

Resource Efficient Boolean Function Solver on Quantum Computer

Xiang Li, Hanxiang Shen, Weiguo Gao, Yingzhou Li

Abstract

Nonlinear boolean equation systems play an important role in a wide range of applications. Grover's algorithm is one of the best-known quantum search algorithms in solving the nonlinear boolean equation system on quantum computers. In this paper, we propose three novel techniques to improve the efficiency under Grover's algorithm framework. A W-cycle circuit construction introduces a recursive idea to increase the solvable number of boolean equations given a fixed number of qubits. Then, a greedy compression technique is proposed to reduce the oracle circuit depth. Finally, a randomized Grover's algorithm randomly chooses a subset of equations to form a random oracle every iteration, which further reduces the circuit depth and the number of ancilla qubits. Numerical results on boolean quadratic equations demonstrate the efficiency of the proposed techniques.

1 Introduction

A nonlinear boolean equation system with R equations and n boolean variables admits,

$$f_j(x) = \bigoplus_{i_1, i_2, \dots, i_n=0}^1 c_{i_1, i_2, \dots, i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} = 0, \quad j = 1, \dots, R, \quad (1)$$

where $c_{i_1, i_2, \dots, i_n} \in \mathbb{F}_2$ are boolean coefficients, $(x_1, \dots, x_n) \in \mathbb{F}_2^n$ is the vector of boolean variables, the \oplus sign is the XOR (logical exclusive disjunction) function which is the addition operation on \mathbb{F}_2 , and the multiplication is the AND (logical conjunction) operation. Nonlinear boolean equation systems appear in a wide range of applications, including but not limited to logic synthesis [14], switching networks [13], cryptography, etc. Among these applications, it has become an important aspect of cryptography. Most cryptography algorithms, for example, RSA [17] and ECC [11], are considered secure based on their negligible success probability for attacks with bounded computational resources, whose complexity comes from solving nonlinear boolean equations. There has already been rich research on attacks with low nonlinearity cases, which is vulnerable to linear approximation attacks [12]. Further, via low order approximation [7, 15], attacks are advanced to a higher level, where the difficulty comes from solving highly nonlinear boolean equations.

As many claimed the achievement of quantum supremacy [1, 20], the quantum computer becomes an attractive platform to address these exponentially scaling problems, including

solving boolean equations and cryptography attacks [5, 18]. Grover’s algorithm [10] provides a $\mathcal{O}(\sqrt{N})$ algorithm for an unsorted database searching problem. Grover’s algorithm also makes it possible to address any numerical optimization problem as long as it can be formulated as a search problem. It reaches the asymptotic optimality and is the best-known quantum algorithm for problems when classical algorithms cannot perform better than brute force searching [6]. Fürer [3] briefed the procedure using Grover’s algorithm as a framework for adaptive global optimization algorithms. Grassl et al. [9] extended Grover’s algorithm to cracking one of the most famous block ciphers, AES, even though it had been suggested to be quantum-safe. It points out that, in principle, a quantum implementation in quantum mechanics is possible once one embeds operations in the cipher into permutations, which is reversible and can be viewed as a subset of all unitary operations.

Finding solutions to nonlinear boolean equations (1) can also be perceived as a searching problem through the truth table of the boolean function. A translation from the equations to the quantum circuit oracle is needed to adapt Grover’s algorithm for nonlinear boolean equations.

In this paper, we propose three novel techniques to improve the efficiency and reduce the cost of solving the boolean quadratic equations under Grover’s algorithm framework. Although all techniques in this paper are numerically tested in solving boolean quadratic equations (BQE), they can be adapted to nonlinear boolean equations of higher algebraic orders.

- (W-cycle Oracle) We propose a W-cycle structure in the construction of an oracle for the nonlinear boolean equations. Such a construction could use fewer qubits at the cost of a deeper circuit. This method provides a flexible trade-off between the number of required qubits and the circuit depth. The maximum number of boolean equations for a fixed number of qubits is carefully calculated. The asymptotic circuit depth is also estimated in this paper.
- (Oracle Compression) A rearranging technique is introduced to significantly reduce the circuit depth. It uses a greedy strategy to change the order of some interchangeable NOT and controlled–NOT gates. Many NOT and controlled–NOT gates are then canceled with each other, and the overall circuit depth is reduced without loss of accuracy.
- (Randomized Grover’s Algorithm) We propose a randomized Grover’s algorithm to establish a trade-off between the computational cost and success rate. The Grover operator varies randomly in each iteration by using only part of the boolean equations. The smaller number of boolean equations leads to a much more shallow quantum circuit.

Finally, we implement all the above three techniques in IBM Qiskit [19] and apply them to address nonlinear boolean equations. Numerical results show the efficiency of all three techniques. Given a quantum computer with 25 qubits, we are able to solve 21 boolean equations with 20 variables.

Figure 1 illustrates how our randomized Grover’s algorithm works via an 8-variable quadratic boolean equations example. It includes 4 snapshots of the probability distribution among all states during the process of iteration. Initiating from a state with equal

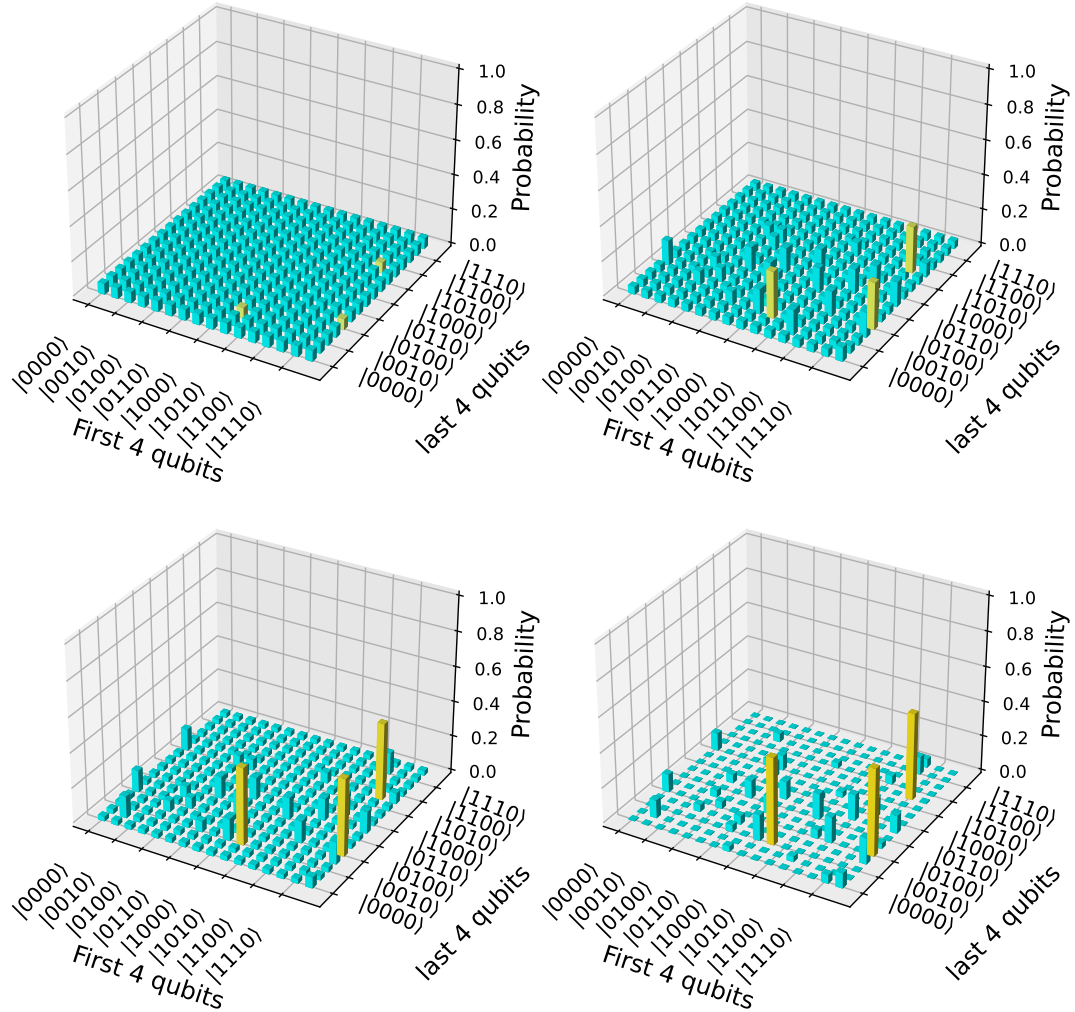


Figure 1: An 8-qubit example as the illustration of the randomized Grover's algorithm. From left to right, top to bottom, the snapshots are the probability distribution among all states after 0, 2, 4, and 6 iterations. We observe that the probability distribution is concentrated more and more on the correct states during these 6 iterations. The sum of the probabilities of the three correct states is 0.012, 0.217, 0.578, and 0.738 respectively. The x -axis and y -axis represent the first and the last 4 qubits respectively. The z -axis is the probability of the corresponding state. Note that for visual clarity, the x -axis labels and y -axis labels are not fully listed.

probabilities across all potential outcomes, the algorithm is observed to incrementally amplify the probabilities associated with the states that correspond to the solutions. Due to the randomized iterations, probabilities at other states are not reduced consistently but vary at different states.

The rest of the paper is organized as follows. Section 2 reviews the vanilla Grover’s algorithm. Section 3 provides details of the technique for constructing the W-cycle oracle and circuit reorganization. Section 4 explains the idea of randomized Grover’s algorithm and discusses the impacts of various iterative schemes. Numerical results are demonstrated in Section 5. Finally, Section 6 concludes the paper with a discussion on future works.

2 Preliminary

Grover’s algorithm has been proposed and developed for decades. This section serves as a review of applying Grover’s algorithm to solve boolean equations. We will first introduce notations and quantum circuit diagrams used throughout the paper in Section 2.1. Then a brief review of Grover’s algorithms is included in Section 2.2.

2.1 Notations and Diagram

Given two 1-qubit orthonormal basis states $|0\rangle$ and $|1\rangle$, any 1-qubit state can be represented as a linear combination, i.e., $|x\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex coefficients satisfying the unit-length constraint, $|\alpha|^2 + |\beta|^2 = 1$. Similarly, an n -qubit state $|x\rangle$ can be represented as a linear combination of n -qubit basis states. The n -qubit orthonormal basis states are tensor products of n 1-qubit basis states and are denoted as $|0\rangle = |0 \cdots 00\rangle$, $|1\rangle = |0 \cdots 01\rangle, \dots, |N-1\rangle = |1 \cdots 11\rangle$ for $N = 2^n$. Alternatively, an n -qubit state $|x\rangle = \sum_{i=0}^{N-1} \alpha_i |i\rangle$ is often represented by its coefficient vector $a = (\alpha_0 \cdots \alpha_{N-1})^\top \in \mathbb{C}^N$, where a is of unit length in 2-norm. The tensor product of two states is denoted as $|x\rangle \otimes |x'\rangle = |x\rangle|x'\rangle$ and the coefficient vector of $|x\rangle|x'\rangle$ is the Kronecker product of their coefficient vectors.

The basic operation in the quantum computer is called the quantum gate, which manipulates qubits by applying some unitary transformations. Several basic gates are widely used throughout the paper as our building blocks, namely, NOT, CNOT, MCX, and MCZ. The NOT gate flips the qubits, i.e., it changes $|0\rangle$ to $|1\rangle$ and changes $|1\rangle$ to $|0\rangle$. More explicitly, it exchanges the coefficient of $|0\rangle$ and $|1\rangle$,

$$\text{NOT}(\alpha|0\rangle + \beta|1\rangle) = \beta|0\rangle + \alpha|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

A CNOT gate is a controlled NOT gate that acts on two qubits, a control qubit and a target qubit. The NOT gate is applied to the target qubit only if the control qubit is in $|1\rangle$. For example, given a 2-qubit state

$$|x\rangle|x'\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle,$$

if we apply a controlled NOT gate on the first qubit, the state will be changed to

$$\text{CNOT}(|x\rangle|x'\rangle) = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{11}|10\rangle + \alpha_{10}|11\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix}.$$

After the CNOT operation, the 2-qubit system is said to be entangled since we cannot represent $\text{CNOT}(|x\rangle|x'\rangle)$ as a tensor product of two states. The MCX gate is a multi-controlled NOT gate that acts on multiple qubits, specifically multiple control qubits and a target qubit. It is defined as applying a NOT gate to the target qubit only if all control qubits are in $|1\rangle$. When there are two control qubits, MCX is also denoted as CCNOT. The MCZ gate is a multi-controlled Z gate similar to MCX, except that the 1-qubit gate been controlled is a Z gate,

$$\text{Z}(\alpha|0\rangle + \beta|1\rangle) = \alpha|0\rangle - \beta|1\rangle = \begin{pmatrix} 1 & \\ & -1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

The matrix representation of the MCZ with one controlling qubit admits

$$\text{MCZ}(|x\rangle|x'\rangle) = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle - \alpha_{11}|11\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix}.$$

For MCZ, the sign of the coefficient is flipped if all qubits, including control qubits and the target qubit, are in $|1\rangle$. In other words, the roles of control qubits and the target qubit could be swapped without affecting the outcomes. Hence, in latter quantum circuit diagrams, the MCZ is depicted without distinguishing the control qubits and the target qubit.

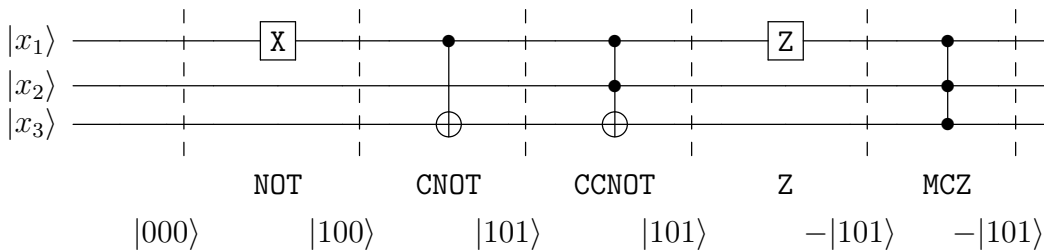


Figure 2: Diagrams of basic quantum gates in quantum circuits. The X in the box represents a NOT gate. For CNOT and CCNOT, the solid dots are on the control qubits, and \oplus is on the target qubit. The Z in the box represents a Z gate, whereas the MCZ gate is represented by solid dots on all the qubits it acts on. A toy example with an initial state $|000\rangle$ is given below the circuit.

Figure 2 illustrates NOT, CNOT, CCNOT, Z, and MCZ quantum gates in a quantum circuit with 3 qubits. These circuit diagrams will be repeatedly used in this paper. In this figure, we give a toy example of a quantum state starting from $|000\rangle$. The first NOT gate flips the first qubit from $|0\rangle$ to $|1\rangle$. Then CNOT flips the third qubit. CCNOT does not change the state since not all control qubits are in $|1\rangle$ state. The following Z gate on the first qubit adds a negative sign. Finally, the MCZ gate acts on $|101\rangle$ and the state remains the same.

A Hardmard gate H is a single qubit gate that transforms the basis states into a superposition of the basis states,

$$H(\alpha|0\rangle + \beta|1\rangle) = \frac{\alpha}{\sqrt{2}}(|0\rangle + |1\rangle) + \frac{\beta}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

One of its important properties is that it can be used to create a uniform superposition of all basis states on multiple qubits that are initialized to $|0\rangle$,

$$H^{\otimes n}(|0\rangle|0\rangle \cdots |0\rangle) = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle.$$

This is a crucial initialization step in many quantum algorithms, including Grover's algorithm.

2.2 Grover's Algorithm

Grover's algorithm is a quantum unstructured search algorithm. Given an oracle circuit O producing a different output for a particular input, Grover's algorithm starts from an equal probability for all possible inputs and applies an iterative procedure to amplify the probability of particular desired inputs. Finally, a measurement would result in one of the particular inputs with high probability. In this section, we review Grover's algorithm and take a boolean function solver as an example.

For an n -variable boolean function as in (1), there are $N = 2^n$ different inputs for x_1, \dots, x_n , and we adapt n qubits to represent the probabilities for all inputs. Another m qubits, known as the oracle workspace or ancillae, are reserved for oracle circuits. A different construction of the oracle circuits for boolean functions results in different m . The vanilla construction, as in Section 3.1, requires m to be at least the number of equations in (1), R . At the beginning of Grover's algorithm, the first n qubits are initialized as equal probabilities for all inputs, i.e.,

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle \quad (2)$$

via applying n Hardmard gate H to $|0\rangle = |0\rangle^{\otimes n}$. The latter m qubits are kept in $|0\rangle = |0\rangle^{\otimes m}$.

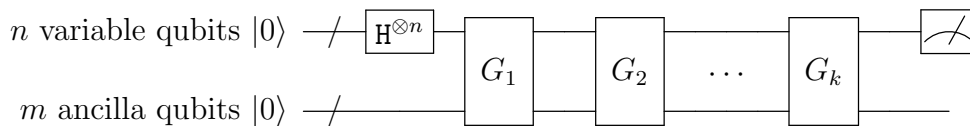


Figure 3: Grover's algorithm. In vanilla Grover's algorithm, all G_i s are $G = WO$ for O and W being defined in (3) and (4) respectively.

After the initialization, Grover's algorithm applies a sequence of Grover iterations. Each Grover iteration applies two circuits: an oracle O and a Householder-like diffusion operator W . The oracle O is application-dependent and often treated as a black box in Grover's algorithm. In general unstructured search problems, the oracle flips the sign of desired

solutions while keeping others unchanged. In solving boolean functions, the oracle flips the sign of $|x\rangle$ if x is a solution of (1). More precisely, the action of the oracle O is

$$O : |x\rangle|0\rangle^{\otimes m} \mapsto (-1)^{g(x)}|x\rangle|0\rangle^{\otimes m}, \quad x = 0, 1, \dots, N-1, \quad (3)$$

where $g(x)$ is the solution indicator function as

$$g(x) = \begin{cases} 1 & \text{if } x \text{ is a solution of (1),} \\ 0 & \text{otherwise.} \end{cases}$$

The quantum circuits for the oracle O of the boolean functions are detailed in Section 3, and efficient constructions are also proposed therein. Importantly, we emphasize that the ancilla qubits have to remain in $|0\rangle = |0\rangle^{\otimes m}$ after the oracle circuit to avoid any side effects. The Householder-like diffusion operation $W = 2|\psi\rangle\langle\psi| - I$ is then applied to the first n qubits, whose action on a general state $\sum_i c_i|i\rangle$ admits

$$W : \sum_i c_i|i\rangle \mapsto \sum_i (-c_i + 2\langle c \rangle) |i\rangle, \quad (4)$$

where $|\psi\rangle$ is as defined in (2), and $\langle c \rangle$ is the average of $\{c_i\}$. This is a crucial step to amplifies the probabilities of solutions. Figure 3 shows an illustration of Grover's algorithm.

Though Grover's algorithm carries an iterative procedure, it is different from classical iterative methods. Instead of converging to a fixed point of the iterative mapping, Grover's algorithm conducts a fixed number of iterations. We take the boolean functions (1) as an example. All N possible boolean variables are split into solutions and non-solutions of (1) and the number of solutions is denoted as M . The set of all solutions is denoted as \mathcal{S} . The initial state $|\psi\rangle$ as in (2) could be rewritten as

$$|\psi\rangle = \cos \frac{\theta}{2} |\alpha\rangle + \sin \frac{\theta}{2} |\beta\rangle,$$

where $|\alpha\rangle$ and $|\beta\rangle$ are the superpositions of non-solutions and solutions, i.e.,

$$|\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_{x \notin \mathcal{S}} |x\rangle, \quad |\beta\rangle = \frac{1}{\sqrt{M}} \sum_{x \in \mathcal{S}} |x\rangle,$$

and θ is determined by $\cos \frac{\theta}{2} = \sqrt{\frac{N-M}{N}}$. The action of G on $|\psi\rangle$ obeys ¹

$$G|\psi\rangle = WO|\psi\rangle = W \left(\cos \frac{\theta}{2} |\alpha\rangle - \sin \frac{\theta}{2} |\beta\rangle \right) = \cos \frac{3\theta}{2} |\alpha\rangle + \sin \frac{3\theta}{2} |\beta\rangle. \quad (5)$$

As illustrated in Figure 4, the action of G could be understood in a geometrical way. Applying the oracle O flips the sign of $|\alpha\rangle$. Then, the Householder-like operation W makes the

¹Actually, $G = WO$ is applied to all $n+m$ qubits, i.e., O is applied to $|\psi\rangle|0\rangle^{\otimes m}$, and W is applied to the first n qubits. For the sake of notations, we omit the m ancilla qubits in (5) and (6).

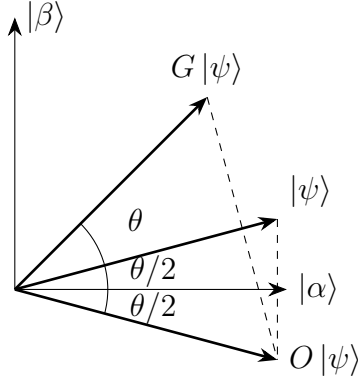


Figure 4: The effect of an iteration of Grover’s algorithm. From the initial state $|\psi\rangle$, the oracle O flips the sign of the solution to $g(x) = 1$, as shown in (3). Then, the diffusion operation W makes a reflection of the state $O|\psi\rangle$ by $|\psi\rangle$. This single step of iteration rotates $|\psi\rangle$ to $G|\psi\rangle$ by an angle of θ . This step reduces the probability of any state in $|\alpha\rangle$ (i.e., wrong solution) being the outcome of a measurement, which is $|\langle\psi|\alpha\rangle|^2$. Note that rotating over $|\beta\rangle$, i.e., repeating the iteration more than K times computed in (7), will contradictorily increase the probability of any state in α being the outcome of a measurement.

state $O|\psi\rangle$ mirrored with respect to $|\psi\rangle$, and results in a state $G|\psi\rangle$ have greater overlapping with $|\beta\rangle$. Applying Grover iteration k times leads to

$$G^k|\psi\rangle = (WO)^k|\psi\rangle = \cos\left(\frac{2k+1}{2}\theta\right)|\alpha\rangle + \sin\left(\frac{2k+1}{2}\theta\right)|\beta\rangle. \quad (6)$$

Repeating the iteration for

$$K = \text{round}\left(\frac{\arccos\sqrt{M/N}}{\theta}\right) \quad (7)$$

times makes $G^K|\psi\rangle$ sufficiently close to $|\beta\rangle$ and the measurement would fall into one of the solutions with a probability $\sin^2\left(\frac{2K+1}{2}\theta\right)$, which is close to 1.

3 Efficient Oracle Construction

The oracle for boolean functions can be constructed either in a qubit-efficient way or in a depth-efficient way. We first show both the qubit-efficient and depth-efficient vanilla constructions of the oracle in Section 3.1. In Section 3.2, we propose a novel recursive oracle construction, which exploits the maximum possible number of equations for a given number of ancillae. Finally, a rearranging technique is introduced in Section 3.3 to further reduce the depth of the circuit by greedily exploring possible parallelization.

3.1 Basic Oracle Construction

We first introduce a straightforward oracle construction for a single boolean equation. Given a boolean equation with n variables, $f(x) = 0$, we adopt $n + 1$ qubits for the oracle, where

the first n qubits represent boolean variables and the last ancilla qubit is used to track the value of $f(x)$. Without loss of generality, we assume $f(x)$ is in a sum of product form, as in (1). Then each product term is implemented by CNOT, CCNOT, or MCX. For example, a CNOT controlling from the i -th qubit to the ancilla represents the x_i term; a CCNOT controlling from the i - and j -th qubits to the ancilla represents the $x_i x_j$ term; other product terms with more than two variables could be implemented by an MCX controlling from qubits corresponding to the variables therein to the ancilla. After applying these gates for all product terms in $f(x)$, the ancilla is in $|0\rangle$ if $|x\rangle$ satisfies $f(x) = 0$, and in $|1\rangle$ if $|x\rangle$ satisfies $f(x) = 1$. Then we apply an X gate followed by a Z gate on the ancilla such that a negative sign is applied for all $|x\rangle$ solves $f(x) = 0$. Finally, to reset the ancilla qubit to $|0\rangle$ after the oracle as in (3), all previous gates except the last Z gate are applied one more time. A detailed quantum circuit of the oracle for $f(x) = x_1 \oplus x_1 x_2 = 0$ as well as its abbreviation are given in Figure 5. In the rest of the paper, the abbreviated quantum circuit is called the *function-controlled-not* gate or *f_i -controlled-not* gate for a particular boolean function f_i . In all later quantum circuit diagrams, we will only draw the abbreviated diagram instead of complex circuits.

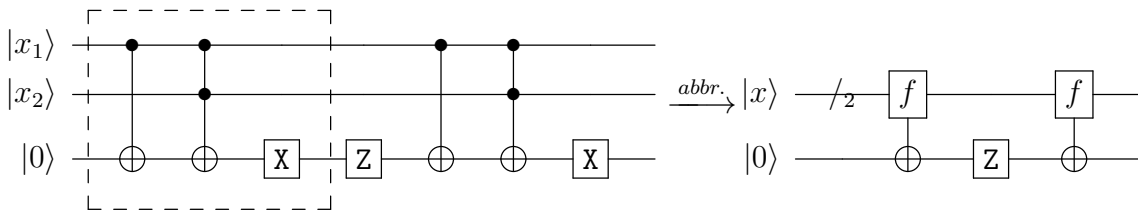


Figure 5: Oracle circuit for boolean equation: $f(x) = x_1 \oplus x_1 x_2 = 0$. The extra X gate ensures the ancilla in state $|1\rangle$ if $f(x) = 0$.

When the boolean equation system involves more than one equation, i.e., R equations in (1) are denoted as f_1, \dots, f_R , there are two basic constructions for the oracle: 1) multiplying equations together and treating them as a single boolean equation; 2) adopting R ancilla qubits to track the values of all boolean equations. We briefly introduce both basic constructions and discuss their drawback. Then a novel recursive construction for oracle is proposed in Section 3.2.

For a set of equations $f_i(x) = 0, i = 1, \dots, R$, one can multiply them together as

$$f(x) \triangleq (f_1(x) \oplus 1)(f_2(x) \oplus 1) \cdots (f_R(x) \oplus 1) \oplus 1. \quad (8)$$

A solution of $f(x^*) = 0$ for $f(x)$ as in (8) requires that $f_i(x^*) = 0$ for all $i = 1, \dots, R$. Hence solving $f(x) = 0$ is equivalent to solve the boolean equation system (1). Then $f(x)$ is transformed to a sum of product form and the oracle is constructed as we discussed above. For example, given a quadratic boolean equation system

$$\begin{aligned} f_1(x) &= x_1 \oplus x_1 x_2 = 0, \\ f_2(x) &= x_3 x_4 = 0, \\ f_3(x) &= x_1 x_4 = 0, \\ f_4(x) &= x_2 \oplus x_3 \oplus x_4 = 0, \end{aligned}$$

the sum of product form of $f(x)$ admits

$$f(x) = x_1 x_2 x_3 \oplus x_1 x_3 x_4 \oplus x_2 x_3 x_4 \oplus x_1 x_2 \oplus x_1 x_3 \oplus x_1 x_4 \oplus x_3 x_4 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4.$$

In transformation to the sum of product form, there are cancellations to reduce the number of product terms. However, the number of product terms in $f(x)$ is often found to be much larger than that in $f_i(x)$ s. Another drawback is caused by the MCX. The number of variables in the product term in $f(x)$ is larger than that in $f_i(x)$ s. Hence the corresponding MCX has more controlling qubits. It is generally considered that a quantum gate operation involving many qubits is more difficult to implement, and an approximated model [16] shows an exponential growth of the circuit depth to the number of controlling qubits. The circuit depth of such an oracle construction is much larger than all later oracle constructions and is often found to be impractical on current quantum devices.

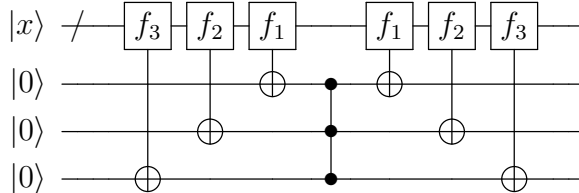


Figure 6: A vanilla stack style oracle construction for a boolean equation system with three equations.

Another basic construction of the oracle for the boolean equation system with R equation is to adopt R ancilla qubits to track the outcomes of all equations. For each $i = 1, \dots, R$, we construct the circuit for $f_i(x)$ with the i -th ancilla qubit. Then R ancilla qubits now keep the outcome of all boolean equations. The i -th ancilla qubit is in $|1\rangle$ if $f_i(x) = 0$, and in $|0\rangle$ otherwise. A solution of (1) satisfies all boolean equations, and hence all ancilla qubits are in $|1\rangle$. An MCZ gate is then applied and flips the sign if all ancilla qubits are in $|1\rangle$. After applying the MCZ gate, we revert all ancilla qubits to their initialized state by applying all previous single equation circuits in the reversed order. Figure 6 provides an example of a boolean equation system with three equations. In the rest of the paper, we call such an oracle construction the vanilla stack style. If we assume that each boolean equation is quadratic and has P product terms, the vanilla stack style oracle construction has circuit depth $RP+C$ and uses R ancilla qubits, where C is the constant circuit depth of MCZ gate.²

3.2 Recursive Oracle Construction

We provide a flexible way of constructing the oracle making a trade-off between the number of qubits and circuit depth. In our novel construction, none of the boolean equations are multiplied together. The idea, in a nutshell, is to build the circuit recursively based on the vanilla stack style. In the following, we propose the recursive oracle construction in detail and explore the maximum number of possible boolean equations that could be constructed on m ancilla qubits. A building block in the recursive oracle construction is denoted as $U_m^{(\ell)}$, where ℓ denotes the recursive level and m denotes the number of ancilla qubits. We will first introduce the quantum circuit $U_m^{(1)}$, which is similar to the vanilla stack style. Then a

²The circuit depth counts the longest sequence of simple gates, including 1-qubit gates and a few multi-qubit gates, i.e., CNOT, CCNOT. The MCZ gate should be decomposed into H and MCX gates, whose depth is denoted as a constant C .

recursive construction from $U_1^{(\ell-1)}, U_2^{(\ell-1)}, \dots, U_{m-1}^{(\ell-1)}$ to $U_m^{(\ell)}$ is proposed. Finally, we construct the oracle quantum circuit based on many $U_m^{(\ell)}$ s.

Quantum circuit for $U_m^{(1)}$. The idea behind $U_m^{(1)}$ is to fully exploit all ancilla qubits while storing the desired information in only the last ancilla qubit. Given $m - 1$ boolean functions, $f_1(x), \dots, f_{m-1}(x)$, we first construct a sequence of function-controlled-not gates controlling from $|x\rangle$ to the 1st, 2nd, \dots , $(m - 1)$ -th ancilla qubits for boolean function f_1, f_2, \dots, f_{m-1} , respectively. Then an MCX gate is applied controlling from ancilla qubits $1, 2, \dots, m - 1$ to the m -th ancilla qubits. After these gate operations, the m -th ancilla qubit is in $|1\rangle$ if $f_1 = 0, f_2 = 0, \dots, f_{m-1} = 0$ are satisfied. Hence the solutions of $f_i(x) = 0$ for $i = 1, 2, \dots, m - 1$ are tracked in the m -th ancilla qubit. In order to reuse other ancilla qubits, we apply the same sequence of function-controlled-not gates in the reversed ordering and reset ancilla qubits $1, 2, \dots, m - 1$ to $|0\rangle$. Figure 7 depicts an example for $U_4^{(1)}$. When $m = 1$, we use function-controlled-not gate directly on the ancilla. Hence $U_1^{(1)}$ is the same as a single function-controlled-not gate.

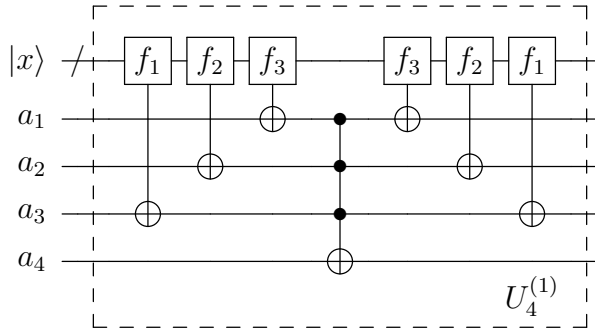


Figure 7: Quantum circuit for $U_4^{(1)}$. The function-controlled-gate is as defined in Figure 5.

Recursion from $U_j^{(\ell-1)}$ s to $U_m^{(\ell)}$. Assume we have constructed all quantum circuits $U_1^{(\ell-1)}, \dots, U_{m-1}^{(\ell-1)}$ at previous level $\ell - 1$. We further assume that after applying $U_j^{(\ell-1)}$ only the j -th ancilla qubit is affected, and all other ancilla qubits remain unchanged. The construction of $U_m^{(\ell)}$ is as follows. We first apply $U_{m-1}^{(\ell-1)}, U_{m-2}^{(\ell-1)}, \dots, U_1^{(\ell-1)}$ in order. Then an MCX gate is applied controlling from ancilla qubits $1, 2, \dots, m - 1$ to the m -th ancilla qubit. After these gate operations, the m -th ancilla qubit is in $|1\rangle$ if all boolean functions behind $U_1^{(\ell-1)}, \dots, U_{m-1}^{(\ell-1)}$ are satisfied. All these boolean functions are called the boolean functions behind $U_m^{(\ell)}$. Similar as that in $U_m^{(1)}$, we apply $U_1^{(\ell-1)}, U_2^{(\ell-1)}, \dots, U_{m-1}^{(\ell-1)}$ in order to reset ancilla qubits $1, 2, \dots, m - 1$ to $|0\rangle$. Importantly, the ordering of $U_1^{(\ell-1)}, \dots, U_{m-1}^{(\ell-1)}$ cannot be changed since $U_j^{(\ell-1)}$ s are not commutable. Noticeably, $U_1^{(2)}$ is not defined under our recursion and so are $U_1^{(\ell)}$ for $\ell > 1$. In our recursion, the actual circuits for $U_1^{(\ell)}$ are $U_1^{(1)}$. For the sake of notation, $U_m^{(\ell)}$ for $\ell > m$ are defined as $U_m^{(\ell)} \triangleq U_m^{(m)}$. Figure 8 depicts the construction of $U_4^{(\ell)}$ from $U_1^{(\ell-1)}, U_2^{(\ell-1)}, U_3^{(\ell-1)}$.

Recursive oracle construction. Given a recursive level $\ell > 1$ and m ancilla qubits, the quantum circuit for the oracle is composed of $U_1^{(\ell-1)}, \dots, U_m^{(\ell-1)}$.³ We first apply $U_m^{(\ell-1)}$

³For an oracle circuit with $\ell = 1$, the construction falls back to the vanilla stack style. Hence we only introduce oracle constructions with $\ell > 1$ in the rest of the paper.

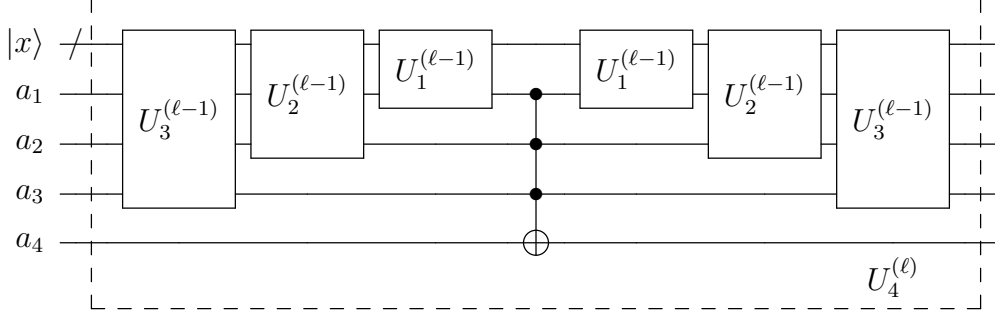


Figure 8: Quantum circuit for $U_4^{(\ell)}$ constructed from $U_1^{(\ell-1)}$, $U_2^{(\ell-1)}$, and $U_3^{(\ell-1)}$.

to $U_1^{(\ell-1)}$ in order. Then an MCZ gate is applied to all ancilla qubits, which introduces the sign flip for solutions. Finally, $U_1^{(\ell-1)}$ to $U_m^{(\ell-1)}$ are applied in order to reset all ancilla qubits to $|0\rangle$. The overall structure for the oracle circuit is similar to that in vanilla stack style except for function-controlled-gates being replaced by our recursive circuits $U_j^{(\ell)}$ s. Figure 9 depicts the oracle circuits with ℓ and $m = 4$. Another example is given in Figure 10 for $\ell = 2$ and $m = 3$, where all $U_j^{(\ell)}$ s are expanded into function-controlled-gates. The level ℓ oracle construction pseudocode is detailed in Algorithm 2 and the recursion pseudocode for $U_m^{(\ell)}$ is in Algorithm 1 for a boolean equation system with R equations and m ancilla qubits. In these pseudocodes, the set of boolean equations $\mathcal{F} = \{f_i\}_{i=1}^R$ is a global variable and $\mathcal{F}.pop()$ pops out a boolean function from the set. If the set \mathcal{F} is empty, $\mathcal{F}.pop()$ then returns a trivial boolean equation $f(x) \equiv 0$. The f -controlled-gate, in this case, is empty, and no quantum gate is appended to the circuit.

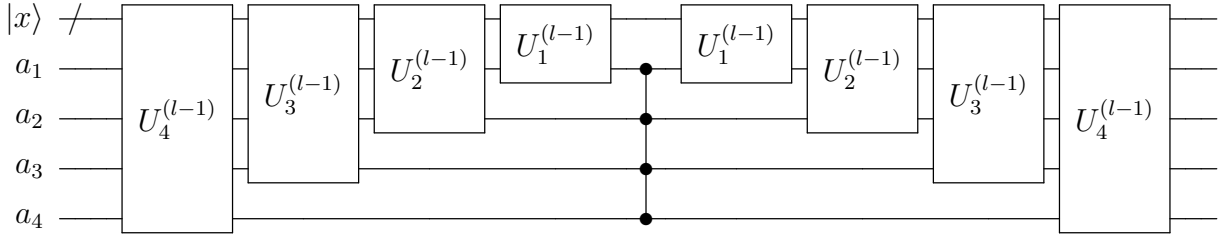


Figure 9: A level ℓ recursive construction of oracle with $m = 4$ ancilla qubits.

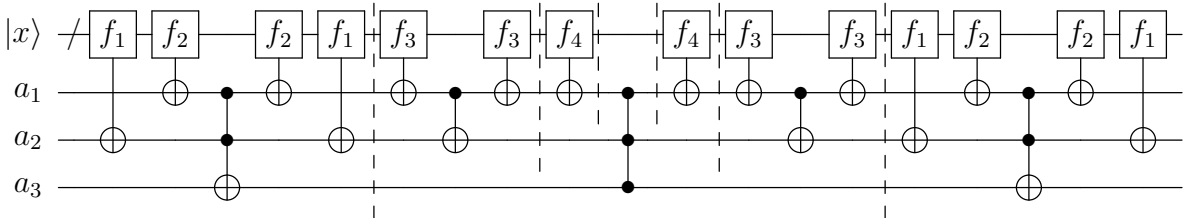


Figure 10: A level $\ell = 2$ recursive construction of oracle with $m = 3$ ancilla qubits for a boolean equation system with 4 equations.

Next, we calculate the capacity of the oracle or $U_m^{(\ell)}$, i.e., the maximum number of boolean equations that could be constructed into the oracle or $U_m^{(\ell)}$. Theorem 1 states the capacity

Algorithmus 1 $U_m^{(\ell)}$ construction.

Input: target ancilla qubits m , recursive level ℓ , and set of boolean equations $\mathcal{F} = \{f_i\}_{i=1}^R$.

Output: quantum circuit for $U_m^{(\ell)}$.

```
1: procedure UCIRCUIT( $\ell, m$ )
2:   if  $\ell > m$  then return UCIRCUIT( $m, m$ ) end if
3:   if  $R = 0$  then return None end if
4:   if  $\ell = 0$  then
5:      $f = \mathcal{F}.\text{pop}()$ 
6:     return  $f$ -controlled gate controlling  $m$ -th ancilla qubit
7:   end if
8:   if  $\ell = 1$  then
9:     if  $m = 1$  then
10:       $f = \mathcal{F}.\text{pop}()$ 
11:      return  $f$ -controlled gate
12:    end if
13:    for  $j = m - 1, m - 2, \dots, 1$  do
14:       $f_j = \mathcal{F}.\text{pop}()$ 
15:      Append  $f_j$ -controlled gate controlling  $j$ -th ancilla qubit to output circuit
16:    end for
17:    Append MCX gate controlling from  $1, \dots, m - 1$  ancilla qubits to  $m$  ancilla qubit
    to output circuit
18:    for  $j = 1, 2, \dots, m - 1$  do
19:      Append  $f_j$ -controlled gate controlling  $j$ -th ancilla qubit to output circuit
20:    end for
21:  else
22:    for  $j = m - 1, m - 2, \dots, 1$  do
23:       $U_j^{(\ell-1)} = \text{UCIRCUIT}(\ell - 1, j)$ 
24:      Append  $U_j^{(\ell-1)}$  to output circuit
25:    end for
26:    Append MCX gate controlling from  $1, \dots, m - 1$  ancilla qubits to  $m$  ancilla qubit
    to output circuit
27:    for  $j = 1, 2, \dots, m - 1$  do
28:      Append  $U_j^{(\ell-1)}$  to output circuit
29:    end for
30:  end if
31:  return output circuit
32: end procedure
```

Algorithmus 2 Oracle construction.

Input: m ancilla qubits, recursive level ℓ , and set of boolean equations $\mathcal{F} = \{f_i\}_{i=1}^R$.

Output: oracle quantum circuit.

```

1: procedure ORACLE( $\ell, m$ )
2:   for  $j = m, m - 1, m - 2, \dots, 1$  do
3:      $U_j^{(\ell-1)} = \text{UCIRCUIT}(\ell - 1, j)$ 
4:     Append  $U_j^{(\ell-1)}$  to output circuit
5:   end for
6:   Append MCZ gate on all ancilla qubits to output circuit
7:   for  $j = 1, 2, \dots, m - 1, m$  do
8:     Append  $U_j^{(\ell-1)}$  to output circuit
9:   end for
10: end procedure

```

of $U_m^{(\ell)}$. Corollary 1 then sums the capacities of $U_j^{(\ell-1)}$ for $j = 1, 2, \dots, m - 1$ and obtains the capacity of a level ℓ oracle with m ancilla qubits.

Theorem 1. Let $N_m^{(\ell)}$ denote the capacity for the level ℓ recursive quantum circuit $U_m^{(\ell)}$ on m ancilla qubits. For various scenarios of ℓ and m , we have

$$N_m^{(\ell)} = \begin{cases} 1 & \ell \geq 1, m = 1, \\ 2^{m-2} & \ell \geq m - 1, m \geq 2, \\ \sum_{j=0}^{\ell} \binom{m-2}{j} & \text{Otherwise.} \end{cases}$$

Proof. For various scenarios of ℓ and m , we have

$$N_m^{(\ell)} = \begin{cases} 1 & \ell = 1, m = 1, \\ m - 1 & \ell = 1, m \geq 2, \\ N_m^{(m)} & \ell > m, \\ \sum_{j=1}^{m-1} N_j^{(\ell-1)} & \text{Otherwise.} \end{cases} \quad (9)$$

The recursive formula (9) can be obtained directly from the process of constructing the circuit $U_m^{(\ell)}$. Obviously, we have $N_1^{(\ell)} = 1$ and $N_2^{(\ell)} = 1$ for all $\ell \geq 1$.

We first derive the expression of $N_m^{(\ell)}$ for $\ell \geq m - 2$ and $m \geq 3$. The claim is that $N_m^{(\ell)} = 2^{m-2}$ for $\ell \geq m - 2$ and $m \geq 3$. From the recursive formula (9), we have $N_3^{(1)} = 2$, $N_3^{(2)} = N_2^{(1)} + N_1^{(1)} = 2$, $N_3^{(3)} = N_2^{(2)} + N_1^{(2)} = 2$, and $N_3^{(\ell)} = N_3^{(3)} = 2$ for all $\ell > 3$. If $N_n^{(j)} = 2^{n-2}$ holds for all $j \geq n - 2$ and $n = 3, \dots, m - 1$, then we have

$$\begin{aligned} N_m^{(m-2)} &= \sum_{j=1}^{m-1} N_j^{(m-3)} = N_1^{(m-3)} + N_2^{(m-3)} + \sum_{j=3}^{m-1} N_j^{(m-3)} = 1 + 1 + \sum_{j=3}^{m-1} 2^{j-2} = 2^{m-2}, \\ N_m^{(m-1)} &= \sum_{j=1}^{m-1} N_j^{(m-2)} = N_1^{(m-2)} + N_2^{(m-2)} + \sum_{j=3}^{m-1} N_j^{(m-2)} = 1 + 1 + \sum_{j=3}^{m-1} 2^{j-2} = 2^{m-2}, \\ N_m^{(\ell)} &= N_m^{(m)} = \sum_{j=1}^{m-1} N_j^{(m-1)} = N_1^{(m-1)} + N_2^{(m-1)} + \sum_{j=3}^{m-1} N_j^{(m-1)} = 1 + 1 + \sum_{j=3}^{m-1} 2^{j-2} = 2^{m-2}, \end{aligned}$$

for all $\ell > m$. By induction, we prove the claim.

Finally, we derive the expression of $N_m^{(\ell)}$ for $2 \leq \ell \leq m - 3$ and $m \geq 5$. The recursive formula (9) could be written as

$$N_m^{(\ell)} = \sum_{j=1}^{m-1} N_j^{(\ell-1)} = N_{m-1}^{(\ell-1)} + \sum_{j=1}^{m-2} N_j^{(\ell-1)} = N_{m-1}^{(\ell-1)} + N_{m-1}^{(\ell)}. \quad (10)$$

Let us introduce an auxiliary variable and its recursive formula,

$$T_m^{(\ell)} \triangleq N_m^{(\ell)} - N_m^{(\ell-1)} = N_{m-1}^{(\ell-1)} + N_{m-1}^{(\ell)} - N_{m-1}^{(\ell-2)} - N_{m-1}^{(\ell-1)} = T_{m-1}^{(\ell)} + T_{m-1}^{(\ell-1)} \quad (11)$$

for $2 \leq \ell \leq m - 2$ and $m \geq 4$. For $m \geq 5$, we have

$$T_m^{(m-2)} = N_m^{(m-2)} - N_m^{(m-3)} = 2^{m-2} - (N_{m-1}^{(m-4)} + N_{m-1}^{(m-3)}) = N_{m-1}^{(m-3)} - N_{m-1}^{(m-4)} = T_{m-1}^{(m-3)},$$

where the first and last equalities adopt the definition of $T_m^{(\ell)}$, and the second and third equalities adopt the recursive formula (10) and $N_m^{(\ell)} = 2^{m-2}$ for $\ell \geq m - 2$. Through a direct calculation, we obtain, $T_m^{(m-2)} = T_5^{(3)} = T_4^{(2)} = 1$ for all $m \geq 4$. This is one boundary condition for the recursive formula of $T_m^{(\ell)}$. For the other boundary, we have

$$T_m^{(2)} = N_m^{(2)} - N_m^{(1)} = \sum_{j=1}^{m-1} N_j^{(1)} - (m-1) = \sum_{j=2}^{m-2} N_j^{(1)} = \sum_{j=1}^{m-3} j = \binom{m-2}{2}$$

for $m \geq 4$. Based on the boundary conditions and recursive formula (11), we notice that $T_m^{(\ell)}$ is a part of Yang Hui's triangle (Pascal's triangle) and admits the expression

$$T_m^{(\ell)} = \binom{m-2}{\ell} \quad (12)$$

for $2 \leq \ell \leq m - 2$ and $m \geq 4$. Solving (11) for $N_m^{(\ell)}$, we obtain

$$N_m^{(\ell)} = T_m^{(\ell)} + N_m^{(\ell-1)} = \sum_{j=2}^{\ell} T_m^{(j)} + N_m^{(1)} = \sum_{j=2}^{\ell} \binom{m-2}{j} + m - 1 = \sum_{j=0}^{\ell} \binom{m-2}{j}.$$

When $\ell = m - 2$ and $m \geq 4$, the expression $\sum_{j=0}^{\ell} \binom{m-2}{j} = 2^{m-2}$ coincides with the above claim. This proves the theorem. \square

Corollary 1. *Let $F_m^{(\ell)}$ denote the capacity for the level ℓ recursive oracle circuit on m ancilla qubits. For various scenarios of ℓ and m , we have*

$$F_m^{(\ell)} = \begin{cases} 2^{m-1}, & \ell \geq m - 1 \\ \sum_{j=0}^{\ell} \binom{m-1}{j}, & \text{Otherwise} \end{cases}. \quad (13)$$

Corollary 1 could be derived directly from Theorem 1. The capacities of $U_{m+1}^{(\ell)}$ and the level ℓ oracle circuit on m ancilla qubits are the same. Hence we could reorganize the capacity in Theorem 1 and lead to Corollary 1.

Besides the capacitance of the oracle given a fixed number of ancilla qubits, another concern for the oracle construction is the circuit depth. We now calculate the total depth in terms of the number of function-controlled gates. The circuit depth is perceived as a measure of the time complexity for the quantum algorithm.

ℓ	m									
	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	1	2	4	7	11	16	22	29	37	46
3	1	2	4	8	15	26	42	64	93	130
4	1	2	4	8	16	31	57	99	163	256
5	1	2	4	8	16	32	63	120	219	382
6	1	2	4	8	16	32	64	127	247	466
7	1	2	4	8	16	32	64	128	255	502
8	1	2	4	8	16	32	64	128	256	511
9	1	2	4	8	16	32	64	128	256	512
10	1	2	4	8	16	32	64	128	256	512

Table 1: Maximum number of boolean equations in oracle circuit, $F_m^{(\ell)}$, for ℓ being the recursive level and m being the number of ancilla qubits.

Theorem 2. Let $K_m^{(\ell)}$ denote the number of function-controlled gates in $U_m^{(\ell)}$. For various scenarios of ℓ and m , we have

$$K_m^{(\ell)} = \begin{cases} 1, & m = 1 \\ 2 \cdot 3^{m-2}, & \ell \geq m - 1, m \geq 2. \\ \sum_{j=1}^{\ell} \binom{m-2}{j-1} 2^{j-1} + \sum_{j=0}^{\ell} \binom{m-2}{j} 2^j, & \text{Otherwise} \end{cases}$$

Corollary 2. Let $G_m^{(\ell)}$ denote the number of function-controlled gates in a level ℓ oracle circuit on m ancilla qubits. For various scenarios of ℓ and m , we have

$$G_m^{(\ell)} = \begin{cases} 2 \cdot 3^{m-1}, & \ell \geq m - 1 \\ \sum_{j=1}^{\ell} \binom{m-1}{j-1} 2^{j-1} + \sum_{j=0}^{\ell} \binom{m-1}{j} 2^j, & \text{Otherwise} \end{cases}$$

In Theorem 1 and Corollary 1, we give both expressions for the circuit capacities. Fixing the recursive level ℓ , both capacities are dominated by $\binom{m-2}{\ell}$ term as m goes large. Hence, given a BQE system with R equations and recursive level ℓ , the number of required ancilla qubits scales as $\mathcal{O}(R^{1/\ell})$. In practice, we could calculate the capacities and find the most proper number of ancilla qubits on classical computers efficiently. Table 1 calculates the capacities of oracles for various ℓ and m for reference.

The asymptotic scaling of the capacity shows the power of our proposed recursive oracle construction method. While in the era of noisy intermediate-scale quantum (NISQ), we are both limited by the number of qubits and quantum circuit depth. Increasing ℓ increases the capacity of the oracle. According to Corollary 2, the circuit depth grows even faster than the capacities with an extra factor 2^ℓ . Hence, given a BQE system, the proper ℓ and m should be carefully chosen based on the property and available resources of the quantum computer.

3.3 Oracle Compression

The recursive oracle circuit above can be optimized to reduce the depth. Recall that the function-controlled gates in the oracle circuit represent product terms in the boolean equation. From the representation of the boolean equation, we know that these gates should be commutable. From the quantum gate perspective, these function-controlled gates consist of (multi-)controlled NOT gates, controlling from some of the first n qubits to an ancilla qubit (including the NOT gates, which is the special case of controlling from none of the qubits). All such controlled NOT gates are commutable. After commuting these gates and other commutable gates, many gates could be eliminated without affecting the circuit outcomes.

Algorithmus 3 Greedy oracle compression.

Input: a list of gates $\mathcal{G} = [g_1, \dots, g_k]$.

Output: a list of rearranged gates \mathcal{R} .

```

1:  $\mathcal{G} = \text{sort}(\mathcal{G})$ 
2: for each  $g_i \in \mathcal{G}$  do
3:   if  $g_i = g_{i+1}^\dagger$  then
4:     Eliminate  $g_i$  and  $g_{i+1}$  from  $\mathcal{G}$ 
5:   end if
6: end for
7:  $\mathcal{R} = []$ 
8: while  $\mathcal{G} \neq \emptyset$  do
9:    $\mathcal{Q} = \emptyset$ 
10:  for each  $g_i \in \mathcal{G}$  do
11:    if qubits of  $g_i$  are not in  $\mathcal{Q}$  then
12:      Add qubits of  $g_i$  to  $\mathcal{Q}$ 
13:      Append  $g_i$  to  $\mathcal{R}$ 
14:      Eliminate  $g_i$  from  $\mathcal{G}$ 
15:    end if
16:  end for
17: end while

```

An exact optimization of the recursive oracle circuit is a job-shop problem with MPT, or $J; m|p_{ij} = 1; \text{fix}_j|C_{\max}$ ⁴ following the widely adopted notation [8]. Brucker et al. [4] proves the strong NP-hardness of $J; 2|p_{ij} = 1; \text{fix}_j|C_{\max}$ which is a special case of 2 qubits in this context. This highlights the complexity of the exact optimization which we consider infeasible in practice. Instead of exact optimization, we use a greedy algorithm to rearrange and reduce the controlled NOT gates. There are two steps in our greedy algorithm: 1) eliminate gate pairs that are complex conjugates of each other; 2) rearrange gates such that they can be maximumly parallelized. In the first step, all commutable gates are sorted according to the qubit indices being acted on. Then for all these gates, if they could cancel with any other one, they should be neighboring to each other after sorting. Hence we go through the

⁴The notation $J; m|p_{ij} = 1; \text{fix}_j|C_{\max}$ describes a job-shop problem with m dedicated machines. Each multiprocessor task (job) requires a unit processing time simultaneously on all the machines it specifies. The optimization task is to minimize the makespan, i.e., the completion time of all tasks.

sorted gate list, check whether neighboring gates are complex conjugates of each other, and eliminate them. In the second step, we search for maximumly parallelizable gates. We start a gate list with a single gate. Then gradually add gates to the list that act on qubits different from all other gates in the list. If no more gates can be added to the list, we start a new gate list and repeat the process until all gates are added to one of the gate lists. If the number of product terms in each boolean equation is much more than the number of qubits, depth can be reduced by a factor of $\frac{1}{m}$ for m being the number of ancillae. Figure 11 gives an example of a quantum circuit before and after applying our greedy algorithm.

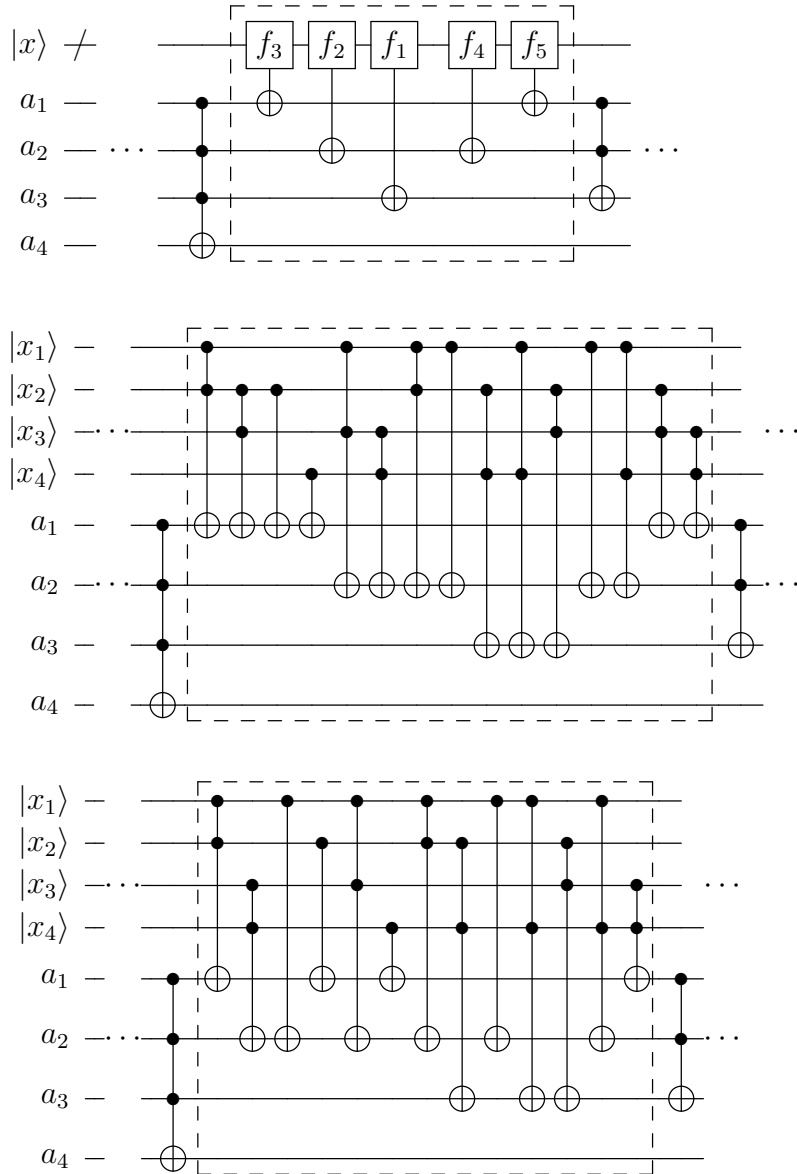


Figure 11: A fragment of oracle for level $\ell = 2$ with $m = 4$ ancilla qubits. The dashed boxes from top to bottom are function-controlled-gates, their expansion into CNOT and MCX gates, and gates after applying Algorithm 3. The original circuit has a depth of 12, whereas the optimized circuit has a depth of 8.

4 Randomized Grover's Algorithm

In this section, we propose the random version of Grover's Algorithm with the idea of randomly splitting boolean equations into groups. Then we analyze the number of iterations required for different schemes of splitting.

With an appropriate way of constructing the oracle, the vanilla Grover's algorithm is complete after K iterations. Recall K for the highest probability of obtaining a correct answer in one measurement is $K = \text{round} \left(\frac{\arccos \sqrt{M/N}}{\theta} \right)$. However, K could be very large in the case when $M \ll N$. As an example, for $M = 1$ and $N = 2^{20}, 2^{25}$, and 2^{30} , K are 568, 3217, and 12198, respectively. Combined with the complexity of each Grover iteration, e.g., the circuit depth of the oracle, the problem quickly becomes impractical.

Randomized Grover's algorithm. We propose a randomized Grover's algorithm that has different Grover operators G in each iteration. As discussed in Section 3, the number of ancilla qubits limits the number of equations it can solve. Splitting boolean equations into groups and using different groups of equations in different Grover iteration allows us to solve a boolean equation system with more equations in given limited quantum resources.

Assume we have R equations in total. Let all boolean equations be split into s groups, where groups are denoted as $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_s$. Each group contains only part of R equations and the union of $\{\mathcal{R}_i\}_{i=1}^s$ is all R equations, i.e., $\bigcup_{i=1}^s \mathcal{R}_i = \{1, 2, \dots, R\}$. The number of equations in \mathcal{R}_s is denoted as R_s . Note that groups can overlap with each other.

The randomized algorithm framework allows each Grover iteration to choose one of \mathcal{R}_i . Denote s_i as the group index for the oracle in the i -th iteration, i.e.,

$$O_i : |x\rangle \mapsto \begin{cases} -|x\rangle, & x \in \{y \mid f_t(y) = 0, \forall t \in \mathcal{R}_{s_i}\} \\ |x\rangle, & \text{otherwise} \end{cases}. \quad (14)$$

Instead of iterating the same oracle, oracles are now different and are labeled O_1, \dots, O_K for a total of K iterations. The corresponding operators are labeled G_1, \dots, G_K , respectively. The circuit before the measurement is

$$G_K G_{K-1} \cdots G_1 H^{\otimes n} |0\rangle = W O_K W O_{K-1} \cdots W O_1 |\psi\rangle, \quad (15)$$

where W is the diffusion operation. Figure 12 illustrates the procedure with measurement.

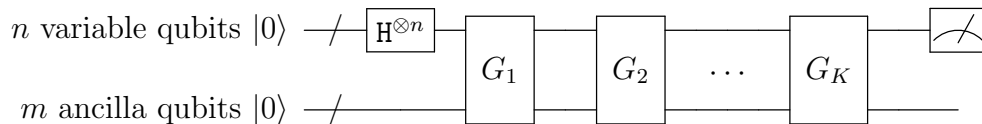


Figure 12: Randomized Grover's algorithm. Different operators G_i contain different oracles.

Many splitting strategies can be applied to the randomized framework. A fixed splitting strategy is to split these equations in the first place, $\bigcup_{i=1}^s \mathcal{R}_i = \{1, \dots, R\}$. Then we iterate using $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_s$ in a cyclic way until we reach the desired number of iterations. Another randomized splitting strategy is to first estimate the number of equations in each iteration,

e.g., q equations. Then, at each iteration, we randomly chose q distinct equations from the entire boolean equation system. Oracle circuits are then constructed just for the current iteration. It is worthwhile to point out that unless intended, each equation should have an equal probability of being chosen in each iteration. Otherwise, the algorithm will be biased to some equations.

The randomized Grover's algorithm is a natural random extension of the vanilla Grover's algorithm. The major reason behind proposing the randomized Grover's algorithm is to reduce the complexity of oracle circuits, i.e., to reduce the depth and the number of ancilla qubits in the oracle quantum circuits. However, the randomized idea does not always work for all boolean equation systems with arbitrary grouping. Here we give a tiny counterexample illustrating the failure of the randomized Grover's algorithm.

Consider a 2-variable boolean equation system with two equations,

$$\begin{aligned} f_1(x_1, x_2) &= x_1 \oplus 1 = 0, & \text{and} \\ f_2(x_1, x_2) &= x_2 \oplus 1 = 0. \end{aligned} \tag{16}$$

Assume the randomized Grover's algorithm splits two boolean equations into two groups and iterates between two groups in a cyclic way. More precisely, the algorithm construct oracles with $f_1(x_1, x_2) = 0$ in G_1, G_3, \dots , and $f_2(x_1, x_2) = 0$ in G_2, G_4, \dots . The state vector before each iteration obeys

$$\frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \xrightarrow{G_1} \frac{1}{2} \begin{pmatrix} -1 \\ -1 \\ 1 \\ 1 \end{pmatrix} \xrightarrow{G_2} \frac{1}{2} \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \xrightarrow{G_3} \frac{1}{2} \begin{pmatrix} -1 \\ 1 \\ -1 \\ 1 \end{pmatrix} \xrightarrow{G_4} \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \xrightarrow{G_5} \frac{1}{2} \begin{pmatrix} -1 \\ -1 \\ 1 \\ 1 \end{pmatrix} \xrightarrow{G_6} \dots$$

As shown in the above iterations, the state vector iterates cyclicly among four vectors, and the amplitude of the solution entry does not change at all. A quantum measurement after any number of iterations will give one of all possible states with equal probability. Hence the randomized Grover's algorithm fails in this case.

For the boolean equation system as in (1), we empirically find that we should not split equations into too many groups with few equations. For a group with few equations, the solution set would be much larger than the solution set of the boolean equation system. Such a splitting would make randomized Grover's algorithm challenging to succeed. A rigorous mathematical analysis for the success of our randomized Grover's algorithm would be interesting for future work.

Estimating iteration numbers. Both Grover's algorithm and our randomized Grover's algorithm cannot iterate forever and stop until they meet some convergence criteria. The randomized Grover's algorithm, similar to the vanilla Grover's algorithm, adopts a fixed number of iterations, whose expression differs from (7). The iteration number can be estimated directly by running the Grover operator and computing the rotation angle of a single iteration, see [2]. In this part, we discuss the estimation of the iteration number for randomized Grover's algorithm leveraging the expectation of randomness, which reduces the computational cost to the evolvment of a 2×2 matrix.

In quantum computing, almost all algorithms bear with randomness. The measurement results are not deterministic. Desired solutions appear in the measurement results with a relatively higher probability. Hence, quantum algorithms, including Grover's algorithm and randomized Grover's algorithm, have to be executed and measured many times until the desired solutions appear once we can validate the solution efficiently. In the case the solution can not be validated efficiently, we have to repeat the execution and measurements many times and confirm the solution through some form of majority voting. In solving nonlinear boolean equations, we are able to validate the solution efficiently in polynomial time complexity. Randomized Grover's algorithm (also Grover's algorithm) further has a trade-off between the iteration number K and the measurement number J . The overall solving time for randomized Grover's algorithm in addressing the boolean equation system (1) is considered as $J \cdot K$. Hence we propose the following constraint optimization problem to obtain the best iteration number K and measurement number J ,

$$\min_{P(J,K) > 1-\varepsilon} J \cdot K, \quad (17)$$

where $P(J, K)$ denotes the success probability of the boolean function solver and ε is a small failure probability of the algorithm.

In general, $P(J, K)$ increases monotonically with respect to both J and K for K smaller than the near-optimal number similar to (7). However, an explicit expression for $P(J, K)$ is unknown for randomized Grover's algorithm. Therefore, solving (17) analytically and exactly becomes infeasible in practice. In the following, we will first propose a simplified probabilistic model for applying randomized Grover's algorithm to solve boolean equation system, and then numerically estimate J and K as an approximated solution of (17).

Recall the n -variable boolean equation system (1) has R boolean equations. Let the number of solutions be M . Without loss of generality, we assume solutions are the first M states, i.e., $|0\rangle, |1\rangle, \dots, |M-1\rangle$. In randomized Grover's algorithm, each group of equations is randomly selected from all R equations, whose solutions contain that of the original boolean equation system. Each oracle O_i could be represented by a $N \times N$ matrix,

$$O_i = \text{diag}\left(\underbrace{-1, \dots, -1}_M, \underbrace{V_1^{(i)}, \dots, V_{N-M}^{(i)}}_{N-M}\right) \quad (18)$$

where $N = 2^n$, and $V_j^{(i)}$ are ± 1 depending on the selected boolean equations in O_i . After K iterations, the quantum state vector of the randomized Grover's algorithm is

$$G_K \cdots G_1 |\psi\rangle = W O_K \cdots W O_1 |\psi\rangle,$$

and the probability of obtaining the i -th correct solution is

$$p_i = |\langle i | G_K \cdots G_1 |\psi\rangle|^2 \quad (19)$$

for $i = 0, 1, \dots, M-1$. Once p_i s are known, the success probability of the boolean function solver $P(J, K)$ can be written down explicitly in terms of p_i s, i.e.,

$$P(J, K) = 1 - (1 - p)^J, \quad \text{where } p = \sum_{i=0}^{M-1} p_i. \quad (20)$$

Note that this requires in the worst case, one has to examine all the J outcomes of the process to find if any of the outcomes represents a correct solution. Checking whether an x satisfies all R equations is of polynomial complexity. By (20), the constraint of optimization (17) is $p > 1 - \varepsilon^{1/J}$.

Now we make two assumptions to simplify the calculation of p_i s and hence $P(J, K)$. First, we assume that all equation groups have exactly \widetilde{M} solutions for $\widetilde{M} > M$. Second, we assume that $V_j^{(i)}$ s are identically independently distributed random variables for all i and j . Those assumptions mean that the probability of every $V_j^{(i)}$ admits

$$\Pr(V = 1) = \frac{N - \widetilde{M}}{N - M}, \quad \Pr(V = -1) = \frac{\widetilde{M} - M}{N - M}.$$

With these two assumptions, we could calculate the expected value of G_i , in the matrix format,

$$\begin{aligned} \mathbb{E}(G_i) = W\mathbb{E}(O_i) &= \frac{2}{N} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \begin{pmatrix} -1 & \cdots & -1 & \frac{N+M-2\widetilde{M}}{N-M} & \cdots & \frac{N+M-2\widetilde{M}}{N-M} \end{pmatrix} \\ &- \text{diag} \left(-1, \dots, -1, \frac{N+M-2\widetilde{M}}{N-M}, \dots, \frac{N+M-2\widetilde{M}}{N-M} \right), \quad \text{for } i = 1, \dots, K, \end{aligned} \quad (21)$$

and the expected quantum state vector is

$$\mathbb{E}(G_K \cdots G_1 |\psi\rangle) = (W\mathbb{E}(O_1))^K |\psi\rangle. \quad (22)$$

Notice that the right-hand side of (21) is independent of i and $\mathbb{E}(WO_i)|x\rangle$ can be represented only by two different numbers. Thus, we could equivalently use a 2-dimensional matrix-vector multiplication scheme to calculate the expected quantum state vector i.e.,

$$|\psi_2^{(i)}\rangle = (G_{2 \times 2})^i |\psi_2^{(0)}\rangle,$$

where $|\psi_2^{(i)}\rangle$ denotes the 2-dimensional expected quantum state vector in i -th iteration, $G_{2 \times 2} = W_{2 \times 2} O_{2 \times 2}$ denotes the 2-dimensional Grover operator and

$$|\psi_2^{(0)}\rangle = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad O_{2 \times 2} = \begin{pmatrix} -1 & \\ & \frac{N+M-2\widetilde{M}}{N-M} \end{pmatrix}, \quad W_{2 \times 2} = \frac{2}{N} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} M & N-M \end{pmatrix} - I_{2 \times 2}.$$

Therefore, the computed probability of obtaining each correct solution is the square of the first element in $|\psi_2^{(K)}\rangle$, $p_i = \left| \langle 0 | \psi_2^{(K)} \rangle \right|^2$, for all $i = 0, 1, \dots, M-1$. The probability of obtaining any correct solution is $p = M \cdot p_i = M \cdot \left| \langle 0 | \psi_2^{(K)} \rangle \right|^2$. The optimization problem (17) is thus

$$\min_{M \cdot \left| \langle 0 | \psi_2^{(K)} \rangle \right|^2 > 1 - \varepsilon^{1/J}} J \cdot K. \quad (23)$$

Given (23), we could conduct a brute force search in the space of J and K for small-to-medium size boolean equation systems and find the near-optimal J and K , which is already useful in NISQ.

Through the above analysis, the assumption that \widetilde{M} remains constant for all oracles is too strong to be true in practice. Further, the number of solutions in oracles is also not known a priori. If an efficient estimator or quantum algorithm in estimating \widetilde{M} is available, we could update our analysis above to be oracle dependent. Otherwise, we have to adopt a rough estimation of \widetilde{M} in practice. For all numerical experiments, we estimate $\widetilde{M} = M \cdot 2^{n-r}$, where r is the number of equations used in each iteration. From our testing, the success of the randomized Grover’s algorithm is related to the estimation of \widetilde{M} but is not very sensitive.

5 Numerical Results

We focus on solving boolean equation systems with only boolean quadratic equations (BQE) in this section. Each BQE is randomly generated in advance, by uniformly selecting from all possible quadratic product terms and linear terms. The number of total terms in BQE is set to a Poisson random variable with the mean $\frac{1}{4}n(n+1)$ which equals half of all available quadratic product terms. The solutions are calculated by brute force for testing purposes. For some tests, we fixed the number of solutions to a specific number or a range. If the number of solutions of the randomly generated BQE system does not match the requirement, we add a new BQE or drop an existing one until the requirement is satisfied. We implement the randomized Grover’s algorithm in Section 4 and run the algorithm in the state vector simulator using the state vector simulation. Limited to the available resource, the maximum number of variables we can simulate is set to $n = 25$ in this section. In all numerical results, programs are written using Qiskit 0.39.3 [19] with Python 3.8 and executed on a Linux server with dual Intel Xeon Gold 6226R (2.90GHZ, 2×16 Cores, 2×32 Threads) and 1 TB memory.

5.1 Efficient Oracle Construction

Capacitance and Recursive Level. We explore the trade-off between the recursion level ℓ and the number of ancilla qubits m in the oracle construction. A higher recursion level will reduce the number of ancillae used in the circuit at the cost of a deeper circuit. As in the previous sections, the number of gates will roughly double when we increase the recursion level ℓ by one.

We run the oracle circuit generation with recursion levels set to $\ell = 1, 2, 3, 4$ on various randomly generated single-solution equations with the number of variables between 10 and 25. The number of ancillae m used in the circuit is computed by counting the minimum ancillae required to contain half of the total number of equations, i.e., $F_m^{(\ell)} \geq R/2$. For each pair of a variable number and an ℓ , we adopt 15 randomly generated BQEs to get rid of the randomness in the equation system generation. The 95% confidence intervals in our results are calculated based on the 15 sample cases. To further explore the power of our recursive oracle construction, we also include the number of ancillae when the boolean equation systems have more than 25 variables. In these cases, the underlying quantum circuits are not explicitly

constructed.

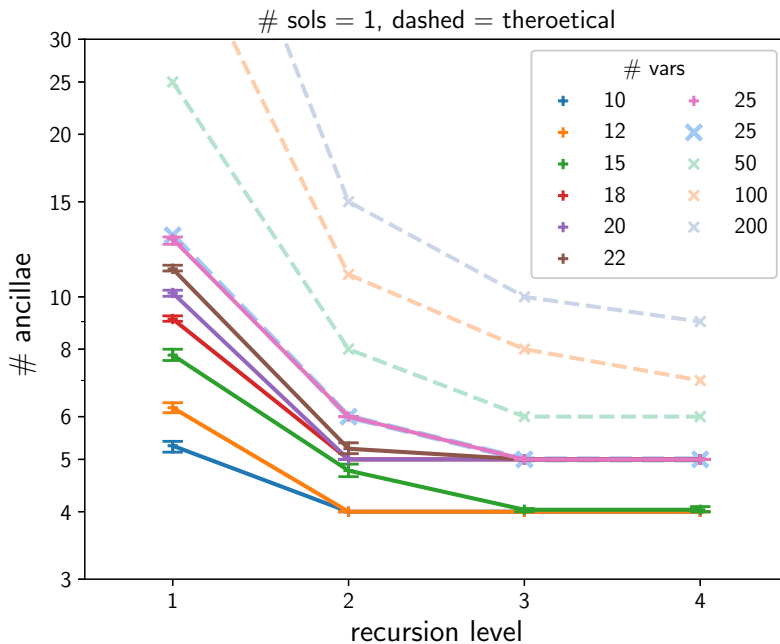


Figure 13: Relation between the recursive level and the required numbers of ancillae. The solid lines with ‘+’ marks are experiments on randomly generated single-solution BQEs. The 95% confidence intervals are plotted. The dashed lines with ‘x’ marks are theoretical results. Note that for $n = 25$, both experimental and theoretical results are plotted and they overlap with each other.

Figure 13 plots the required number of ancillae for the oracles with different recursion levels. Solid lines correspond to cases with actual quantum circuit constructions in practice whereas dash lines correspond to our theoretical estimations. The solid lines are plotted with confidence intervals. The variance of ancillae usage comes from the variance of the number of equations, and is arguably small, indicating that the scaling will not change much for different BQEs. For $n = 25$, we plot both the experimental and theoretical results. From the figure, these two results agree with each other. As illustrated in Figure 13, increasing the recursion level will significantly reduce the number of required ancillae, especially when the number of variables is large. For BQEs with a small-to-medium number of variables, increasing the recursion level from 1 to 2 drastically decreases the number of required ancillae. While further increasing ℓ to 3 or above has little impact on decreasing the number of required ancillae. Hence, for BQEs with a small-to-medium number of variables, we suggest using $\ell = 2$, whereas for BQEs with a large number of variables, we suggest using the theoretical result to find a proper ℓ such that the required number of ancillae is feasible on quantum computers.

Oracle compression. We investigate the circuit depth before and after our greedy oracle compression, Algorithm 3. We test 15 single-solution BQEs for each parameter setup, where the BQEs are generated as described at the beginning of this section.

n	m	Level ℓ	# Eq. per iter	Depth of one iteration		
				w/o compression	w/ compression	ratio
15	7	1	7	839	149	-82.24%
15	4	2	7	1459	802	-44.98%
15	5	2	7	2373	1062	-55.22%
20	4	2	7	2580	1382	-46.44%
20	5	2	11	4198	1818	-56.69%
20	5	3	15	8880	5333	-39.94%
25	5	2	11	6520	2788	-57.25%
25	6	2	16	9931	3527	-64.49%
25	5	3	15	13904	8258	-40.61%

Table 2: Depth of a Grover iteration with and without greedy oracle compression. Various sizes of BQEs are tested and results are listed for comparison. For each circuit, the depth of one iteration is the average circuit depth from the whole circuit. The depth of a circuit is counted based on the predefined simple gates in qiskit. The displayed figures in the table are further averaged among 15 distinct BQEs (15 distinct circuits).

Table 2 presents the averaged circuit depth with and without our greedy oracle compression. First of all, for all cases in Table 2, our greedy oracle compression reduces the circuit depth by at least $\sim 40\%$. In most cases, the circuit depths are reduced by half. From the 6th and last row of Table 2, where the recursion level is relatively high $\ell = 3$, we find that their compression ratios are low. As the circuit contains more and more MCX gates on the same set of qubits and ancilla qubits, the number of quantum gates that could be commuted and parallelized are reduced. Hence the compression ratio is low in these cases. Therefore, empirically, we observe that our greedy oracle compression technique is more powerful when the recursion level ℓ is relatively small. For BQEs with a small-to-medium number of variables, we again suggest using $\ell = 2$.

5.2 Randomized Grover’s Algorithm

Total circuit depth. Splitting equations into various number of groups in randomized Grover’s algorithm will affect both the total depth of the circuit and the number of ancilla qubits required. In this part, we will explore the trade-off between the number of groups and the total depth.

In this experiment, we compare the depth with different splitting strategies for BQEs with 15 and 20 variables. We randomly generate a sequence of BQEs with the solution number smaller than 90 for both $n = 15, 20$. We group these BQEs into five based on their solution numbers, namely, single-solution, 2-4 solutions, 5-9 solutions, 10-19 solutions, and 20-89 solutions. Each group has 40-100 BQEs. For each BQE and a split factor, the quantum circuit is constructed with recursion level $\ell = 2$ and Grover iteration number achieving a nominal success rate of 99.9% in 1024 shots. The split factor is defined as the total number of equations over the number of equations chosen per Grover iteration. A higher split factor means each iteration contains fewer equations. The nominal success rate is the one estimated

by our computing model in Section 4, which is different from the actual success rate.

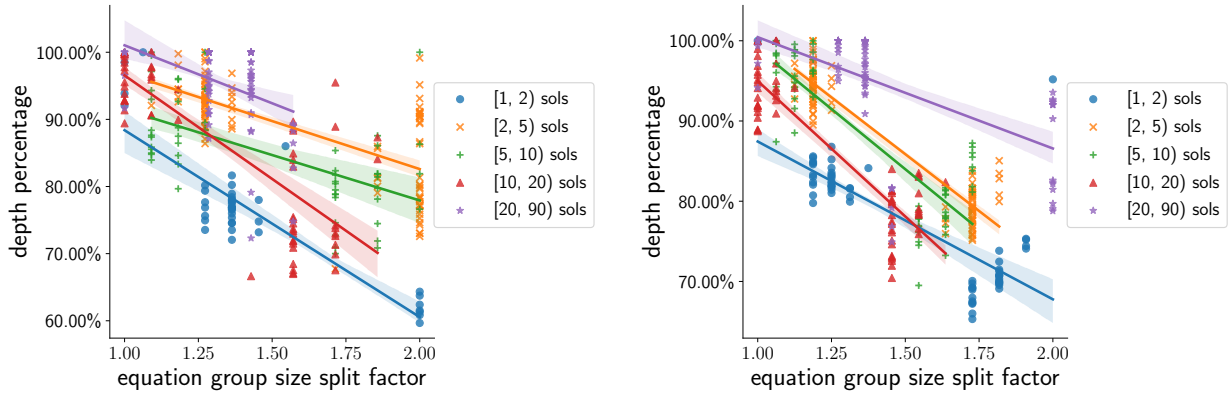


Figure 14: Relation between splitting and the total depth. The left figure plots 15 variables BQEs and the right plots 20 variable BQEs. For both figures, the x -axis is the split factor and the y -axis shows the relative depth in each group, so the longest circuit in each group is normalized as 100%, and others are the relative depth to the longest one. The regression line for each group is plotted with the 95% confidence interval.

Figure 14 illustrates the relative depth of the circuit required to obtain a 99.9% success rate in 1024 shots. The relative depth is defined as the depth of the circuit over the maximum depth within the group. Each BQE with a split factor contributes to a point in Figure 14. For a better understanding of the data, we conduct a linear regression for each BQE group and plot the regression lines together with their 95% confidence interval. The regression line of the 15-variable 20-to-90-solution group is shorter since the number of equations in each iteration is already small.

For both figures in Figure 14, all slopes of the regression lines are negative. Hence, the total depth of the quantum circuit is reduced as the proportion of equations used in each iteration decreases. Here, we only explore the split factor between 1.0 and 2.0. For a small-to-medium BQE, a large split factor is not helpful. In general, the circuit depth will first decrease and then increase as the split factor increases. When the split factor is relatively close to 1, the splitting strategy effectively reduces the total circuit depth and our computing model in Section 4 is trustable. However, when the split factor is relatively large, each split group has too few equations and yields a vast solution space. In this scenario, the number of Grover iterations must be large and the total circuit depth is also large. Empirically, we find that a split factor around 2 is efficient for BQEs with a small-to-medium number of variables.

Success rate. On some very small-scale problems, the failure rate of randomized Grover’s algorithm will be abnormally high, due to the inaccurate estimate of \widetilde{M} . For these small-scale problems, the original solution space is large enough to converge in a very small amount of iterations, making saving resources less important. The variance in estimating the iterations dominates the actual value, leading to a very inaccurate outcome. We aggregate 15 randomly generated circuits from 10 to 18 variables and summarize the observed success rate for different iteration numbers in Figure 15, where each experiment sets the nominal success rate to 80.0%. Firstly, the success rate for most groups has reached the nominal success rate,

especially for all groups with 256 shots. The average success probability is 92.4%, which is higher than the nominal success rate. Secondly, due to the limit of accurately estimating the number of solutions in each iteration of Grover’s algorithm, we observed that the success rate is lower than the nominal success rate for some cases. Given the number of shots, this discrepancy becomes more prominent as the number of variables increases from 10 to 18. This is because, for a fixed number of shots, a higher number of Grover iterations is required with sparser solutions, which increases the likelihood of overshooting or undershooting the desired result. It is better to increase the number of shots, rather than the number of iterations, to obtain a higher success rate. Finally, note that some circuits cannot reach a nominal success rate of 80% if the number of shots is set to less than 16. This is caused by the success rate of a single shot cannot be arbitrary large, and the fail rate of a single shot to the power of the number of shots will not fall below 20%. In this case, one has to increase the number of shots to obtain a higher success rate.

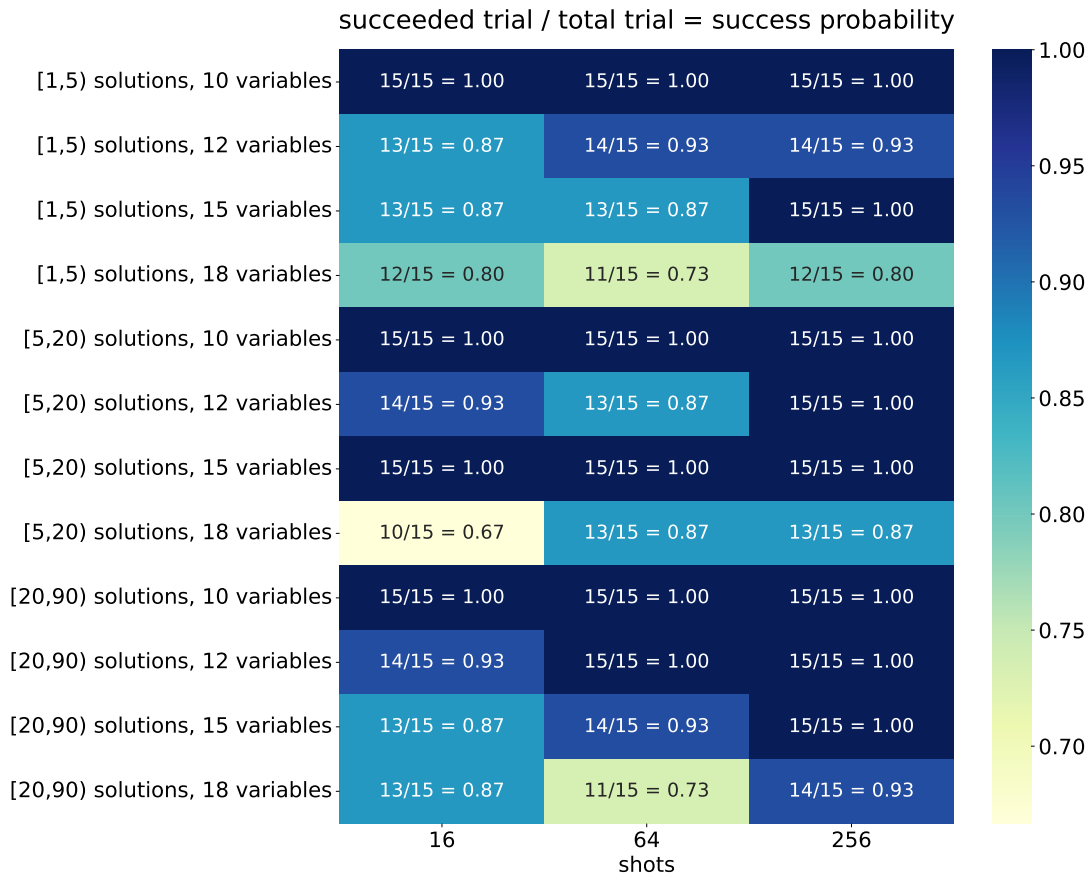


Figure 15: The success rate of each circuit running several numbers of shots. The x -axis shows the number of shots we ran for each circuit. The y -axis contains the number of variables and the number of solutions for each circuit. In each cell of the heatmap, we annotate the number of succeeded circuits, the number of total circuits we ran, and the success rate. The color bar plots the value of the success rate. Due to the constraint of computational resources, we can only include test cases with no more than 18 variables.

6 Conclusion and Future Work

We proposed three novel techniques to efficiently solve nonlinear boolean equations on quantum computers under Grover’s algorithm framework. Three techniques are W-cycle oracle construction, oracle compression, and randomized Grover’s algorithm. For the first technique, W-cycle oracle construction improves the capacity in encoding the boolean equations into the quantum circuit given a fixed number of ancilla qubits. W-cycle oracle construction introduces a recursive circuit construction idea to maximally reuse ancilla qubits to keep solution information. Given m ancilla qubits, vanilla quantum circuit construction could encode m boolean equations, whereas our W-cycle oracle construction could at most encode 2^m boolean equations at the cost of deeper circuits. The W-cycle oracle construction also introduces a way to conduct the trade-off between the number of ancilla qubits and the circuit depth. The flexible trade-off is very important in the NISQ era. The second technique, oracle compression, adopts a greedy strategy to swap commutable quantum gates. Oracle compression eliminates redundant quantum gate pairs and rearranges the quantum gates in a parallelizable way. Numerical experiments show that the oracle compression technique leads the quantum circuit depth saving by a factor between 40% to 80%. The saving rate, in general, is larger when the recursive level ℓ is small in the W-cycle oracle construction. The third technique, randomized Grover’s algorithm, uses random combinations of boolean equations in each iteration to construct the oracle and reduces the oracle circuit depth. In each iteration, the algorithm chooses parts of the boolean equation system. The solution set of the chosen equations contains that of the original system. Hence, through the randomized Grover iteration, amplitudes of the original solutions are always amplified. While the amplitudes of those virtual solutions (solutions of chosen equations but not the original system) are amplified in some iterations and damped in other iterations. However, the algorithm is not guaranteed to converge. To improve the convergence, we empirically make the number of equations per iteration relatively large. An estimation of the number of iterations for randomized Grover’s algorithm is also proposed. The estimation analysis is carried out under the assumption that the number of solutions in each iteration is a constant. Numerically, we find that our randomized Grover’s algorithm is efficient and the estimation of iteration number is useful.

There are a few interesting future directions. The most interesting one would be extending the randomized Grover’s algorithm to a wider range of applications. The idea behind randomized Grover’s algorithm is to randomly relax the constraints on the solutions so that the quantum circuit for the oracle could be significantly simplified. Such an idea would be useful not only in NISQ but also in beyond-NISQ era. The rigorous analysis for randomized Grover’s algorithm would be another interesting future direction. A careful and rigorous analysis would hint at the choices for selecting boolean equations in each iteration and also lead to a more accurate estimation of the iteration number. Other interesting future directions include extending the W-cycle circuit construction idea to other applications and exploring other optimizing techniques like multiplying some equations into one.

References

- [1] F. ARUTE, K. ARYA, R. BABBUSH, D. BACON, J. C. BARDIN, R. BARENDTS, R. BISWAS, S. BOIXO, F. G. BRANDAO, D. A. BUELL, ET AL., *Quantum supremacy using a programmable superconducting processor*, Nature, 574 (2019), pp. 505–510.
- [2] J. BALEWSKI, D. CAMPS, K. KLYMKO, AND A. TRITT, *Efficient quantum counting and quantum content-addressable memory for dna similarity*, 2023.
- [3] W. P. BARITOMPA, D. W. BULGER, AND G. R. WOOD, *Grover’s quantum algorithm applied to global optimization*, SIAM Journal on Optimization, 15 (2005), pp. 1170–1184.
- [4] P. BRUCKER AND A. KRÄMER, *Shop scheduling problems with multiprocessor tasks on dedicated processors*, Annals of Operations Research, 57 (1995), pp. 13–27.
- [5] Y.-A. CHEN AND X.-S. GAO, *Quantum algorithm for boolean equation solving and quantum algebraic attack on cryptosystems*, Journal of Systems Science and Complexity, (2022), pp. 1–40.
- [6] M. FÜRER, *Solving np-complete problems with quantum search*, in Latin American Symposium on Theoretical Informatics, Springer, 2008, pp. 784–792.
- [7] J. D. GOLIC, *Fast low order approximation of cryptographic functions*, in International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 1996, pp. 268–282.
- [8] R. L. GRAHAM, E. L. LAWLER, J. K. LENSTRA, AND A. H. G. R. KAN, *Optimization and approximation in deterministic sequencing and scheduling: A survey*, in Annals of discrete mathematics, vol. 5, Elsevier, 1979, pp. 287–326.
- [9] M. GRASSL, B. LANGENBERG, M. ROETTELER, AND R. STEINWANDT, *Applying grover’s algorithm to aes: quantum resource estimates*, in International Workshop on Post-Quantum Cryptography, Springer, 2016, pp. 29–43.
- [10] L. K. GROVER, *A fast quantum mechanical algorithm for database search*, in Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, 1996, pp. 212–219.
- [11] N. KOBLITZ, *Elliptic curve cryptosystems*, Mathematics of Computation, 48 (1987), pp. 203–209.
- [12] M. MATSUI, *Linear cryptanalysis method for des cipher*, in Workshop on the Theory and Application of of Cryptographic Techniques, Springer, 1993, pp. 386–397.
- [13] E. J. MCCLUSKEY, *Minimization of boolean functions*, The Bell System Technical Journal, 35 (1956), pp. 1417–1444.
- [14] G. D. MICHELI, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill Higher Education, 1994.

- [15] W. MILLAN, *Low order approximation of cipher functions*, in International Conference on Cryptography: Policy and Algorithms, Springer, 1995, pp. 144–155.
- [16] M. A. NIELSEN AND I. L. CHUANG, *Quantum computation and quantum information*, Cambridge university press, 2010.
- [17] R. L. RIVEST, A. SHAMIR, AND L. ADLEMAN, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, 21 (1978), pp. 120–126.
- [18] P. W. SHOR, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM review, 41 (1999), pp. 303–332.
- [19] M. TREINISH, J. GAMBETTA, P. NATION, QISKIT BOT, P. KASSEBAUM, D. M. RODRÍGUEZ, S. DE LA PUENTE GONZÁLEZ, S. HU, J. LISHMAN, K. KRSULICH, J. GARRISON, L. BELLO, J. YU, M. MARQUES, J. GACON, D. MCKAY, J. GOMEZ, L. CAPELLUTO, TRAVIS-S-IBM, A. PANIGRAHI, LERONGIL, R. I. RAHMAN, S. WOOD, T. ITOKO, A. MITCHELL, A. POZAS-KERSTJENS, C. J. WOOD, D. SINGH, D. RISINGER, AND E. ARBEL, *Qiskit/qiskit: Qiskit 0.39.3*, 2022.
- [20] Y. WU, W.-S. BAO, S. CAO, F. CHEN, M.-C. CHEN, X. CHEN, T.-H. CHUNG, H. DENG, Y. DU, D. FAN, ET AL., *Strong quantum computational advantage using a superconducting quantum processor*, Physical Review Letters, 127 (2021), p. 180501.