



entropy



Article

Quantum Algorithm for Variant Maximum Satisfiability

Abdirahman Alasow, Peter Jin and Marek Perkowski

Special Issue

Advances in Quantum Computing

Edited by

Dr. Brian R. La Cour and Dr. Giuliano Benenti



<https://doi.org/10.3390/e24111615>

Quantum Algorithm for Variant Maximum Satisfiability [†]

Abdirahman Alasow *, Peter Jin and Marek Perkowski *

Department of Electrical & Computer Engineering, Portland State University, Portland, OR 97207, USA

* Correspondence: alasow@pdx.edu (A.A.); mperkows@ee.pdx.edu (M.P.)

[†] This paper is an extended version of our paper published in the IEEE 52nd International Symposium on Multiple-Valued Logic (ISMVL 2022) 18–20 May 2022.

Abstract: In this paper, we proposed a novel quantum algorithm for the maximum satisfiability problem. Satisfiability (SAT) is to find the set of assignment values of input variables for the given Boolean function that evaluates this function as TRUE or prove that such satisfying values do not exist. For a POS SAT problem, we proposed 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 information on how far the given Boolean function is from the SAT satisfaction. We used Grover's algorithm with a new block called quantum counter in the oracle circuit. The proposed circuit can be adapted for various forms of satisfiability expressions and several satisfiability-like problems. Using the quantum counter and mirrors for SAT terms reduces the need for ancilla qubits and realizes a large Toffoli gate that is then not needed. Our circuit reduces the number of ancilla qubits for the terms T of the Boolean function from T of ancilla qubits to $\approx \log_2 T + 1$. We analyzed and compared the quantum cost of the traditional oracle design with our design which gives a low quantum cost.

Keywords: satisfiability; maximum satisfiability; quantum counter; Grover search algorithm; quantum circuit

**Citation:** Alasow, A.; Jin, P.;

Perkowski, M. Quantum Algorithm for Variant Maximum Satisfiability.

Entropy **2022**, *24*, 1615. <https://doi.org/10.3390/e24111615>

Academic Editors: Brian R. La Cour and Giuliano Benenti

Received: 18 October 2022

Accepted: 2 November 2022

Published: 5 November 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

1.1. Satisfiability

The satisfiability (SAT) problem for a given Boolean function is the problem of determining if there exists a set of assignment values of input variables for the given Boolean function that evaluates this function to TRUE. Boolean or propositional-logic expressions are formed using operators AND, OR, EXOR, and NOT from input variables. Satisfiability expression (circuit) is often expressed as a product-of-sum (POS) form. POS is a logical ANDs of OR terms, where each OR term is an inclusive sum of literals. For instance, the POS SAT 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. Another example, $f(a, b) = (a + b)(\bar{a} + \bar{b})(\bar{a} + b)(a + \bar{b})$ is not satisfiable because 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 also have 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.

1.2. Maximum Satisfiability

Maximum satisfiability (MAX-SAT) is an optimization version of the SAT problem. MAX-SAT finds the maximum number of constraints of a given Boolean function that are satisfied.

Suppose a Boolean function in the POS form contains thousands of sum (OR) terms (also called clauses). The MAX-SAT problem is to examine the maximum number 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 to find the decision satisfied/unsatisfied (yes/no) but also to provide the maximum number of terms (clauses) that are satisfied with the actual satisfying assignment values for the variables in case the formula is not SAT satisfiable. The MAX-SAT is considered to be an NP-hard problem [7].

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 sets 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 [8,9]:

1. Weighted MAX-SAT: Each clause has an 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.

The applications of these different variants will be discussed in the next section.

2. Related Work

2.1. Maximum Satisfiability Applications

There are many optimization problems and real-world applications that can be encoded to 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 [8]. We will briefly discuss some of these applications.

2.1.1. Data Analysis and Machine Learning

MAX-SAT has been used in many problems in Data Analysis, Artificial Intelligence (AI) and Machine Learning [10]. Correlation clustering is a well-studied problem in data analysis and AI in which data are divided into subgroups in a meaningful way. Discovering an optimal way of making such a division is a computational challenge. There are many approaches to find the optimal clustering, including a greedy local-search and approximation algorithms, which cannot find optimal clusterings. Solving exact formulations of the correlation clustering as MAX-SAT based approach leads to cost-optimal correlation clustering [11]. Bayesian Network Structure Learning (BNSL) is a computationally hard problem of finding a directed acyclic graph structure that optimally describes a given data structure. These problems use learning that can be based on probabilistic or exact inference methods. Using MAX-SAT as exact inference has been shown to yield a competitive approach to learning optimal bounded tree-width Bayesian network structures (BTW-BNSL) [12]. There are many other AI applications and data analysis approaches formulated as MAX-SAT, including causal structure discovery [13], and deriving interpretable classification rules [14].

2.1.2. Planning and Scheduling

MAX-SAT can be applied in linear temporal logic (LTL) specifications for robotic motion planning and control of autonomous systems. Suppose that we want to design a controller for a robotic museum guide; the robot has to give a tour of the exhibitions in a specific order, which constitutes the hard specification. Preferably, it also avoids certain locations, such as the staff's office, the library, or the passage when it is occupied. These

preferences are encoded in the soft specifications [15]. This is an example of a partial MAX-SAT formulation. There are other planning problems that can be encoded as MAX-SAT for cost-optimal planning [16,17].

Scheduling problems are well-known problems that appear in various contexts, including health care, airlines, transportation services, and various financial and money transfer problems in organizations. These scheduling problems can be encoded as a weighted partial MAX-SAT problem [18].

2.1.3. Verification and Security

Functional verification tasks dominate the effort of contemporary VLSI and SoC design cycles. A major step of functional verification is design debugging, which determines the root cause of failed verification tasks such as simulation or equivalence checking. The MAX-SAT formulation is used as a pre-processing step to construct a highly optimized debugging framework [19–21]. One of the techniques for debugging both hardware and software is fault localization, where the goal is to pinpoint the localization of bugs. Fault localization is performed using the MAX-SAT approach to reduce and improve automation for error localization, which can speed up the debugging process [22,23].

MAX-SAT has many applications in security. Starting with solving the user authorization query problem [24], reconstructing AES key schedule images [25], detecting hardware Trojans [26], and malware detection [27].

2.1.4. Bioinformatics

MAX-SAT has many applications in the bioinformatics field, such as cancer therapy, finding the optimal set of drugs to fix or rectify the fault areas of the gene regulatory network [28], modeling biological networks and checking their consistency [29], finding the maximum similarity between RNA sequences [30] and finding the minimum-cardinality set of haplotypes that explains a given set of genotypes [31].

2.1.5. Combinatorial Optimization Problems

Combinatorial optimization problems are widely studied in fundamental academic research and in solving real-life problems. Many of these problems are NP-hard, where an exhaustive search is not tractable. For instance, MAX-SAT has been used to encode and solve such problems as the Max-Clique problem [32–34], given a group of vertices. The maximal clique is the largest subset of vertices in which each point is directly connected to every other vertex in the subset.

Other applications within this domain that have been encoded into MAX-SAT consist of determining the Treewidth of a graph [35] and finding solutions for the maximum quartet consistency problem [36].

2.2. Classical Algorithm for Maximum Satisfiability Problem

There are many classical algorithms for solving MAX-SAT problems: exact algorithms, stochastic local search algorithms [37–39], evolutionary algorithms [40,41], and hybrids of local search and evolutionary algorithms [42,43]. Exact algorithms are often used for small or medium size problems that can be easily verified as satisfied or unsatisfied. The exact algorithms are based on the Davis–Putnam–Logemann–Loveland algorithm (DPLL) [44], an example being the Branch-and-Bound algorithm [45,46] 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. Large size problems use stochastic local search algorithms and evolutionary algorithms which can potentially provide a high-quality solution [42,47].

2.3. Quantum Algorithms for Maximum Satisfiability Problem

MAX-SAT is an NP-hard problem and is one of the most widely studied optimization problems in classical algorithms. These NP-hard problems can be potentially solved by quantum algorithms which would offer significant improvements over the classical algorithms, assuming the 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 [48]. 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 currently available quantum computers. Thus, due to the limited number of qubits available, some algorithms suggested reducing the number of qubits. For instance, the quantum cooperative search algorithm for 3-SAT [49] 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 a reduced quantum cost. The main idea is to avoid large Toffoli gates that have high quantum costs and lead to decoherence. Our novel quantum oracle circuit design requires fewer 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 computers and simulators have a limited total number of qubits, our quantum algorithm allows us to solve larger MAX-SAT problems. However, because of a limited number of qubits, it is not competing with modern software MAX-SAT solvers.

3. Definitions and Preliminaries

In this section, we will define some basic concepts related to quantum gates and quantum cost. A few useful gates are shown in Figure 1.

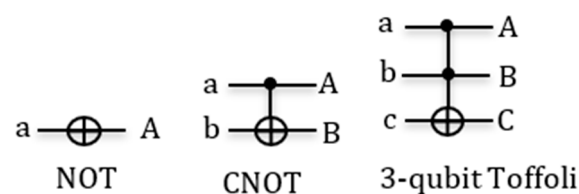


Figure 1. Gate symbol: NOT, CNOT, 3-qubit Toffoli gates.

Definition 1: Reversible gate is $n \times n$ quantum gate that has n input variables and n output variables. A quantum gate is reversible if it maps an n -input binary vector into a unique n -output binary vector. In addition, it is a one-to-one mapping or a permutation of vectors. For example, the NOT gate is reversible because if the output is 0, then you know the input must be 1, and vice versa.

Definition 2: Controlled-NOT (CNOT) is a 2-qubit gate, where the first qubit is called control and the second qubit is called target. CNOT applies the NOT gate on the target qubit when the control qubit is one. The value of the control qubit is not affected. Thus $A = a$, $B = a \oplus b$. The CNOT gate is also called the Feynman gate. Using Definition 1, the reader can check that this function is reversible.

Definition 3: *n*-control Toffoli gate consists of *n*-control qubits and one target qubit. The target qubit is inverted if all control qubits are 1. Otherwise, the target qubit is unchanged: $C = ab \oplus c$. The values of all control qubits are not changed, thus $A = a$, $B = b$, etc. This is the universal reversible gate; it realizes AND with $c = 0$ and NAND with $c = 1$.

Definition 4: Ancilla qubits are extra qubits to allow extra working space during the computation. They are necessary to convert arbitrary Boolean functions to reversible Boolean functions.

For instance, the Boolean function $X = a \cdot b$ is not reversible, but function $X = a \cdot b \oplus c$ is a reversible gate with $c = 0$.

Although the iterative 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 explaining these gates first.

3.1. Nth Root of Not Gate

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

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

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 .

$W = \sqrt{V} = \sqrt[4]{NOT}$

$G = \sqrt{W} = \sqrt[8]{NOT}$

$VV = NOT$

$VV^\dagger = I$

$WW = V$

$GG = W$

3.2. Controlled-Nth Root of NOT Gate

The controlled-*N*th 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 *N*-th 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-*N*th root of NOT gate is:

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

The inverse of *N*-th root of NOT gate and controlled-*N*th root of NOT gate are constructed from a matrix where the plus and minus signs are reversed.

Figure 2 shows examples of various controlled-*N*th root of NOT gates that we will use in our design of large Peres gates used in counters.

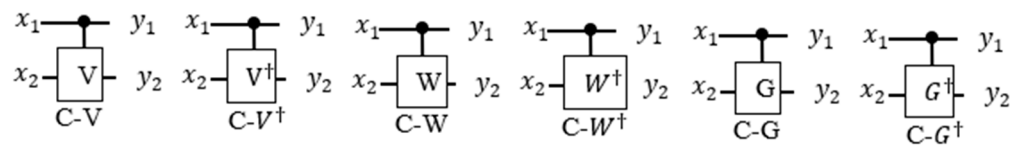


Figure 2. Some symbols for quantum gates of Controlled- n th root of NOT gate and their inverse (+) dagger or conjugate.

3.3. 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 that are used to calculate the quantum cost: NOT, controlled- n th 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 [50] but for this paper we use the approximate costs defined as above.)

Toffoli gate could be built using controlled- n th root of NOT gate [51]. A 3-bit Toffoli gate from Figure 3 has two control qubits and one target qubit and is built from controlled V/V^\dagger 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 [52] is equal to $2^{m+1} - 3$.

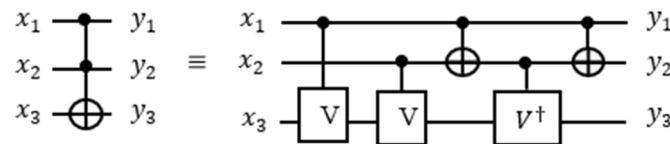


Figure 3. 3-bit Toffoli gate represented as controlled- V/V^\dagger and CNOT gates.

3.4. Peres Gate

The Peres gate [53] 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 (Figure 4I). When the 3-bit Toffoli and CNOT gates are implemented separately, the cost would be six (Figure 4II). However, the 3-bit Peres gate costs four 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.

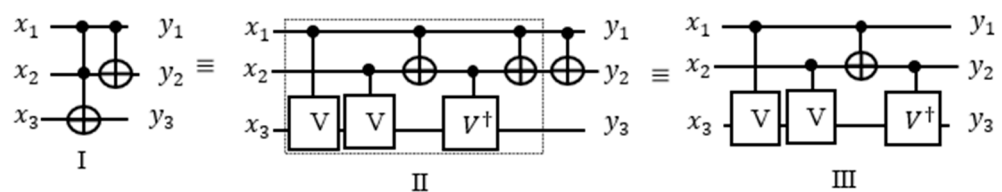


Figure 4. (I) 3-bit Peres gate (II) decomposed Toffoli gate with CNOT. (III) 3-bit Peres gate and its representation using controlled- V/V^\dagger and CNOT gates.

The Figure 4III:

- 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 behaves 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 consists 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 a few additional controlled gates. The reader can appreciate this recursive way of building counter blocks of any size by analyzing Figure 5 in which a 4-controlled gate at the right uses the 3-controlled Peres gate in four upper qubits.

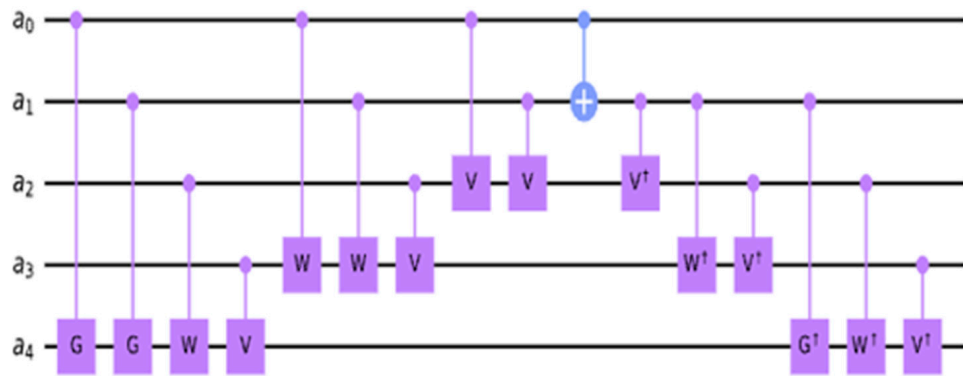


Figure 5. A Peres gate realized on five qubits.

As shown in Figure 5, the 5-qubit Peres gate uses the 4-qubit Peres gate as its sub-circuit. Figures 4 and 5 illustrate that the general formula for the quantum cost of m -controlled Peres gate [54] is equal to m^2 . For a larger design, the Peres gate can be designed as recursive blocks as shown in Figure 6.

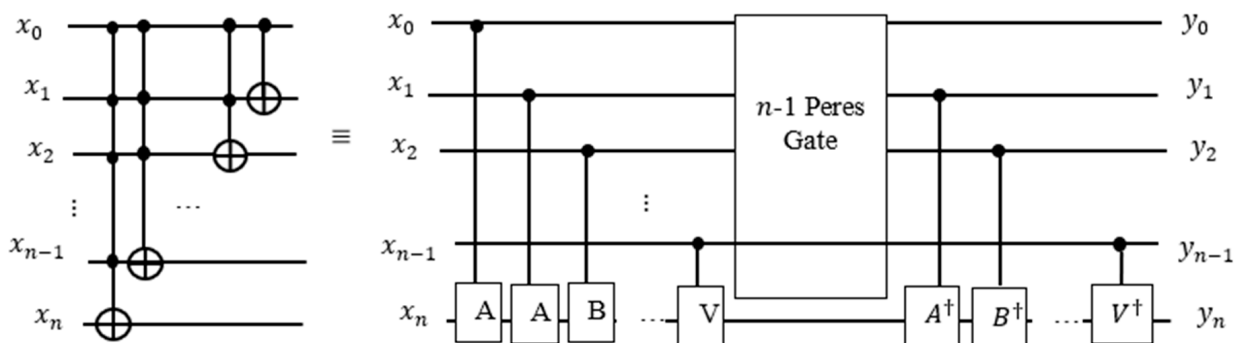


Figure 6. Generalized Peres gate realized on n qubits.

3.5. Quantum Oracle

An oracle is a black box operation that takes an input and gives an output that is a yes/no decision. A 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 also has to replicate all input variables on the respective output qubits. If the 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 $f(x)$ that 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:

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

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

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

4. Proposed Quantum Algorithm for Maximum Satisfiability

In traditional Grover's algorithm, oracles are composed of Toffoli and NOT gates; one needs to keep the 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 Grover's oracle needs 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 a power of 2 or $\lceil \log_2 T \rceil + 2$ ancilla qubits when T is a power of 2 in order to keep the circuit from growing too large. Our design also improves the overall runtime. For example, in traditional oracles if there are 1,000,000 terms, then we need the same number as 1,000,000 ancilla qubits, but for our design, we need only 21 ancilla qubits. To eliminate the need for ancilla qubits, we make use of the concept of an iterative quantum counter built from blocks, with 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 a quantum counter allows us to send the result from the Toffoli gate representing one OR term to the counter circuit, hence eliminating 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.

4.1. Grover's Search Algorithm

Grover's Algorithm [55] searches an unordered array of N elements to find a particular element with a given property. Grover's algorithm is often used as a subroutine in other quantum algorithms [56–58]. In classical computations, in the worst case, this search takes N queries (tests, evaluations of the classical oracle). In 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. The 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 $f = 1$. Grover's search algorithm uses another trick called inversion about the mean (average), which is also known as diffusion operation or amplitude amplification. Inversion about the 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 a high probability near 1.

As shown in Figure 7a, 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 number of required iterations for Grover operator is: $R \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil$ where N is number of all search space elements and M is number of solutions. The Grover Loop G is a quantum subroutine which can be broken into four steps as shown in Figure 7b:

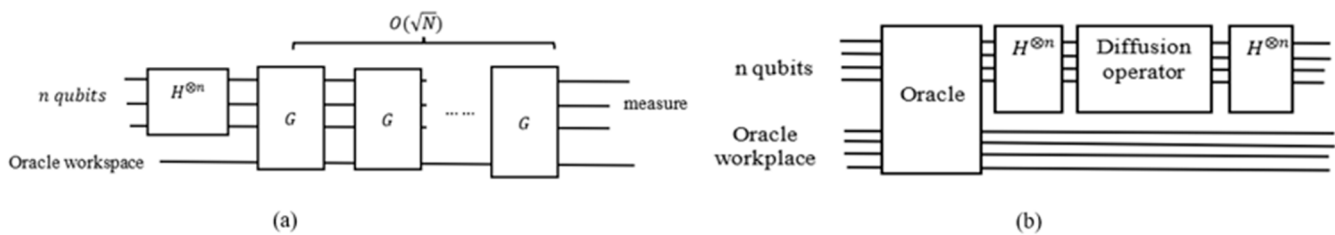


Figure 7. (a) Schematic circuit for Grover’s algorithm [55]. (b) Grover operator G .

1. Phase inversion: apply the oracle. If the oracle recognizes the solution, then the phase of the desired state is inverted
2. Apply the Hadamard transform $H^{\oplus n}$ ($H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$)
3. Zero state phase shift: Perform the condition phase shift, in which all states receive a phase shift of -1 except for the zero state $|0\rangle$.
4. Apply the Hadamard transform $H^{\oplus n}$

4.2. Quantum Counter

As described in Section 3.3, the quantum counter block should be constructed from multiple-controlled Peres gates, where the first qubit of the Peres gate is applied a constant 1 with other variables combined, and the Peres gate is then turned into a quantum counter. (This qubit will be next taken from the 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 Figure 8.

Here z is the least significant qubit and x the most significant. The outputs of CNOT and two of the Toffoli gates are $1 \oplus z$, $1 \oplus z \oplus y$, 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 $1 \cdot z \oplus y = 1 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$. 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 $\text{input}+1$.

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.

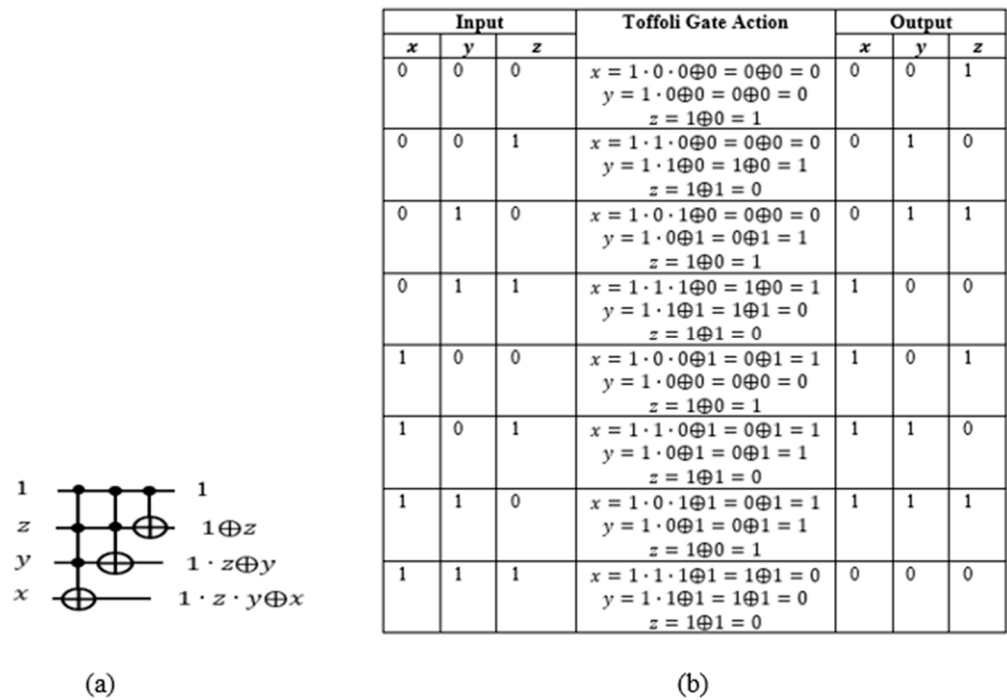


Figure 8. (a) Three-qubit quantum counter. (b) Analysis of 3-qbit quantum counter block from (a).

4.3. 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 $a + b + c = \overline{\overline{a + b + c}} = \overline{\overline{a} \cdot \overline{b} \cdot \overline{c}}$. With the XOR operation, $1 \oplus a = \overline{a}$. Hence $a + b + c = \overline{\overline{a} \cdot \overline{b} \cdot \overline{c}} = 1 \oplus \overline{\overline{a} \cdot \overline{b} \cdot \overline{c}}$. The corresponding quantum circuit using a Toffoli gate is shown in Figure 9.

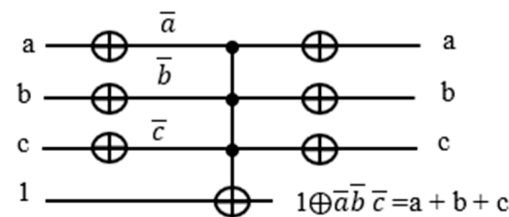


Figure 9. Convert sum term to product term using De Morgan’s law.

Suppose we have a Boolean function $f(a, b, c) = (a + b + \overline{c})(\overline{a} + \overline{b} + c)(b + c)$ from Karnaugh map in Table 1. As one can see in Table 1, there are four 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 1 is a marked element and potential solution to the Grover Algorithm. However, in one run of Grover’s search algorithm, only one solution is found.

Table 1. Karnaugh map of POS for the Boolean function $f(a, b, c) = (a + b + \overline{c})(\overline{a} + \overline{b} + c)(b + c)$.

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.

$$a + b + \bar{c} = \overline{\overline{a + b + \bar{c}}} = \overline{\bar{a}\bar{b}c} = \overline{\bar{a}\bar{b}}\bar{c}$$

$$\bar{a} + \bar{b} + c = \overline{\overline{\bar{a} + \bar{b} + c}} = \overline{\overline{\bar{a}\bar{b}}c} = \overline{\bar{a}\bar{b}}\bar{c}$$

$$b + c = \overline{\overline{b + c}} = \overline{\bar{b}\bar{c}}$$

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 Figure 10:

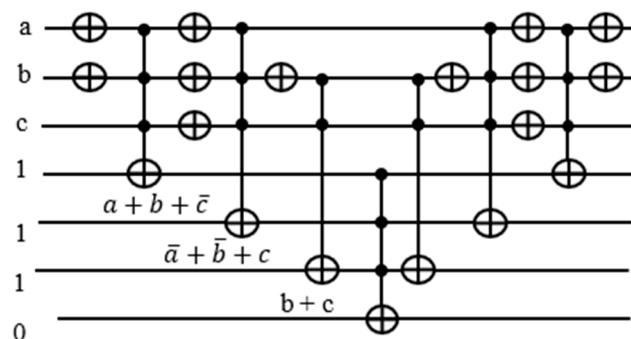


Figure 10. Traditional oracle for Multiple input Toffoli gate used as global AND gate $f = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$.

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 us 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.

4.4. 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 whether 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.

The target output of each Toffoli gate realizing an OR term is used to activate the counter block corresponding to it. In Figure 11, notice that there are two NOT gates adjacent to each other, canceling each other out. Hence, we can remove those gates from our circuit.

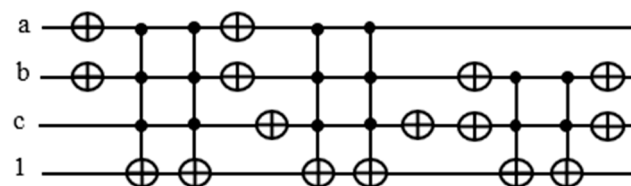


Figure 11. Improved version of the part the oracle $f = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$.

There are eight NOT and six Toffoli gates in this design in Figure 12 as opposed to 12 NOT and 7 Toffoli gates in the traditional design in Figure 10. The reason we need ancilla qubits in the traditional design is that 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 the quantum counter, we are able to reset the output line back to 1.

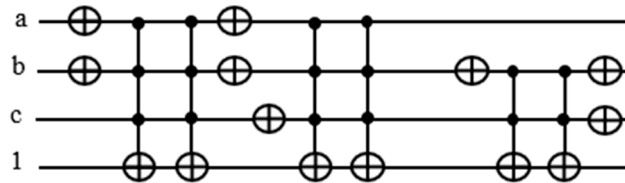


Figure 12. Improved and optimized version of the part the oracle $f = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$.

The count for the number of satisfied terms is output on the xy qubits. In this case, we have three satisfied terms and want to have three as the output expressed as 11 which are 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 Figure 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 1, and $N = 8$ is the number of all search space elements (cells of the Karnaugh map from Table 1). In general, the value of M is calculated using Quantum Counting algorithm [55], but an unsolved problem, the value of M , is taken as 1 to run the Grover iterations R .

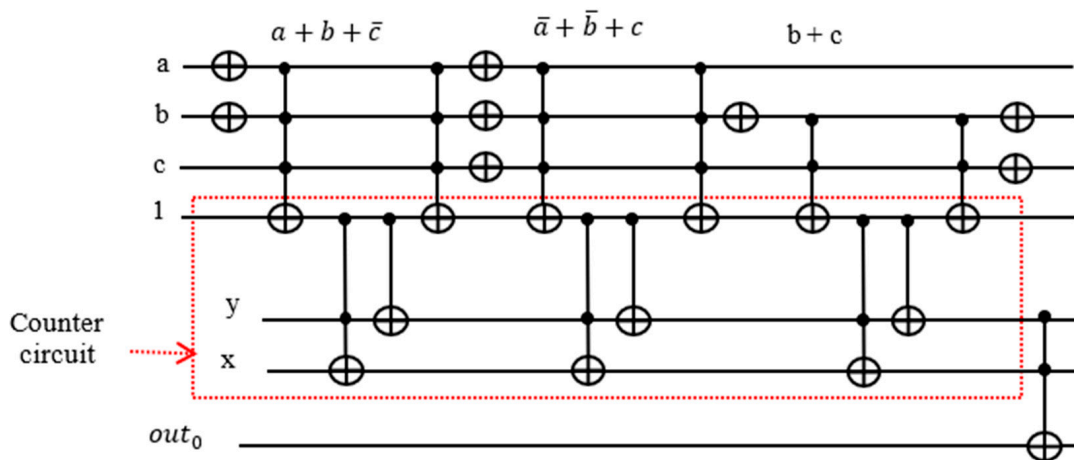


Figure 13. Improved complete oracle using quantum counter.

In Figure 14, we run the circuit on the ‘qasm_simulator’ from QISKIT for 1024 shots (independent runs to obtain high precision probability) for which the circuit produces the correct answers. We measured a_0, a_1, a_2 and out_0 in Figure 14 where a_0, a_1, a_2 correspond to the Boolean variables, a, b, c , respectively in Figure 13. As can be seen in Figure 15, it illustrates the QISKIT [59] output graphics for the simulated circuit. The measured values with high probability are 1010, 1101, 1110, and 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 1. For the unsatisfied, the measured values with low probability are 0000, 0001, 0011, and 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.

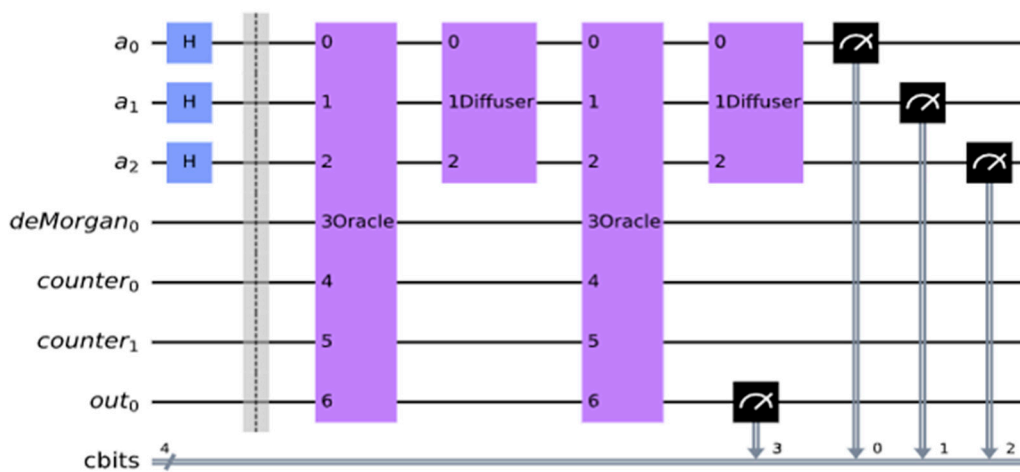


Figure 14. MAX-SAT applied Grover’s search algorithm. $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$.

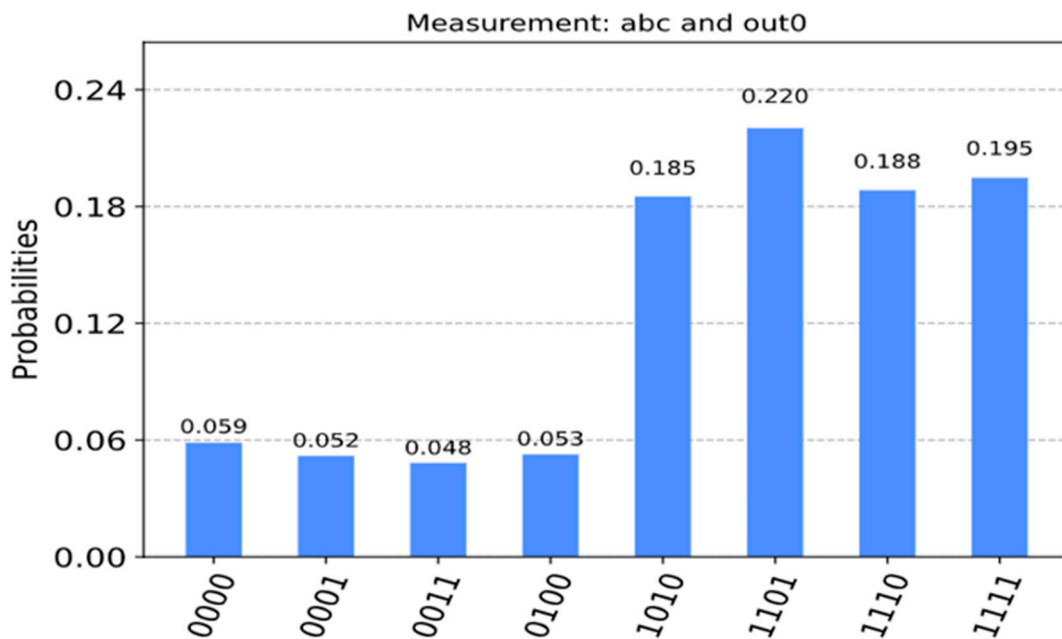


Figure 15. Measurement of the Boolean variables and the outcome of function $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$.

As can be seen in Figure 15, the four values 000, 001, 011, and 100 have some value with less probability because of noise created by the simulator. However, we verified the solutions by applying the number of iterations R , and the output from the simulation with high probability 010, 101, 110, and 111 matches the theoretical values, which can be verified manually. We also applied different shots to test, and the results were closely similar, with a high probability for all satisfying values.

4.5. Verifying an Unsatisfiable Function

Suppose a function with four OR terms $f(a, b) = (a + b)(\bar{a} + \bar{b})(a + \bar{b})(\bar{a} + b)$ 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.

$$a + b = \overline{\overline{a + b}} = \overline{\bar{a} \cdot \bar{b}}$$

$$\bar{a} + \bar{b} = \overline{\overline{\bar{a} + \bar{b}}} = \overline{\overline{\bar{a}} \cdot \overline{\bar{b}}} = \overline{a \cdot b}$$

$$a + \bar{b} = \overline{\overline{a + \bar{b}}} = \overline{\bar{a} \cdot b} = \bar{a} \cdot \bar{b}$$

$$\bar{a} + \bar{b} = \overline{\overline{\bar{a} + \bar{b}}} = \overline{\bar{a} \cdot b} = \bar{a} \cdot \bar{b}$$

The four 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 four 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 Figure 16 is not satisfiable, so comparing to a value of 4 in the last gate would not generate any correct solution. Grover’s algorithm will give a few random values that can be verified on the satisfiability formula outside Grover’s Algorithm using function $f(a, b)$. Therefore, we remove the two NOT gates in block (B) to get the maximum satisfied terms of the function.

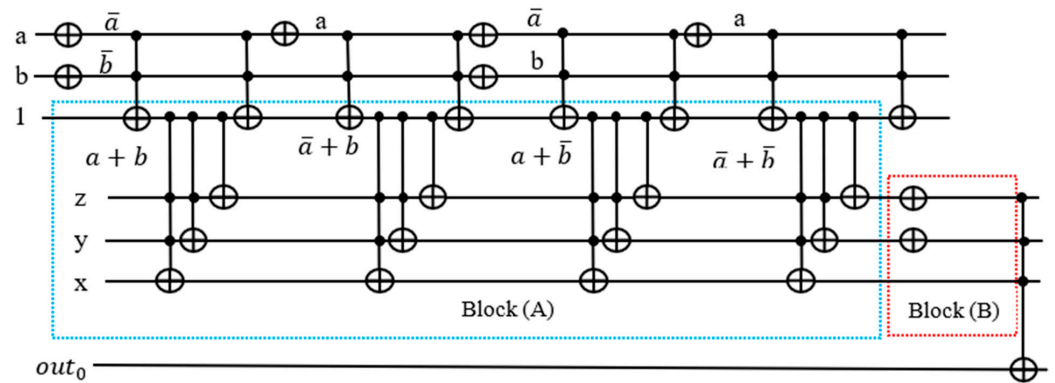


Figure 16. Oracle with counter $f(a, b) = (a + b)(\bar{a} + b)(a + \bar{b})(\bar{a} + \bar{b})$.

In a more general case in Figure 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, the 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 the 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 the counter value. For instance, if there are 10 constraints in a given function, but it should satisfy a minimum seven constraints, then set the threshold to seven and check if the counter equals to seven. There are applications based on the method of the threshold with a comparator, such as finding the minimum set of support [60].

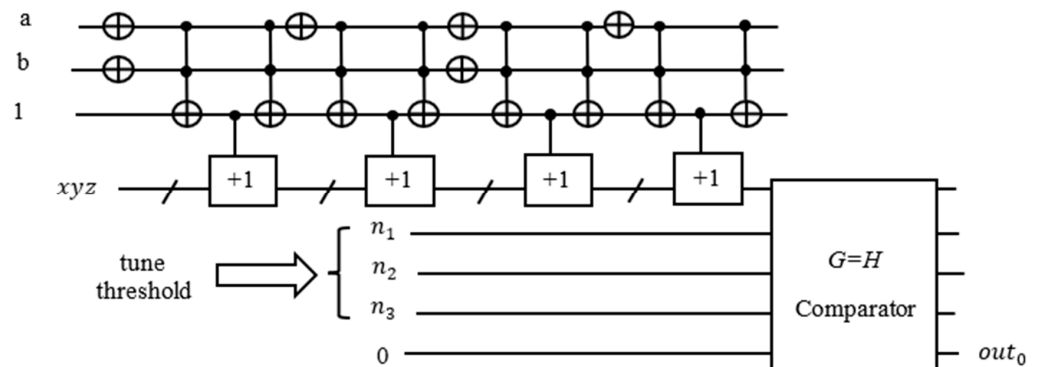


Figure 17. Oracle with counter circuit and threshold with comparator.

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

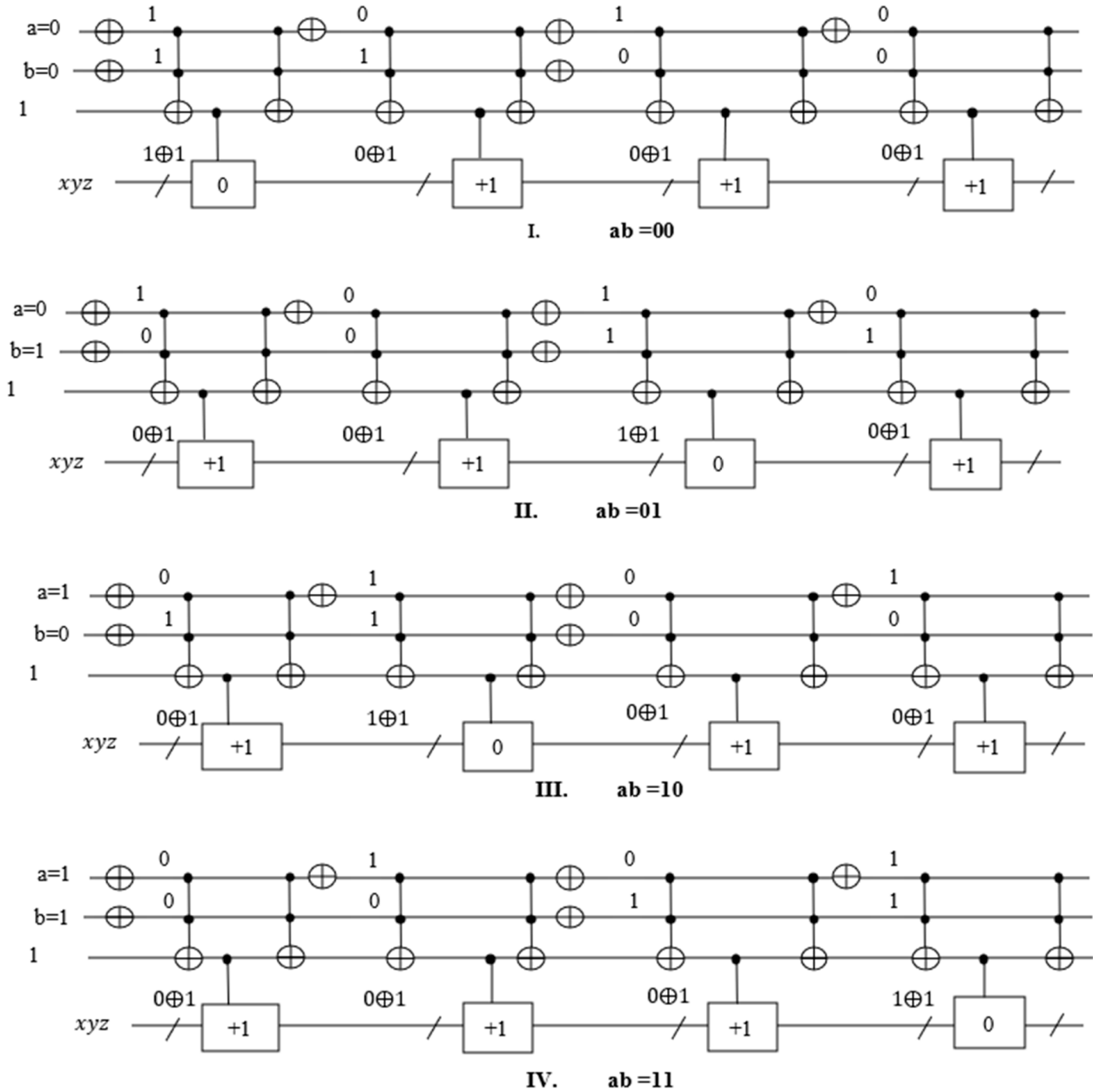


Figure 18. MAX-SAT verification.

In Figure 19, we measured the Boolean variables, counter, and output. In Figure 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, and 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 the Boolean function. The counter qubits are the second, third, and fourth qubits from the most significant qubit, which in this case is 011.

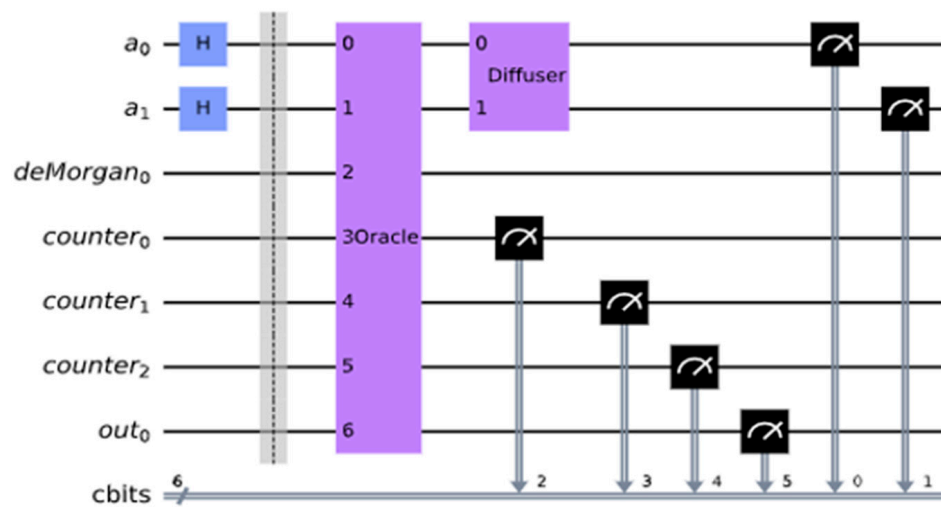


Figure 19. $f(a, b) = (a + b)(\bar{a} + b)(a + \bar{b})(\bar{a} + \bar{b})$ applied Grover’s algorithm.

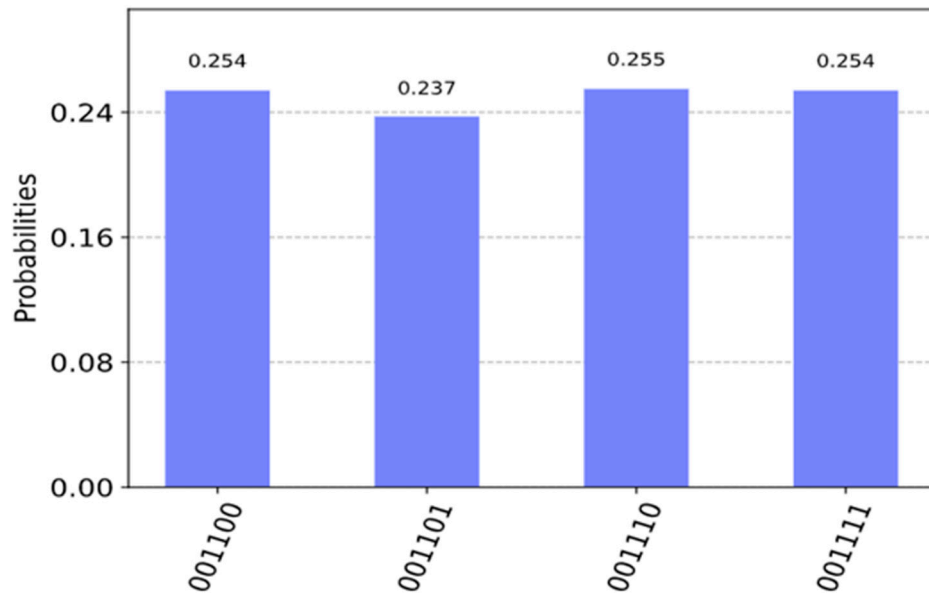


Figure 20. Measurement of $f = ((a, b, c) a + b)(\bar{a} + b)(a + \bar{b})(\bar{a} + \bar{b})$.

5. Calculation of Quantum Cost

5.1. Calculation of Quantum Counter Size

In the Table 2 shows the required number of qubits for the quantum counter which each term is not required for one ancilla qubit, but many terms require a few ancilla qubits.

In general, if there are T terms in given Boolean function then the total number of qubits that need for quantum counter is:

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

As shown in Figure 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 the quantum counter, each term is not required for one ancilla qubit, but many terms are required for a few ancilla qubits.

Table 2. Quantum counter size; total qubits for counter.

Number of Terms (Clauses)	Total Qubits for Quantum Counter
2	$\lceil \log_2 T \rceil + 2 = 3$
3	$\lceil \log_2 T \rceil + 1 = 3$
4	$\lceil \log_2 T \rceil + 2 = 4$
5 ... 7	$\lceil \log_2 T \rceil + 1 = 4$
8	$\lceil \log_2 T \rceil + 2 = 5$
9 ... 15	$\lceil \log_2 T \rceil + 1 = 5$
16	$\lceil \log_2 T \rceil + 2 = 6$
17 ... 31	$\lceil \log_2 T \rceil + 1 = 6$
32	$\lceil \log_2 T \rceil + 1 = 7$
33 ... 63	$\lceil \log_2 T \rceil + 1 = 7$
64	$\lceil \log_2 T \rceil + 2 = 8$
65 ... 127	$\lceil \log_2 T \rceil + 1 = 8$
128	$\lceil \log_2 T \rceil + 2 = 9$
129 ... 255	$\lceil \log_2 T \rceil + 1 = 9$
256	$\lceil \log_2 T \rceil + 2 = 10$
257 ... 511	$\lceil \log_2 T \rceil + 1 = 10$
...	...
...	...
T	$\begin{cases} \lceil \log_2 T \rceil + 1, & \text{if } T \text{ is not power of } 2 \\ \lceil \log_2 T \rceil + 2, & \text{if } T \text{ is power of } 2 \end{cases}$

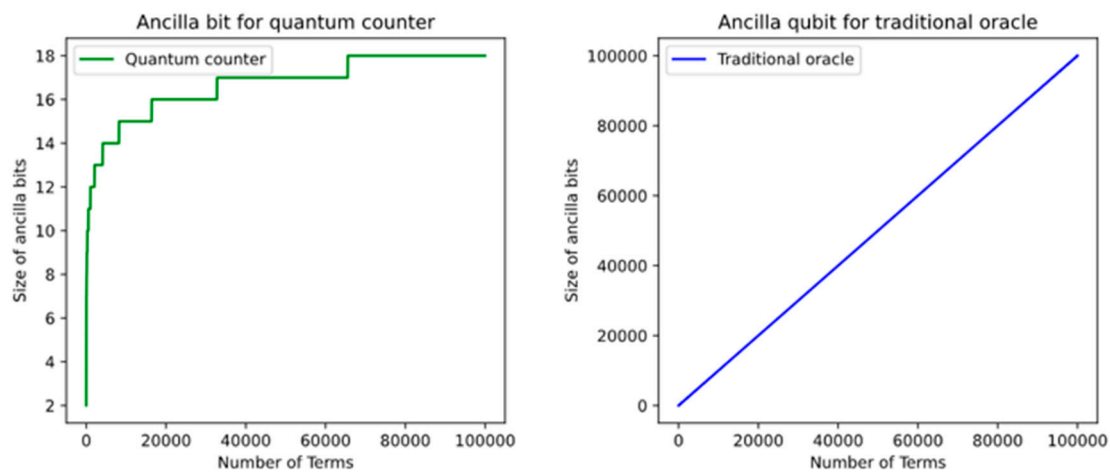


Figure 21. Comparison of required numbers of ancilla qubits for our oracle and the traditional oracle.

5.2. Quantum Cost Calculation for Quantum Counter

Each term in the 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 is important to have low cost quantum circuits for this high demand for n -bit Toffoli gate. Since the Peres gate is a low-cost quantum circuit, we replaced the Toffoli gates with Peres gates for cost reduction [52]. The formula of quantum cost for m -controlled bits of Peres gate is m^2 and for Toffoli gate is $2^{m+1} - 3$.

In Figure 22 a three-qubit counter (3-control qubits) consists of three Toffoli gates which are 3-control, 2-control, and 1-control (CNOT) gates. For 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, and the quantum cost is also calculated separately: $(2^{4+1} - 3) + (2^{3+1} - 3) + (2^{2+1} - 3) + (2^{1+1} - 3) = 60 - 12 = 48$.

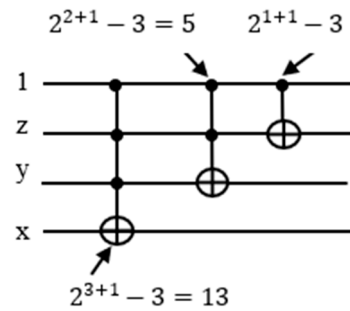


Figure 22. Quantum cost for 3-bit counter.

Thus, we can derive a general formula for the quantum cost of m -bit quantum counter using the Toffoli gate:

$$2^{m+2} - 4 - 3m.$$

The total quantum cost of the quantum counter for each term T is:

$$\text{Peres cost} = T * m^2. \tag{1}$$

$$\text{Toffoli cost} = T * (2^{m+2} - 4 - 3m). \tag{2}$$

Based on these two Formulas (1) and (2), the Toffoli gate has a higher quantum cost than Peres gate. Thus, we used in our design the Peres gates. As we mentioned before, our final counter uses Peres gates, so we built our oracle using the Peres gate, and it is mapping to the n th root of NOT gates which leads to low quantum cost. The recursive design method from Peres gate was used.

6. Variants of SAT Oracles Using Quantum Counter

Following our preliminary work [61], in this section, we discuss some other applications of the quantum counter in variants of satisfiability, such as the product of SOPs SAT.

6.1. Oracle for SOPs

MAX-SAT can be solved for a Product of any function. In particular, this can be a Product of SOPs. The SOP functions can be implemented with a counter by summing the digits of the counter at the end, using De Morgan’s rule. Each product term is simply a Toffoli gate, and the counter can be checked in a similar way to a regular sum term. Figure 23 presents an example circuit for the function $ab + b\bar{c} + \bar{a}\bar{c}$.

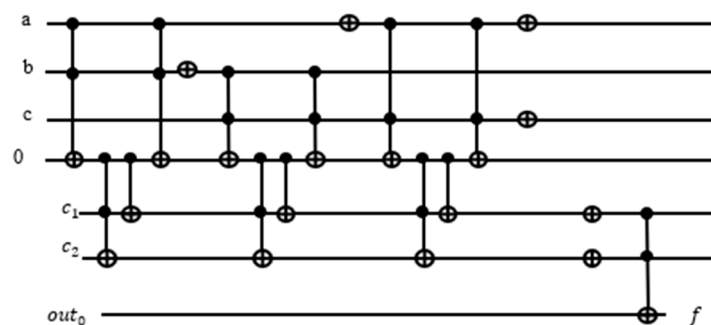


Figure 23. Part of the product of SOP oracle that realizes SOP function $f = ab + b\bar{c} + \bar{a}\bar{c}$.

6.2. Oracle for Product of SOPs (POSOP SAT)

POSOP functions consist of products of SOP functions. We were not able to find any references to this form of SAT. However, we can take advantage of the fact that every term must be true in a product for the product to be true, and thus we can check against a counter value of the number of terms in order to construct the oracle for POSOP. For example, Figure 24 presents the circuit for function $(\bar{a}b + \bar{a}c)(abc + \bar{b}\bar{c})$.

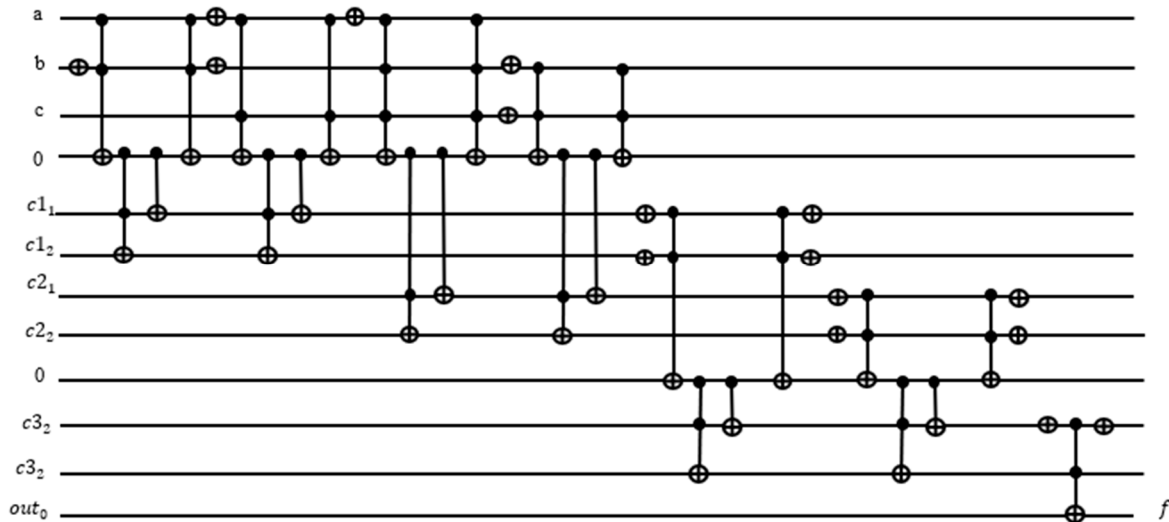


Figure 24. Realization of the Oracle for $f = (\bar{a}b + \bar{a}c)(abc + \bar{b}\bar{c})$, with POSOP SAT.

POSOP circuits are much larger than traditional SOP circuits since an additional counter is required for each SOP term. As such, it may be more advantageous to convert POSOP to a more standard form, such as SOP or POS to be implemented in reversible logic. This depends on a particular problem instance.

6.3. Oracle for Exclusive-or-Sum-of-Products (ESOP)

An Exclusive-or-Sum-of-Products (ESOP) form is an exclusive sum (using the ' \oplus ') operator of product terms. There is not much published on ESOP SAT except for [62], although this is an interesting subject. Grover's Oracle can be trivially applied to ESOP SAT, a problem that has also not been discussed yet. The advantage of ESOP SAT over OR SAT presented in the previous section is that ESOP SAT can be realized without the need for a large AND gate or a counter. Since every product in the EXOR sum can be implemented as a Toffoli gate, SAT with ESOP can be formulated with just the input qubits and one output qubit. For example, given a function such as $ab \oplus b\bar{c} \oplus \bar{a}\bar{c}$, we can implement Grover's Oracle, as shown in Figure 25.

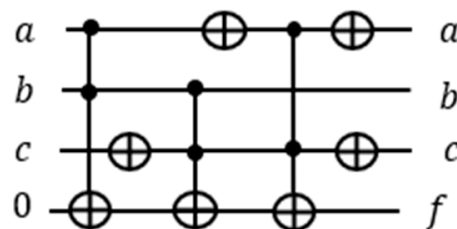


Figure 25. Realization of Oracle $f = ab \oplus b\bar{c} \oplus \bar{a}\bar{c}$ for ESOP SAT realized in Grover's Algorithm.

7. OR Satisfiability Problems for Electronic Design Automation

In this section, we will show that many EDA (Electronic Design Automation) problems can be reduced to SAT and MAX-SAT. In the most general case, the Satisfiability Decision Function problem is formulated as an arbitrary binary-valued-input, binary-output, and

single-output function. For instance, a product of sums of literals, (the literals are variables negated or not), EXOR of products of literals, and product of sums of products of literals. These functions are created by transforming some natural language or mathematical decision problems, such as, for instance, cryptographic puzzles. The question is to find out for which values of variables the formula for SAT or MAX-SAT is satisfied. In some problems, one has to find all solutions; in some other problems we look for just one solution or only some solutions. For all these variants, we have some freedom to modify Grover's Algorithm, and/or call it several times with modified oracles [60].

Below we will systematically formulate several satisfiability types of problems, starting from the simplest ones. We concentrate on problems that have applications in EDA. Each of these basic problems below can have in addition several variants related to specific applications. Given is a product of terms, each term being a Boolean sum of literals, and each literal being a Boolean variable or its negation. We are interested in the following problems.

Problem 1 (Satisfiability): *Answer Yes if there exists a product of literals that satisfies all terms or No if such a product does not exist. Give the solution as a set of literals.*

Problem 2 (Optimization of the Generalized Petrick function): *Find a product with the minimum number of literals that satisfies all terms or prove that such a product does not exist.*

Problem 3 (Optimization of the Generalized Petrick function-nonnegated literal variant): *Find such a product of literals that satisfies all terms and in which a minimum number of literals is not negated or prove that no such product exists. (The not negated literals will also be called positive literals). In particular, the Petrick Function is positive unate, which it means has only positive literals.*

Problem 4 (MAX-SAT): *Find such set of literals that satisfies the maximum number of terms.*

Problem 5 (Tautology Checking): *Verify whether a function is a Sum of Product Form is a Boolean tautology. Function F is a tautology (all input combinations are 1) when its negation \bar{F} is not satisfiable (all combinations are 0).*

In some variants of these problems, depending on a particular application, we can look for all solutions, all optimal solutions, some optimal solutions, or for a single optimal solution. The central role of the Problem 1 is well-established in computer science. All NP-complete combinational decision problems are equivalent to the Satisfiability Problem [63]. Many reductions of practically important problems to other above problems were shown, including problems from VLSI Design Automation, especially in logic design and state machine design. SAT and MAX-SAT also have many applications in logistics, scheduling, AI, and robotics. Ashenhurst/Curtis Decomposition of Boolean functions can be done in an algorithm that repeatedly applies Satisfiability [64]. Generalized Ashenhurst/Curtis Decomposition was also realized by building a complex oracle for Grover's Algorithm based on the mathematics of Partition Calculus [65]. These SAT-like problem formulations are also of fundamental importance in many algorithms for Boolean minimization, factorization, and multi-level design. The set covering problem is reduced to the minimization of Petrick Function. The reductions of many practically important NP-hard combinatorial optimization problems can also be found in the literature. For instance, the minimization of the Sum of Products Boolean functions can be reduced to the Covering Problem [66] and Covering Problem can be further reduced to the Petrick Function Optimization Problem (PFOP) [67]. Many other problems, like test minimization, can also be reduced to the Covering Problem [66,68]. The problems of Partial Satisfiability and its applications are discussed by K. Lieberherr [69]. Many other reductions to the formulated above problems are discussed in [63,70]. Paper [71] discusses the reduction of three-level NAND circuits, TANT, to the covering-closure problem solved similarly to SAT. A similar problem of the synthesis of networks from negative gates uses the same reduction [72]. A design

automation system [73] was created, in which many problems were first reduced to the few selected “generic” combinatorial optimization problems. These problems include some of the problems listed above.

The problem of minimization of Finite State Machines includes: (1) the Maximum Clique Problem and (2) the problem of finding the minimum closed and a complete subgraph of a graph (Closure/Covering Problem) [71]. The first of these problems, (1), can be reduced to the Petrick Function Optimization Problem (PFOP). The problem of optimum output phase optimization of PLA [74] can be reduced to PFOP. The second problem, (2), can be reduced to the Generalized Petrick Function Optimization Problem (GPFOP), introduced above and illustrated below. Many other problems, like AND/OR graph searching [75], were reduced to the Closure/Covering Problem.

A number of problems (including Boolean minimization [76], layout compaction, and minimization of the number of registers in hardware compilation can be reduced to the Minimal Graph Coloring Problem. Regular layout problems can be reduced to SAT [77]. The Minimal Graph Coloring can be reduced to the Problem of Finding the Maximum Independent Sets, and next the Covering Problem (Maghout algorithm). The Problem of Finding the Maximum Independent Sets can be reduced to PFOP. The PFOP is a particular case of the GPFOP. The role and importance of Tautology and conversion methods from SOP to POS and vice versa in logic design are well known. These problems can also be solved using SAT.

Concluding on OR SAT. In theory, every NP problem can be polynomially reduced to SAT and also to OR 3-SAT. But this is not practical. Many problems can be reduced to graph coloring or maximum clique problems that can be in turn reduced to satisfiability problems.

As we see now, many problems can be solved with quadratic speedup using future quantum computers. A hybrid classical/quantum computer based on Grover tuned to solve variants of SAT problems of various types would be a tremendous asset to all these problems [60].

8. Conclusions

We have designed a novel quantum oracle circuit that requires a 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 overall in 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 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 the IBM QISKIT simulator [23] that provided the expected results. We presented other Variants of SAT oracles that can be designed for the oracle circuit using the quantum counter. Also, we discussed many other potential problems in the area of EDA that can be reduced to SAT and MAX-SAT such that the oracle can be constructed the quantum counter idea.

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 functions. This type of problem is 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.

Author Contributions: Conceptualization, A.A. and P.J.; Data curation, A.A.; Formal analysis, A.A.; Investigation, A.A.; Methodology, A.A., P.J. and M.P.; Project administration, M.P.; Resources, A.A.; Software, A.A.; Supervision, M.P.; Validation, A.A. and M.P.; Visualization, A.A.; Writing—original

draft, A.A. and P.J.; Writing—review & editing, A.A. and M.P. All authors have read and agreed to the published version of the manuscript.

Funding: This article’s publication was funded by the Portland State University Open Access Article Processing Charge Fund, grant number: PO 23-24362, administered by the Portland State University Library.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Marques-Silva, J.; Glass, T. Combinational equivalence checking using satisfiability and recursive learning. In Proceedings of the Conference on Design, Automation and Test in Europe, Munich, Germany, 1 January 1999; p. 33.
2. Konuk, H.; Larrabee, T. Explorations of sequential ATPG using Boolean satisfiability. In Proceedings of the Digest of Papers Eleventh Annual 1993 IEEE VLSI Test Symposium, Atlantic City, NJ, USA, 6–8 April 1993; IEEE: Manhattan, NY, USA, 1993; pp. 85–90.
3. Biere, A.; Cimatti, A.; Clarke, E.M.; Fujita, M.; Zhu, Y. Symbolic model checking using SAT procedures instead of BDDs. In Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, New Orleans, LA, USA, 21–25 June 1999; 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.; Mitra, S. QED: Quick error detection tests for effective post-silicon validation. In Proceedings of the 2010 IEEE International Test Conference, Austin, TX, USA, 2–4 November 2010; IEEE: Manhattan, NY, USA, 2010; pp. 1–10.
5. Wang, P.W.; Donti, P.; Wilder, B.; Kolter, Z. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 10–15 June 2019; PMLR: New York, NY, USA, 2019; pp. 6545–6554.
6. Cook, S.A. The complexity of theorem-proving procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing, Shaker Heights, OH, USA, 3–5 May 1971; pp. 151–158.
7. Kohli, R.; Krishnamurti, R.; Mirchandani, P. The minimum satisfiability problem. *SIAM J. Discret. Math.* **1994**, *7*, 275–283. [[CrossRef](#)]
8. Biere, A.; Heule, M.; van Maaren, H. (Eds.) *Handbook of Satisfiability*; IOS Press: Washington, DC, USA, 2009; Volume 185.
9. Fu, Z.; Malik, S. On solving the partial MAX-SAT problem. In Proceedings of the International Conference on Theory and Applications of Satisfiability Testing, Seattle, WA, USA, 12–15 August 2006; Springer: Berlin/Heidelberg, Germany, 2006; pp. 252–265.
10. Berg, O.J.; Hyttinen, A.J.; Järvisalo, M.J. Applications of MaxSAT in data analysis. In Proceedings of the Pragmatics of SAT 2015 and 2018, Oxford, UK, 7 July 2018.
11. Berg, J.; Järvisalo, M. Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. *Artif. Intell.* **2017**, *244*, 110–142. [[CrossRef](#)]
12. Berg, J.; Järvisalo, M.; Malone, B. Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In Proceedings of the Artificial Intelligence and Statistics, Reykjavik, Iceland, 22–25 April 2014; PMLR: New York, NY, USA, 2014; pp. 86–95.
13. Hyttinen, A.; Saikko, P.; Järvisalo, M. A core-guided approach to learning optimal causal graphs. In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017), Melbourne, Australia, 19–25 August 2017.
14. Malioutov, D.; Meel, K.S. MLIC: A MaxSAT-based framework for learning interpretable classification rules. In Proceedings of the International Conference on Principles and Practice of Constraint Programming, Lille, France, 27–31 August 2018; Springer: Cham, Switzerland, 2018; pp. 312–327.
15. Dimitrova, R.; Ghasemi, M.; Topcu, U. Maximum realizability for linear temporal logic specifications. In Proceedings of the International Symposium on Automated Technology for Verification and Analysis, Los Angeles, CA, USA, 7–10 October 2018; Springer: Cham, Switzerland, 2018; pp. 458–475.
16. Zhang, L.; Bacchus, F. MAXSAT heuristics for cost optimal planning. In Proceedings of the AAAI Conference on Artificial Intelligence, Toronto, ON, Canada, 22–26 July 2012; Volume 26, pp. 1846–1852.
17. Muise, C.; Beck, J.C.; McIlraith, S.A. Optimal partial-order plan relaxation via MaxSAT. *J. Artif. Intell. Res.* **2016**, *57*, 113–149. [[CrossRef](#)]
18. Demirović, E.; Musliu, N.; Winter, F. Modeling and solving staff scheduling with partial weighted maxSAT. *Ann. Oper. Res.* **2019**, *275*, 79–99. [[CrossRef](#)]
19. Safarpour, S.; Mangassarian, H.; Veneris, A.; Liffiton, M.H.; Sakallah, K.A. November. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer Aided Design (FMCAD’07)*; IEEE: Cham, Switzerland, 2007; pp. 13–19.
20. Chen, Y.; Safarpour, S.; Veneris, A.; Marques-Silva, J. Spatial and temporal design debug using partial MaxSAT. In Proceedings of the 19th ACM Great Lakes symposium on VLSI, Boston Area, MA, USA, 10–12 May 2009; pp. 345–350.
21. Chen, Y.; Safarpour, S.; Marques-Silva, J.; Veneris, A. Automated design debugging with maximum satisfiability. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2010**, *29*, 1804–1817. [[CrossRef](#)]

22. Jose, M.; Majumdar, R. Cause clue clauses: Error localization using maximum satisfiability. *ACM SIGPLAN Not.* **2011**, *46*, 437–446. [[CrossRef](#)]
23. Zhu, C.S.; Weissenbacher, G.; Malik, S. Post-silicon fault localisation using maximum satisfiability and backbones. In Proceedings of the 2011 Formal Methods in Computer-Aided Design (FMCAD), Austin, TX, USA, 30 October–2 November 2011; IEEE: Cham, Switzerland, 2011; pp. 63–66.
24. Wickramarachchi, G.T.; Qardaji, W.H.; Li, N. An efficient framework for user authorization queries in RBAC systems. In Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, 3–5 June 2009; pp. 23–32.
25. Liao, X.; Zhang, H.; Koshimura, M. Reconstructing AES key schedule images with SAT and MaxSAT. *IEICE Trans. Inf. Syst.* **2016**, *99*, 141–150. [[CrossRef](#)]
26. Shabani, A.; Alizadeh, B. PMTP: A MAX-SAT-based approach to detect hardware trojan using propagation of maximum transition probability. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2018**, *39*, 25–33. [[CrossRef](#)]
27. Feng, Y.; Bastani, O.; Martins, R.; Dillig, I.; Anand, S. Automated synthesis of semantic malware signatures using maximum satisfiability. *arXiv* **2016**, arXiv:1608.06254.
28. Lin, P.C.K.; Khatri, S.P. Application of Max-SAT-based ATPG to optimal cancer therapy design. *BMC Genom.* **2012**, *13*, 1–10. [[CrossRef](#)]
29. Guerra, J.; Lynce, I. Reasoning over biological networks using maximum satisfiability. In Proceedings of the International Conference on Principles and Practice of Constraint Programming, Québec City, QC, Canada, 8–12 October 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 941–956.
30. Martins, R. Solving RNA alignment with MaxSAT. In *MaxSAT Evaluation*; University of Helsinki: Helsinki, Finland, 2017; p. 29.
31. Graça, A.; Marques-Silva, J.; Lynce, I.; Oliveira, A.L. Haplotype inference with pseudo-Boolean optimization. *Ann. Oper. Res.* **2011**, *184*, 137–162. [[CrossRef](#)]
32. Li, C.M.; Quan, Z. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence, Atlanta, GA, USA, 11–15 July 2010.
33. Li, C.M.; Jiang, H.; Xu, R.C. Incremental MaxSAT reasoning to reduce branches in a branch-and-bound algorithm for MaxClique. In Proceedings of the International Conference on Learning and Intelligent Optimization, Lille, France, 12–15 January 2015; Springer: Cham, Switzerland, 2015; pp. 268–274.
34. Fang, Z.; Li, C.M.; Qiao, K.; Feng, X.; Xu, K. Solving Maximum Weight Clique Using Maximum Satisfiability Reasoning. In Proceedings of the ECAI, Prague, Czech, 18–22 August 2014; pp. 303–308.
35. Berg, J.; Järvisalo, M. SAT-based approaches to treewidth computation: An evaluation. In Proceedings of the 2014 IEEE 26th International Conference on Tools with Artificial Intelligence, Limassol, Cyprus, 10–12 November 2014; IEEE: Cham, Switzerland, 2014; pp. 328–335.
36. Morgado, A.; Marques-Silva, J. Combinatorial optimization solutions for the maximum quartet consistency problem. *Fundam. Inform.* **2010**, *102*, 363–389. [[CrossRef](#)]
37. Smyth, K.; Hoos, H.H.; Stützle, T. Iterated robust tabu search for MAX-SAT. In Proceedings of the Conference of the Canadian Society for Computational Studies of Intelligence, Halifax, NS, Canada, 11–13 June 2003; Springer: Berlin/Heidelberg, Germany, 2003; pp. 129–144.
38. Mastrolilli, M.; Gambardella, L.M. Maximum satisfiability: How good are tabu search and plateau moves in the worst-case? *Eur. J. Oper. Res.* **2005**, *166*, 63–76. [[CrossRef](#)]
39. Cai, S.; Lei, Z. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artif. Intell.* **2020**, *287*, 103354. [[CrossRef](#)]
40. Marchiori, E.; Rossi, C. A flipping genetic algorithm for hard 3-SAT problems. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99), Orlando, FL, USA, 13–17 July 1999; pp. 393–400.
41. Layeb, A.; Deneche, A.H.; Meshoul, S. A new artificial immune system for solving the maximum satisfiability problem. In Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, Cordoba, Spain, 1–4 June 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 136–142.
42. Munawar, A.; Wahib, M.; Munetomo, M.; Akama, K. Hybrid of genetic algorithm and local search to solve max-sat problem using NVIDIA CUDA framework. *Genet. Program. Evolvable Mach.* **2009**, *10*, 391–415. [[CrossRef](#)]
43. Lardeux, F.; Saubion, F.; Hao, J.K. GASAT: A genetic local search algorithm for the satisfiability problem. *Evol. Comput.* **2006**, *14*, 223–253. [[CrossRef](#)]
44. Davis, M.; Logemann, G.; Loveland, D. A machine program for theorem-proving. *Commun. ACM* **1962**, *5*, 394–397. [[CrossRef](#)]
45. Li, C.M.; Manyà, F.; Planes, J. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In Proceedings of the International Conference on Principles and Practice of Constraint Programming, Sitges, Spain, 1–5 October 2005; Springer: Berlin/Heidelberg, Germany, 2005; pp. 403–414.
46. Li, C.M.; Xu, Z.; Coll, J.; Manyà, F.; Habet, D.; He, K. Combining clause learning and branch and bound for MaxSAT. In Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming (CP 2021), Montpellier, France, 25–29 October 2021; Schloss Dagstuhl-Leibniz-Zentrum für Informatik: Wadern, Germany, 2021.
47. Gu, J. Efficient local search for very large-scale satisfiability problems. *ACM SIGART Bull.* **1992**, *3*, 8–12. [[CrossRef](#)]

48. Bian, Z.; Chudak, F.; Macready, W.; Roy, A.; Sebastiani, R.; Varotti, S. Solving sat and maxsat with a quantum annealer: Foundations and a preliminary report. In Proceedings of the International Symposium on Frontiers of Combining Systems, Brasília, Brazil, 27–29 September 2017; Springer: Cham, Switzerland, 2017; pp. 153–171.
49. Cheng, S.T.; Tao, M.H. Quantum cooperative search algorithm for 3-SAT. *J. Comput. Syst. Sci.* **2007**, *73*, 123–136. [[CrossRef](#)]
50. Lee, S.; Lee, S.-J.; Kim, T.; Biamonte, J.; Perkowski, M. The cost of Quantum Gate Primitives. *J. Mult. Valued Log. Soft Comput.* **2006**, *12*, 561–573.
51. Barenco, A.; Bennett, C.H.; Cleve, R.; DiVincenzo, D.P.; Margolus, N.; Shor, P.; Sleator, T.; Smolin, J.A.; Weinfurter, H. Elementary gates for quantum computation. *Phys. Rev. A* **1999**, *52*, 3457. [[CrossRef](#)] [[PubMed](#)]
52. Maslov, D.; Dueck, G.W. Improved quantum cost for n-bit Toffoli gates. *Electron. Lett.* **2003**, *39*, 1790–1791. [[CrossRef](#)]
53. Peres, A. Reversible logic and quantum computers. *Phys. Rev. A* **1985**, *32*, 3266. [[CrossRef](#)] [[PubMed](#)]
54. Szyppowski, M.; Kerntopf, P. Low quantum cost realization of generalized peres and toffoli gates with multiple-control signals. In Proceedings of the 2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013), Beijing, China, 5–8 August 2013; IEEE: Manhattan, NY, USA, 2013; pp. 802–807.
55. Nielsen, M.A.; Chuang, I. *Quantum Computation and Quantum Information*; Cambridge University Press: Cambridge, UK, 2002.
56. Wong, R.; Chang, W.L. Quantum speedup for protein structure prediction. *IEEE Trans. NanoBiosci.* **2021**, *20*, 323–330. [[CrossRef](#)]
57. Chang, W.L.; Chen, J.C.; Chung, W.Y.; Hsiao, C.Y.; Wong, R.; Vasilakos, A.V. Quantum speedup and mathematical solutions of implementing bio-molecular solutions for the independent set problem on IBM quantum computers. *IEEE Trans. NanoBiosci.* **2021**, *20*, 354–376. [[CrossRef](#)]
58. Wong, R.; Chang, W.L. Fast quantum algorithm for protein structure prediction in hydrophobic-hydrophilic model. *J. Parallel Distrib. Comput.* **2022**, *164*, 178–190. [[CrossRef](#)]
59. 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.; et al. *Qiskit: An Open-Source Framework for Quantum Computing*; Zenodo: Honolulu, HI, USA, 2019.
60. Perkowski, M. Inverse Problems, Constraint Satisfaction, Reversible Logic, Invertible Logic and Grover Quantum Oracles for Practical Problems. In Proceedings of the International Conference on Reversible Computation, Oslo, Norway, 9–10 July 2020; Springer: Cham, Switzerland, 2020; pp. 3–32.
61. Alasow, A.; Perkowski, M. Quantum Algorithm for Maximum Satisfiability. In Proceedings of the 2022 IEEE 52nd International Symposium on Multiple-Valued Logic (ISMVL), Dallas, TX, USA, 18–20 May 2022; pp. 27–34.
62. Csanky, L. On the Generalized Reed-Muller Canonical Form of Boolean Functions: Research Project. Ph.D. Thesis, University of California, Berkeley, CA, USA, 1972.
63. Garey, M.R.; Johnson, D.S. *A Guide to the Theory of NP-Completeness. Computers and Intractability*; W. H. Freeman & Co: New York, NY, USA, 1979.
64. Lin, H.P.; Jiang, J.H.R.; Lee, R.R. To SAT or not to SAT: Ashenhurst decomposition in a large scale. In Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, USA, 10–13 November 2008; IEEE: Manhattan, NY, USA, 2008; pp. 32–37.
65. Li, Y.; Tsai, E.; Perkowski, M.; Song, X. Grover-based Ashenhurst-Curtis decomposition using quantum language quipper. *Quantum Inf. Comput.* **2019**, *19*, 35–66. [[CrossRef](#)]
66. Breuer, M.A.; Preiss, R.J. *Design Automation of Digital Systems*; Prentice Hall: Hoboken, NJ, USA, 1972; Volume 1.
67. Slagle, J.R. *Artificial Intelligence: The Heuristic Programming Approach*; McGraw-Hill: New York, NY, USA, 1971.
68. Kohavi, Z.; Jha, N.K. *Switching and Finite Automata Theory*; Cambridge University Press: Cambridge, UK, 2009.
69. Lieberherr, K.; Specker, E. Complexity of partial satisfaction. In Proceedings of the 20th Annual Symposium on Foundations of Computer Science (sfcs 1979), Washington, DC, USA, 29–31 October 1979; IEEE: Manhattan, NY, USA, 1979; pp. 132–139.
70. Perkowski, M. *State-Space Approach to the Design of a Multipurpose Problem-Solver for Logic Design*; KU Leuven: North Holland, Amsterdam, 1978.
71. Perkowski, M. Synthesis of multioutput three level NAND networks. In Proceedings of the Seminar on Computer Aided Design, Budapest, Hungary, 3–5 November 1976; pp. 238–265.
72. Perkowski, M. Minimization of two-level networks from negative gates. In Proceedings of the Midwest, Lincoln, Nebraska, 10–12 August 1986; Volume 86, pp. 756–761.
73. Perkowski, M.; Liu, J.; Brown, J. A System for Fast Prototyping of Logic Design Programs. In Proceedings of the 1987 Midwest Symposium on Circuits and Systems, Syracuse, NY, USA, 17–18 August 1987.
74. Sasao, T. Input variable assignment and output phase optimization of PLA's. *IEEE Trans. Comput.* **1984**, *33*, 879–894. [[CrossRef](#)]
75. Nilsson, J. *Problem-Solving Methods in Artificial Intelligence*; McGraw-Hill: New York, NY, USA, 1971.
76. Nguyen, L.B.; Perkowdki, M.A.; Goldstein, N.B. Palmini—Fast Boolean minimizer for personal computers. In Proceedings of the 24th ACM/IEEE Design Automation Conference, Miami Beach, FL, USA, 28 June–1 July 1987; pp. 615–621.
77. Perkowski, M.; Mishchenko, A. Logic synthesis for regular layout using satisfiability. In Proceedings of the International Symposium on Boolean Problems, Freiberg, Germany, 19–20 September 2002.