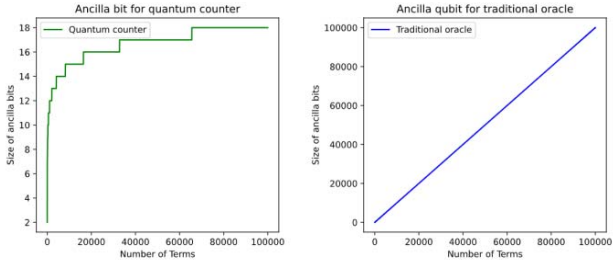

Fig. 21. Comparison of required numbers of ancilla qubits for our oracle and the traditional oracle.

As shown in Fig. 21, for instance, if there are 100,000 terms then the number of required ancilla qubits in traditional oracle is 100,000 but in our design the quantum counter requires only $\lceil \log_2 T \rceil + 1 = 18$ ancilla qubits. Using quantum counter each term is not required for one ancillia qubit but many terms are rquired a few anicllia qubits.

B. Quantum cost calculation for quantum counter

Each term in Boolean function is represented as n-bit Toffoli gate and the satisfiability result is passed down to the counter. We need as many counter blocks as there are terms in the given POS Boolean function. The counter can be built from Toffoli gates or Peres gates. It's important to have low-cost quantum circuit for this high demand for n-bit Toffoli gate. Since the Peres gate is a low-cost quantum circuit, we replaced Toffoli gates with Peres gates for cost reduction [11]. The formula of quantum cost for m-controlled bits of Peres gate is m^2 and for Toffoli gate is $2^{m+1} - 3$.

Three-qubit counter (3-control qubits) consist of three Toffoli gates which are 3-control, 2-control and 1-control (CNOT) gates in Fig. 8. Each of these Toffoli gates, the quantum cost is calculated separately: $(2^{3+1} - 3) + (2^{2+1} - 3) + (2^{1+1} - 3) = 28 - 9 = 19$. Four-qubit counter consists of four Toffoli gates, the quantum cost is also calculated separately: $(2^{4+1} - 3) + (2^{3+1} - 3) + (2^{2+1} - 3) + (2^{1+1} - 3) = 60 - 12 = 48$. Thus, we can drive a general formula for quantum cost of m-bit quantum counter using Toffoli gate: $2^{m+2} - 4 - 3m$. The total quantum cost of quantum counter for each term T is: Peres cost $= T * m^2$. Toffoli cost $= T * (2^{m+2} - 4 - 3m)$. Based on these two formulas, the Toffoli gate has higher quantum cost then Peres gate. Thus, we used in our design the Peres gates. As we mentioned before, our final counter uses

Peres gates, so we build our oracle using the Peres gate and it's mapping to *n*th root of NOT gates which leads to low quantum cost. The recursive design method from Peres gate was used.

VI. Conclusion

We have designed a novel quantum oracle circuit which requires the logarithmically reduced number of qubits for solving SAT and MAX-SAT problems. The oracle circuit uses the iterative quantum counter circuit which replaces the ancilla qubits of a global large AND gate for traditional oracle design. Our design showed a significant reduction over all for the number of qubits in Grover's search algorithm for MAX-SAT. Also, our design calculates the quantum measurable number of the maximum satisfiable OR terms for unsatisfiable Boolean functions. We also compared between using Peres and Toffoli gates in terms of quantum cost, where the Peres gates built from truly quantum primitives provide lower quantum costs. Finally, we tested and showed two examples on IBM QISKIT simulator [23] that provided the expected results.

Suppose one wants to calculate the number of satisfied true minterms for a SAT or MAX-SAT problems. This corresponds to the number of ones in certain Boolean function. This type of problems are solved using the Quantum Counting Algorithm [14], which in turn is based on Quantum Phase Estimation. Also, many other quantum algorithms use oracles with large AND gate at the output. We plan to work on finding solutions to these problems. The obvious improvement and generalization will be that the yes/no solutions will be extended to solutions for non-solvable problems where the answer will be given to tell how far we are from the solution by creating the "MAX versions" of the problems instead of the current "YES/NO" versions.

References

[1] Marques-Silva, J. and Glass, T., 1999, January. Combinational equivalence checking using satisfiability and recursive learning. In Proceedings of the conference on Design, automation and test in Europe (pp. 33-es).

[2] Konuk, H. and Larrabee, T., 1993, April. Explorations of sequential ATPG using Boolean satisfiability. In Digest of Papers Eleventh Annual 1993 IEEE VLSI Test Symposium (pp. 85-90). IEEE.

[3] Biere, A., Cimatti, A., Clarke, E.M., Fujita, M. and Zhu, Y., 1999, June. Symbolic model checking using SAT procedures instead of BDDs. In Proceedings of the 36th annual ACM/IEEE Design Automation Conference (pp. 317-320).

[4] Hong, T., Li, Y., Park, S.B., Mui, D., Lin, D., Kaleq, Z.A., Hakim, N., Naeimi, H., Gardner, D.S. and Mitra, S., 2010, November. QED: Quick error detection tests for effective post-silicon validation. In 2010 IEEE International Test Conference (pp. 1-10). IEEE.

[5] Wang, P.W., Donti, P., Wilder, B. and Kolter, Z., 2019, May. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In International Conference on Machine Learning (pp. 6545-6554). PMLR.

[6] Cook, S.A., 1971, May. The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on Theory of computing (pp. 151-158).

[7] Peres, A., 1985. Reversible logic and quantum computers. Physical Review A, 32(6), p.3266.

[8] Li, C.M., Manya, F. and Planes, J., 2005, October. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In International conference on principles and practice of constraint programming (pp. 403-414). Springer, Berlin, Heidelberg.

[9] Bian, Z., Chudak, F., Macready, W., Roy, A., Sebastiani, R. and Varotti, S., 2017, September. Solving sat and maxsat with a quantum annealer: Foundations and a preliminary report. In International Symposium on Frontiers of Combining Systems (pp. 153-171). Springer, Cham.

[10] Cheng, S.T. and Tao, M.H., 2007. Quantum cooperative search algorithm for 3-SAT. Journal of Computer and System Sciences, 73(1), pp.123-136.

[11] Maslov, D. and Dueck, G.W., 2003. Improved quantum cost for n-bit Toffoli gates. Electronics Letters, 39(25), pp.1790-1791.

[12] Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A. and Weinfurter, H., 1995. Elementary gates for quantum computation. Physical Review A, 52(5), p.3457.

[13] Szyprowski, M. and Kerntopf, P., 2013, August. Low quantum cost realization of generalized peres and toffoli gates with multiple-control signals. In 2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013) (pp. 802-807). IEEE.

[14] Nielsen, M.A. and Chuang, I., 2002. Quantum computation and quantum information. Cambridge University Press, 2000.

[15] Kohli, R., Krishnamurti, R. and Mirchandani, P., 1994. The minimum satisfiability problem. SIAM Journal on Discrete Mathematics, 7(2), pp.275-283.

[16] Smyth, K., Hoos, H.H. and Stützle, T., 2003, June. Iterated robust tabu search for MAX-SAT. In Conference of the Canadian Society for Computational Studies of Intelligence (pp. 129-144). Springer, Berlin, Heidelberg.

[17] Mastrolilli, M. and Gambardella, L.M., 2005. Maximum satisfiability: how good are tabu search and plateau moves in the worst-case?. European Journal of Operational Research, 166(1), pp.63-76.

[18] Marchiori, E. and Rossi, C., 1999. A flipping genetic algorithm for hard 3-SAT problems.

[19] Layeb, A., Deneche, A.H. and Meshoul, S., 2010, June. A new artificial immune system for solving the maximum satisfiability problem. In International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (pp. 136-142). Springer, Berlin, Heidelberg.

[20] Munawar, A., Wahib, M., Munetomo, M. and Akama, K., 2009. Hybrid of genetic algorithm and local search to solve max-sat problem using NVIDIA CUDA framework. Genetic Programming and Evolvable Machines, 10(4), pp.391-415.

[21] Lardeux, F., Saubion, F. and Hao, J.K., 2006. GASAT: a genetic local search algorithm for the satisfiability problem. Evolutionary Computation, 14(2), pp.223-253.

[22] Davis, M., Logemann, G. and Loveland, D., 1962. A machine program for theorem-proving. Communications of the ACM, 5(7), pp.394-397.

[23] Aleksandrowicz, G., Alexander, T., Barkoutsos, P., Bello, L., Ben-Haim, Y., Bucher, D., Cabrera-Hernández, F.J., Carballo-Franquis, J., Chen, A., Chen, C.F. and Chow, J.M., 2019. Qiskit: An open-source framework for quantum computing. Accessed on: Mar, 16.

[24] Biere, A., Heule, M. and van Maaren, H. eds., 2009. *Handbook of satisfiability* (Vol. 185). IOS press.

[25] Gu, J., 1992. Efficient local search for very large-scale satisfiability problems. *ACM SIGART Bulletin*, *3*(1), pp.8-12.

[26] Lee, S., Lee, S-J, Kim, T., Biamonte, J and M. Perkowski, The cost of Quantum Gate Primitives, J.Multiple-Valued Logic and Soft Computing, 12(5-6), pp. 561-573, 2006.

[27] Perkowski, M., 2020, July. Inverse Problems, Constraint Satisfaction, Reversible Logic, Invertible Logic and Grover Quantum Oracles for Practical Problems. In *International Conference on Reversible Computation* (pp. 3-32). Springer, Cham.

[28] Fu, Z. and Malik, S., 2006, August. On solving the partial MAX-SAT problem. In International Conference on Theory and Applications of Satisfiability Testing (pp. 252-265). Springer, Berlin, Heidelberg.

[29] Li, C.M., Xu, Z., Coll, J., Manyà, F., Habet, D. and He, K., 2021. Combining clause learning and branch and bound for MaxSAT. In 27th International Conference on Principles and Practice of Constraint Programming (CP 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[30] Cai, S. and Lei, Z., 2020. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. Artificial Intelligence, 287, p.103354.