12-2023

Quantum Algorithms for Unate and Binate Covering Problems with Application to Finite State Machine Minimization

Abdirahman Alasow
Portland State University, alasow@pdx.edu

Marek Perkowski
Portland State University, marek.perkowski@pdx.edu

Quantum Algorithms for Unate and Binate Covering Problems with Application to Finite State Machine Minimization

Abdirahman Alasow
Portland State University, Portland, OR, USA
alasow@pdx.edu

Marek Perkowski
Portland State University, Portland, OR, USA
mperkows@ee.pdx.edu

Abstract

Covering problems find applications in many areas of computer science and engineering, such that numerous combinatorial problems can be formulated as covering problems. Combinatorial optimization problems are generally NP-hard problems that require an extensive search to find the optimal solution. Exploiting the benefits of quantum computing, we present a quantum oracle design for covering problems, taking advantage of Grover's search algorithm to achieve quadratic speedup. This paper also discusses applications of the quantum counter in unate covering problems and binate covering problems with some important practical applications, such as finding prime implicants of a Boolean function, implication graphs, and minimization of incompletely specified Finite State Machines.

1 Introduction

Many optimization problems can be formulated as the selection of a subset from a larger set. One familiar form is the covering problems [8, 20, 26]. Covering problems are minimization problems for combinatorial optimization problems in which a certain combinatorial structure covers another. For instance, in logic design, the covering problems can be formulated as a set of minterms to be covered by a set of subsets of minterms (such as the prime implicants for a specific example). Minterms for a Boolean function of n variables are products of literals for all variables of this

function. A literal is a variable or the negation of a variable. True minterms are those for which the value of the function is 1. The prime implicant is the product of literals that cannot be extended by removing some of the literals and which covers some subset of true minterms. Covering problems are generally given as a table with rows corresponding to the set elements and columns corresponding to the subsets.

Covering problems include two main types: unate covering problem (UCP) and binate covering problem (BCP) [28, 12, 18]. The unate covering problem is the problem of finding a minimum cost assignment to variables to which a given Boolean function f is equal to 1. The literals are all in positive form (uncomplemented). The Binate covering problem has the extra constraint that negative literals may be present. The covering problem can be generalized by assuming that the choice of a subset implies the choice of another subset. This additional constraint can be represented by an implication clause. For example, if the selection of group a implies the choice of b ($a \Rightarrow b$), then the clause $\bar{a} + b$ is added. Note that the implication clause makes the product of sums form binate in variable a, because a (uncomplemented) is also part of some covering clauses. Therefore, this class of problems is called binate covering or covering with closure. There are two main ways to express the covering problems: constraints in the form of a matrix or a product of sum (POS) form of a Boolean equation $f = 1$. This formulation of POS is known as Petrick's method [33]. If the covering problems are expressed in the form of a matrix, then the corresponding matrix (covering matrix) of the UCP is filled with elements from the set $\{1, 0\}$ while the BCP is filled with elements from the set $\{1, -1, 0\}$. A -1 entry corresponds to a complemented variable, and a 1 entry to an uncomplemented one. In the covering matrix, the rows correspond to each term of the expression, and the columns correspond to each variable. Covering problems can also be expressed in the form of a POS formula, such that $f(x_1, x_2, x_3, x_4) = (x_1 + x_2)(x_2 + x_3)(x_1 + x_4)x_3x_4 = 1$ then this is called an unate covering, which always has a solution. If some of the variables in the function appear both positive and negative (complement) such that $f(x_1, x_2, x_3, x_4) = (x_1 + x_3 + x_4)(x_1 + \overline{x_2} + x_4)(\overline{x_2} + x_3 + x_4)(x_2 + \overline{x_3} + x_4)$ then it is called binate covering, which may or may not have a solution. In this paper, we will express the covering problems in POS form.

2 Related Work

The fundamental covering problem known from computer science has many practical applications in electronic design automation (EDA) and digital systems.

2.1 Applications of Covering Problems

There are many combinatorial optimization problems for covering problems in various areas, such as logic minimization [35], scheduling [27], parallel computation on a GPU [30], and allocation, encoding, and routing [7] for unate covering problem. Binate covering problem is used in finite state machine minimization [12, 17], technology mapping [28], Boolean relations [15], and Directed Acyclic Graphs (DAG) covering problems [35, 24, 25]. Logic minimization is the process of finding the optimal implementation of a logic function. The minimum-cost implementation is achieved when the cover of a given function consists of the minimum number of prime implicants to represent the function. In another variant, the solution should have the total minimum cost of selected prime implicants. The Unate covering can be used to achieve exact logic minimization [35]. Scheduling problems can be formulated as UCP, such as vehicle and crew scheduling problems in [27], to minimize the combined vehicle and crew cost. Parallel computation on a GPU [30, 40, 39, 38] was explored using the Unate covering problem. The classical task of the GPU is matrix multiplication, which can be mapped to the UCP implementation. Also, UCP has many other applications in logistic problems such as allocation, encoding, and routing [7]. Finite state machine minimization [12, 17] can be formulated as a binate covering problem such that the number of internal states in the finite state machine can be reduced. A standard-cell library must first comply with the available library primitives in VLSI, a process known as technology mapping. Finding the optimal mapping of logic gates to VLSI library cells can be accomplished via binate covering [28]. Boolean relations such as two-level logic minimization under the Sum-of-Product (SOP) representation can be solved based on the binate covering problem formulation [15]. Finding the minimum set of nodes or paths covering every node in a directed acyclic graph is called DAG covering. This can be formulated as a binate covering problem [35, 24, 25] by constructing the closure condition of the graph.

2.2 Classical Algorithms for Covering Problems

The covering problem is considered NP-hard [34, 16], and much effort has been spent on it because of its wide applications. Several classical algorithms are proposed for covering problems based on exact and heuristic algorithms. Several exact algorithms are proposed for covering problems, such as the most widely known approach, the branch and bound algorithm [41, 15, 36, 22], with many techniques suggested for lower bound and upper bound improvement using pruning techniques. In the branch and bound technique, the covering table is expressed as the POS of the constraints,

and the problem solution explores, in the worst case, all possible solution instances. Branch and bound employs the upper bound and lower bound methods. For each solution to the constraints, upper bounds on the value of the cost function are identified, and lower bounds are estimated using the current set of variable assignments. The upper bound value is updated each time a new lower-cost solution is discovered. When the lower bound estimation is greater than or equal to the most recently computed upper bound, the search can be pruned. As a result, a better solution will not be found using the current variable assignments, allowing us to prune the search. Branch and bound used reduction and bounding techniques of search space for solutions to avoid the generation of some of the suboptimal solutions. The upper bound is the cost of the cheapest solution seen so far. Eventually, it will be the cost of the optimal solution. While the lower bound is an estimate of the minimum cost of a solution for the problem. The lower bound is the most important factor for runtime.

A Binary Decision Diagrams (BDDs) based algorithm [41] was proposed to solve the covering problem such that finding the solution only requires computing the shortest path in the BDD. The number of variables in the BDD is equal to the number of columns in the binate table. However, the BDD tree is too large to be built when there are many variables. A mixed technique of both branch-bound and BDD-based algorithms was proposed in [15], such that the constraints are represented as a conjunction of BDDs. This method leads to an effective method to compute a lower bound on the cost of the solution. Exact algorithms are computationally expensive for large problems because of exhaustive searches. Although there are several improvements using pruning and reduction techniques for exact algorithms, in general, the computational time can be exponential. Thus, a heuristic approach [16, 37] has been proposed for covering problems that provide suboptimal solutions. The heuristic approach in [37] has time complexity $O(n^2m)$ where n is the number of variables and m the number of terms in the covering problem. While the literature proposes both exact and heuristic approaches for classical algorithms, there is still a need for efficient algorithms that may take advantage of quantum algorithms to solve covering problems efficiently.

3 Quantum Algorithm for Covering Problem

Since classical optimization techniques are inefficient for solving NP-hard problems in terms of computational complexity, we present a quantum algorithm for solving the covering problems. To the best of our knowledge, this is the first quantum algorithm for solving covering problems. Algorithms with quantum oracles are bet-

ter than the corresponding classical search algorithms because they operate using quantum parallelism and quantum superposition, with all vectors being potential solutions simultaneously (all minterms). Thus, a quantum oracle that iterates sufficiently many times highly increases the probability of finding one of the solutions in a single measurement of all input qubits. Grover's algorithm implemented in quantum circuits gives a quadratic speedup when compared to an exhaustive classical algorithm for the same problem. Our paper reduces all covering problems to Grover's algorithm with an innovative way of building the quantum oracle that allows the number of qubits to be reduced logarithmically and at the same time solves both the decision and optimization problems in various variants.

A hybrid algorithm (a combination of classical and quantum) can be used to solve the covering problems, which assumes an arbitrary number of terms and variables. This algorithm would be a direct generalization of the algorithm presented in this paper. A classical computer can use any type of heuristic or algorithmic search method to expand a search tree. The upper part of the tree is created on a classical computer using all kinds of general search strategies, heuristic functions, cost functions, parameters, and constraints such as those discussed in [32, 18, 22]. This way, the sizes of the macro-leaves of the tree are reduced step by step such that the number of terms in them is less than m and the number of variables is less than n. For each macro-leaf, a quantum computer is called and executes a full search based on Grover's algorithm. Suppose the problem in the macro-leaves with less than m terms and less than n variables is recognized as a special type of problem. In that case, a special algorithm on a classical computer is executed for this reduced tree. This is a standard method used recently by several authors because Grover's algorithm gives a quadratic speedup only for problems for which a more efficient algorithm than a complete search does not exist. Also, this method allows for excellent scalability by sharing subtasks between the classical and quantum processors based on the availability of the quantum computer size (parameters of m and n).

3.1 Grover's Search Algorithm

Grover's algorithm [29, 42], 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 and evaluations of the classical oracle). In the average case, a particular element will be found in $\frac{N}{2}$ queries. Grover's algorithm can find elements in $\sqrt{N}$ queries. Thus, Grover's algorithm can be used to find all possible solutions for covering problems. Grover's algorithm is a quantum search algorithm that speeds up a classical search algorithm of complexity $O(N)$ to $O(\sqrt{N})$ in the space of N objects. Hence, Grover's algorithm gives a quadratic speedup. To

solve all optimal solutions of the covering problem, Grover's algorithm has to be repeated.

The covering problem contains n variables from the given Boolean function that are used to represent the search space of $N = 2^n$ elements. To solve the covering problem 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), 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 measuring the marked states, so that measuring the final states will return the target solution with a high probability near to 1.

An oracle is a black box operation that takes an input and gives an output, a yes/no decision. A quantum oracle is realized as a binary reversible circuit that is used in quantum algorithms for the estimation of the value of the Boolean function realized in it. The quantum oracle also has to replicate all input variables on the respective output qubits. If the function of the oracle is not reversible, we use ancilla qubits to make the function reversible. 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 $f(x)$ 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:

$$|x\rangle|y\rangle \rightarrow |x\rangle|q \oplus f(x)\rangle$$

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

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

As shown in Figure 1a, the n qubits in the superposition state result from applying a vector of Hadamard gates to the initial state $|0\rangle^n$. Next, we applied a repeated operator G which is called the Grover's Loop. After the iteration of the Grover's 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's Loop G is a quantum subroutine that can be broken into four steps, as shown in Figure 1b:

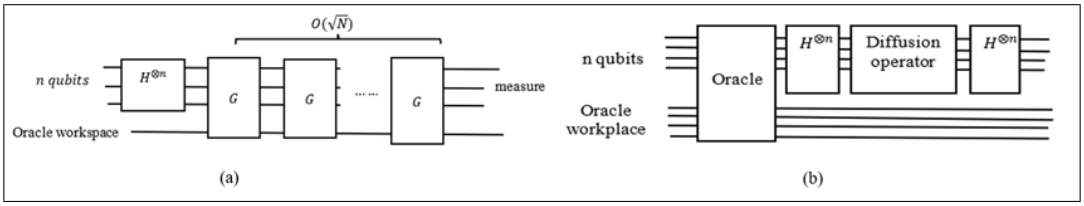


Figure 1: (a) Schematic circuit for Grover's algorithm [29]. (b) Grover's Loop Operator

1. Apply the Oracle O. This step is phase inversion.
2. Apply the Hadamard transform $H^{\otimes n}$, where $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
3. 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.
4. Apply the Hadamard transform $H^{\otimes n}$

The number of required iterations for Grover's 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. The literature includes many methods for efficient and optimal design of circuits in quantum oracle [6, 13, 19, 11].

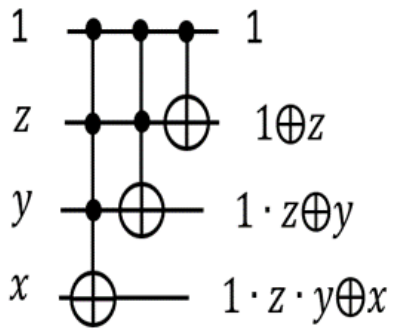
3.2 Quantum Counter

Each term of the clause required one multi-input Toffoli gate that has n qubit for the clause literals and one extra ancilla qubit to store the result of the clause. This was the traditional design of the quantum oracle, where each clause would have one ancilla qubit. For large problems, the number of ancilla qubits is very large, so even future quantum computers could not handle this approach. We propose an advanced quantum oracle design based on a quantum counter [3] that logarithmically reduces the number of required qubits. The quantum counter block is built from multi-input Toffoli and CNOT gates, where the first qubit of the quantum counter is applied as a constant 1 with the other qubits together.

In Figure 2a 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 $1 \cdot z \oplus y = 1 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$. 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. Below is a table in Figure 2b for all the input combinations of the 3-qubit quantum counter.



(a)

Input			Toffoli Gate Action	Output		
x	y	z		x	y	z
0	0	0	$x = 1 \cdot 0 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$ $y = 1 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$ $z = 1 \oplus 0 = 1$	0	0	1
0	0	1	$x = 1 \cdot 1 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$ $y = 1 \cdot 1 \oplus 0 = 1 \oplus 0 = 1$ $z = 1 \oplus 1 = 0$	0	1	0
0	1	0	$x = 1 \cdot 0 \cdot 1 \oplus 0 = 0 \oplus 0 = 0$ $y = 1 \cdot 0 \oplus 1 = 0 \oplus 1 = 1$ $z = 1 \oplus 0 = 1$	0	1	1
0	1	1	$x = 1 \cdot 1 \cdot 1 \oplus 0 = 1 \oplus 0 = 1$ $y = 1 \cdot 1 \oplus 1 = 1 \oplus 1 = 0$ $z = 1 \oplus 1 = 0$	1	0	0
1	0	0	$x = 1 \cdot 0 \cdot 0 \oplus 1 = 0 \oplus 1 = 1$ $y = 1 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$ $z = 1 \oplus 0 = 1$	1	0	1
1	0	1	$x = 1 \cdot 1 \cdot 0 \oplus 1 = 0 \oplus 1 = 1$ $y = 1 \cdot 0 \oplus 1 = 0 \oplus 1 = 1$ $z = 1 \oplus 1 = 0$	1	1	0
1	1	0	$x = 1 \cdot 0 \cdot 1 \oplus 1 = 0 \oplus 1 = 1$ $y = 1 \cdot 0 \oplus 1 = 0 \oplus 1 = 1$ $z = 1 \oplus 0 = 1$	1	1	1
1	1	1	$x = 1 \cdot 1 \cdot 1 \oplus 1 = 1 \oplus 1 = 0$ $y = 1 \cdot 1 \oplus 1 = 1 \oplus 1 = 0$ $z = 1 \oplus 1 = 0$	0	0	0

(b)

Figure 2: (a) Three-qubit quantum counter. (b) Analysis of a 3-qubit quantum counter block from (a)

We assign a counter block for each OR term (clause) and individual terms in the Boolean function for the covering problem, where the result of the clause is used as one of the control qubits of the counter. When the clause evaluates to 0, the counter forwards its input to the output without change. 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.

Our proposed concept of using a quantum counter can be used to design a quantum oracle for both decision and optimization problems, such as SAT-like, MAX-SAT problems, and many other problems in machine learning, such as mining frequent pattern generation [4]. In traditional quantum oracle design for SAT-like, MAX-SAT, and covering problems, every clause is built as a multi-input Toffoli gate. The number of qubits in the multi-input Toffoli gate is equal to the number of variables in the clause plus one extra ancilla qubit to save the result of the clause. In our design, there is no need for extra ancilla qubits for each clause, but several clauses have a quantum counter which shares ancilla qubits. For instance, if there are 30 clauses, our design requires only 5 ancilla qubits for all 30 clauses, rather than the 30 ancilla qubits required in the traditional quantum oracle. Thus, our design reduces the number of qubits logarithmically.

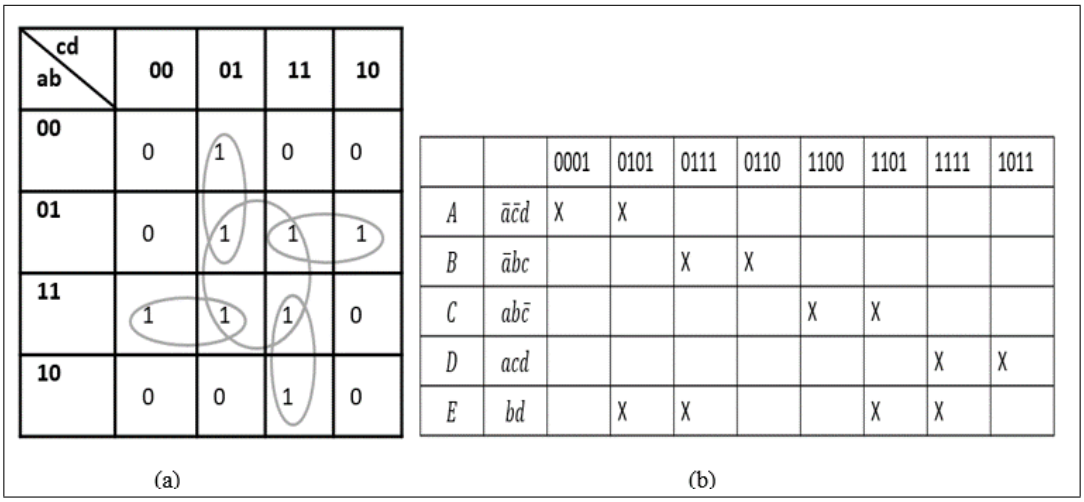
4 Grover Algorithm for Unate Covering Problem

The unate covering problem (also called the set covering problem [21]) is to find the minimum cost assignment of variables that satisfy a Boolean equation $f = 1$. The literals in the POS function f are all in the positive form (uncomplemented variables).

4.1 Finding All Prime Implicants for the Exact Minimum Covering of a SOP Circuit

Minterms for a Boolean function of n variables are products of literals for all variables of the function. A literal is a variable or the negation of a variable. True minterms are those for which the value of the function is 1. The prime implicant in a sum-of-product (SOP) structure is a product of literals that cannot be extended by removing some of the literals and which covers some subset of true minterms. For instance, given a function from Figure 3a, all its prime implicants are marked as ovals (loops). Using the minterm compatibility graph G, all prime implicants are found as maximum cliques. Prime implicants can also be found as maximum independent sets of graphs (G complement). Based on the truth table (or a Karnaugh Map) and the prime implicants, the covering table from Figure 3b is created. In this table,

every row represents a prime implicant as a product of Boolean literals, and each column represents a true minterm. The covering table is filled with symbol X for every minterm included in a prime implicant.



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

(a)

		0001	0101	0111	0110	1100	1101	1111	1011
A	$\bar{a}\bar{c}d$	X	X						
B	$\bar{a}bc$			X	X				
C	$ab\bar{c}$					X	X		
D	acd							X	X
E	bd		X	X			X	X	

(b)

Figure 3: (a) Truth table for all prime implicants of a SOP circuit. (b) Covering table for SOP function from (a)

From the table in Figure 3b, denoting rows A, B, C, D, E we compile the Petrick function in a standard way such that each column is created as one term by adding the variables in the row corresponding with symbol X. For instance, column 0101 has two cells filled with X. Adding the two rows $A+E$ as one term that corresponds to the symbols X in column 0101. In such a way, equation (1) is created:

$$A(A+E)(B+E)BD(D+E)(C+E)C = 1 \tag{1}$$

Equation (1) can be simplified using the Boolean law: $A(A+E) = A \cdot A + A \cdot E = A(1+E) = A$ to the following equation: $1 = A \cdot B \cdot C \cdot D$. Therefore, $f = A + B + C + D = \bar{a}\bar{c}d + \bar{a}bc + acd + ab\bar{c}$ is the minimum sum of products of expression of function f. In the case of many variables and clauses, solving this problem exactly is very difficult.

The main goal of this example is to show how SOP minimization can be solved using the UCP. Then the quantum oracle is built from UCP to apply Grover's algorithm. The quantum oracle is built from UCP (1). First, we need to convert each OR term into a product using De Morgan's Law $A+E = \overline{\overline{A+E}} = \overline{\overline{A}\overline{E}} = 1 \oplus \overline{A}\overline{E}$; $B+E = \overline{\overline{B+E}} = \overline{\overline{B}\overline{E}} = 1 \oplus \overline{B}\overline{E}$; $C+E = \overline{\overline{C+E}} = \overline{\overline{C}\overline{E}} = 1 \oplus \overline{C}\overline{E}$; $D+E = \overline{\overline{D+E}} = \overline{\overline{D}\overline{E}} = 1 \oplus \overline{D}\overline{E}$. From (1) we can rearrange $A \cdot B \cdot C \cdot D(A+E)(B+E)(D+E)(C+E)$.

To simplify the oracle design, we consider $A \cdot B \cdot C \cdot D$ as one term, which needs one quantum circuit. Then we connect one block of the iterative quantum counter after each Toffoli gate representing the OR term of the function POS formula. We put the ancilla qubit back to its original state by mirroring each Toffoli gate after each counter. In this case, we need five quantum counter circuits, as can be seen in Figure 4. Also, we need to add the NOT gate in the output circuit block, which makes the last qubit out_0 to produce 1 if the variable xyz in counter circuit is 101 such that the Boolean function in equation (1) is equal to 1.

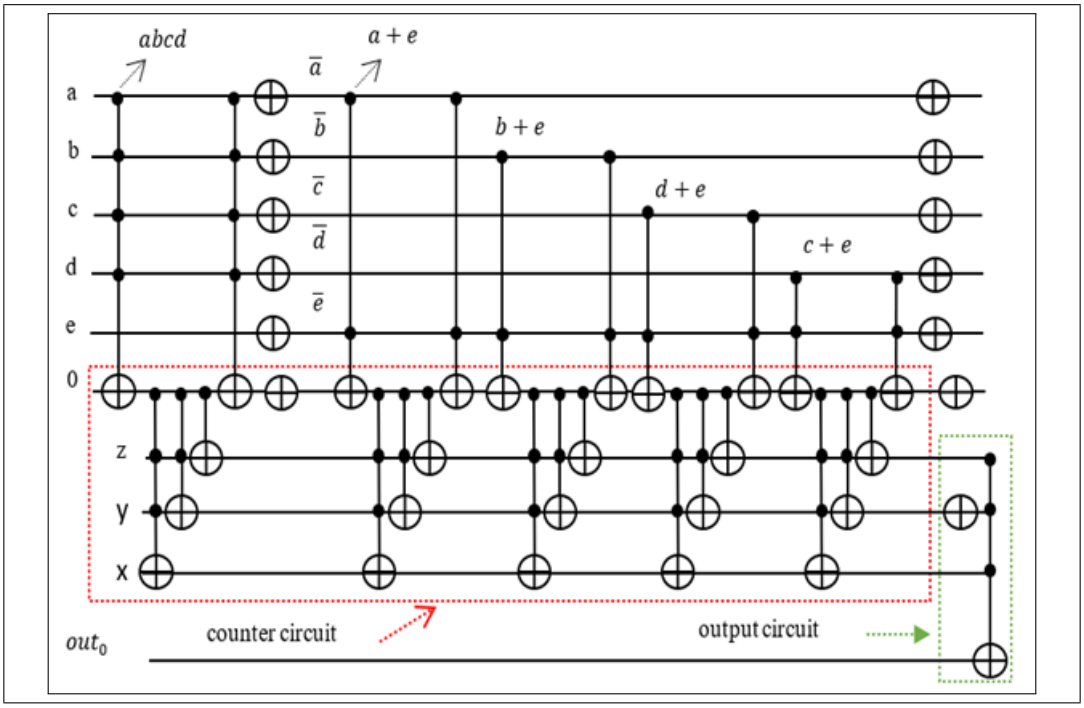


Figure 4: Quantum oracle for $A \cdot B \cdot C \cdot D(A+E)(B+E)(D+E)(C+E)$

In Figure 5, we applied the oracle circuit in Figure 4 in the Grover's search algorithm for iterations $R = 4$ from this formula: $R \leq \left\lceil \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rceil$ where $N = 2^5 = 32$ is the number of all search space elements since there are 5 variables for A, B, C, D, E. $M = 2$ is the number of solutions. $M = 2$ because in equation (1) is equal to 1 either $ABCDE = 11110$ or $ABCDE = 11111$. 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, a_3 and a_4 in Figure 5 where a_0, a_1, a_2, a_3, a_4 correspond to the Boolean variables A, B, C, D, E re-

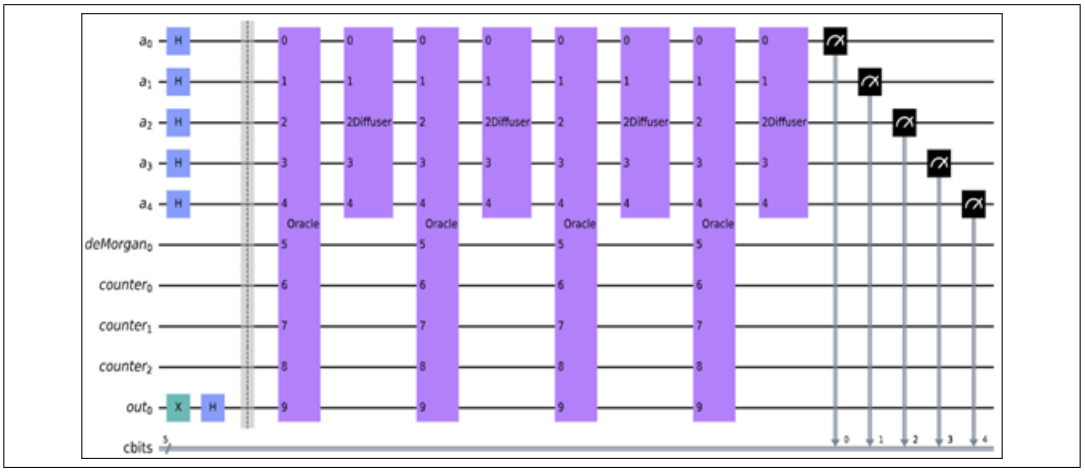


Figure 5: Grover's algorithm with 4 iterations using the oracle circuit from Figure 4

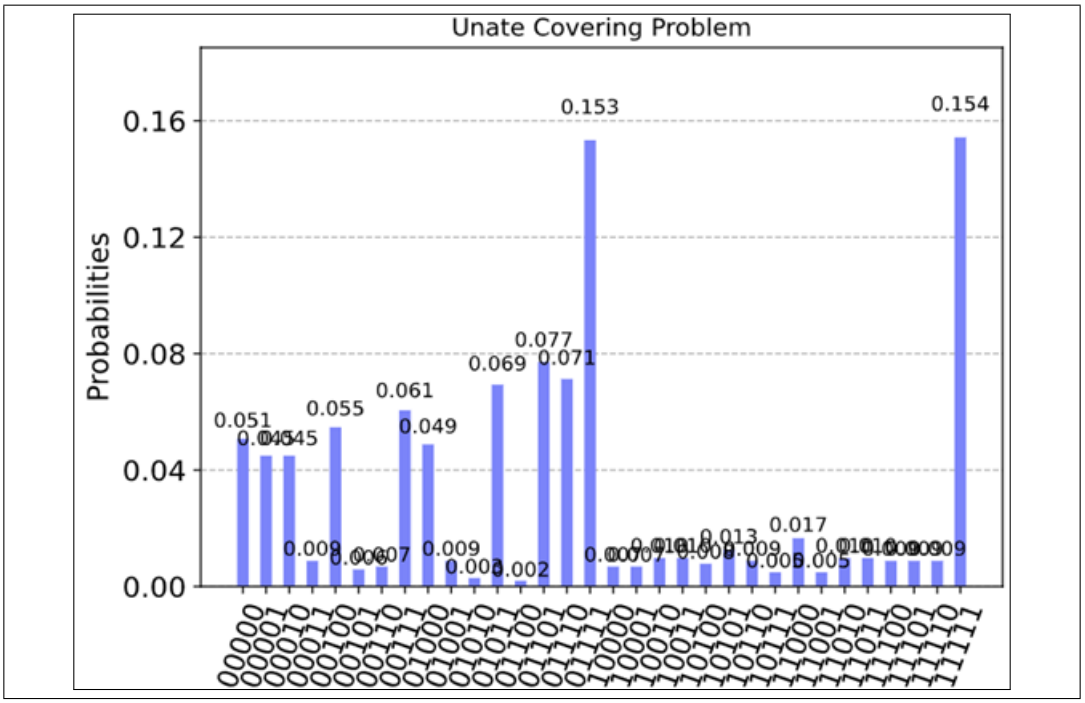


Figure 6: Measurement of the Boolean variables from $A \cdot B \cdot C \cdot D(A + E)(B + E)(D + E)(C + E)$

spectively. As can be seen in Figure 6, the diagram illustrates the QISKIT [5] output graphics for the simulated circuit. The measured values $a_4a_3a_2a_1a_0$ with high probability are 11111, 01111. The values 11111 and 01111 correspond to E, D, C, B, A respectively which are two solutions to equation (1): A, B, C, D and A, B, C, D, E.

This example explains that the above method can be applied to an arbitrary POS formula with x variables and y clauses. Therefore, our method is scalable to an arbitrary size of unate covering problem, assuming a sufficiently large number of qubits in a quantum computer or in a quantum component of a hybrid computer.

5 Grover's Algorithm for Binate Covering Problem

Binate covering problem is the same as unate covering problem with the additional constraint that BCP can contain negative literals. There are some cases that have no solution, and therefore, the BCP should reliably notify the user about this fact. We presented three examples of BCP problems: (1) Finding the minimum covering for an implication graph. (2) Finding a minimum cost constraint. (3) Minimization of incompletely specified Finite State Machines.

5.1 Finding the Minimum Covering for an Implication Graph

An implication graph is a directed acyclic graph (DAG) where each node represents a variable assignment. An implication graph represents the implication relations between pairs of variable assignments. Given a set S and a family of subsets $F = \{s_1, s_2, \cdots, s_n\}$, a closure conditions are represented as an implication graph. For instance, assuming given a family of subsets of the set $\{1, 2, 3, 4, 5, 6\}$ and the implication graph from in Figure 7, the optimization task is to select a subset of nodes from the set A, B, C, D, E that satisfies all three conditions:

- Covering condition: all items from the set $\{1, 2, 3, 4, 5, 6\}$ must be covered by the selected nodes.
- All closure conditions must be satisfied.
- The set of selected nodes must have the minimum number of elements.

The general optimization can be solved by constructing a covering and closure table. For a particular problem, as specified above, the table is shown in Figure 8. Here, the rows correspond to the nodes (subsets) of the implication graph in Figure 7, while the columns correspond to the individual elements of the set $\{1, 2, 3, 4, 5, 6\}$ for covering table and the columns for the closure table correspond to the nodes in the implication graph which are the same subsets as the rows of the table.

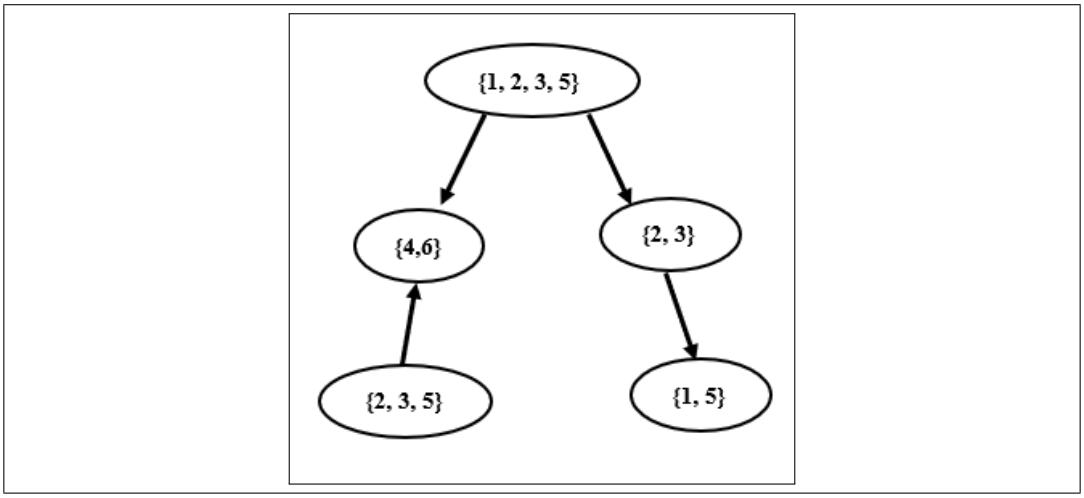


Figure 7: Implication graph for the set $\{1, 2, 3, 4, 5, 6\}$

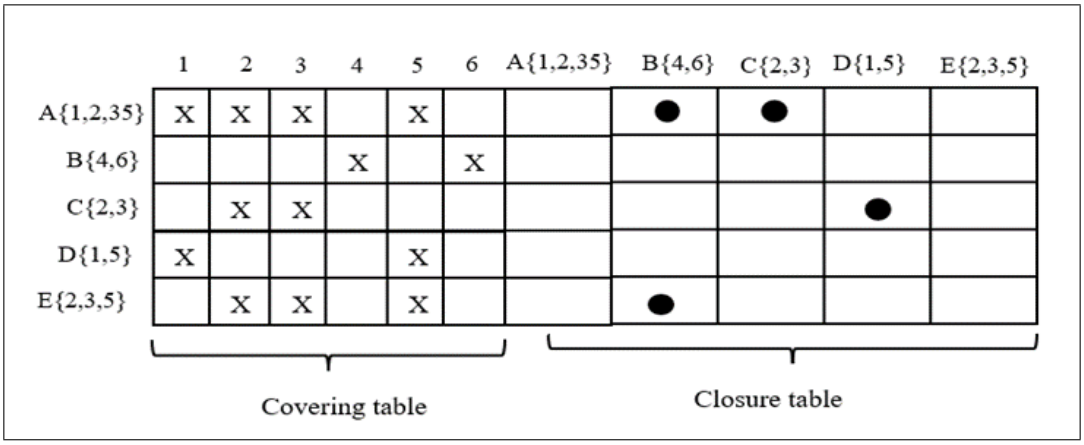


Figure 8: Covering-closure table based on the implication graph from Figure 7

The closure conditions are illustrated in Figure 7. Assuming that we select set $A = \{1, 2, 3, 5\}$ on top of the implication graph, it is implied that the set $B = \{4, 6\}$ and the set $C = \{2, 3\}$ must be also selected. Set $\{2, 3\}$ implies set $\{1, 5\}$ and set $\{2, 3, 5\}$ implies set $\{4, 6\}$ (see Figure 7 above). This way, the table from Figure 8 is created. For instance, the black dots in the row A mean that set $A = \{1, 2, 3, 5\}$ implies sets $B = \{4, 6\}$ and $C = \{2, 3\}$. Set $C = \{2, 3\}$ implies sets $D = \{1, 5\}$. Set $E = \{2, 3, 5\}$ implies sets $B = \{4, 6\}$. Based on the covering and closure table from Figure 8, Petrick's method creates a Boolean formula in equation (2). Next, this

formula is transformed into the general POS formula from equation (3)

$$(A+D)(A+C+E)B(A+D+E)(A \Rightarrow BC)(C \Rightarrow D)(E \Rightarrow B) = 1 \qquad (2)$$

In equation (2), the first four terms describe the covering conditions, and the last three terms correspond to closure conditions (closure constraints). Equation (2) can be converted to equation (3) by using the logic transformation rule $(A \Rightarrow B) \Leftrightarrow (\overline{A} + B)$

$$(A+D)(A+C+E)B(A+D+E)(\overline{A}+BC)(\overline{C}+D)(\overline{E}+B) \qquad (3)$$

Next , we build a quantum oracle for the general POS formula from equation (3) by converting each OR term into Products using De Morgan's law $A + D = \overline{\overline{A+D}} = \overline{\overline{A}\overline{D}} = 1 \oplus \overline{A}\overline{D}$; $A + C + E = \overline{\overline{A+C+E}} = \overline{\overline{A}\overline{C}\overline{E}} = 1 \oplus \overline{A}\overline{C}\overline{E}$; $A + D + E = \overline{\overline{A+D+E}} = \overline{\overline{A}\overline{D}\overline{E}} = 1 \oplus \overline{A}\overline{D}\overline{E}$; $\overline{A} + BC = \overline{\overline{\overline{A}+BC}} = \overline{A\overline{BC}} = 1 \oplus A\overline{BC} = 1 \oplus A(1 \oplus BC)$; $\overline{C} + D = \overline{\overline{\overline{C}+D}} = \overline{C\overline{D}} = 1 \oplus C\overline{D}$; $\overline{E} + B = \overline{\overline{\overline{E}+B}} = \overline{E\overline{B}} = 1 \oplus E\overline{B}$. Then we connect one block of the iterative quantum counter after each Toffoli gate, representing the OR term of the function POS formula. We put the ancilla qubit back to its original state by mirroring each Toffoli gate after each counter. In this case, we need seven quantum counter circuits (one for B term and six for each POS term), as can be seen in Figure 9. The oracle circuit in Figure 9 is inserted into Grover's Oracle. This is similar to previous examples that demonstrate one more time how inefficient using the global AND is, even for very small practical binate covering problems. Rather than using AND, we use a quantum counter.

This example illustrates that an arbitrary size problem of solving an implication graph can be reduced to algorithmically creating Grover's oracle with x variable qubits and y terms. Therefore, the implication graph problem can be solved by Grover's algorithm with a quadratic speedup.

In Figure 10, we applied the oracle circuit from Figure 9 in Grover's search algorithm for iterations $R = 2$ from this formula: $R \leq \left\lceil \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rceil$ where $N = 2^5 = 32$ and $M = 5$ (this can be verified by a truth table). We run the circuit on the 'qasm_simulator' from QISKIT for 1024 shots, and the circuit produces the correct answers. We measured a_0, a_1, a_2, a_3 and a_4 in Figure 10 where a_0, a_1, a_2, a_3, a_4 corresponds to the Boolean variables A, B, C, D, E respectively. As can be seen in the histogram in Figure 11, this illustrates the QISKIT [5] output graphics for the simulated circuit. The measured values $a_4a_3a_2a_1a_0$ with high probability are 01110, 01111, 11010, 11110, and 11111 corresponding to variables E, D, C, B, A respectively. For instance, the vector 01110 corresponds to the solution of DCB.

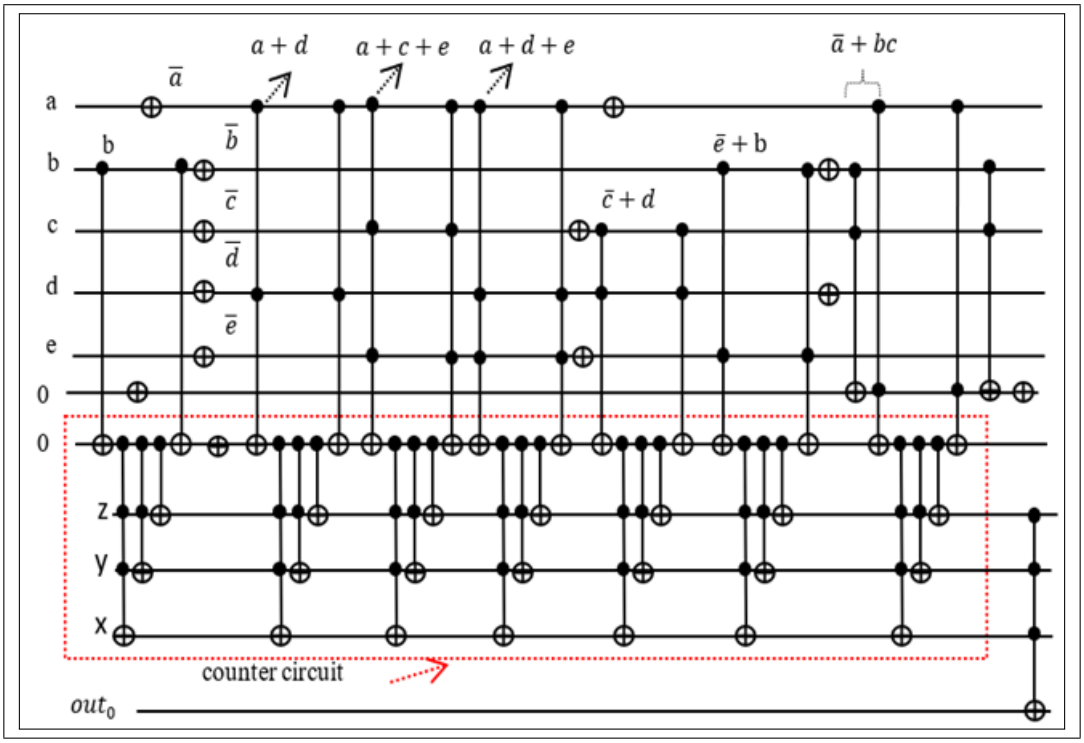


Figure 9: Quantum oracle for $(A + D)(A + C + E)B(A + D + E)(\overline{A} + BC)(\overline{C} + D)(\overline{E} + B)$

In the measurement, the solutions have much higher probabilities than the non-solutions. These solutions are verified outside of Grover's algorithm, just by using the oracle with function $(A+D)(A+C+E)B(A+D+E)(\overline{A}+BC)(\overline{C}+D)(\overline{E}+B)$.

5.2 Finding Minimum Cost Constraint

Minimum cost constraint minimizes the cost of satisfying the assignment for the given problem. Given m constraints on n Boolean variables, the goal is to find an assignment that satisfies all constraints such that:

$$minimizing \sum_{i=0}^{n-1} w_i x_i$$

subject to $f = y_0 \wedge y_1 \wedge \cdots \wedge y_{m-1} = 1$.

Where $w_i \geq 0$ is the weight of variable x_i and y_m is a clause which means a sum of x_i. This is called a weighted binate covering. The practical application of finding

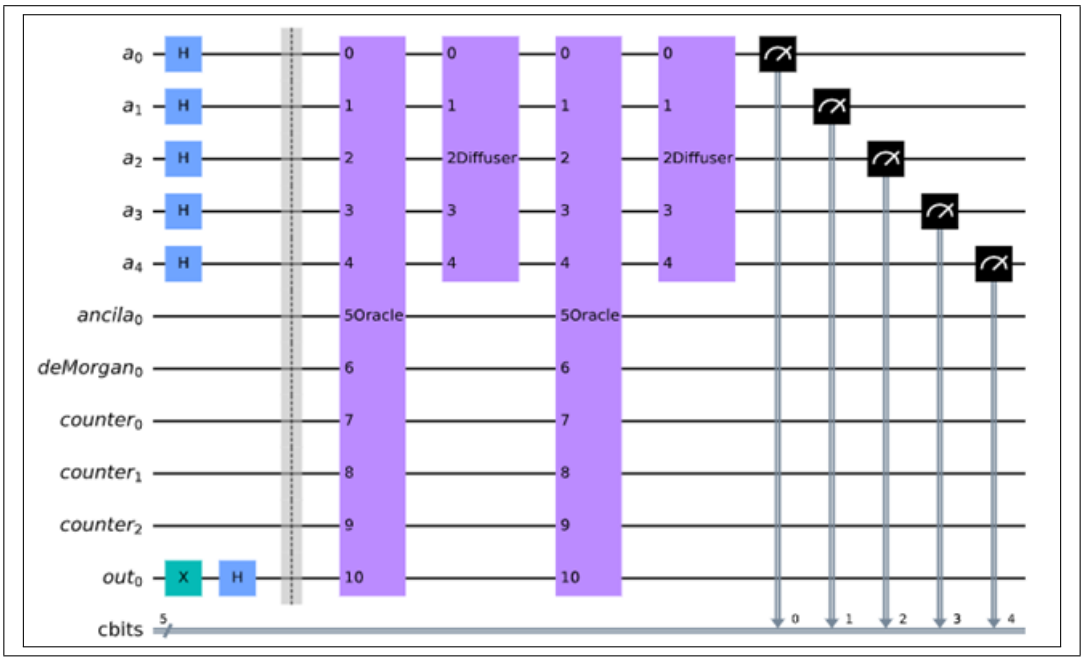


Figure 10: Grover's algorithm with 2 iterations using the oracle circuit

the minimum cost constraint can be found in [21], which can be formulated as a binate covering problem. However, we present in this paper a general example of a Boolean function that can be solved using Grover's algorithm. For instance, given is the following set of clauses Y:

$$y_0 = x_0 + x_3$$

$$y_1 = x_2 + \overline{x_3}$$

$$y_2 = x_1 + \overline{x_2}$$

$$y_3 = x_1 + x_2 + x_3$$

and let the weight w_i for each value of x_i as follow: $w_0 = 4, w_1 = 2, w_2 = 1$, and $w_3 = 1$. The optimization task is to find the minimum cost assignment based on the given weight. First, we construct the POS function, and then we design a quantum oracle for the Grover's algorithm to find the exact minimum solution. Based on the solution, we apply the weight for each clause to find the minimum cost constraint. Here the Y clauses are represented by a POS Boolean formula:

$$(x_0 + x_3)(x_2 + \overline{x_3})(x_1 + \overline{x_2})(x_1 + x_2 + x_3) \tag{4}$$

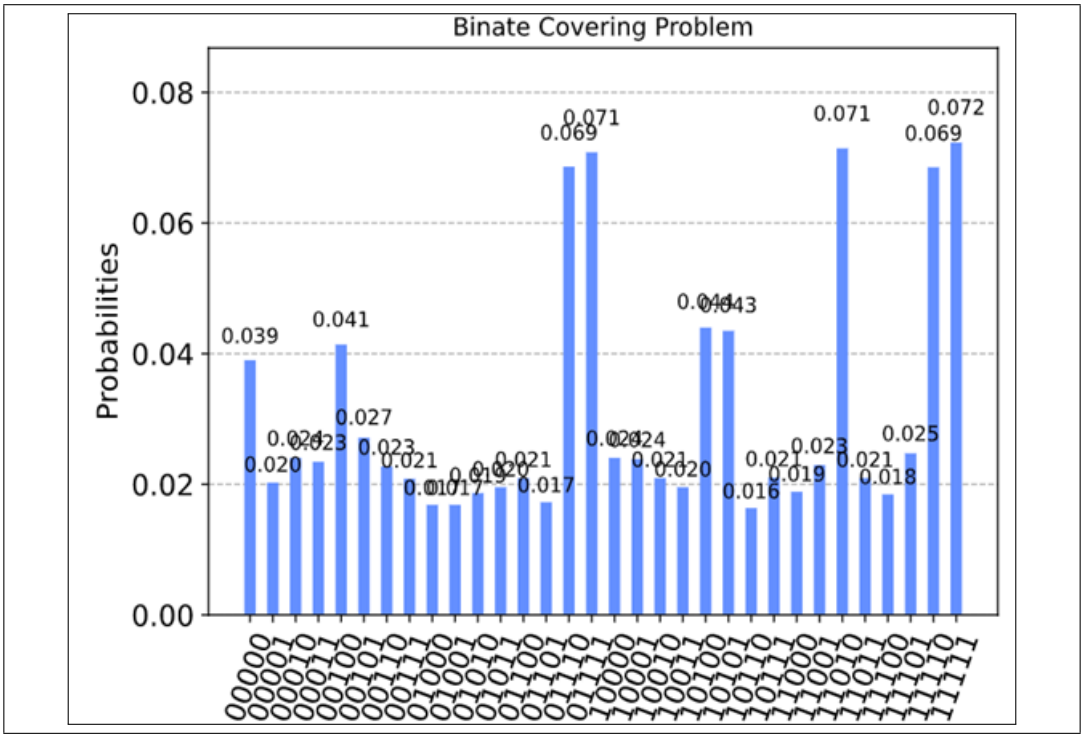


Figure 11: Measurement of the Boolean variables from Figure 10 based on $ABCDE$

The solution found in Grover's algorithm is subject to the equation (5) for the minimum cost function:

$$4x_0 + 2x_1 + x_2 + x_3 \tag{5}$$

Equation (5) is an arithmetic function that is computed in a classical processor of a hybrid computer, while equation (4) is computed in Grover's algorithm. First, we build a quantum oracle from equation (4), similar to the previous examples, after applying De Morgan's low: $x_0 + x_3 = \overline{x_0}\overline{x_3} \oplus 1$; $x_2 + \overline{x_3} = \overline{x_2}x_3 \oplus 1$; $x_1 + \overline{x_2} = \overline{x_1}x_2 \oplus 1$; $x_1 + x_2 + x_3 = \overline{x_1}\overline{x_2}\overline{x_3} \oplus 1$

We need to add the two NOT gates in the output circuit block, which makes the last qubit out_0 to produce one if the variable xyz in counter circuit is 100, such that the Boolean function in equation (4) is equal to 1.

In Figure 13, we applied the oracle circuit in Figure 12 in Grover's search algorithm for iterations R=2 from this formula: $R \leq \left\lceil \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rceil$ where $N = 2^4 = 16$ is the number of all search space elements. $M = 4$ is the number of solutions that can be verified by creating a truth table. We run the circuit on the 'qasm_simulator' from

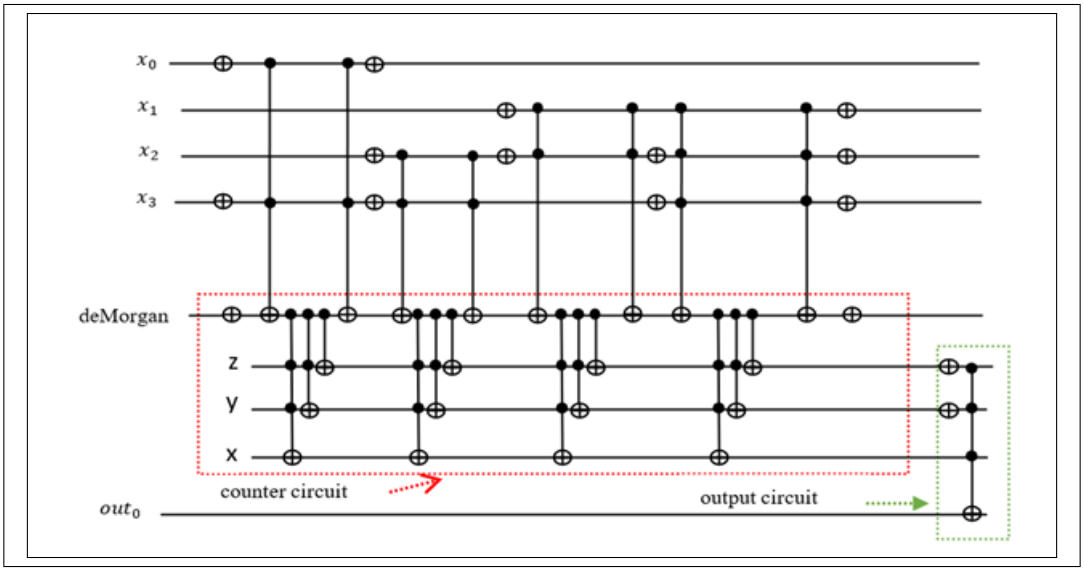


Figure 12: Quantum oracle for $(x_0 + x_3)(x_2 + \overline{x_3})(x_1 + \overline{x_2})(x_1 + x_2 + x_3)$

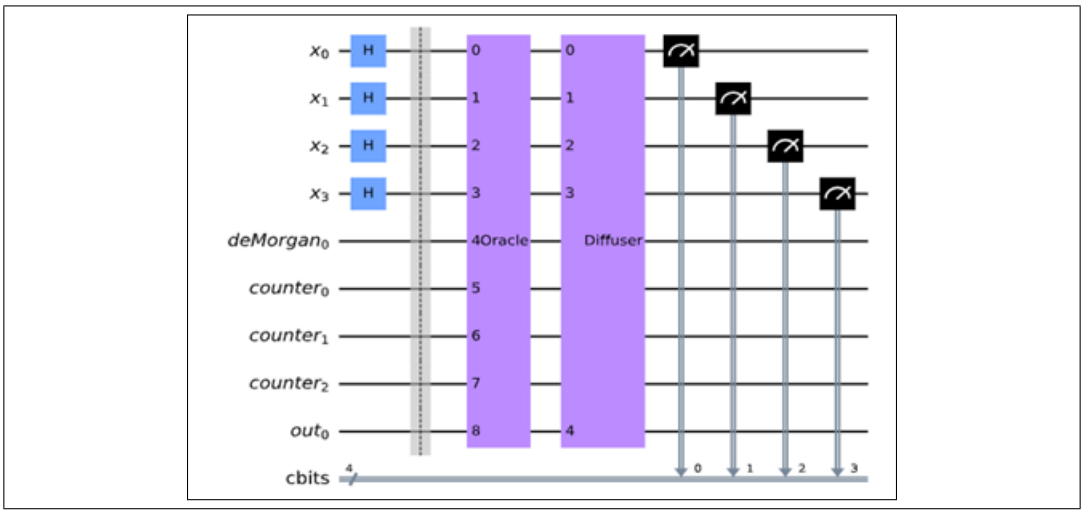


Figure 13: Grover's algorithm with 2 iterations using the oracle circuit from Figure 12

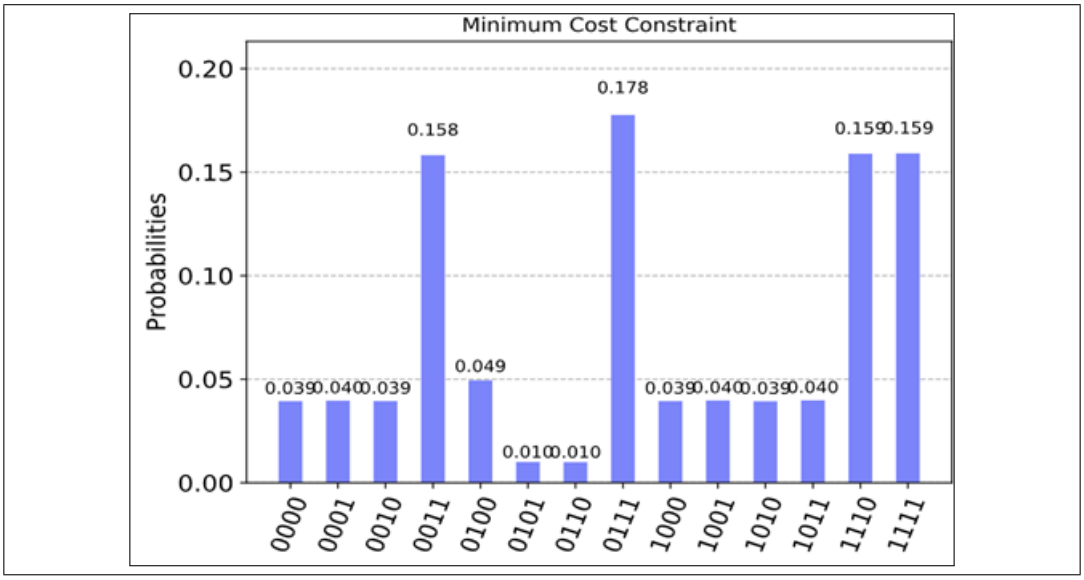


Figure 14: Measurement of the Boolean variables from Figure 13 based on $x_0x_1x_2x_3$

QISKIT for 1024 shots, and the circuit produces the correct answers. We measured $x_3x_2x_1x_0$ in Figure 13. As can be seen in Figure 14, the measured values $x_3x_2x_1x_0$ with high probability are {1111,1110,0111,0011}. Applying these values to equation (4), we found 1110, which gives the minimum cost of 4.

$$1110 : 4x_0 + 2x_1 + x_2 + x_3 = 4 * 0 + 2 * 1 + 1 * 1 + 1 * 1 = 4$$

Based on this value, we choose 1110 from the answer {1111,1110,0111,0011}, which corresponds to the solution of $x_3x_2x_1$ respectively with the minimum cost of 4. In another variant of our method, the arithmetic calculation is built into a quantum counter inside the oracle, such that for clause i instead of value 1, the value of w_i is added. This general method, however, is not practical for current quantum simulators. Concluding, the presented method is scalable to arbitrary size problem of minimizing the minimum cost constraint can be reduced to algorithmically creating Grover's oracle with x variable qubits and y terms. Therefore, finding the minimum cost constraint can be solved by Grover's algorithm with a quadratic speedup.

5.3 Minimization of Incompletely Specified Finite State Machines

A Finite State Machine (FSM) is an abstract model used in design to model problems in various fields of science and engineering. A finite state machine consists

of input states, output states, and internal states that can change from one state to another state based on input. The change of the internal state is described by a transition function. For various reasons, finite state machines can have incompletely specified output functions and transition functions. The minimization of incompletely specified finite state machines is considered an NP-hard problem [17]. We present the complete oracle design for Grover's algorithm for the well-known classical problem of minimization of the number of states of incompletely specified finite state machines. For a given incompletely specified finite state machine, the solution is achieved by the following steps:

1. Classical computers create a triangular table to obtain compatible states.
2. Based on the triangular table, the classical computer creates a compatibility graph.
3. Quantum computer finds all maximum cliques in the compatibility graph.
4. The classical computers create the covering table component of the covering-closure based on the maximum cliques.
5. The classical computer creates a closure table component of the covering-closure table only for compatible states.
6. The classical computer creates a Boolean function for the oracle from the covering-closure table.
7. A quantum oracle is designed by a classical computer.
8. Grover's algorithm is called on a quantum computer with the oracle found in point 7.

Below we will illustrate the above general hybrid algorithm on a particular example. Given is an incompletely specified Mealy finite state machine described as a transition/output table from Figure 15a. Dashes represent don't care in internal states or output states. This table has internal states A, B, C, D, E, F, two input signals for columns, and one binary output under a slash symbol in cells of the map. A triangular table in Figure 15b was generated based on the table from Figure 15a. The table from Figure 15b covers all possible cases to minimize the number of states in the finite state machine. The 'X' symbol in the table indicates no possibility for grouping the corresponding states. Symbol 'V' in the table indicates that the states can be combined without any problem. A pair of state variables in a cell of the triangular map V indicates that states can be grouped only if the states mentioned

in the block can be combined without any problem. For instance, states B and F can be combined under the condition that states C, F are compatible (can be combined). The method of creating the triangular table is well-known from [20].

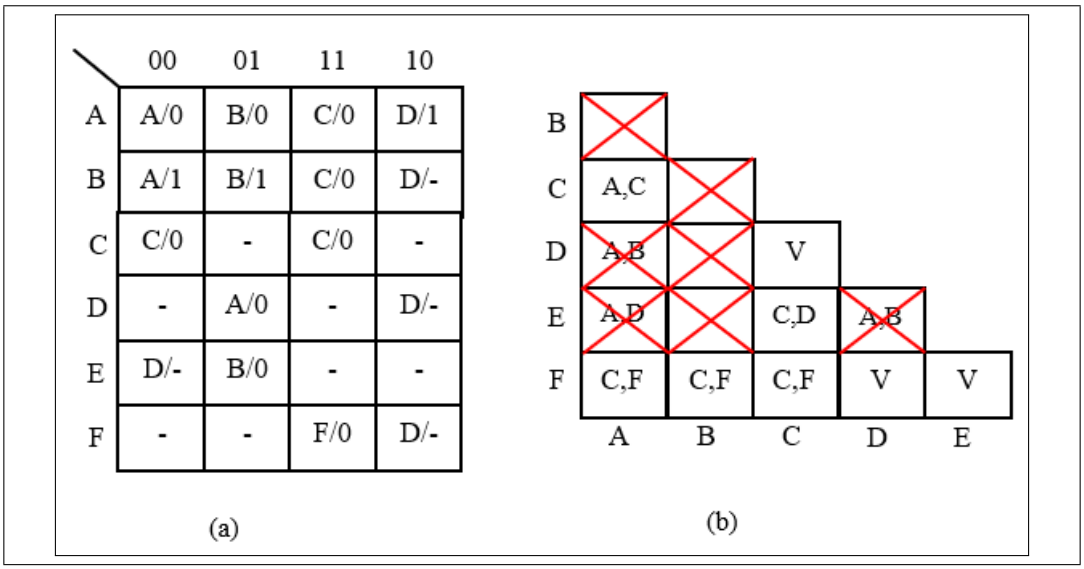


Figure 15: (a) Truth table of FSM (b) FSM triangular table generated based on the truth table from (a)

Based on the triangular table, a compatibility graph for the state machine is generated, as shown in Figure 16a. Every cell in the triangular map that has no symbol X corresponds to a compatible pair of states. For instance, the cell at the intersection of row C and column A is compatible. The cell at the intersection of row B and column A is not compatible as it has a symbol X in it. Similarly, states A and D are not compatible. In Figure 15b, states A and D are compatible under the condition that states A and B are compatible. But we found earlier that states A and B are not compatible; thus, states A and D are not compatible. The cell at the intersection of row D and column A is crossed-out as symbol X. In the same way, the cell at the intersection of column A and row E is replaced with X because states A and D are not compatible. A simple recursive classical algorithm creates symbols X for every incompatible pair of internal states.

From Figure 16a, the maximum cliques of the graph are identified as $\{A, C, F\}$, $\{B, F\}$, $\{C, D, F\}$, $\{C, E, F\}$. Finding of all cliques in a graph is done by a SAT-based algorithm similar to those discussed earlier in this paper. The compatibility graph from Figure 16a is a complement of the incompatibility graph from Figure

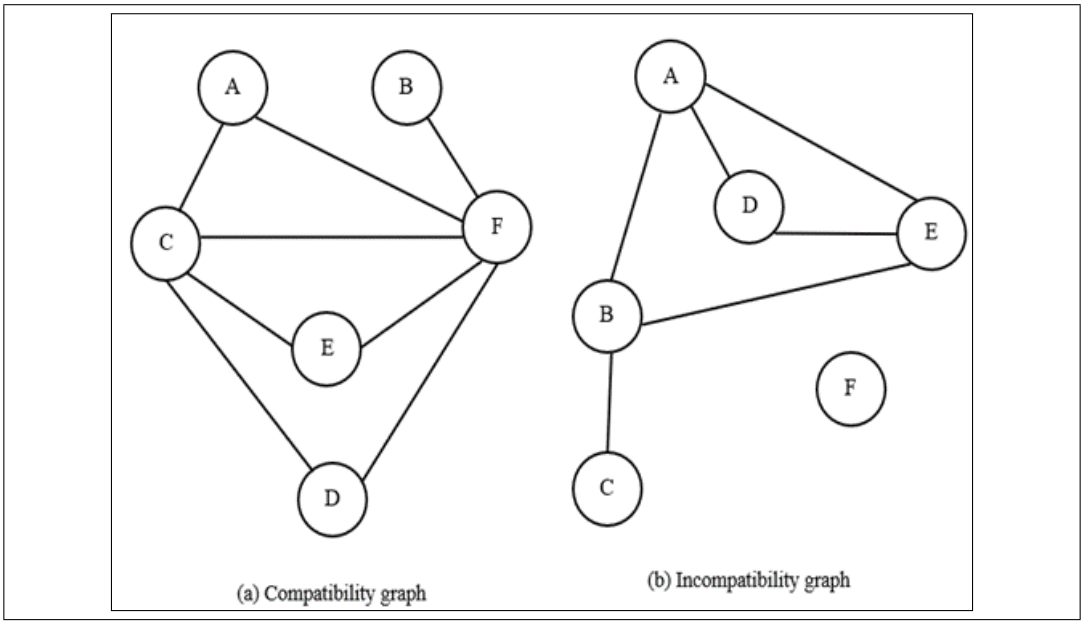


Figure 16: (a) Compatibility graph (b) Incompatibility graph.

16b. As we see, the maximum cliques in the compatibility graph are the same as the maximum independent sets in the incompatibility graph from Figure 16b. These maximum independent sets can be found from the graph coloring [14] of the incompatibility graph. The graph coloring can be solved by a special Grover's oracle. It can be solved by finding the Maximum Independent Sets and then solving the unate covering problem with them. These problems are reducible to SAT-like oracles for Grover's algorithm. This example explains the relation between the graph coloring of the incompatibility graph, finding the maximum cliques of the compatibility graph, and covering problems. These partial quantum algorithms are also useful in a quantum algorithm for solving the Ashenhurst-Curtis Decomposition [23].

To minimize the finite state machine, a covering-closure table, shown in Figure 17, is created by considering the maximum cliques and all their subsets as rows of the table. All of the states $\{A, B, C, D, E, F\}$ in the machine correspond to columns of the covering table, and the implications $\{A, C\}, \{C, F\}, \{C, D\}$ in the compatibility graph from the cell of the triangular table correspond to columns of the closure table. From the table in Figure 17, a binate covering problem can be specified using the equation:

$$(X + V + P)Y(X + Z + U + V + Q + R + T)(Z + R + S)(U + T + W)(X + Y +$$

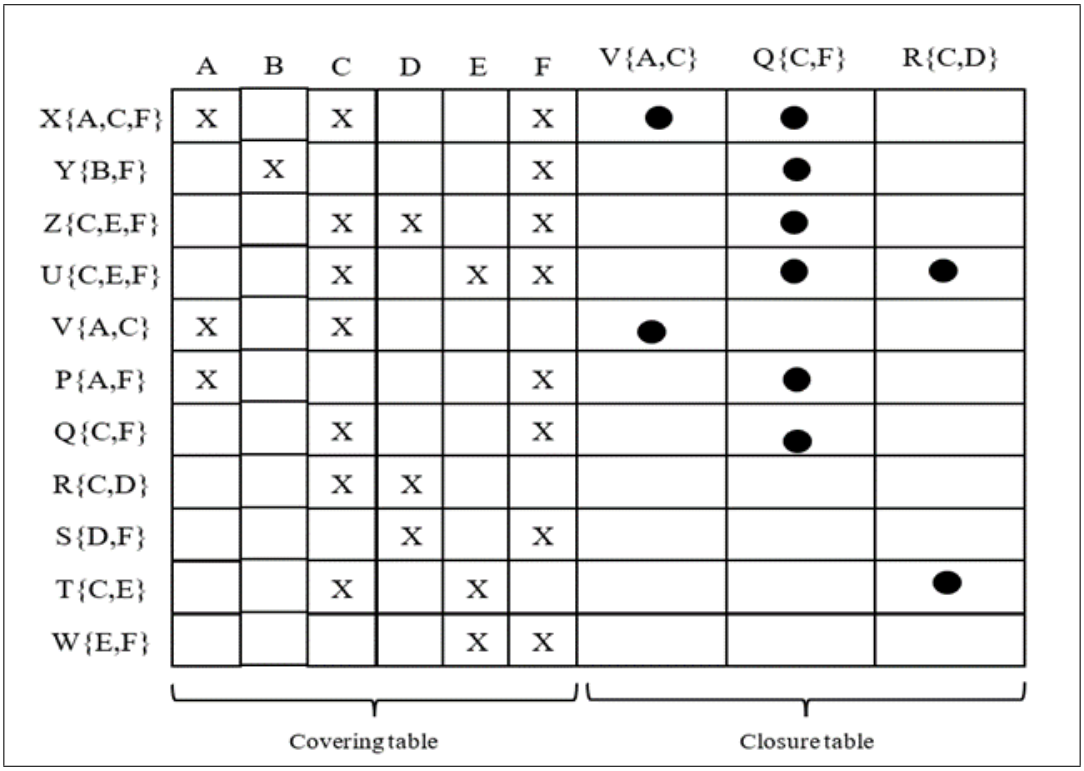


Figure 17: Covering-Closure table for the FSM

$Z + U + P + Q + S + W)(X \Rightarrow VQ)$
$(Y \Rightarrow Q)(Z \Rightarrow Q)(U \Rightarrow QR)(V \Rightarrow V)(P \Rightarrow Q)(Q \Rightarrow Q)(T \Rightarrow R) = 1$

The function can be simplified using the Boolean laws $A \Rightarrow B \Leftrightarrow (\overline{A} + B)$.

$$\begin{cases} (X+V+P)Y(X+Z+U+V+Q+R+T)(Z+R+S)(U+T+ \\ W)(X+Y+Z+U+P+Q+S+W)(\overline{X}+VQ)(\overline{Y}+Q)(\overline{Z}+Q)(\overline{U}+ \\ QR)(\overline{V}+V)(\overline{P}+Q)(\overline{Q}+Q)(\overline{T}+R) = 1 \end{cases} \tag{6}$$

The number of search space $N = 2^{11} = 2048$ where 11 is the number of variables in the rows of covering-closure table. There are 155 solutions that the Boolean equation (6) is equal to 1. The presented method is not yet practical as contemporary quantum computers have not enough qubits. However, with a sufficient number of qubits, the presented algorithm will allow to minimize large machines with quadratic speedup. To visualize all these solutions in a histogram is difficult such that we use a more general case in Figure 18, we repeat the Grover's algorithm for iterations $R = 3$ with tuning values of thresholds until equal to counter value. The comparator $G = H$

compares the output from the counter with the threshold value given as constant values n_1, n_2, n_3 and n_4. (Using the threshold with a comparator has many other applications such as finding the minimum set of support [31]).

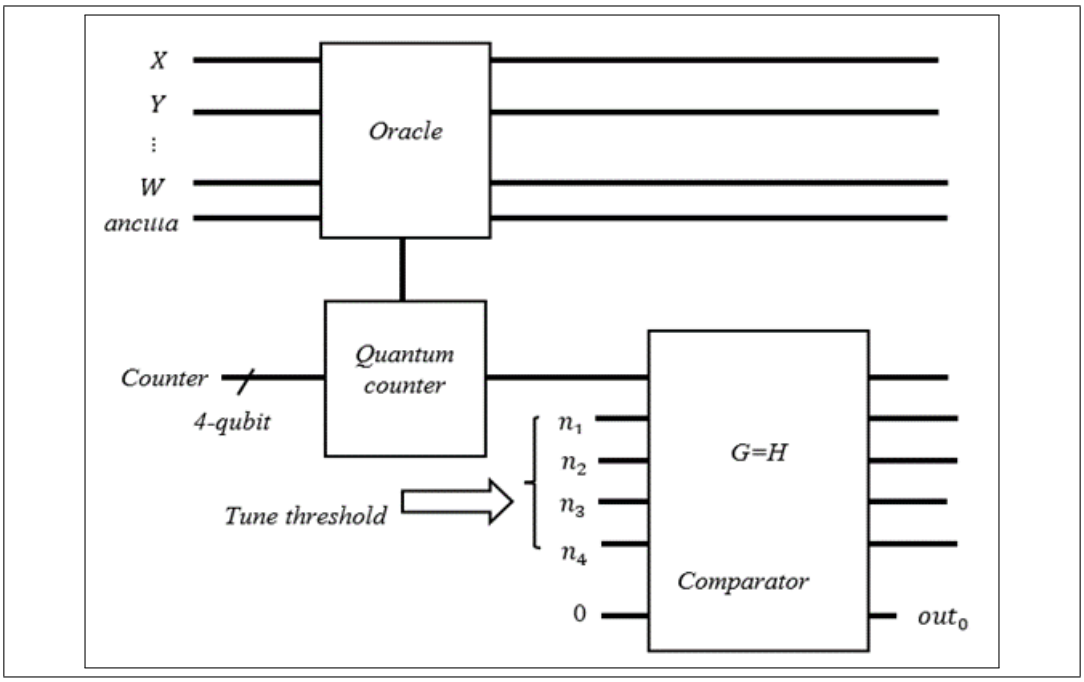


Figure 18: FSM Oracle Design with counter circuit and threshold with comparator

The solution with the minimum number of positive literals is $\{V, Q, R, Y, U\}$ which simplifies to $\{V, R, Y, U\}$ because group $Q = \{C, F\}$ is included in $U = \{C, E, F\}$, thus $Q \Rightarrow U$. Symbol V requires combining states A and C, symbol Y requires combining states B and F, symbol R combines states C and D and symbol U combines states C, E, F together. As the result, we obtain the minimized state machine from Figure 19a. We combine states from the respective states of Figure 15. Thus, combining in column 00 for row V we obtain symbols A and C and output 0. This way, combining states from groups V, Y, R and U the entire table from Figure 19a is created. Now for every subset of initial states A, B, C, D, E, F corresponding to each symbol from set of sets $\{V, Y, R, U\}$ we check to which set this subset belongs. For instance state C is included in sets V, R and U. Therefore symbol C in the table from Figure 19a is replaced with symbols V, R and U in the transition cells. This way, the non-deterministic machine from Figure 19b is created. Now select any state among V, R and U to create one of the deterministic machines

described by the non-deterministic machine. Choose every row U in column 11 in order to improve the logic realization of the machine. Similarly, for the purpose of good encoding (encoding not explained here), select state R in column 00 to have two transitions to V and two transitions to R in this column. One final finite state machine minimized in this way is shown in Figure 19c.

a)

	00	01	11	10
$V(A,C)$	$A,C/0$	$B/0$	$C/0$	$D/1$
$Y(B,F)$	$A/1$	$B/1$	$C,F/0$	$D/-$
$R(C,D)$	$C/0$	$A/0$	$C/0$	$D/-$
$U(C,E,F)$	$C,D/0$	$B/0$	$C,F/0$	$D/-$

b)

	00	01	11	10
V	$V/0$	$Y/0$	$V,R,U/0$	$R/1$
Y	$V/1$	$Y/1$	$U/0$	$R/-$
R	$V,R,U/0$	$V/0$	$V,R,U/0$	$R/-$
U	$R/0$	$Y/0$	$U/0$	$R/-$

c)

	00	01	11	10
V	$V/0$	$Y/0$	$U/0$	$R/1$
Y	$V/1$	$Y/1$	$U/0$	$R/-$
R	$R/0$	$V/0$	$U/0$	$R/-$
U	$R/0$	$Y/0$	$U/0$	$R/-$

Figure 19: Steps to create an exactly minimized deterministic FSM using binate covering problem. (a) the table created directly from the solution to the covering-closure problem; (b) a non-deterministic automaton created from the table in (a); (c) one deterministic automaton created from the non-deterministic automaton in (b)

There are not yet benchmarks for quantum algorithms. However, there exist benchmarks for classical algorithms, such as those in [18, 22, 1, 9]. The current quantum computers are too small to run the classical benchmarks on them. One can, however, speculate on the speedup of future quantum and hybrid computers based on these classical benchmarks. Suppose a benchmark takes m terms and n variables, using our method, this benchmark would require n qubits for variables and $\lceil \log_2 m \rceil$ ancilla qubits for terms to represent the problem in a quantum algorithm design. In contrast, the traditional quantum oracle design would require n qubits for variables and $m+1$ ancilla qubits for terms [2]. Thus, when compared to the traditional Grover, our proposed design requires fewer qubits with a quadratic

speedup of Grover's algorithm. As can be seen from Figure 20, for instance, if a given covering problem consists of 100,000 clauses, then our quantum oracle design requires only 18 ancilla qubits, while the traditional quantum oracle would require 100,000 ancilla qubits, which is the same as the clause number in the given problem. Assuming a complete search, the complexity of the classical algorithm would be $N = 2^n$. The complexity of our quantum algorithm would be $O(\sqrt{N})$. When the quantum computers have enough qubits, comparing practical benchmarks will be possible. Because IBM aims to build a quantum computer with 100,000 qubits in 10 years [10], we hope that in this time frame, our quantum algorithm for EDA problems will become practical.

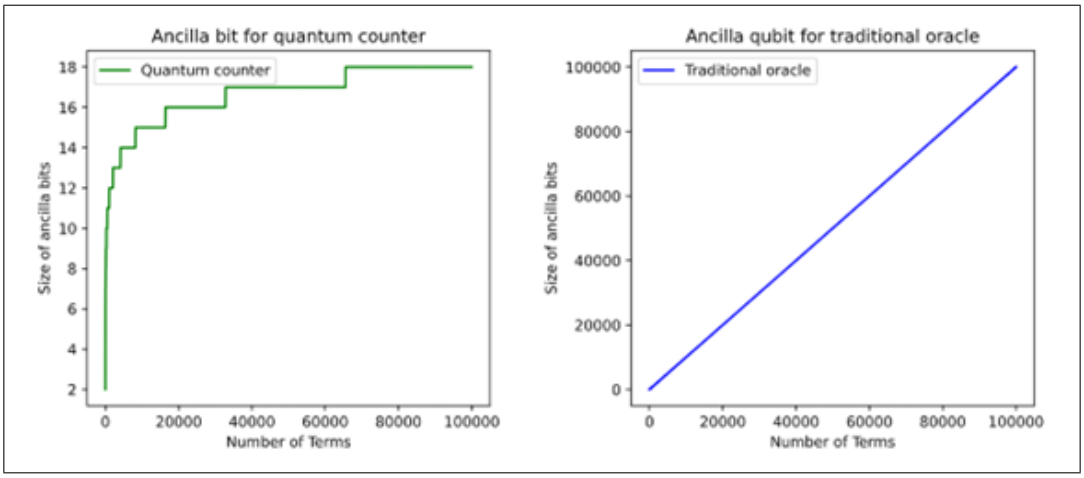


Figure 20: Comparison of the required numbers of ancilla qubits for our quantum oracle design (left) and the traditional quantum oracle design (right) [2]

6 Conclusion

We presented new quantum algorithms to solve with quadratic speedup several fundamental problems of classical logic circuits and finite state machine design. All these algorithms either use Grover, repeated Grover, or hybrid algorithms that use Grover's algorithm. Optimization problems are reduced to a repetition of constraint satisfaction problems solved by Grover' algorithm. Our approach is based on creating various oracles, which are, however, based on the same basic principle. We presented quantum oracle designs for various well-known EDA applications of the Unate and Binate covering problems. Innovative quantum algorithms for exact an incompletely specified FSM minimization have been presented here for the first

time. Each of these problems is converted to a general quantum oracle. Our quantum oracle design uses an iterative quantum counter block used inside the oracles for the Grover-like algorithms. The concept of introducing this quantum counter can be applied to all these algorithms, allowing them to solve in a uniform way both SAT-like and MAX-SAT-like problems. Most importantly, this approach reduces the number of qubits logarithmically [3]. The introduction of the iterative quantum counter circuit replaces the ancilla qubits of the global large AND gate for traditional quantum oracle design for SAT-like problems. This is important because it reduces not only the number of required qubits but also avoids designing a quantum AND gate with many inputs, which is known to be a very complicated gate. We presented experimental results from the QISKIT simulator that showed the circuit works correctly.

Our future research will investigate using quantum counting based on combining Grover's algorithm with quantum phase estimation [29] to more specifically estimate the number of Grover's Loop repetitions for larger problems, which are difficult to calculate manually. Further improvement is to extend the quantum oracle design to add an arithmetic circuit for the weighted covering problems.

References

[1] The mcnc benchmark problems for vlsi floorplanning. `http://www.mcnc.org`. [Online].

[2] Abdirahman Alasow, Peter Jin, and Marek Perkowski. Quantum algorithm for variant maximum satisfiability. *Entropy*, 24(11), 2022.

[3] Abdirahman Alasow and Marek Perkowski. Quantum algorithm for maximum satisfiability. In *2022 IEEE 52nd International Symposium on Multiple-Valued Logic (ISMVL)*, pages 27–34. IEEE, 2022.

[4] Abdirahman Alasow and Marek Perkowski. Quantum algorithm for mining frequent patterns for association rule mining. *Journal of Quantum Information Science*, 13(1):1–23, 2023.

[5] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, F Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, et al. Qiskit: An open-source framework for quantum computing. *Accessed on: Mar*, 16, 2019.

[6] Carolina Allende, Efrain Buksman, and André Fonseca De Oliveira. Quantum circuit design using neural networks assisted by entanglement. In *2021 IEEE URUCON*, pages 316–319. IEEE, 2021.

[7] Julien Bramel and David Simchi-Levi. On the effectiveness of set covering formulations for the vehicle routing problem with time windows. *Operations Research*, 45(2):295–301, 1997.

[8] Melvin A Breuer. *Design Automation of Digital Systems: Vol. 1.: Theory and Techniques*. Prentice-Hall, 1972.

[9] Franc Brglez. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proc. Intl. Symp. Circuits and Systems, 1985*, 1985.

[10] Michael Brooks. Ibm wants to build a 100,000-qubit quantum computer. `https://www.technologyreview.com/2023/05/25/1073606/ibm-wants-to-build-a-100000-qubit-quantum-computer`. Accessed: 2023-06-15.

[11] Hongxiang Fan, Ce Guo, and Wayne Luk. Optimizing quantum circuit placement via machine learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 19–24, 2022.

[12] Gary D Hachtel and Fabio Somenzi. *Logic synthesis and verification algorithms*. Springer Science & Business Media, 2005.

[13] Chi-Chuan Hwang, Chu-Yuan Tseng, and Cheng-Fang Su. Quantum circuit design for computer-assisted shor's algorithm. 2022.

[14] Tommy R Jensen and Bjarne Toft. *Graph coloring problems*. John Wiley & Sons, 2011.

[15] Jeong and Somenzi. A new algorithm for the binate covering problem and its application to the minimization of boolean relations. In *1992 IEEE/ACM International Conference on Computer-Aided Design*, pages 417–420. IEEE, 1992.

[16] Ankit Kagliwal and Shankar Balachandran. Set-cover heuristics for two-level logic minimization. In *2012 25th International Conference on VLSI Design*, pages 197–202. IEEE, 2012.

[17] Timothy Kam, Tiziano Villa, Robert Brayton, and Alberto Sangiovanni-Vincentelli. A fully implicit algorithm for exact state minimization. In *Proceedings of the 31st annual Design Automation Conference*, pages 684–690, 1994.

[18] Timothy Kam, Tiziano Villa, Robert K Brayton, and Alberto L Sangiovanni-Vincentelli. *Synthesis of finite state machines: functional optimization*. Springer Science & Business Media, 2013.

[19] Sumeet Khatri, Ryan LaRose, Alexander Poremba, Lukasz Cincio, Andrew T Sornborger, and Patrick J Coles. Quantum-assisted quantum compiling. *Quantum*, 3:140, 2019.

[20] Zvi Kohavi and Niraj K Jha. *Switching and finite automata theory*. Cambridge University Press, 2009.

[21] Xiao Yu Li. *Optimization algorithms for the minimum-cost satisfiability problem*. North Carolina State University, 2004.

[22] Xiao Yu Li, Matthias F Stallmann, and Franc Brglez. Effective bounding techniques for solving unate and binate covering problems. In *Proceedings of the 42nd annual Design Automation Conference*, pages 385–390, 2005.

[23] Yiwei Li, Edison Tsai, Marek Perkowski, and Xiaoyu Song. Grover-based ashenhurst-curtis decomposition using quantum language quipper. *Quantum Information & Computation*, 19(1-2):35–66, 2019.

[24] Stan Liao, Srinivas Devadas, Kurt Keutzer, and Steve Tjiang. Instruction selection using binate covering for code size optimization. In *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pages 393–399. IEEE, 1995.

[25] Stan Liao, Kurt Keutzer, STEVEN Tjiang, and Srinivas Devadas. A new viewpoint on code generation for directed acyclic graphs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 3(1):51–75, 1998.

[26] Edward J McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, 1956.

[27] Marta Mesquita and Ana Paias. Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem. *Computers & Operations Research*, 35(5):1562–1575, 2008.

[28] Giovanni De Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.

[29] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[30] Eric Paul, Bernd Steinbach, and Marek Perkowski. Application of cuda in the boolean domain for the unate covering problem. 2010.

[31] Marek Perkowski. Inverse problems, constraint satisfaction, reversible logic, invertible logic and grover quantum oracles for practical problems. In *Reversible Computation: 12th International Conference, RC 2020, Oslo, Norway, July 9-10, 2020, Proceedings 12*, pages 3–32. Springer, 2020.

[32] Marek A Perkowski, Jiuling Liu, and James E Brown. Rapid software prototyping: Cad design of digital cad algorithms. In *Progress in computer-aided VLSI design*, pages 353–401. 1990.

[33] Stanley R Petrick. A direct determination of the irredundant forms of a boolean function from the set of prime implicants. *Air Force Cambridge Res. Center Tech. Report*, pages 56–110, 1956.

[34] Richard L Rudell. *Multiple-valued logic minimization for PLA synthesis*. Electronics Research Laboratory, College of Engineering, University of . . . , 1986.

[35] Richard L Rudell. *Logic synthesis for VLSI design*. University of California, Berkeley, 1989.

[36] Tsutomu Sasao. *Logic synthesis and optimization*, volume 2. Springer, 1993.

[37] Michal Servit and Jan Zamazal. Heuristic approach to binate covering problem. In *Proceedings The European Conference on Design Automation*, pages 123–124. IEEE Computer Society, 1992.

[38] Bernd Steinbach and Christian Posthoff. Sources and obstacles for parallelization-a comprehensive exploration of the unate covering problem using both cpu and gpu. *GPU Computing with Applications in Digital Logic*, 63, 2012.

[39] Bernd Steinbach and Christian Posthoff. Fast calculation of exact minimal unate coverings on both the cpu and the gpu. In *Computer Aided Systems Theory-EUROCAST 2013: 14th International Conference, Las Palmas de Gran Canaria, Spain, February*

10-15, 2013. Revised Selected Papers, Part II 14, pages 234–241. Springer, 2013.

[40] Bernd Steinbach and Matthias Werner. Alternative approaches for fast boolean calculations using the gpu. In *Computational Intelligence and Efficiency in Engineering Systems*, pages 17–31. Springer, 2015.

[41] Tiziano Villa, Timothy Kam, Robert K Brayton, and Alberto L Sangiovanni-Vincenteili. Explicit and implicit algorithms for binate covering problems. *IEEE Transactions on computer-Aided Design of integrated Circuits and Systems*, 16(7):677–691, 1997.

[42] Wikipedia. Grover's algorithm — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Grover's%20algorithm&oldid=1169154668`. accessed: 2023-06-27.

Received