8-9-2021

Quantum Grover's Oracles with Symmetry Boolean Functions

Peng Gao
Portland State University

Quantum Grover's Oracles with Symmetry Boolean Functions

by

Peng Gao

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Electrical and Computer Engineering

Dissertation Committee:
Xiaoyu Song, Chair
Marek Perkowski
Fu Li
Jingke Li

Portland State University
2021

Abstract

Quantum computing has become an important research field of computer science and engineering. Among many quantum algorithms, Grover's algorithm is one of the most famous ones. Designing an effective quantum oracle poses a challenging conundrum in circuit and system-level design for practical application realization of Grover's algorithm.

In this dissertation, we present a new method to build quantum oracles for Grover's algorithm to solve graph theory problems. We explore generalized Boolean symmetric functions with lattice diagrams to develop a low quantum cost and area efficient quantum oracle. We study two graph theory problems: cycle detection of undirected graphs and generalized hypercube partitioning. We present a novel method to design a quantum oracle to solve Boolean function minimization problems which occur in classical circuit optimization.

Acknowledgments

This dissertation would not have been accomplished without the countless support I received during my study, to which I express my sincere gratitude and appreciation.

First and foremost, I would like to thank my advisor Prof. Song for his engorgement and exceptional advice, throughout my doctoral program. His research and working attitude are a great standard for me to learn.

I would also like to wholeheartedly thank Prof. Perkowski for his enormous help on my career. The weekly meetings with him are greatly helpful for my career, these meetings introduced me to the research area of quantum computing.

I would like to express my sincerest gratitude to my advisory committee. I am grateful for their sacrifice of valuable time.

My life in Portland would not have been so enjoyable without friends and colleagues: Yiwei Li, Huajie Yang, Bin Lin, Quchun Yu, Xiao Li, Paul Canarsky, Yingcong Li, Nelson Tan, Amer Aimen, Thinh Nguyen.

Last but not the least, this dissertation is dedicated to my family. I would like to thank my parents and other relatives for their endless support and encouragement.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Overview

At a conference on physics and computation at the Massachusetts Institute of Technology (MIT) in 1981, Richard Feynman, one of the greatest physicists of his time, asked the question, "*Can we simulate physics on a computer? The answer is no, at least not all the physics, but one of its branches called quantum mechanics*." Studies on the laws of nature at the atom and particle level have been of interest in the field of quantum mechanics. If we tried to simulate this on a standard computer, then we would end up with the problem of having to deal with too many variables that the computer would not be able to handle. For instance, if a particle is described by two variables, then for n particles, we would need 2^n variables. Therefore, if we have 1,000 particles, then we would need $2^{1,000}$ variables, and the computers that we have at present will seldom, if ever, have enough memory to store such large quantities of values. Hence, instead of trying to simulate quantum mechanics on a computer, our goal was changed to building a quantum mechanical computer that would be dramatically better than ordinary computers.

The question of how to practically design a quantum computer was answered for the first time with the prototype for a quantum computer demonstrated on February 13, 2007, by D-Wave Systems, Inc. at the Computer History Museum in Mountain View, California. It

consisted of a 16-qubit quantum annealing processor. On May 11, 2011, D-Wave announced D-Wave One, "The world's 1st commercially-available quantum computer" operating on a 128-qubit chipset via quantum annealing to solve optimization problems, and on August 20, 2015, they also announced the general availability of a 1,000+ qubit quantum computer called the D-Wave 2X system, which was designed to handle complex problems.

There are several technologies and types of quantum computers being developed. Technology giants like Google, Microsoft, Intel, and IBM are coming together in quantum computer competitions in different areas, such as quantum design automation, quantum simulators, quantum algorithms, quantum chip design, and quantum cloud computing.

There are three concepts in the IBM 2020 quantum roadmap: kernels, algorithms, models. The kernel developers are focusing on creating high-performance quantum circuits. Algorithm developers rely on these circuits to develop groundbreaking quantum algorithms that might provide an advantage over current classical computers. Model developers apply these algorithms to solve real-world problems in chemistry, physics, biology, machine learning, and optimization by creating different quantum models.

In this dissertation, we propose a quantum algorithm design methodology with which to solve graph-theory-related problems in which we make use of the symmetry of Boolean functions. Our methodology is based on lattice diagrams and the Grover algorithm. We transferred the graph problems to constraint satisfaction problem that uses Boolean symmetry. We took the area-efficiency advantage of the lattice diagram into account to create our quantum oracles with reduced complexity and, thus, reduced quantum-realization costs.

1.2 Research Goals

When researchers use Grover's algorithm to solve practical problems, they face several issues when building an efficient oracle. These issues can potentially influence quantum computation performance.

1. The quantum gates are used in the oracle. How is the use of certain gates reflected in the total quantum cost the simulation time of a quantum oracle? How will the synthesis methods for binary quantum gates affect the number of ancilla qubits, the performance of simulation, and the costs of the circuit of the oracle and the entire Grover algorithm?
2. How can the quantum oracle be synthesized with fewer ancilla qubits? How does the cost of ancilla qubits affect the performance of the quantum oracle?

3. Normally quantum oracles are synthesized with Feynman, swap, inverter, and Toffoli gates. However, compared to the classical switching circuit, quantum mechanics have flexibility in creating the base gates. Can we find better quantum gates with which to synthesize an oracle?
4. Because it is limited to the number of qubits a quantum computer can operate, a solution based on a pure quantum algorithm is still not applicable using current technology. A hybrid classical/quantum algorithm is the most promising method to use to solve many problems. What improvements can be achieved by using such a combination?

I focus on creating an efficient quantum oracle for Grover's algorithm to solve constraint satisfiability problems (CSPs) and other problems that can be transformed into CSPs. My methodology for building a quantum oracle is based on symmetrical Boolean functions and lattice diagrams and aims to reduce quantum computing costs and improve efficiency by using lattice diagrams to implement the oracle in the form of a symmetrical Boolean function. I proposed transforming non-symmetric functions into symmetric functions by using the reminder function. My contributions also include building several quantum oracles and hybrid classical/quantum algorithms with which to solve practical problems, such as the Hamiltonian cycle detection problem, how to minimize the switching function, and how to solve some logic puzzles.

1.3 Related works

There are four categories of problems discussed in my dissertation: Cycle detection and hypercube partitioning problems, Switching function minimization problems, Maximum Independent Set problems (MISp), CSPs.

Problems 1 and 3 are classical graph-theory-related problems, and there are many related papers [58–77] on this problem in quantum computing as well. Problem 2 is an electronic design automation (EDA) problem. It has been well studied in classical computing but has not attract enough attention in quantum computing yet. The river crossing puzzle is categorized into Problem 4. Few researchers [78–80] have studied this kind of problem in quantum computing.

Graph-theory-related problems have been well-studied from a variety of aspects. Most of the related papers involve modeling an arbitrary graph with the adjacent matrix model or the adjacency list model based on Christoph's paper [58, 59]. In that paper [58], Christoph et al. studied the quantum query complexity of these two graph models with the minimum spanning tree problem. In their next paper [59], they extended the idea using Grover's algorithm and presented a quantum exponential searching algorithm based on Grover's algorithm. The ideas in these two papers inspired many subsequent works in quantum

computing. There are many papers on the cycle detection problem and Hamiltonian cycle detection problem [60–65]. Most of them involve the use of Grover's algorithm. The papers [60, 61] present an overview of how to solve the Hamiltonian cycle problem in quantum computing. Vidya [60] briefly introduced the Hamiltonian cycle problem and a possible way to speed up Grover's algorithm for this problem. Rudolph [61] presented an adjacent matrix model with a quantum circuit that can be used as the oracle function in Grover's algorithm. Similar to Rudolph's idea, Burger [62] presented a detailed quantum circuit of the adjacent matrix model to solve the Hamiltonian cycle detection problem. The authors of both of these papers used Grover's algorithm to check the connections of all vertices in the adjacent matrix model to determine whether a cycle exists in the graph. Da-Jian [63] introduced a quantum algorithm on an adiabatic quantum processor. This algorithm takes advantage of Grover's algorithm in brutal searching. Similarly, Yimin's algorithm [64] is a Grover-based algorithm, they present a hybrid framework to solve the Hamiltonian cycle problem. The difference between Yimin's method and Da-jian's method is in the graph modeling and encoding part. Anuradha [65] presented a quantum algorithm with which to solve the Hamiltonian cycle problem based on quadratic unconstrained binary optimization (QUBO). This method is similar to that of a Grover-based algorithm but with a different problem formulation. In QUBO, the problem needs to be formulated into a group of linear equations while Grover's oracle can be expressed using a Boolean function.

The traveling salesman problem (TSP) can be transformed into a Hamiltonian cycle detection problem, and there are also some papers [66–70] on this topic. Eppstein [66] conducted a scientific analysis of the classical algorithm to solve the Hamiltonian cycle and TSP problems. Based on Eppstein's paper, Mingyu introduced his quantum algorithms [67]. Minyu et al. presented an exponential quantum algorithm using branch searching to solve the TSP problem in a degree-3 graph. In their next paper [68], they extended their algorithm to a degree-4 graph. Dominic [69] used a similar technique to improve Xiao's algorithm with higher-degree graphs. Karthik [70] showed a gate-level implementation of Grover's algorithm to solve the TSP on IBM's quantum simulator. Karthik also modeled the distance matrix of a four-city TSP and got the result from IBM's quantum simulator.

Another problem discussed in my dissertation is the problem of hypercube partitioning. Hayato [71] and Eugenio [72] proposed different graph partitioning algorithms for a quantum annealing system. Hayato's provided a generalized graph partitioning algorithm using QUBO. His work was inspired by biology and chemical applications. Eugenio focused on partitioning a graph into Hamiltonian subgraphs. With the power of D-Wave, the maximum size of a graph that can be processed is 4,000 vertices with 5,200 edges.

The maximum independent set problem (MIS) and its related problem, the maximum clique (MC) problem, are famous graph problems, there are many papers [73–77] that use these problems as an example to prove the advantages of quantum computing. Alan's paper [73] contains an overview of how to use Grover's algorithm to solve MC problems. The author of this paper explained Grover's algorithm and its application to MC problems. Arpita's paper [74] contains a circuit-level solution to problems. The authors designed a quantum circuit to check the maximum clique in a 3-vertex graph and a 4-vertex graph and then simulated their circuit using IBM's quantum simulator. Another quantum algorithm that involves the use of quantum walking was introduced in Xi's paper [75]. Quantum walking is a different searching algorithm from Grover's algorithm. Xi proposed the use of three algorithms for graphs with different degrees. These papers [76, 77] investigated the quantum solutions to the MC problem in a quantum annealing model. A quantum annealer can provide higher accuracy and more quantum registers than adiabatic architecture. The most famous product of this technology is D-wave. Elijah et al. [76] proposed a framework for a quantum decomposition algorithm for solving MC problems. In their paper, they analyzed the lower bound of their algorithm for the MC problem and predicted the performance of their algorithm on future D-wave architectures. Similar to Elijah et al., Guillaume's group [77] proposed a quantum algorithm with which to solve MC problems using a quantum annealer and QUBO.

Thanks to Grover's algorithm and other quantum searching algorithms, the CSP is a promising area that can show the advantage of quantum computing. Ashley [78] provided a knowledgeable introduction to the quantum walk and Grover's algorithm in CSP by giving examples of backtracking trees. Zhengbing [79] provides a quantum algorithm on a quantum annealing system for CSP. This algorithm is based on Boolean constraints and function decompositions. In a recent paper, Koen et al. [80] provided a benchmark QPack that contains many CSPs like MaxCut, domination set, and the traveling salesman problem. Their benchmark considered not only running time and complexity analysis but also the cost and accuracy of quantum algorithms. Hong [81] presented three quantum-classical algorithms with which to solve weighted constraint satisfaction problems (WCSPs), one for Boolean WCSPs and the other two for general WCSPs on quantum annealers with the QUBO. Despite that progress, there remain many unsolved questions about the circuit-level implementation of the quantum algorithm and the advantages of quantum computing compared to classical computing.

1.4 Dissertation Outline

This dissertation is organized as outlined below:

Chapter 2: Different quantum gates are introduced, along with a few basic concepts of quantum mechanics. Grover algorithms are explained in detail. Two examples of Grover searching are presented to show the importance of quantum oracle in Grover algorithms.

Chapter 3: Lattice diagrams are introduced. Shannon, positive, and negative Davio lattices are presented, along with how they are realized in quantum circuits. Different structures of lattice diagrams (single output/multiple outputs) are discussed, and the simplification of the lattice diagrams is explored.

Chapter 4: The quantum oracle design methodology is introduced, and graph-theory-related problems are formulated using examples to demonstrate the details of our methodology.

Chapter 5: By transferring the Boolean minimization problems into graph problems. We created a hybrid quantum algorithm to solve the Boolean minimization problem and designed a methodology for generating the quantum oracle.

Chapter 6: We Investigated two logic puzzles with our quantum oracle building methodology—the river crossing puzzle and the MIS problem.

Chapter 7: Conclusion and future work. This chapter contains a summary of the research results of this dissertation. Further research work and approaches are also discussed.

Chapter 2: Background

The first papers that verify quantum mechanics and quantum models for computation by experiments were published in the 1980s [27,28]. The landmark paper about the quantum computing model was published in 1985 by Deutsch [29], this paper considered the relation between the quantum computation model and the Turing machine, which is an important issue to address when building a quantum computer.

In this chapter, we summary quantum gates and circuits are used in this dissertation. The quantum gates are introduced by a unitary matrix. For the quantum circuits, examples are provided to demonstrate their characteristics. At the end of this section, Grover's algorithm is introduced by equation derivation and simulation, this is the kernel algorithm of our methodology.

2.1 Quantum Gates and Circuits

Before introducing the quantum gates, we want to give a brief introduction to Dirac notation, also known as bra-key notation, which is used to present vectors in quantum mechanics.

Dirac Bra-Ket Notation

Dirac notation was introduced by Paul Dirac in 1939 [55], the notation uses angle bracket " $\rangle$ " and vertical bar " $|$ " to denote the product of vectors or the action of linear function

on vector in complex vector space. Inner product, outer product, and tensor product are the three most used functions in quantum mechanics.

$$|V\rangle = \begin{bmatrix} a \\ b \end{bmatrix}, \langle V| = [\, a^* \quad b^* \,], |W\rangle = \begin{bmatrix} c^* \\ d^* \end{bmatrix}, \langle W| = [\, c \quad d \,]$$

Inner product: $\langle W|\, V\rangle = [\, c \quad d \,] \begin{bmatrix} a \\ b \end{bmatrix} = ac + bd$

Outer product: $|V\rangle \langle W| = \begin{bmatrix} a \\ b \end{bmatrix} [\, c \quad d \,] = \begin{bmatrix} ac & ad \\ bc & bd \end{bmatrix}$

Tensor product : $|V\rangle \otimes |W\rangle = \begin{bmatrix} ac^* \\ bc^* \\ ad^* \\ bd^* \end{bmatrix}$

$|V\rangle$ is called a ket , it is usually represented as a column vector, $\langle V|$ is called bra, it is the complex conjugate of ket (V^*), usually represented as a row vector. In the quantum computer research area, Bra-ket notation is a standard notation to describe quantum bits(qubits).

In contrast to the classical computer, a quantum computer would work with quantum bits, *qubit*, Qubit is denoted with eigenstates(states) $|0\rangle$ and $|1\rangle$, and they exist in a superposition of these states: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α,β are complex numbers

satisfying the normalization condition: $|\alpha|^2 + |\beta|^2 = 1$ [3]. The superposition of a qubit can be present on a Bloch Sphere [3] in Fig 2.1. The basis state are $|0\rangle$ and $|1\rangle$, and the superposition of $|\psi\rangle$ can be converted to unique coordinates on the sphere by $|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$.

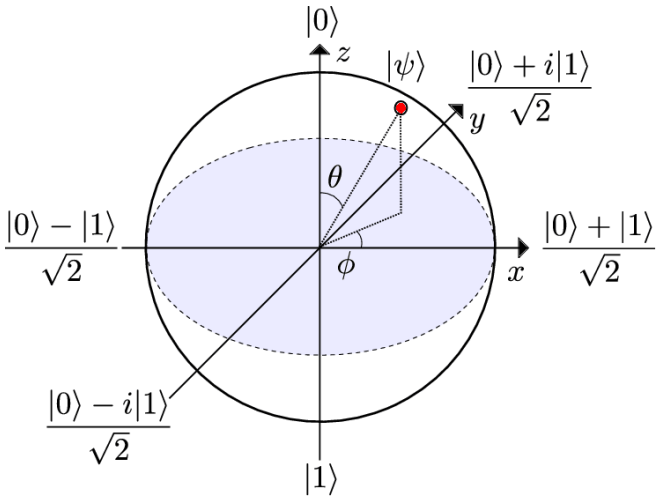


Figure 2.1 Bloch sphere [35].

A qubit can be operated by a quantum logic gate, resulting in the rotation of the state vector to a different location on the Bloch Sphere [3]. For example, a qubit from basis state $|0\rangle$, which is the top point on the z-axis, when $\theta = \frac{\pi}{2}$, $\phi = 0$, the state of this qubit moves to

$\frac{|0\rangle + |1\rangle}{\sqrt{2}}$, later we will introduce, this rotation can be performed by Hadamard gate.

Comparing to a classical logic bit which only uses state 0 and 1, the quantum superposition state like $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$ means this qubit can be state $|0\rangle$ and $|1\rangle$ at the same time, these two states have a probability to be measured, and the state of this qubit can be decided only when it is observed in the measurement. The superposition of a qubit enables parallelization of computation. Many quantum algorithms are taking this advantage to provide a great speed-up for solving sorts of classical problems [29 -33].

Like the classical logic gates that are the building blocks of a classical circuit, we have quantum gates that act on qubits and are the basic building blocks in quantum circuits. The special asset of these gates is the power of reversibility. Unitary matrices may also represent the functioning of these gates.

Next, we would introduce some of the commonly used quantum gates [1]: Hadamard gate, Pauli rotation gates, Swap gate, CNOT (Controlled NOT) gate, Toffoli gate, Fredkin gate. We can find similarity in function of some quantum gates and classical logic gates, like

Pauli rotation gate, Pauli X gate is changing the state from $|0\rangle$ to $|1\rangle$ or $|1\rangle$ to $|0\rangle$, its function is similar as the classical inverter.

Hadamard Gate:

This gate maps basis state $|0\rangle$ to $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$ and basis state $|1\rangle$ to $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$

It is represented by the unitary matrix: $H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$.

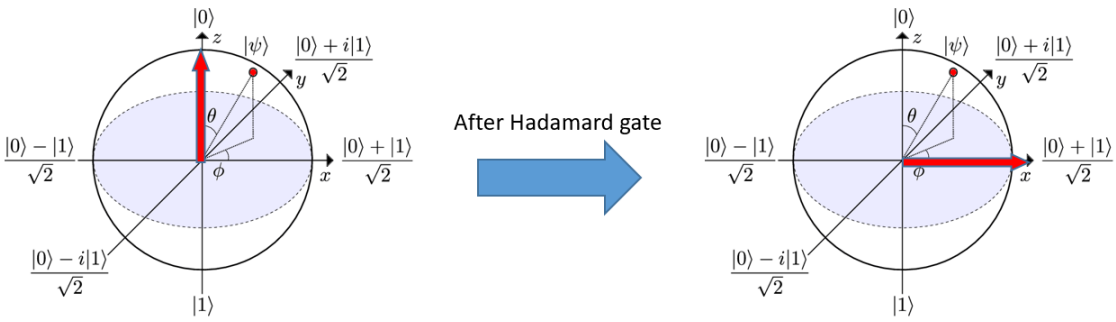


Figure 2.2 One qubit Hadamard Gate [35]

In Fig.2.2, the initial state is $|0\rangle = 1|0\rangle + 0|1\rangle$, it can be resented in matrix $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$, after we apply the Hadamard gate, the state changed from $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ to

$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix}=\frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 1\end{bmatrix}$, the new state $|\psi\rangle$ is $|\psi\rangle=\frac{|0\rangle}{\sqrt{2}}+\frac{|1\rangle}{\sqrt{2}}=\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, the probability of observing $|0\rangle$ and $|1\rangle$ in this qubit are equal. This is an important feature of Hadamard gate.

When we apply the Hadamard gate to a parallel n-qubit system prepared in a state $|0\rangle$, the output state produced by the Hadamard gate is an equal superposition of all the integers in the range from 0 to 2^n-1

i.e. $H|0\rangle \oplus H|0\rangle \oplus ... \oplus H|0\rangle = \frac{1}{\sqrt{2^n}}\sum_{j=0}^{2^n-1}|j\rangle$ where, $|j\rangle$ is the computational basis state represented by the binary number that maps to the corresponding number 'j' in base {1,0}. Take vector state $|000\rangle$ for an example, the entire space of the output of Hadamard gate is created with 2^3 basic states, such as:

$$\frac{1}{\sqrt{2^3}}(|000\rangle+|001\rangle+|010\rangle+|011\rangle+|100\rangle+|101\rangle+|110\rangle+|111\rangle)$$

Pauli X, Y, and Z Gates:

The Pauli X gate is the quantum equivalent of the NOT gate in classical logic, while the Y and Z gates rotate the qubit along the Y and Z axis, respectively, on the Bloch sphere in Fig2.1.

The Pauli X gate maps $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$, the equivalent Pauli matrix is:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.1}$$

It corresponds to the rotation of the Bloch ball around the X-axis by π (180) radians.

The Pauli Y gate maps $|0\rangle$ to $i\ |1\rangle$ and $|1\rangle$ to $-i\ |0\rangle$, the Pauli Y matrix is:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{2.2}$$

It equates to the rotation of the Bloch ball around theY-axis by π (180) radians.

The Pauli Z gate maps $|1\rangle$ to - $|1\rangle$ and leaves state $|0\rangle$ as is, the Pauli Z matrix is:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.3}$$

It equates to the rotation of the Bloch ball around the Z-axis by π (180) radians.

Swap Gate:

This gate swaps two qubits as per its basis state, it is given by the matrix in Fig 2.3:

$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2.3 Matrix of Swap gate

Figure 2.4 Symbol of SWAP gate

For example, if the input state is $|01\rangle$, its matrix representation is :

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{matrix} \tag{2.4}$$

When we apply the Swap gate to this state:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{2.5}$$

From the out of Swap gate, we can find the state changed from $|01\rangle$ to $|10\rangle$.

The Swap gate can only perform a swap operation between two qubits, but we can use multiple Swap gates to perform a swap in a quantum circuit which input size is larger than two.

CNOT Gate:

This is a controlled-NOT gate that acts on two or more qubits; the first line/qubit is a control line/qubit that decides whether to flip the rest of the qubits. When the first qubit is set to $|1\rangle$, it performs a NOT operation on the rest of the qubits; otherwise, it leaves them unchanged. It is represented by the matrix in Fig. 2.5:

$$\mathrm{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 2.5 Matrix of CNOT gate

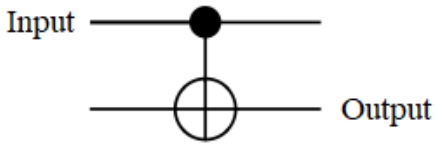


Figure 2.6 Symbol of CNOT

Toffoli Gate:

This is also called the Universal quantum gate as the other gates can be deduced from this gate. It functions as a CCNOT (CONTROLLED CONTROLLED NOT) gate on three qubits. If the first two qubits are in state $|1\rangle$ then it flips the third qubit, else it leaves it unchanged. It is given by the following matrix in Fig. 2.7:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 2.7 Matrix of Toffoli gate

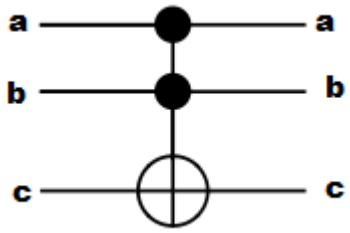


Figure 2.8 Symbol of the Toffoli gate

We use a truth table to give a straightforward view of the Toffoli gate. This gate maps input state $|a,b,c\rangle$ to output $|a,b,(c \oplus ab)\rangle$ as:

Table 2.1. Truth Table of the Toffoli Gate

Input	Output
$\|000\rangle$	$\|000\rangle$
$\|001\rangle$	$\|001\rangle$
$\|010\rangle$	$\|010\rangle$
$\|011\rangle$	$\|011\rangle$
$\|100\rangle$	$\|100\rangle$
$\|101\rangle$	$\|101\rangle$
$\|110\rangle$	$\|111\rangle$
$\|111\rangle$	$\|110\rangle$

Fredkin Gate

This gate is also called a CSWAP (Controlled- Swap) gate. It is a three-qubit gate used to perform a controlled swap, it only swaps the lower two qubits when the first line is

activated. The benefit of this gate is its conservation of 0's and 1's throughout. The matrix representation is in Fig. 2.9.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2.9 Matrix of Fredkin gate

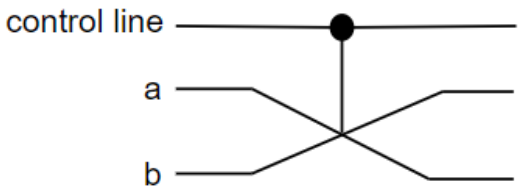


Figure 2.10 Symbol of Fredkin gate

Next, we would like introduce some quantum circuits which are used in our methodology.

Quantum Counter (Quantum Accumulator)

This circuit is a basic component used in our oracle, the function of this circuit is to count and sum up the number of targets.

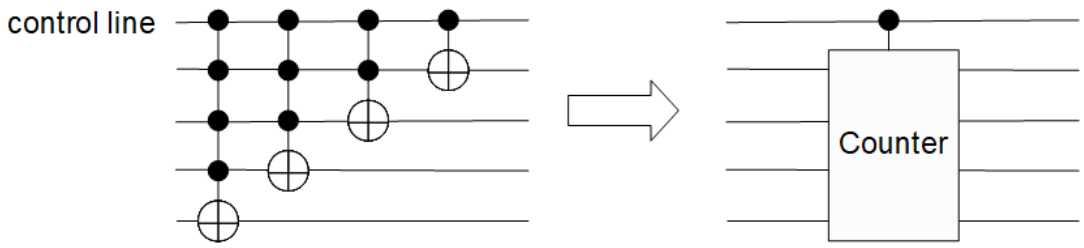


Figure 2.11 4-qubit quantum accumulator

The top line in Fig. 2.11 is called the control line, this circuit only accumulates when the control line equals to $|1\rangle$. The remaining four lines are the output lines, they record how many times the control line has been activated, then output the number in the format of a vector. The output lines need to be initialized with $|0\rangle^{n}$. The quantum accumulator needs to be implemented in cascade connection in the circuit, the number of counters is decided by the number of targets that need to be counted.

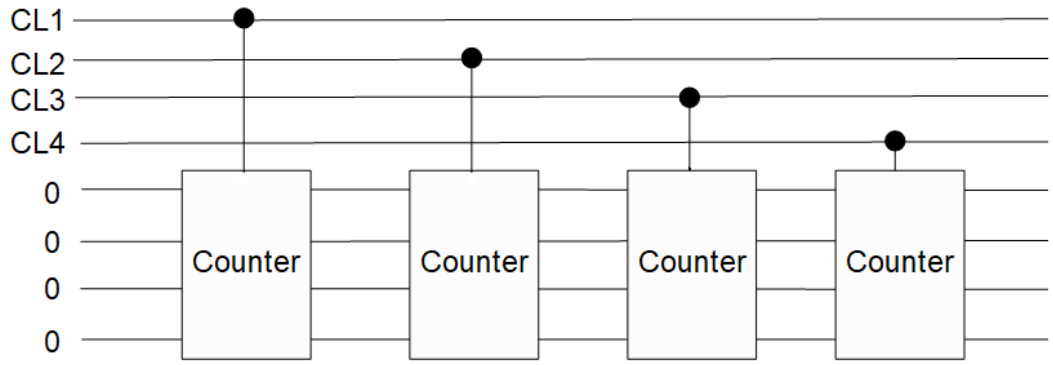


Figure 2.12 Example of a quantum accumulator

For example, in the circuit of Fig 2.12, there are four counters to accumulate the results from related targets. If and only if the related control line (CL) is true, the counter is activated. Suppose all the control lines are true in Fig. 2.12, let us look back to Fig. 2.11 for the inner structure of each counter. For the first counter, only the Feynman gate at the right side is on, this gate flips the initialized value 0 to 1, and the value of the rest qubits just pass through, the output of this circuit is [1000]. The data input of the accumulator is fed by the output of the previous one, this connection makes the gates inside of each accumulator be triggered sequentially. For example, the input of the second accumulator is [1000], the Feynman gate and two-input Toffoli gate in Fig 2.11 are triggered on, the output of the accumulator is [0100], taking the most significant bit (MSB) from the right side of this vector, this binary vector equals to 2.

Quantum Inequality Comparator

Quantum inequality comparator, as shown in Fig 2.13, is another important circuit template used in our oracle, the function of this circuit is comparing the value of two inputs.

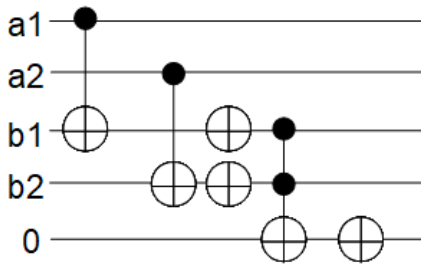


Figure. 2.13 2-qubit comparator

In Fig 2.13, it is a two-qubit inequality comparator. The bottom data line is the output of this inequality comparator, when the input data are the same qubit vector, the output line returns 1; otherwise, the output line returns 0. Similarly, Equality Comparator and other comparators can be built to be used in Grover Oracles.

Because of the quantum cost of large Toffoli gate is extremely expensive, our oracle uses the accumulator and comparator to replace the Toffoli gate. Besides the quantum cost, the combination of accumulator and comparator also is more flexible in constraint satisfaction problem (CSPs), for example, each of the control line is a result of a constraint, if the oracle need to disable some of the constraints, for Toffoli gate, the blocked constraints must be specific, otherwise, the oracle can not decide where to put the invertor on the Toffoli gate. However, in accumulator and comparator structure, the oracle only concerns the numbers of satisfied constraints, it doesn't need to check which specific constraint is true. This property provides more flexibility in some CSPs comparing to the Toffoli gate.

2.2 Quantum Circuit Simulation

This dissertation is entirely devoted to the Grover Algorithm, which is the most useful quantum algorithm with many practical applications in engineering problems. The name Grover is derived from its inventor, but it is somewhat misleading. This is not an algorithm; rather, it is a general method to speed-up loops that search without additional information. Therefore, the Grover algorithm can be applied to many problems giving quadratic speedup for each loop of some "higher-level algorithm."

For the speedup of Grover Searching, it is like looking for a white marble among three black marbles in an urn. The worst case is, after picking out every black marble, we finally find the white marble. However, with Grover searching, if we know we are looking for the white marble, we can take it out firsthand; that is the power of quantum searching, find the target in only one shot. In general, with N marbles the classical algorithm would need in the worst case N searches (evaluations of the oracle), but the quantum algorithm of Grover would need only $\sqrt{N}$!

A simple example is shown in Figure 2.14. We do not know the function of the Boolean inside the oracle, but we know that this function has only one true minterm. In this case, it is $\left(b1, \overline{b2}, b3\right)$ for minterm it will be: 101(binary vector). Assuming that it is the only solution, the problem is finding the solution between the possible eight minterms of the 3-

bit length data, which are 000, 001, 010, 011, 100, 101, 110, and 111. The normal method to solve this problem would need to test the classical circuit 7 times to find the solution. In a quantum algorithm, only one oracle is enough, as illustrated below. The blocks with symbol X and Z in Fig. 2.14 are Pauli X gate and Pauli Z gate, the block with H is Hadamard gate.

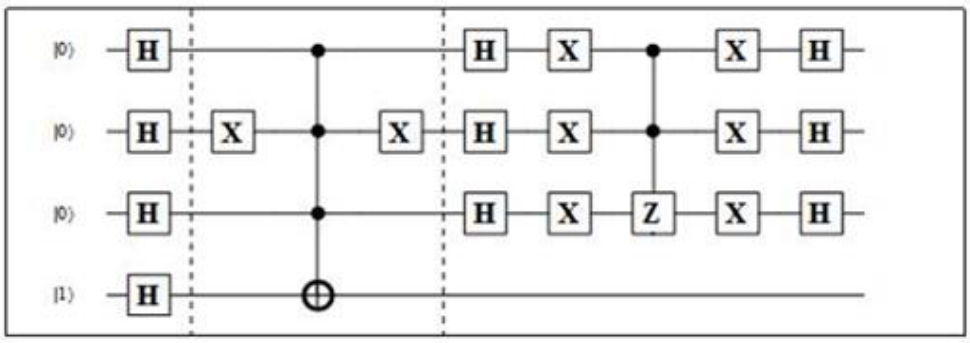


Figure 2.14: Example of Grover algorithm with a single Toffoli gate to realize the minterm of three-variable function.

The number of qubits is the length of data in this problem. Our target uses hidden items; for easy reading, it uses decimal numbers here. Like the stem graph shown above, we will have a relatively high probability of getting the solution compared with other minterms after two iterations.

The next example is a Boolean satisfaction problem (SAT); the probability of changing every input qubit is shown in detail in this example.

Example 2.1

In this SAT example, the oracle is a Boolean equation: F(a,b,c) = (a+b')(a'+c), its quantum circuit implementaltion is Fig 2.15

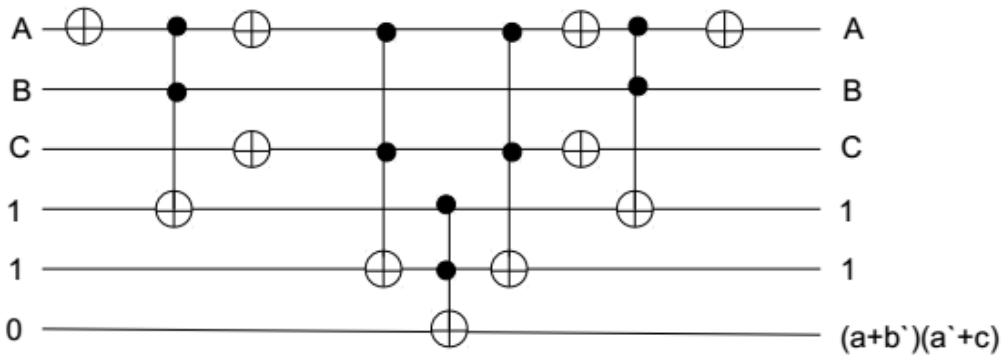


Figure 2.15 Oracle of this equation

ab/c	0	1
00	1	1
01	0	0

11	0	1
10	0	1

Figure 2.16 K-map of this equation

From the K-map, there are four solutions for this SAT problem; the Unique solution Grover searching is a special case of Multiple solution searching; for the Multiple solution problems, theoretically, we will obtain an equal probability of each solution by Grover searching after the maximum iteration.

Fig. 2.17 is the same implementation of F(a,b,c); the different part is, there are many swap gates used here. Based on the current technology, qubits can only talk to their neighbor's lines. Then we need to use a swap gate to pull those qubits up or down for the correct operation. This optimization is not only for a circuit but also needed inside the realizetion of the Toffoli gate. In Fig 2.17, we use the standard symbol of the Toffoli gate. However, inside, we also add two swap gates for the same reason, shown in Fig 2.18. After applying the swap gate, this Toffoli gate model can be simplified because of the two Feynman gates in a red block; the simplified version is shown in Fig 2.18.b

The rest of the quantum schematic in this dissertation all use the same technology, for the reader’s convenience, the Swap gate will not be draw in the rest of the dissertation.

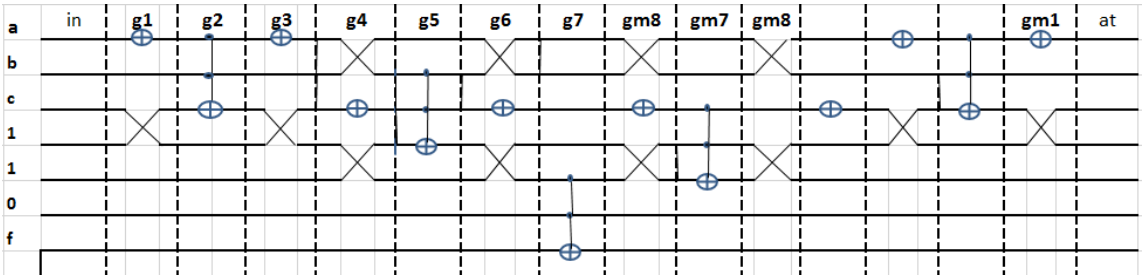


Figure 2.17 Extended version of the SAT oracle with Swap gates for the Grover algorithm

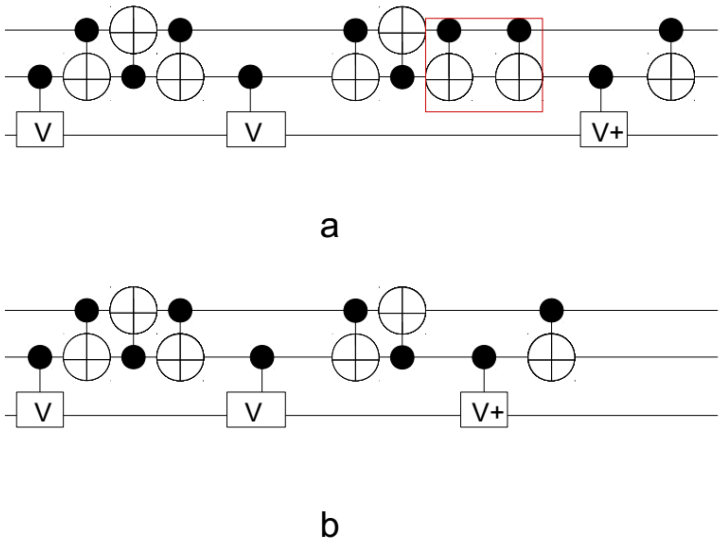


Figure 2.18 (a): Inner structure of Toffoli gate (b): Simplified Toffoli gate

2.3 Grover's Algorithm

2.3.1. Introduction

To search an unsorted database with N entries, Grover's algorithm requires N dimensional state space, represented by n= $\log_2 N$ qubits. The best classical algorithm requires time in the order of O(N) to search over the unordered set, while Grover [2] takes only $O\left(\sqrt{N}\right)$ units of time for the same, resulting in a quadratic speedup. Such speedup is achieved due to the quantum superposition of the states. The working is as follows:

For an 'n' qubit system, the search space is $N = 2^n$ with all bits initialized to state $|0\rangle$ as:

$$|0\rangle^{\otimes n} = |0\rangle \tag{2.6}$$

To bring this into a superposition state, a Hadamard transformation is done as:

$$|\psi\rangle = H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2}}\sum_{x=0}^{2^n-1}|x\rangle \tag{2.7}$$

This ensures equal amplitude of $\frac{1}{\sqrt{2^n}}$ associated with every possible configuration of qubits in the system and an $\frac{1}{2^n}$ equal probability that the system will be in one of the 2^n

states. Quantum algorithms use amplitude amplification to achieve the qualities of quantum amplitudes which differentiate them from simple probabilities. The main idea is to select the right kind of phase-shifting operator that satisfies some conditions at each iteration. For example, a phase shift of π is nothing but the multiplication of the amplitude by -1, this changes the orientation of the vector but not the probability of its being in the state. In this algorithm, subsequent transformations are done on the system to take advantage of that difference in amplitude to separate that state(s) of a differing phase(s) and to ultimately increase the probability of the system being in that state(s).

The next set of transformations is referred to as the *Grover Iteration*, which performs amplitude amplification and is repeated: $\frac{\pi}{4}\sqrt{N}$ times. This ensures that we attain the optimal probability of the state being observed to be the correct one and according to Grover [2] this is achieved at a rotation of $\pi/4$ radians. The first step to the iteration is the use of an oracle called the *Quantum Oracle* O. This oracle is a black-box function, meaning it can observe and modify the system without collapsing to a classical state and it indicates if the system is in the correct state by rotating it by π radians, else it does nothing. Quantum oracle implementations often use extra bits(ancilla bits), which are unnecessary in this implementation, so the oracle's impact on $|x\rangle$ may be written as:

$$|x\rangle \xrightarrow{O} (-1)^{f(x)}|x\rangle \tag{2.8}$$

Where *f(x) = 1* if x is in the correct state and *f(x) = 0* otherwise. The implementation of f(x) depends on the problem at hand.

The next step is called the diffusion transform, it is the key point of Grover's Algorithm. After the phase inversion from previous step, the diffusion transform flips the amplitude of every states by an operator called Grover diffusion operator $2|0\rangle\langle 0| - I$. The function of this operator is shown as following:

$$[2|0\rangle\langle 0| - I]|0\rangle = 2|0\rangle\langle 0|\,0\rangle - I = |0\rangle \tag{2.9}$$

$$[2|0\rangle\langle 0| - I]|x\rangle = 2|0\rangle\langle 0|\,x\rangle - I = -|x\rangle \tag{2.10}$$

The diffusion operator flips the amplitude of state $|x\rangle$, but for the state $|0\rangle$, this operator didn't change the amplitude.

The entire diffusion operator may be written as:

$$H^{\otimes n}[2|0\rangle\langle 0| - I]H^{\otimes n} = 2H^{\otimes n}|0\rangle\langle 0|H^{\otimes n} - I = 2|\psi\rangle\langle\psi| - I$$

For arbitrary state $\sum_i a_i \ |i\rangle$, where a_i is the amplitude, when we apply the diffusion operator to it:

$$(2|\psi\rangle\langle\psi| - I)\sum_i a_i \ |i\rangle = \sum_i (2a_m - a_i)|i\rangle \tag{2.11}$$

Where a_m is the mean amplitude of every states, $a_m = \frac{\sum_i a_i}{N}$, N is the number of items in the database. From the equation one can see that the amplitude of each state is flipped by the mean amplitude a_m [34].

Regarding the runtime of the Grover iteration, the exact runtime of the oracle depends on the problem at hand, so the call to the oracle is viewed as one operation. Hence the total runtime for one iteration is $\Theta(2n)$, from the two Hadamard transformations, plus $O(n)$ for the phase shift gate leading to $O(\sqrt{N}) = O(\sqrt{2^n}) = O(2^{n/2})$ iterations each with a runtime of $O(n)$ is $O(2^{n/2})$.

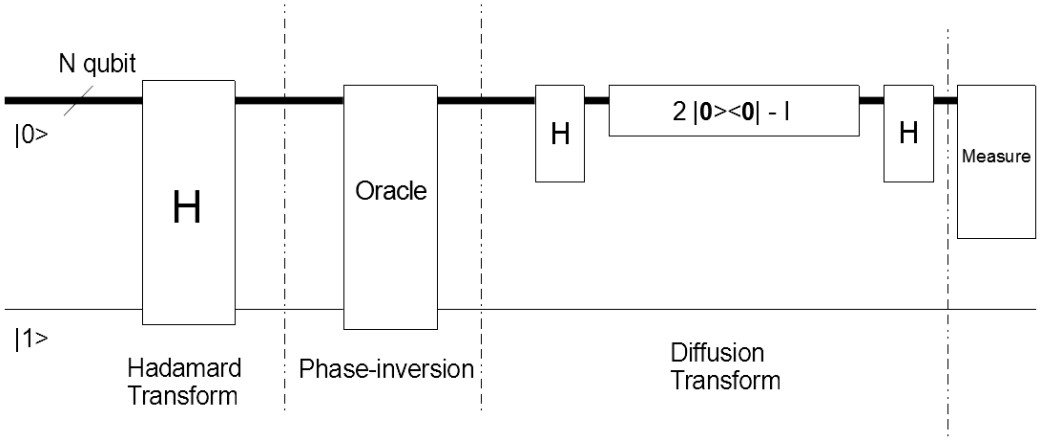


Figure 2.19 Circuit diagram for Grover's algorithm for the oracle.

Example 2.2:

Consider a system of N = 8 = 2^3 states and the state being searched is 011. This system can be represented by n = 3 qubits as follows:

$$|x\rangle = a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|100\rangle + a_5|101\rangle + a_6|110\rangle + a_7|111\rangle$$

Where a_i is the amplitude of the quantum state $|i\rangle$. The system is initialized to state zero. Applying the first set of Hadamard transformations to obtain a superposition of states with equal amplitude probabilities is:

$$1/\sqrt{N} = 1/\sqrt{8} = 1/2\sqrt{2} = h1$$

$$H^3|000\rangle = h1|000\rangle + h1|001\rangle + h1|010\rangle + h1|011\rangle + h1|100\rangle + h1|101\rangle + h1|110\rangle + h1|111\rangle = \frac{1}{2\sqrt{2}}\sum_{x=0}^{7}|x\rangle = |\psi\rangle \quad (2.12)$$

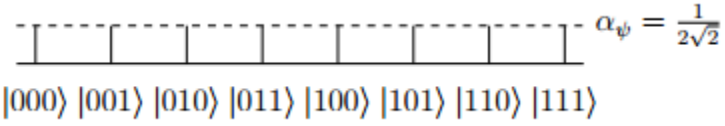


Figure 2.20 Initialized amplitude of quantum states

The number of iterations to find all solutions: $\frac{\sqrt{8}\pi}{4} \approx 2.22$ and is rounded to 2 iterations.

For iteration 1, the oracle query will negate the amplitude of the state |011> (as we are in search of this state from the superposition) giving:

$$|x\rangle = h1|000\rangle + h1|001\rangle + h1|010\rangle - h1|011\rangle + h1|100\rangle + h1|101\rangle + h1|110\rangle + h1|111\rangle \quad (2.13)$$

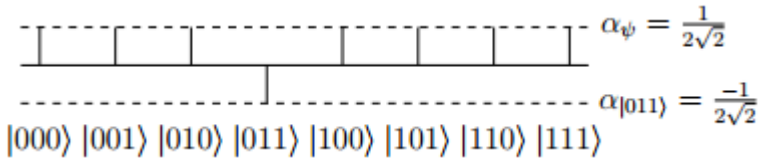


Figure 2.21 Amplitudes of quantum states after the phase inversion

Now, the diffusion transform is performed as $2|\psi\rangle\langle\psi| - I$, which will increase the amplitudes by their difference from the average, decreasing if the difference is negative:

$$\begin{aligned}
&[2|\psi\rangle\langle\psi| - I]|x\rangle \\
&= [2|\psi\rangle\langle\psi| - I]\left[|\psi\rangle - \frac{2}{2\sqrt{2}}|011\rangle\right] \\
&= 2|\psi\rangle\langle\psi|\psi\rangle - |\psi\rangle - \frac{2}{\sqrt{2}}|\psi\rangle\langle\psi|011\rangle + \frac{1}{\sqrt{2}}|011\rangle
\end{aligned} \tag{2.14}$$

Apply the following property of identity:

$$<\psi|\psi>= N\frac{1}{\sqrt{N}}\cdot\frac{1}{\sqrt{N}} = 8\frac{1}{\sqrt{8}}\cdot\frac{1}{\sqrt{8}} = 1 \tag{2.15}$$

$$<\psi|001>=<001|\psi>= \frac{1}{\sqrt{N}} = \frac{1}{\sqrt{8}} \tag{2.16}$$

$$2|\psi\rangle\langle\psi|\,\psi\rangle - |\psi\rangle - \frac{2}{\sqrt{2}}|\psi\rangle\langle\psi|\,011\rangle + \frac{1}{\sqrt{2}}|011\rangle$$

$$= 2|\psi\rangle - |\psi\rangle - \frac{2}{\sqrt{2}}\frac{1}{\sqrt{8}}|\psi\rangle + \frac{1}{\sqrt{2}}|011\rangle$$

$$= |\psi\rangle - \frac{1}{2}|\psi\rangle + \frac{1}{\sqrt{2}}|011\rangle$$

$$= \frac{1}{2}|\psi\rangle + \frac{1}{\sqrt{2}}|011\rangle$$

$$= \frac{1}{2}\left(\frac{1}{2\sqrt{2}}\sum_{x=0}^{7}|x\rangle\right) + \frac{1}{\sqrt{2}}|011\rangle$$

$$= \frac{1}{4\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) + \left(\frac{1}{4\sqrt{2}} + \frac{1}{\sqrt{2}}\right)|011\rangle$$

$$= \frac{1}{4\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) + \frac{5}{4\sqrt{2}}|011\rangle$$

Hence,

$$|x\rangle = \frac{1}{4\sqrt{2}}|000\rangle + \frac{1}{4\sqrt{2}}|001\rangle + \frac{1}{4\sqrt{2}}|010\rangle + \frac{5}{4\sqrt{2}}|011\rangle + ... + \frac{1}{4\sqrt{2}}|111\rangle \qquad (2.17)$$

Which appears as in Fig. 2.22

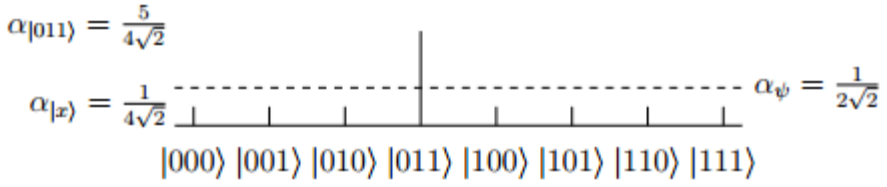


Figure 2.22 Amplitudes of quantum states after the first Grover iteration

The same transformations are applied for the second iterations as follows:

$$|x\rangle = \frac{1}{4\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) + \frac{5}{4\sqrt{2}}|011\rangle$$

$$= \frac{1}{4\sqrt{2}}\sum_{x=0}^{7}|x\rangle - \frac{6}{4\sqrt{2}}|011\rangle$$

$$= \frac{1}{2}|\psi\rangle - \frac{3}{2\sqrt{2}}|011\rangle$$

On the second round of diffusion transform, apply the diffusion operator:

$$[2|\psi\rangle\langle\psi| - I]\left[\frac{1}{2}|\psi\rangle - \frac{3}{2\sqrt{2}}|011\rangle\right]$$

$$= |\psi\rangle\langle\psi|\,\psi\rangle - \frac{1}{2}|\psi\rangle - \frac{3}{\sqrt{2}}|\psi\rangle\langle\psi|\,011\rangle + \frac{3}{2\sqrt{2}}|011\rangle$$

$$= |\psi\rangle - \frac{1}{2}|\psi\rangle - \frac{3}{4}|\psi\rangle + \frac{3}{2\sqrt{2}}|011\rangle$$

$$= -\frac{1}{4}|\psi\rangle + \frac{3}{2\sqrt{2}}|011\rangle$$

$$= -\frac{1}{8\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) + \frac{11}{8\sqrt{2}}|011\rangle$$

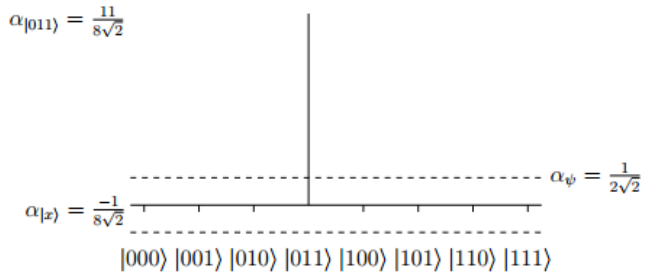


Figure 2.23 Amplitudes of quantum states after the second Grover iteration

2.3.2 Simulations

We will use here a simulation to show the internal quantum data change during the execution of the Grover Algorithm. The simulations have been performed on MATLAB and the amplitude of each state will be exported into an Excel worksheet.

The circuit for the Grover implementation is as follows:

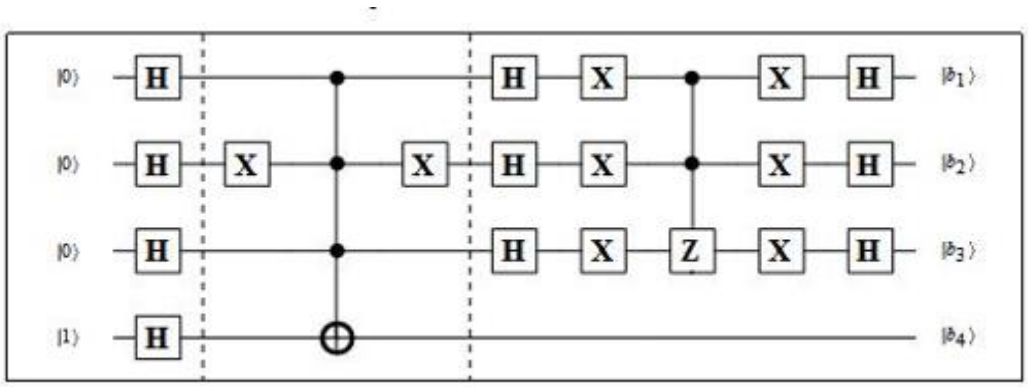


Figure 2.24 Implementation of Grover search for a three-input Toffoli gate

The code implemented allows the user to select a size for Grover implementation. The algorithm was initially tested for a Toffoli gate oracle and then extended to SAT problems. Also, each gate is defined as a separate function for ease of access.

Toffoli Oracle:

This is a three-bit gate (but the algorithm can have this extended to N bits) as follows:

Table 2.2 Truth table of example oracle

a	b	b	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1(Target)

For a three-qubit Grover, $N = \sqrt{2^3} \approx 2$. It needs two iterations to obtain the correct measurement. The following graphs are simulation results from MATLAB.

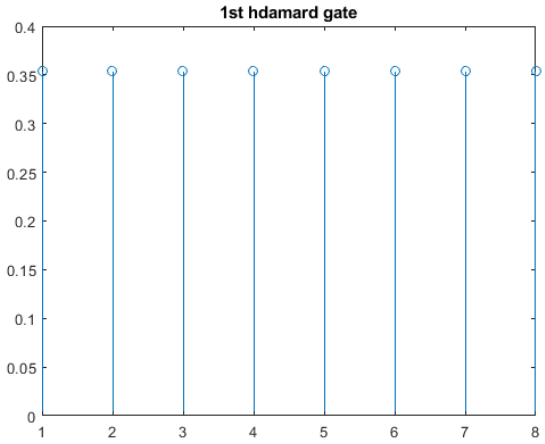


Figure 2.25 Amplitude of each state after Hadamard gate.

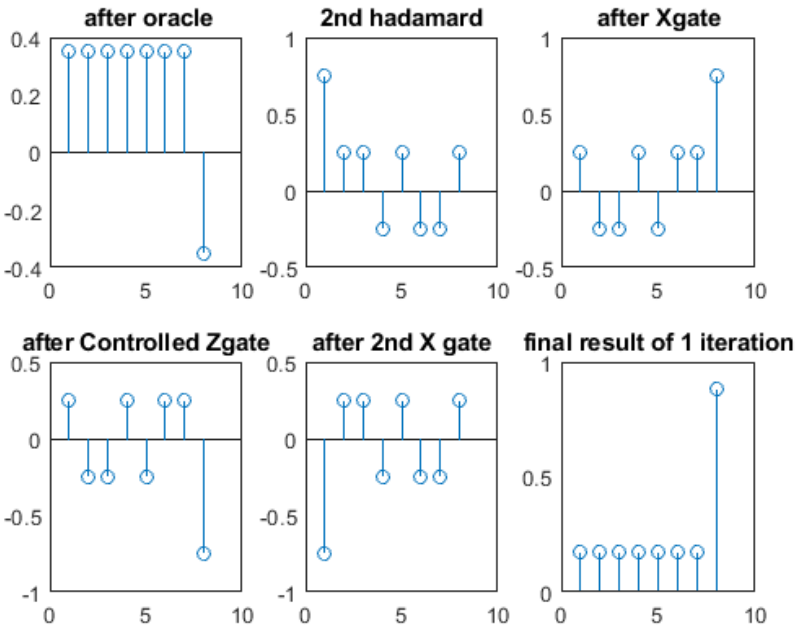


Figure 2.26 Result of the first iteration

Fig 2.26 shows the result of each stage in the first round of Grover's algorithm, the phase inversion can be observed clearly at the sub-figure "after oracle" in Fig.2.26, and the amplitude implication can be found in the final result.

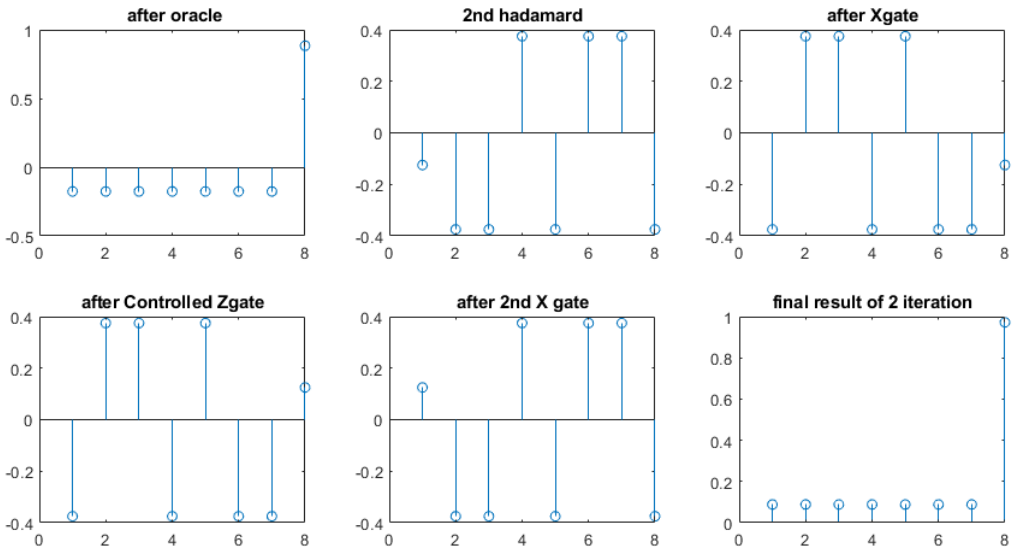


Figure 2.27 Result of the second iteration

In last figure of Fig. 2.27, the amplitude of rest items is reduced below 0.2 comparing to the value around 0.3 in Fig.2.26, this the trick of Grover's algorithm. As mentioned previously, there is a maximum rounds of Grover's algorithm, for this example, the number is 2. If we perform the third round, then the superposition of target item would be collapsed. [14]

2.3.3 Quantum Oracle

This oracle is specific to the Satisfiability problem (SAT): F = (a+b') (a'+c)

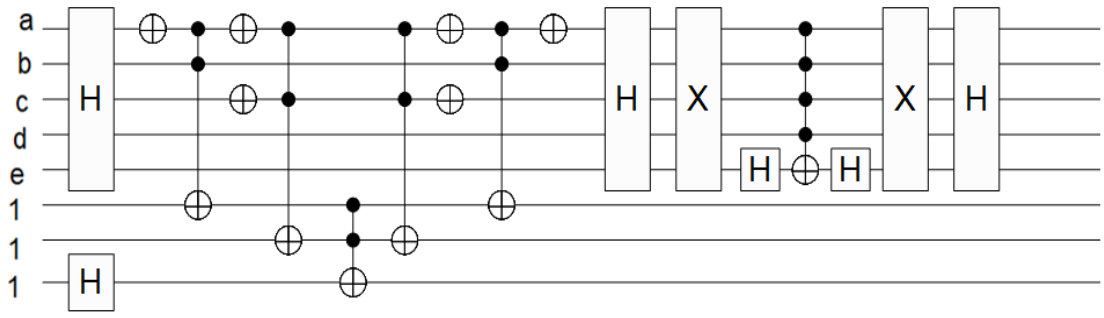


Figure 2.28 Oracle for the 2-SAT problem

This is designed to test the working of Grover's algorithm on more practical applications as many problems may be reduced to SAT-based problems. By SAT-based problems I mean not only POS SAT (Product of Sums SAT) but any other Boolean formula that can be satisfied or not. For instance, ESOP or Product of ESOPs. The Toffoli oracle mentioned previously has been replaced with this oracle and the results will be slightly different from the previous one due to the presence of multiple satisfying conditions as shown in the Tab. 2.3 below:

Table 2.3 Truth table for F(a, b, c, d, e)

a	b	c	F=(a+b')(a'+c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Variable *d* and *e* are not listed in the table because they do not affect the result of the output of this function. Based on the number of input variables, there needs to be multiple iterations to get the final results with the highest possible measurement probability.

The results of simulation for each iteration are as follows:

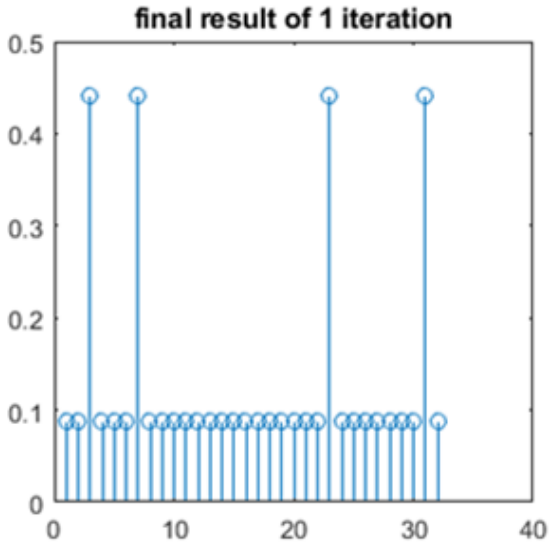


Figure 2.29 Results of the first iteration

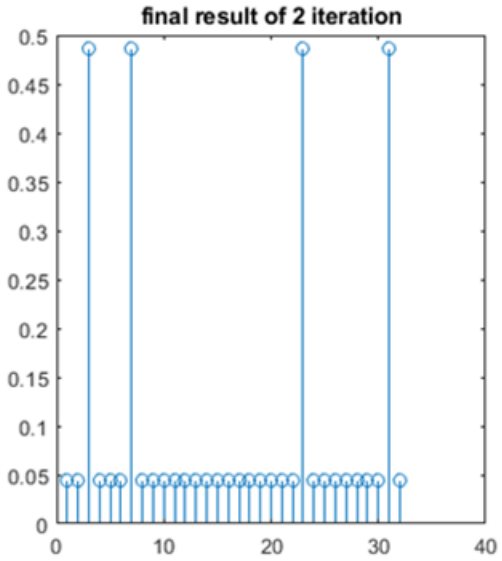


Figure 2.30 Results of the second iteration

After two iterations of the Grover search, we can find that the largest amplitude items are 3, 7, 23, and 31. Since there are two ancillary bits here, we need to decode those results to get the final result; for state 3, its binary representation is 00011, the 2 LSBs (Least Significant Bit) are ancillary bits ”11”, so the solution will be a=0 B=0 and C=0.

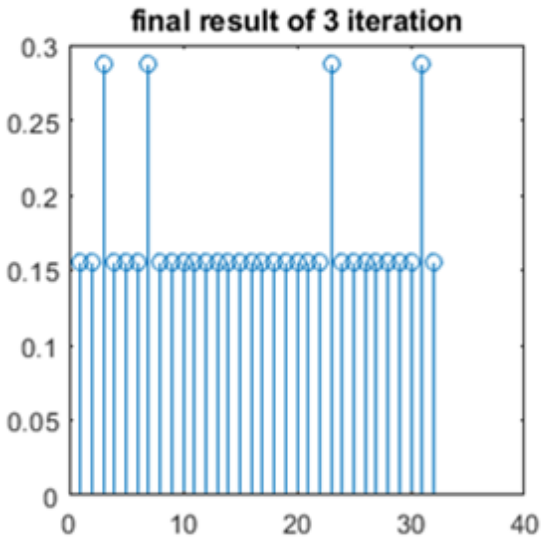


Figure 2.31 Results of the third iteration

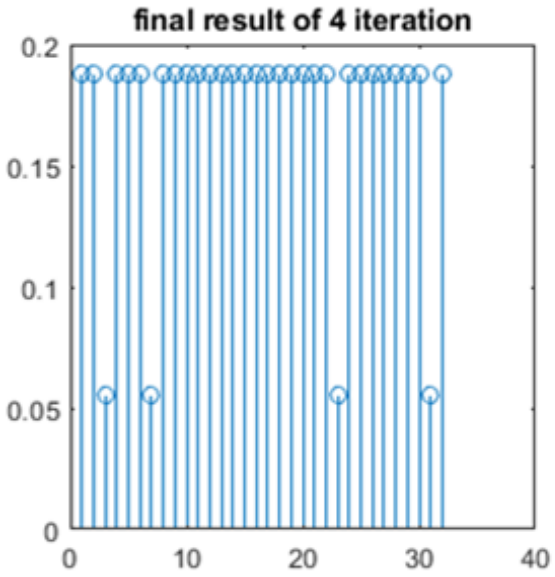


Figure 2.32 Results of the fourth iteration

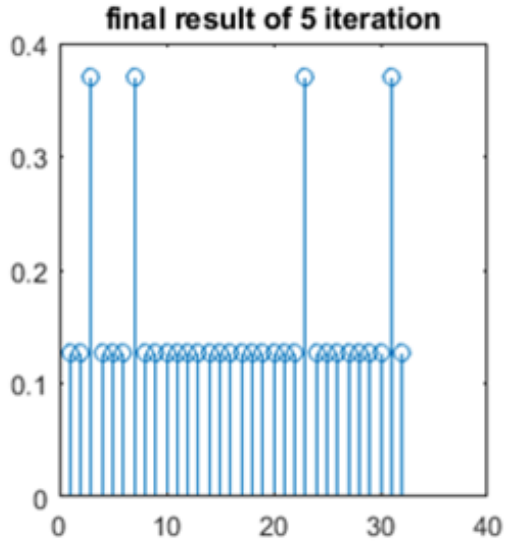


Figure 2.33 Results of the fifth iteration

In this example, because there are five input variables, to get the best result, the number of iterations should be $\sqrt{2^5} \approx 5$.

But from the simulation results, we noticed that the amplitude of targets in the second iteration is ten times greater than others, meanwhile that number in the fifth iteration is four. The phenomenon is caused by the redundant input variables *d* and *e*, these two variables is independent with the output of the oracle function. So in this case, when we build the circuit, the redundant input variables should be removed.

The oracle function is built from a Boolean logic function, the problem of redundancy removal is similar to the same problem in classical Boolean logic, there are many methods to solve this problem like Binary decision diagrams (BDDs) [56,57].

There are four solutions to this problem: at iteration 1, we can find the phase-amplitude (AMP) of these four solutions is 0.42, compared with other minterms is 0.1, the probability amplitude of them is 0.42^2=0.17, 0.1^2=0.01. We can find that the probability of the measured solution is 17 times larger than non-solution. In the next Grover searching iteration, the AMP is changed to 0.48 and 0.05; the probability amplitudes are 0.23 and 0.0025, 0.23/0.0025 = 92.

Based on the condition of the Grover algorithm, this iteration is the upper bound. In the next iteration, the AMP will collapse, and the solution's probability amplitude will

decrease. From Figure 2.29 to 2.33, we can prove this phenomenon; in iteration 3, the amplitudes of solution and non-solution are 0.28 and 0.15, and the related probability amplitudes will be 0.08 and 0.02. With the Grover algorithm, we have a higher probability of finding one solution in the multiple solution searching problems; but what if we want to find solutions?

Suppose we have a problem; after Grover searching, there are four solutions: A, B, C, D. The probability of finding a good solution is 100%. The probability of finding each solution is the same. The probability of finding one different solution at first search is 100%. Since all solutions are different, there is a 100% probability of getting a different solution at the first time.

The combinations of picking two solutions between four solutions are:

$$C_2^5 = \frac{5!}{2!\,3!} = 10 \tag{2.18}$$

In these ten cases, there are four combinations (AA, BB, CC, DD) that contains the same solutions.

So the probability of finding two different solutions in the second search is 6/10=0.6

combinations of picking three solutions are:

$$C_3^6 = \frac{6!}{3!\,3!} = 20 \tag{2.19}$$

Similarly, there are ten combinations (AAA…DDD, AAB, AAC, …, CCD) contains same solutions.

So the probability of finding three different solutions in the third search is 10/20=0.5.

The probability of picking 4 solutions is:

$$C_4^7 = \frac{7!}{4!\,3!} = 35 \tag{2.20}$$

There are 23 duplicate combinations for this case, and the probability is 12/35=0.34

Number of searches	Probability to find the target
1st	100%
2nd	60%
3rd	50%
4th	35%

2.4 Summary of Chapter 2

In this chapter, Quantum gates and related details from quantum mechanics were introduced, based on those quantum gates, quantum counter and comparator were presented, which are used in our quantum oracle design methodology. Quantum Grover's algorithm was presented step by step with equation derivation and simulation. Grover algorithm was shown to be efficient in single target searching problem. However, there are not many papers that would explore the situation with multiple solutions, this problem was discussed at the end of this chapter.

Chapter 3: Boolean Symmetric Function and Quantum Lattice Diagram

Note: Some of the contents of this chapter have been published in the following paper:

P. Gao, Y.Li, M.A. Perkowski and X. Song "Realization of Quantum Oracles using Symmetries of Boolean Functions", *Quantum Inf. Comput*. vol. 20 (5&6), 2020.

The kernel of a quantum oracle is an implementation of a decision function, which the output of this function is only 0 or 1. Synthesizing the oracle is therefore a problem in Boolean Circuit specification and minimization. This decision function is implemented with quantum gates in the oracle, the number of the quantum gates and the complexity of the circuit greatly affect the quantum cost of an oracle. From the current size of quantum cells, the quantum cost is still a problem to be concerned about when designing quantum oracles. How to use Boolean symmetric function and Lattice diagram to design an area-efficient oracle are mainly discussed in this chapter. This approach of designing Grover oracle bottom-up using reversible logic and blocks with ancilla qubits is the new contribution of my dissertation.

3.1 Boolean Symmetric Functions

Let f be a total Boolean function: $B^n \rightarrow B$, where $B = \{0,1\}$ and $n>1$.

Definition 1 (Totally Symmetric Boolean Function)

A Boolean function *f* is totally symmetric if its output is invariant under any permutation of its input bits:

$f(x_1, x_2, \ldots, x_n) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$ for all permutations σ of {1, …, n}.

A single index symmetric function *S* can be denoted S^k such that, for every true minterm m_i of *S*, the number of positive literals in all true minterms of this function is *k* and the number of negative literals is *n-k*.

For example, a symmetric function $F(a,b,c) = a\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c$ is denoted S^1, where $\bar{a}$ means *a* is a negative literal in the minterm $\bar{a}b\bar{c}$. Each minterm of F(a, b, c): $a\bar{b}\bar{c}$, $\bar{a}b\bar{c}$, $\bar{a}\bar{b}c$, has only one positive literal.

The symmetry property of S^1 can be shown by permuting the input from (a, b, c) to (a, c, b), which means the order of input variables *b* and *c* are transposed. The new equation is $a\bar{b}\bar{c} + \bar{a}\bar{b}c + \bar{a}b\bar{c}$ after changing the order, which is the same function as S^1. This property should be satisfied for any pair of variables.

Increasing *k* from 1 to 2, we create another single index symmetric function in which all the minterms have two positive literals, such as $S^2 = ab\bar{c} + a\bar{b}c + \bar{a}bc$ By combining multiple single index symmetric functions, we obtain a multiple index symmetric function, such as $S^1+S^2=S^{1,2}$.

Definition 2 (Polarity Vector)

Given a Boolean function $f(x_1, x_2, \ldots, x_n)$, its polarity vector is a vector of $(y_1, y_2, \ldots, y_n)$, where y_i is either a positive or negative literal of variable x_i.

By introducing the polarity vector, the symmetric function can be extended to the concept of a generalized symmetric function. The original definition of the totally symmetric function is a special case in which the polarity vector contains only positive literals.

Definition 3 (Generalized symmetric function)

Given a Boolean function $f(x_1, x_2, \ldots, x_n)$ and a polarity vector $(y_1, y_2, \ldots, y_n)$, $y_i \in \{\overline{x}_i, x_i\}$. By substituting each literal in the polarity vector with $f(x_1, x_2, \ldots, x_n)$, a new function $g(x_1, x_2, \ldots, x_n)$ is derived; if $g(x_1, x_2, \ldots, x_n)$ is a symmetric function, then $f(x_1, x_2, \ldots, x_n)$ is a generalized symmetric function.

Generalized symmetric functions can be denoted by polarity vectors as $S^k_{(y1, y2, \ldots, yn)}$ such that, for every minterm m_i of S, the total number of literals identical to the literal y_i in polarity vector $(y_1, y_2, \ldots, y_n)$ is k. For the n-variable Boolean function, there are 2^n polarity vectors and there are 2^n ways to create a generalized symmetric function.

For example, a Boolean function $f(a,b,c) = \bar{a}b\bar{c} + \bar{a}\,\bar{b}c + abc$ is not symmetric by Definition 1, since the minterm *abc* has no negative literals. However, if we use a polarity vector $\left(\bar{a},b,c\right)$ to generate a new function *g(a, b, c),* where $\bar{a},b,c$ in $f(a,b,c)$ will be replaced by *a, b, c* in *g(a, b, c),* $g(a,b,c) = ab\bar{c} + a\bar{b}c + \bar{a}bc$. With this replacement, the new function *g(x, y, z)* becomes a symmetric function S^2, thus making *f(a, b, c)* a generalized symmetric function with polarity vector $\left(\bar{a},b,c\right)$ denoted $S^2_{\left(\bar{a},b,c\right)}$.

3.2 Lattice Diagram

3.2.1 Introduction

There exist several structures to realize Boolean symmetric functions [4,5,6]. Universal Akers arrays (UAAs) are well-known because of their area efficiency and planar layouts [7]. Lattice diagrams [4] inherit this property from UAAs but, in several cases, are even more efficient. First, comparing them to UAAs' rectangular shapes, lattice diagrams start with tree expansion and then combine non-isomorphic nodes at the same levels, thus always forming triangle or trapezoid shapes that keep only the minimum necessary size of repeated variables. Second, instead of assuming only Shannon expansion in UAAs, lattice diagrams can use Shannon expansion and positive and/or negative Davio expansions. In

the case of synthesizing quantum circuits consisting of natural Toffoli gates, the Davio expansions lead to more efficient quantum circuits than the Shannon expansions.

Based on the structure and number of outputs, lattice diagrams can be classified into single-output or multi-output lattices. Both these types can use Shannon or any Davio expansion, any of the two Davio expansions, or any combination of these.

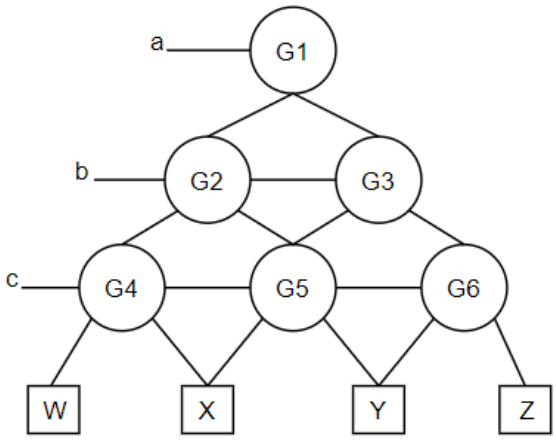


Figure 3.1. Single-output lattice diagram.

Fig. 3.1 shows a directed acyclic graph of a three-level generalized lattice diagram for a symmetric function with three input variables. In this example, G1 is the output cell controlled by variable *a*. For a lattice realizing a symmetric function of three variables, the signals W, X, Y, and Z are always constants.

Expansion type is also a critical characterization of lattice diagrams. If every cell in a lattice diagram has a uniform expansion, then the lattice can be named after the expansion typs. For instance, in a Shannon lattice, all the cells use only Shannon expansions.

Let f_0 and f_1 be the positive and negative cofactors of a Boolean function f with respect to variable x_1, respectively. Here, f_0 is $f(0, x_2, \ldots, x_n)$, where x_1 is replaced by 0, while f_1 is $f(1, x_2, \ldots, x_n)$, where x_1 is replaced by 1. $f_2 = f_0 \oplus f_1$, where the symbol $\oplus$ means exclusive-OR (XOR). We define $f_2 = f_0 \oplus f_1$ as a Boolean difference for x_1.

The Shannon expansion of $f(x_1, x_2, \ldots, x_n)$ with respect to variable x_1 [5] is defined as follows:

$$f(x_1, x_2, \ldots, x_n) = \overline{x}_1 f_0 \oplus x_1 f_1 \tag{3.1}$$

The positive Davio expansion [5] of $f(x_1, x_2, \ldots, x_n)$ with respect to variable x_1 is the following:

$$f(x_1, x_2, \ldots, x_n) = f_0 \oplus x_1 f_2 \tag{3.2}$$

The negative Davio expansion of $f(x_1, x_2, \ldots, x_n)$ with respect to variable x_1 is the following:

$$f(x_1, x_2, \ldots, x_n) = f_1 \oplus \overline{x_1} f_2 \tag{3.3}$$

In the Shannon lattice, the expansion function of each cell can be represented by a 2-to-1 multiplexer [6], the logic formulation of which is $f(a,x,y) = x \cdot \bar{a} + y \cdot a$, where *a* is the selecting signal and *x, y* are data inputs. A multiplexer symbol is shown in Fig. 3.2.

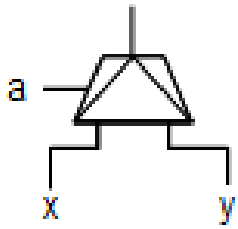


Figure 3.2. A 2-to-1 multiplexer with control variable *a* and data inputs *x, y*.

There are two types of lattice hierarchies: single-output and multi-output. Fig. 3.3 (a) is a single-output Shannon lattice, and (b) is a multi-output Shannon lattice. The lattice in Fig. 3.3 (a) has three input variables: *a, b, c*. The single-output lattice has fewer constant inputs than the multi-output lattice. The multi-output Shannon lattice has one extra level to realize a symmetric function with the same number of input variables. The top multiplexer in Fig. 3.3 (b) is hypothetical; it can be replaced by a constant in a real circuit. Both lattice diagrams can be simplified by substituting signals 0 or 1 to constant symbols.

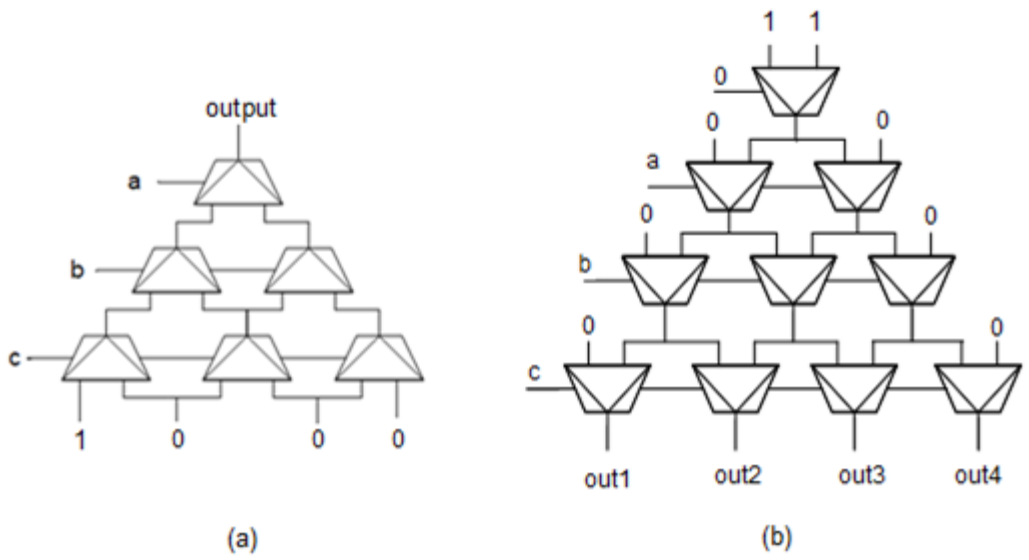


Figure. 3.3 Shannon lattices for three-variable functions: (a) single-output and (b) multiple-output.

For a positive Davio lattice, its expansion function for each cell is $f(s,a,b) = a \oplus (b \cdot s)$, which can be realized by XOR and AND gates. Fig. 3.4 shows its symbol. By changing the polarity of *s*, this cell can be used in the negative Davio lattice as well. For demonstration purposes, we will use positive Davio lattices as an example in Fig. 3.5.

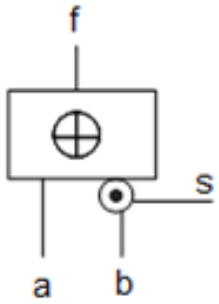


Figure 3.4 Davio gate.

Similar to Shannon lattices, Davio lattices also have two hierarchies.

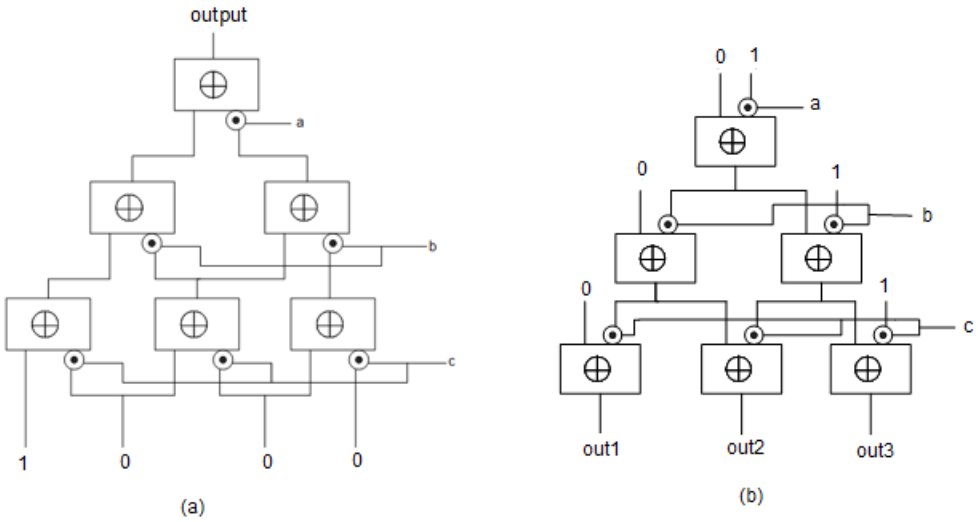


Figure 3.5 Davio lattices for three-variable functions: (a) a single-output Davio lattice and (b) a multi-output Davio lattice.

The constants of a single-output lattice can be changed for different symmetric functions. The next section discusses constants and symmetric functions in more detail. For multi-output lattices, constants are determined for symmetric functions, since this lattice can generate all symmetric functions by adding simple functions, such as OR or XOR, to outputs.

In addition to Shannon lattices and positive/negative Davio lattices, the lattice cell can also be built using mixtures of these three expansions. Fig. 3.6 shows an example of a generalized lattice. It uses Shannon expansion for variable *a*, positive Davio expansion for variable *b*, and negative Davio for variable *c*.

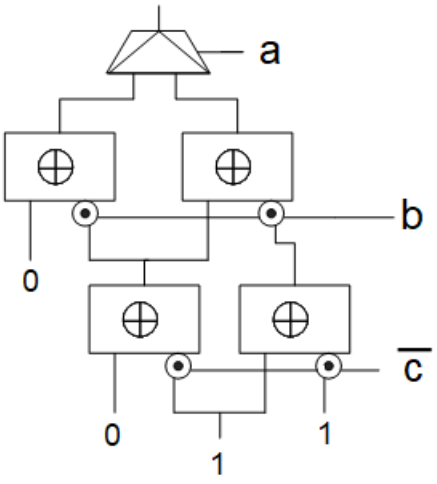


Figure 3.6 Generalized lattice diagram.

3.2.2 Realization Boolean Symmetric Functions using Lattice Diagrams

This subsection presents both single-output and multi-output Shannon lattices and discusses the relation between input constants in lattice and a related symmetric function.

A Boolean function Q(*a*,*b*,*c*), it can be implemented by lattice layout from Fig. 3.7. The expansion function of each cell is a Shannon expansion, realized by a 2-to-1 multiplexer. The level controlled by variable *c* is the terminal level of this lattice.

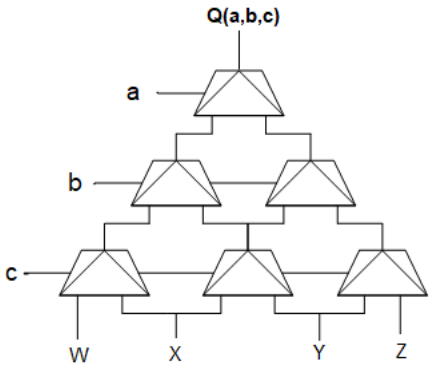


Figure 3.7 Single-output Shannon lattice.

The output function of each cell in the lattice can be derived by applying the equation of multiplexer. The output function of the lattice in Fig. 3.7 is:

$$Q(a,b,c)=W\cdot \bar{a}\,\bar{b}\,\bar{c}+X\cdot(a\,\bar{b}\,\bar{c}+\bar{a}\,b\,\bar{c}+\bar{a}\,\bar{b}\,c)+\ Y\cdot(ab\,\bar{c}+\bar{a}\,bc+a\,\bar{b}\,c)+\ Z\cdot abc \tag{3.4}$$

In this equation, each coefficient constant is linked with different terms of Q(*a*, *b* ,*c*), and each term is a symmetric index of Q(*a*, *b*, *c*). For example, $\bar{a}\bar{b}\bar{c}$ is a symmetric function

$S^0_{(a,b,c)}(a,b,c)$ = W. If we set W to 1 and X, Y, Z to 0, respectively then $S^0_{(a,b,c)}(a,b,c)$ will be the output of the lattice. The examples in this section are demonstrated with polarity vector (a, b, c) to keep the consistency. We will use $S^0(a, b, c)$, instead of $S^0_{(a,b,c)}(a,b,c)$, for a short notation in the remaining of this section.

The symmetric function Q(a, b, c) from Fig. 3.7 can be rewritten to the following format to emphasize the role of the constants W, X, Y, Z, and individual symmetry indices.

$$Q(a,b,c)=W\cdot S^0+X\cdot S^1+Y\cdot S^2+Z\cdot S^3 \tag{3.5}$$

By selecting different coefficient values, the related symmetric function will be created at the output of this lattice. The sequence of coefficient constants W, X, Y, Z at the bottom of the lattice is called a symmetry vector. We create a table for all the single symmetric indices of function Q(a, b, c) in Tab. 3.1.

Table 3.1. Symmetric function with the related vector for Shannon Lattice Diagram

Symmetric Function	Symmetry vector [W,X,Y,Z]
S^0	[1,0,0,0]
S^1	[0,1,0,0]
S^2	[0,0,1,0]
S^3	[0,0,0,1]

By changing the symmetry vector, the lattice can generate all the symmetric functions of Q(a,b,c). For example, [0,1,0,1] generates the symmetric function $S^{1,3}(a,b,c)$.

In the multi-output lattice, we need to shift the control variables to the second level and change the top multiplexer to the output constant 1. The top two levels can be simplified into outputting *a* and $\bar{a}$, an optimization discussed in section 3.3 with more detail.

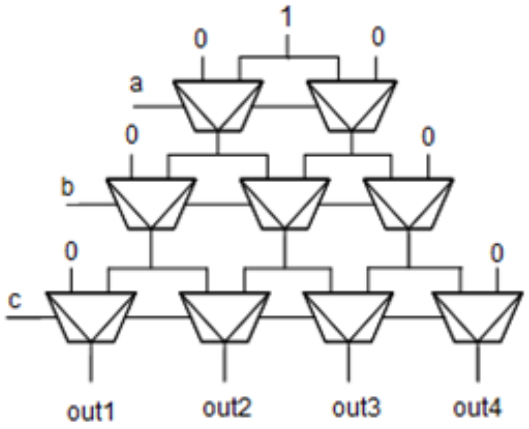


Figure 3.8. Multi-output Shannon lattice.

While the lattice in Fig. 3.7 can realize any single-output symmetric function by selecting a symmetry vector, the lattice in Fig. 3.8 has the property that all symmetric indices of Q(a,b,c) can be realized by simple logic operations on outputs out1, out2, out3, and out4.

$$
\begin{aligned}
out1 &= abc = S^3(a,b,c) \\
out2 &= ab\bar{c} + a\bar{b}c + \bar{a}bc = S^2(a,b,c) \\
out3 &= a\bar{b}\,\bar{c} + \bar{a}b\bar{c} + \bar{a}\,\bar{b}c = S^1(a,b,c) \\
out4 &= \bar{a}\,\bar{b}\,\bar{c} = S^0(a,b,c)
\end{aligned}
\qquad (3.6 \sim 3.9)
$$

This multi-output lattice is useful for multi-output Boolean functions, such as full-adder, compressor, counter, and arithmetic functions. For example, a 1-bit full-adder has three

inputs *a*, *b*, c_{in}, and two outputs *sum* and *carry-out*, both of which can be transformed into symmetric notation:

$$Sum = a \oplus b \oplus c_{in} = S^{1,3}(a,b,c_{in})$$
$$Carry\text{-}out = ab + c_{in}(a \oplus b) = S^{2,3}(a,b,c_{in}) \qquad (3.10 \sim 3.11)$$

Two XOR gates that can combine *out1* and *out2* for carry-out and *out1* and *out3* for a sum are needed to realize these two functions with a three-level multi-output lattice. Since the lattice in Fig. 3.7 realizes only a single-output function, to realize a multi-output function like the above adder, one would need two single-output lattices, as the multi-output lattice structure uses fewer gates.

3.2.3 Realizing Symmetric Function with Davio Lattices

Compared to Shannon lattices, Davio lattices use fewer quantum gates for each cell and fewer ancilla lines [10] as well as lower resource cost for a quantum circuit layout for the linear nearest-neighbor model (LNN) [10].

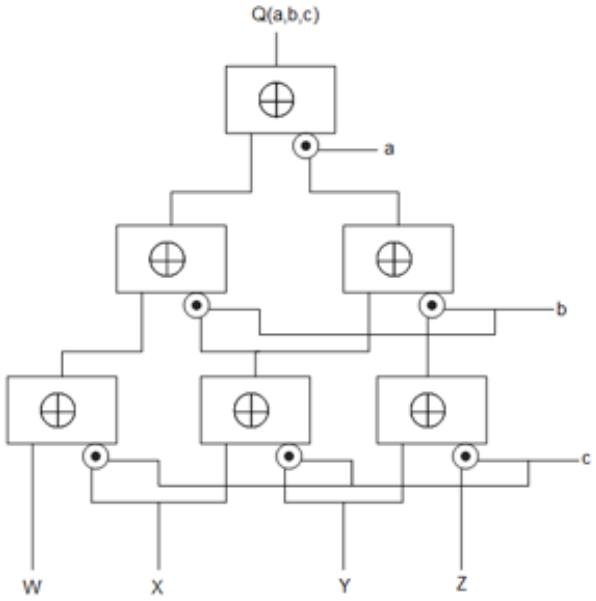


Figure 3.9. Davio lattice structure that realizes all three variable symmetric functions with W, X, Y, and Z as constants and functions with more variables in case W, X, Y, and Z are variables or simple functions.

We use the same function to show the difference in constant values between Shannon and Davio lattices. Fig. 3.9 shows a three-level positive Davio lattice, the output equation of which can be derived:

$$Q(a,b,c)=W\bigoplus X\cdot(a\bigoplus b\bigoplus c)\bigoplus Y\cdot(ab\bigoplus bc\bigoplus ac)\bigoplus Z\cdot(abc) \tag{3.12}$$

In this equation, each constant is multiplied by a term so that each is a symmetric function. For example $a\oplus b\oplus c$ contains the minterms $ab\bar{c}, a\bar{b}c, \bar{a}bc, abc$, which is $S^{1,3}$(a,b,c).

Different from the polynomial normal form of Shannon expansion, Davio expansion uses the Zhegalkin normal form; its polynomial expansion can be derived by a binary matrix called the "Zhegalkin polynomial matrix" [11], which is generated recurrently:

$$D_0=1,\ D_j = \begin{Bmatrix} D_{j-1}, D_{j-1} \\ 0, D_{j-1} \end{Bmatrix} \tag{3.13}$$

where j=1,2,…,m. Fig. 3.10 shows the matrix used in our example.

$$D_2=\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

	S^0	S^1	S^2	S^3
W	1	1	1	1
X	0	1	0	1
Y	0	0	1	1
Z	0	0	0	1

Figure 3.10. Zhegalkin polynomial matrix for three variables.

In this matrix, every row represents the constants in a Davio lattice and every column represents symmetric functions from S^0 to S^3. For example, the second row [0 1 0 1] means the related symmetric function of constant X is $S^{1,3}$. The equation of Q(a,b,c) can be expressed with symmetric functions as follows:

$$Q(a,b,c) = W \cdot S^{0,1,2,3} \bigoplus X \cdot S^{1,3} \bigoplus Y \cdot S^{2,3} \bigoplus Z \cdot S^{3} \quad (3.14)$$

Different from Shannon expansion, each constant is associated with one multiple index symmetric function. We can get the same symmetric function indices as in a Shannon lattice by selecting a different order of constants. For example, if we need function S^1, we can make both X and Z 1 and the remaining constants 0. In a Davio lattice, each part is connected with an XOR gate, so the equation $Q(a,b,c) = 0 \oplus S^{1,3} \oplus S^3 \oplus 0 = S^1$. Tab. 3.2 shows symmetric functions with related binary vectors.

Table 3.2. Symmetric function with the related vector for Positive Davio Lattice Diagram

Symmetric Function	Symmetry Vector [W,X,Y,Z]
S^0	[1,1,1,1]
S^1	[0,1,0,1]
S^2	[0,0,1,1]
S^3	[0,0,0,1]

Like multi-output Shannon lattices, Davio lattices can also be built in reverse, as shown in Fig. 3.11.

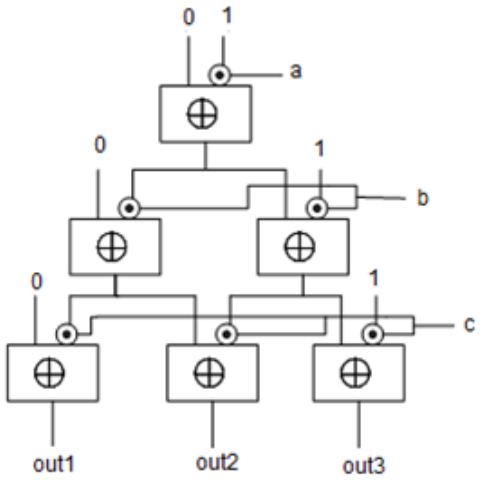


Figure 3.11. Multi-output Davio lattice with a reversed shape.

The outputs of this lattice are the following:

$$out1 = abc = S^{3}(a,b,c)$$
$$out2 = ab \oplus bc \oplus ac = S^{2,3}(a,b,c)$$
$$out3 = a \oplus b \oplus c = S^{1,3}(a,b,c) \qquad (3.15\sim3.17)$$

By using the XOR operator, we can get all multiple index symmetric functions. For example, to get $S^0(a,b,c)$, because S^0 can be derived into $S^0 = 1 \oplus S^3 \oplus S^{1,3} \oplus S^{2,3}$, we can obtain S^0 by connecting out1, out2, out3, and 1 with an XOR gate.

3.3 Quantum Implementation of Boolean Symmetric Function with Lattice Diagrams

In the previous section, we presented the layout of the Lattice diagram for the Boolean symmetric function in various formats. For the Davio Lattice diagram, its base cell Davio gate can be mapped into the quantum circuit by Toffoli gate (Introduced in Chapter 2). Shannon Lattice diagram needs a few more steps because its base cell is a 2-to-1 Multiplexer.

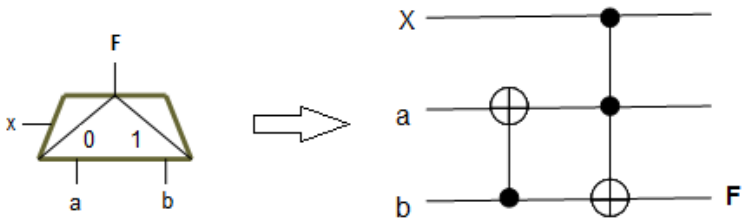


Figure 3.12 Quantum implementation of 2-to-1 Multiplexer $F(x,a,b) = \bar{x}a + xb$.

For the whole Lattice diagram, because the inputs of the root node in the Lattice diagram are always constants, the structure can be simplified. Fig. 3.13 presents an example of simplifying symmetric function S^1(a,b,c). In Fig. 3.13 (b), since both the input of the right corner cell are 0, the output of this cell is not related to c, its value is always 0. We can remove this cell and replace that place with a 0. The remaining figures follow the same method.

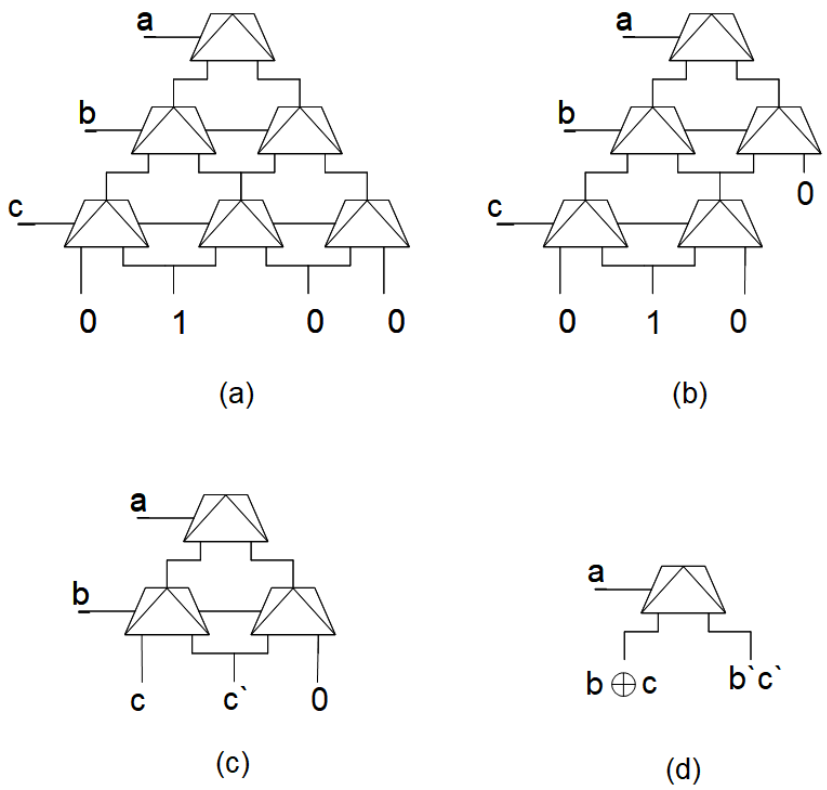


Figure 3.13 (a) the original Lattice Diagram of $S^1(a,b,c)$. (b) Removing the bottom-right cell. (c) Removing the level of input variable c. (d) Removing the level of input variable b.

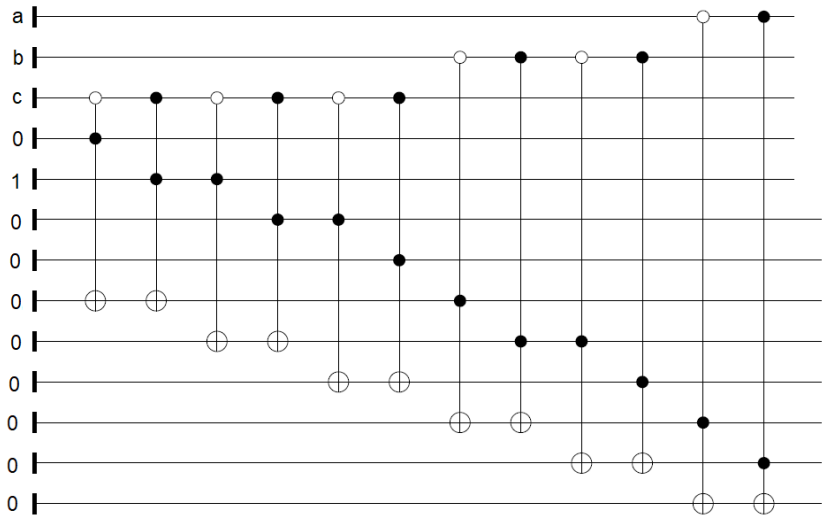


Figure 3.14 Quantum circuit of figure 3.13 (a).

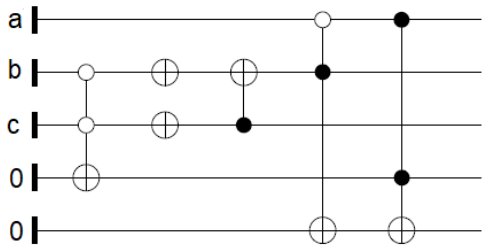


Figure 3.15 Simplified quantum circuit of figure 3.14 (a).

The original Lattice uses 16 Toffoli gates and six inverters, the simplified circuit used three Toffoli gates, one Feynman gate, and seven inverters, which obviously uses fewer gates than the original circuit and reduces significantly the quantum cost for large functions.

3.4 Realizing Non-Symmetric Functions with Lattice Diagrams

In the previous sections, we showed the advantage of Lattice Diagrams in realizing symmetric functions. For non-symmetric function, there are many methods to transform them into symmetric [36,37,38]. We purposed a new method based on reminder function decomposition.

Our decomposition method creates a circuit composed of two parts: an upper part that is a remainder function usually expressed in ESOP [38] and a lower part that is a symmetric function implemented with a lattice diagram. These two parts are connected with the XOR operator. The following is a brief outline of the decomposition methodology for function F.

(1) Choose a random polarity vector to start.

(2) Create all or some polarity vectors with corresponding lattice structures. Select a few with the simplest symmetric functions based on those polarity vectors.

(3) For all symmetric functions from step 2, calculate their remainder functions with the original function F. The remainder functions are denoted by R_0, R_1, . . ., R_k.

(4) Evaluate the cost of the remainder function by the number of literals and operators, select the minimal cost remainder function, and create the lattice diagram for its symmetric part.

The following example presents details of the above decomposition method. Consider this non-symmetric function:

$$f(a,b,c,d) = a\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c}d + \bar{a}\bar{b}c\bar{d} + bcd \tag{3.18}$$

Its Karnaugh map is Tab. 3.3 as following:

Table 3.3 Karnaugh map for function *f(a,b,c,d)*

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

First, we decide that the order of variables in the polarity vector is *a, b, c, d*. The expansion type is positive Davio expansion. In the second step, we create all 16 different symmetric functions based on polarity vectors. Next, for each symmetric function, the corresponding remainder function is calculated. For example, the decomposition of a symmetric function

with polarity vector $\left(\bar{a},\bar{b},c,d\right)$ is $S^{3}_{\left(\bar{a},\bar{b},c,d\right)}(a,b,c,d) \oplus acd \oplus a\bar{b}\,\bar{c}\,\bar{d}$. The function $acd \oplus a\bar{b}\,\bar{c}\,\bar{d}$ is the remainder function.

For polarity vector (a,b,c,d), the decomposition to a lattice and remainder is $S^{4}_{(a,b,c,d)}(a,b,c,d) \oplus \bar{a}cd \oplus \bar{a}\,\bar{b}d \oplus \bar{a}\,\bar{b}c\bar{d} \oplus a\bar{b}\,\bar{c}\,\bar{d}$. It has a more complicated remainder function: $\bar{a}cd \oplus \bar{a}\,\bar{b}d \oplus \bar{a}\,\bar{b}c\bar{d} \oplus a\bar{b}\,\bar{c}\,\bar{d}$. After comparing all possible remainder functions, we find the polarity vector $\left(\bar{a},\bar{b},c,d\right)$ leads to the minimal cost remainder function.

3.5 Summary of Chapter 3

In this chapter, The Boolean symmetric functions were introduced. Two types of their expansions: Shannon expansion, Positive/Negative Davio expansion were discussed in detail and illustrated with examples. Different structures of quantum Lattice diagrams are explored. The simplification of the quantum Lattice diagram was presented, this example was to demonstrate the area-efficient property of quantum Lattice diagrams. At the end of this section, a method of transforming non-symmetric function to symmetric function was introduced.

Chapter 4: Design Quantum Oracle for Graph Theory Problems

Note: Some of the contents of this chapter have been published below:

P. Gao, Y.Li, M.A. Perkowski and X. Song "Realization of Quantum Oracles using Symmetries of Boolean Functions", *Quantum Inf. Comput*. vol. 20 (5&6), 2020

This chapter introduces a methodology of building quantum oracles with Symmetric Boolean functions to solve graph theory problems. Graph theory problems are an excellent candidate for using quantum oracle to solve because many classical problems can be transformed into the graph theory problems.

4.1 Modeling a Graph with a Boolean Expression

In graph theory, the graph-partitioning problem is finding a subset of edges or vertices of the original graph, each of which is called a "subgraph." Based on the properties of subgraphs, there are many different decompositions. Here, we are mainly interested in searching symmetric loop in arbitrary graph and partitioning an arbitrary graph into symmetric graphs, because the symmetric graph [12] is a vital category in graph theory and topology.

Definition 1 (Graph) [47]

A graph G is a pair G = (V,E), consisting a nonempty set V of *vertices (nodes)*, a set E, disjoint from V, of edges of graph G, where $V = \{v_1,v_2,\ldots,v_i\}$, $E = \{e_1,e_2,\ldots,e_i\}$.

Definition 2 (Path, Cycle, Loop (self-loop), Degree) [47]

A *path* in a graph is a sequence of distinct vertices v_1, v_2, …, v_n such that (v_i, v_{i+1}) is an edge for each $i = 1, \ldots, n-1$. When two vertices are the end points of an edge, they are called *adjacent*.

A *cycle* is a graph with the equal number of vertices and edges whose vertices can be placed around a circle that two vertices are adjacent if and only if they appear consecutively along the circle.

A *self-loop* is an edge whose endpoints are equal.

The *degree* of vertex $v \subseteq V$, denoted *deg(v),* is the number of edges incident to *v*. If *deg(v)* =0, then vertex *v* is called *isolated*.

Definition 3 (Simple graph, Subgraph, Clique) [47]

A *simple graph* is a graph having no self-loops or multiple edges. Multiple edges are edges having the same pair of endpoints.

A *subgraph* of a graph G is a graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

In a graph, a set of pairwise adjacent vertices is called a *clique.* For a clique with n vertices, denoted as K_n.

Definition 4 (Symmetric Loop)

A graph G(E,V), if all pairs of vertices $u, v \in V$, there exists a path from *u* to *v*, and the *deg(u) = deg(v),* then graph G is a symmetric loop.

We notice that there are some interesting properties of the symmetric loop, which can be modeled by a Boolean function. Given a simple graph G(E,V), there exists a subgraph H (V(H),E(H)), in which H is a symmetric loop.

Our model begins with the feature of a vertex on the symmetric loop. Given a vertex $u \in$ V, there exists a set of edges $E_u = \{e_1, e_2, \ldots, e_i\}$, $E_u \in E$, $i = 1, \ldots, m$, where *m* is the number of edges that are connected to vertex *u*. Set E_u can be encoded with a Boolean vector B ={

$b_1, b_2, \ldots, b_i$ }, $i = 1, \ldots, m$. Boolean vector B can be represented by literal $\left\{ \hat{b}_1, \hat{b}_2, ..., \hat{b}_n \right\}$, $\hat{b}_i$ is positive if and only if its related edge $e_i \cap E(H) = e_i$, otherwise it is negative.

Rule (Symmetric Loop)

If vertex u is in symmetric loop L, then there must be at least two positive literals in its Boolean vector B, because there is one edge for entering this vertex, and another edge for exiting this vertex.

For a vertex *u* connected with *n* edges, this rule can be written into the following Boolean expression:

$$u = \bigcup_{k=1}^{\binom{n}{2}} \hat{b}_1, \hat{b}_2, ..., \hat{b}_n \tag{4.1}$$

For a symmetric loop L with *t* vertices, its expression is the following:

$$L = \bigcap_{m=1}^{t} \left(\bigcup_{k=1}^{\binom{n}{2}} \hat{b}_1, \hat{b}_2, ..., \hat{b}_n \right) \tag{4.2}$$

Theorem: For $L = \bigcap_{m=1}^{t} \left(\bigcup_{k=1}^{\binom{n}{2}} \overset{\wedge}{b_1}, \overset{\wedge}{b_2}, ..., \overset{\wedge}{b_n} \right)$, if there is a satisfied result, then a symmetric loop exists in the graph.

Proof: Based on the definition of the symmetric loop, all nodes in the loop can be expressed by a Boolean function *u*, which input is the edges connected to this node. Because if all node functions are satisfied at the same time which is the Equation *L*, the input of every function *u* shows a path which is a symmetric loop.

Based on Definition 4, we can add more rules to model some specific graphs like Hamilton cycle, spanning tree, and hypercube. In this dissertation, we are interested in the Hamilton cycles and in the hypercubes.

Definition 5 (Hamilton cycle)

Let G be a graph with n≥3 vertices. A cycle that contains every vertex of G is called a Hamilton cycle.

Similar to a symmetric loop, we can derive a rule for a Hamilton cycle. The input and output of the Boolean function is the same as the symmetric loop, the different part is using Exclusive-OR(XOR) instead of Inclusive-OR(OR). This is because a vertex should only be passed once in a Hamilton cycle.

Rule (Hamilton cycle)

The rule of Hamilton cycle H with *t* vertices can be written into the following Boolean expression:

$$H = \bigoplus_{m=1}^{t} \left(\bigcup_{k=1}^{\binom{n}{2}} \hat{b}_1, \hat{b}_2, ..., \hat{b}_n \right) \tag{4.3}$$

Where n is the number of edges connected to a vertex in Hamilton cycle H.

Definition 6 (Hypercube)

The hypercube graph Q_n is an undirected regular graph with 2^n vertices, where each vertex corresponds to a binary string of length *n*. Two vertices labeled by string *x* and *y* are connected by an edge if and only if *x* can be obtained from *y* by changing a single bit.

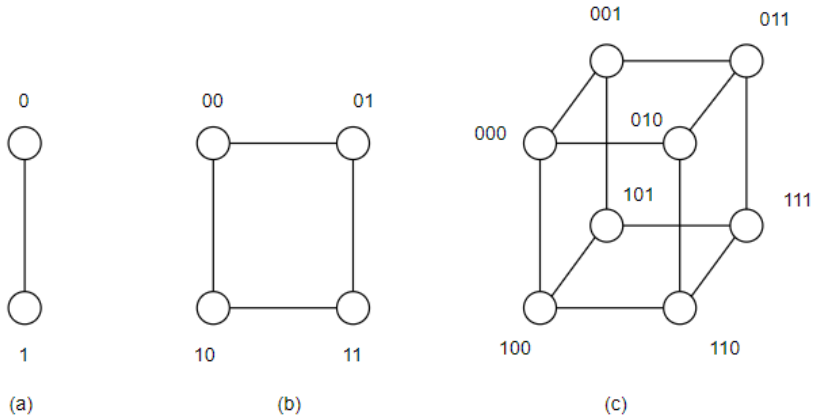


Figure 4.1 n-cube graph. (a) Q_1 for n=1 (b) Q_2 for n=2 (c) Q_3 for n=3

In Fig. 4.1, three hypercube graphs Q_1, Q_2, Q_3. Let's take Q_2 for example to explain how to get a hypercube graph following the Definition 6. There are four nodes in Q_2, and the binary strings for every adjacent node should only have one different position. For hypercube with 2^n vertices, Q_n can be defined recursively in terms of the Cartesian product of two graphs as follow:

$$Q_1 = K_2,\ Q_n\ =\ K_2 \times Q_{n-1} \tag{4.3}$$

K_2 is a clique with two nodes. It's the isomorphic graph of Q_1.

From the equation 4.3, the hypercube Q_2 can be derived using the equation $Q2 = K_2 \times Q_1$, Hypercube Q_1 can be represented by a set {0,1} which is the binary string of every node in Q_1, because Q_1 and K_2 are isomorphic. Then they can be represented by the same set. Q_2= {0,1}×{0,1} = {00,01,10,11}, because the set of binary strings of Q_2 is derived. The next step is to assign the binary string to each node and make the connections between nodes (00,01), (00,10), (01,11), (10,11).

F. Harary, J. P. Hayes's work [17] extends the original definition of hypercube to a generalized hypercube by introducing a multivalued string. In their paper, they proved that binary string encoding can be replaced by a multivalued string such as ternary or quaternary. The generalized hypercube can be defined in a Cartesian product of two clique graphs as follows:

$$Q_{(m,n)} = K_m \times K_n \ , \ m,n \neq 0 \tag{4.4}$$

Variables *m* and *n* are the number of nodes for the cliques. For example, let's assume m=2, n=3. K_2 is the same graph that we used in the previous example. For K_3, it is shown in Fig.4.2 (b), it requires ternary string to encode each node. After encoded K_2 and K_3, the next step to get the generalized hypercube $Q_{(2,3)}$ is by Cartesian product, $Q_{(2,3)}$= {0,1} ×{0,1,2} = {00,01,02,10,11,12}, Fig. 4.2 (c) shows $Q_{(2,3)}$.

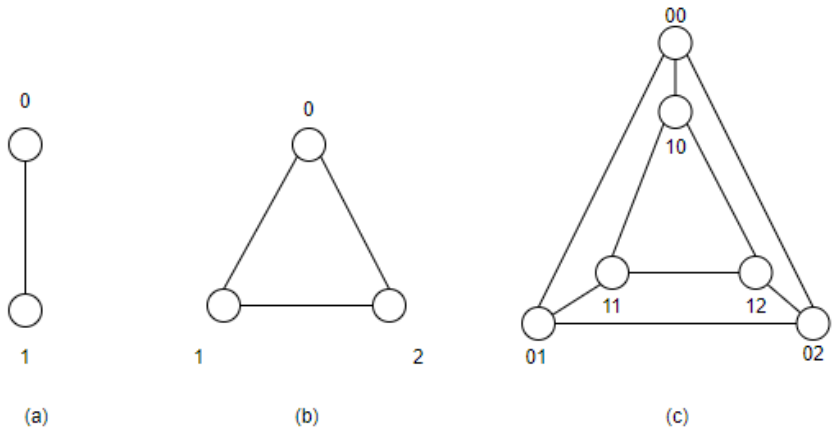


Figure 4.2 Example of generalized hypercube graph. (a) Clique K_2 (b) Clique K_3 (c) Generalized hypercube $Q_{(2,3)}$

4.2 Methodology of Building Quantum Oracles for Hamilton Cycle and Hypercube Graph

In this section, we introduce our methodology of building a quantum oracle using the Grover algorithm to solve graph partition problems related to symmetric loop, Hamiltonian cycle, and hypercube.

From the previous section, we found the Boolean expression of the symmetric loop, Hamiltonian cycle, and hypercube is only counted on the number of positive literals, which can be written into a Boolean symmetric function. With the advantage of Boolean symmetric functions and Lattice Diagrams, the oracle is made more efficient than by building it directly from Boolean equations. This is important at least because of the quantum decoherence.

There is a little different to build oracle for a hypercube, as compared to a symmetric loop and a Hamiltonian cycle. Because the degree of the hypercube is not only 2, and to model some hypercube graph requires more than one index in symmetric function, this part is discussed at the end of this section. The following steps are applicable for those three graphs and any further graph which can be transformed into a Boolean function.

Let us first consider the symmetric loop as an example. Given a graph G(E,V) with *n* vertices there exists a subgraph $L(E_L,V_L)$ which is a symmetric loop. Our problem is to find the partition of $L(E_L,V_L)$.

Step 1. A Boolean symmetric equation of every vertex needs to be derived. The classical computer would fetch the information from graph G, find the degree of all vertices, and then group them by the same degree because the vertices with the same degree have the same structure in the Lattice diagram.

Step 2. After we get the Boolean symmetric functions, we need to use a satisfiability solver (SAT solver) on the classical computer to check this equation. If this equation is not satisfiable, then we stopped here, because there is no a symmetric loop in the graph. Otherwise, we record the result of SAT solver as a reference solution to our quantum oracle.

Step 3. Based on the previous step, the size of Lattice diagrams is given to the classical computer and generates the required Lattice Diagram. Let us take the quantum loop for example, from the previous section, we know its lattice diagram is based on symmetric function S^2. The classical computer also generates a mirror circuit S^{-2} like in Fig 4.3, this mirror circuit will reset the data on the input wire to the initial state. Therefore, this data

can be used for the next blocks. The mirror circuit is an inverse circuit of S^2. The bold line with input 0s is a bus which contains results of each symmetric blocks, this bus line is the control signal of the quantum counter block as mentioned in Chapter 2. The input of the Lattice diagram is the edge set E. It is sent into the oracle in the format of bit vector.

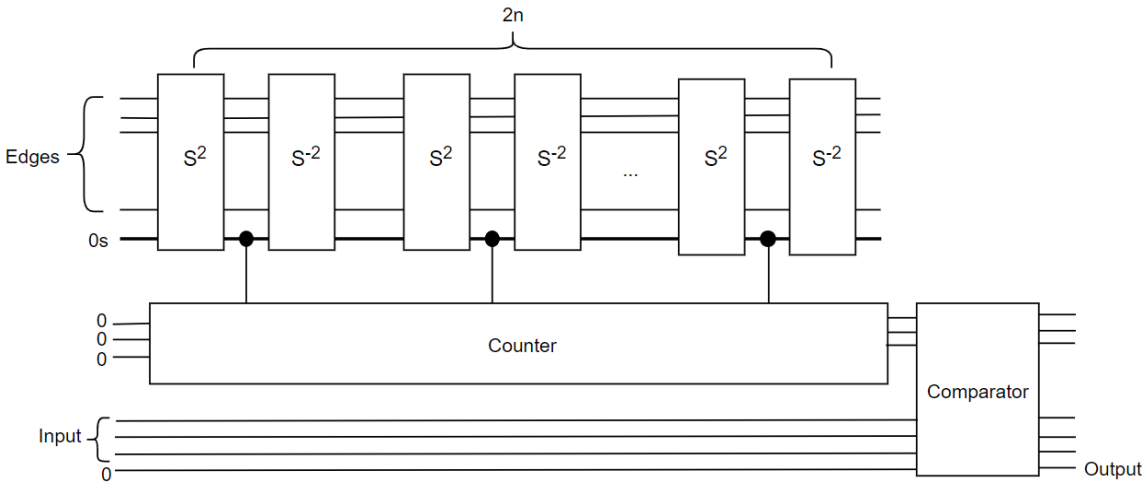


Figure 4.3 Quantum oracle for graph problem with *n* vertices

Step 4. In the oracle, we use a quantum counter and comparator (Find the details in Chapter 2) to replace the large AND gate. Because the quantum cost of the AND gate is huge (Find the details in Chapter 2), the input of the comparator is the number of vertices in graph G, we use counter here to collect the number of satisfied symmetric functions. Then this number is compared to the number of vertices, if the result of the equality comparator is 1, it means that the symmetric functions of all the vertices are satisfied.

Step 5. After the oracle is ready, we can connect it to the Grover searching module. Fig. 4.3 is a block diagram of our oracle, in Grover search we need to run the Grover search multiple iterations to get the final result, which means this oracle needs to repeat many times as well. After all the oracle and Grover searching module are set up, the simulation can start.

Step 6. After the result is returned, we can use the result from SAT solver to verify the correctness of our oracle. In some case, the problem may have multiple solutions, but Grover can only return one result per time, and during the next searching round, this result still has the same probability as other results to be found. To solve this problem, we need to remove this result from the searching space, after every round of Grover searching, we add a circuit of this result to the oracle, which makes the output gratitude of this input combination equals 0. This add-on circuit is connected after the original oracle.

Example 4.1 describes a quantum oracle designed with our methodology.

Example 4.1

Consider the following graph G=<V, E>, where V={A, B, C, D} and E={e_1, e_2, e_3, e_4, e_5, e_6} (Fig. 4.4). All Hamiltonian cycles of this graph are obtained by sets of edges from the oracle: {e_1, e_2, e_3, e_4}, {e_1, e_6, e_3, e_5}, and {e_2, e_6, e_4, e_5}.

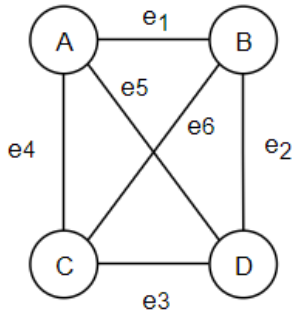


Figure 4.4 Graph G of example 4.1

For this example, we relate a symmetric function to each node (Tab. 4.1).

Table 4.1. Symmetric functions for verities in Fig. 4.4

	Vertex function	Symmetric
A	$F_A = e_1 e_4 \overline{e_5} \oplus e_1 \overline{e_4} e_5 \oplus \overline{e_1} e_4 e_5$	$S_A^2(e_1, e_4, e_5)$
B	$F_B = e_1 e_2 \overline{e_6} \oplus e_1 \overline{e_2} e_6 \oplus \overline{e_1} e_2 e_6$	$S_B^2(e_1, e_2, e_6)$
C	$F_C = e_3 e_4 \overline{e_6} \oplus e_3 \overline{e_4} e_6 \oplus \overline{e_3} e_4 e_6$	$S_C^2(e_3, e_4, e_6)$

D	$F_D = e_2 e_3 \overline{e_6} \oplus e_2 \overline{e_3} e_6 \oplus \overline{e_2} e_3 e_6$	$S_D^2(e_2, e_3, e_6)$

For node A, it Boolean function is $F_A(e_1, e_4, e_5) = e_1 e_4 \overline{e_5} \oplus e_1 \overline{e_4} e_5 \oplus \overline{e_1} e_4 e_5$, the term $e_1 e_4 \overline{e_5}$ in this expression means edge e_1 and edge e_4 are selected, edge e_5 is not selected, that is because in a Hamiltonian cycle there should be only two edges selected for a vertex. If there exists a set of selected edges that satisfy F, then at least one cycle exists. Moreover, the set of edges is a Hamiltonian cycle. As we see, we convert the graph partition problem into a Boolean satisfiability problem with several symmetric functions. Function F can then be represented in the form of a product of symmetric functions:

$$F(e_1, e_2, e_3, e_4, e_5, e_6) = S_A^2(e_1, e_4, e_5) S_B^2(e_1, e_2, e_6) S_C^2(e_3, e_4, e_6) S_D^2(e_2, e_3, e_6) \quad (4.5)$$

Each symmetric function defines a constraint for one node.

Please note that only symmetric function S^2 is used in our method here. For different graph cases, the input variables of symmetric functions need to be changed based on the edges incident to the related vertex. Based on the consistent format of symmetric function, there is a pattern of symmetry vector for different numbers of the input variables. Considering the quantum cost, we use Davio lattice in this problem. (see Chapter 2)

In Fig.4.4, we can find the symmetry vector of S^2 for three variables, which is [0 0 1 1]. As discussed in Chapter 2, we can extend the Zhegalkin polynomial matrix to get any symmetric functions with the different number of input variables, for example, Fig. 4.5 is the Zhegalkin polynomial matrix for symmetric function f (e_1, e_2, e_3, e_4, e_5, e_6).

$$\begin{pmatrix} 1\,1 & 1\,1 & 1\,1 & 1\,1 \\ 0\,1 & 0\,1 & 0\,1 & 0\,1 \\ 0\,0 & 1\,1 & 0\,0 & 1\,1 \\ 0\,0 & 0\,1 & 0\,0 & 0\,1 \\ 0\,0 & 0\,0 & 1\,1 & 1\,1 \\ 0\,0 & 0\,0 & 0\,1 & 0\,1 \\ 0\,0 & 0\,0 & 0\,0 & 1\,1 \\ 0\,0 & 0\,0 & 0\,0 & 0\,1 \end{pmatrix}$$

Figure 4.5 Zhegalkin polynomial matrix for seven input variables.

To get $S^2(e_1, e_2, e_3, e_4, e_5, e_6)$, the symmetry vector is [0 0 1 1 0 0 1]. Because of the recurrent structure in the matrix, the pattern "0 0 1 1" is repeated here. It can be easily proved that for different numbers of input variables, the symmetry vector of S^2 always contains the pattern "0 0 1 1", for example, the symmetry vector of a 5-variables function S^2(a, b, c, d, e) is [0 0 1 1 0 0]. With this property, we can estimate the number of Davio

gates needed in the general case of symmetric function S^2, it is $\frac{a^2 - a}{2}$, where *a* is the number of input variables.

Our approach to partitioning graphs into cycles uses the Grover algorithm [14]. To detect if there is a cycle in a graph, we must check all possible combinations of the selected edges in the graph, and the Grover algorithm promises a quadratic acceleration when searching through all possible combinations of edges. The edges are encoded into a binary vector as the input of the oracle. Meanwhile, the output result is also returned as a binary vector. Knowing the incidence matrix of the graph, the node base representation of all cycles can be found by a classical computer.

The oracle for solving this problem needs three blocks: a set of symmetric functions (one for each node), a counter of satisfied nodes, and an equality comparator. The symmetric function block is built with lattice diagrams, as shown in the previous Chapter. The counter used here counts the number of satisfied symmetric functions in Equation F. This block is also known as a "counter of ones". Information about the structure of the counter and comparator can be found in Chapter 2. We realized this counter by repeating increment gates [15], a single increment gate adds 1 to the value of the input when its control is activated. When it is not activated, the gate does nothing. This step can also be realized

with a multiple-input Toffoli gate, but with the higher number of inputs, the quantum cost of a Toffoli gate rises dramatically. By using the counter instead of a Toffoli gate as a global AND, we reduce the number of respective ancilla qubits from *n* to log *n*. It was for this reason that we have chosen the counter in our design. Because we use the counter instead of a Toffoli gate, we need an equality comparator to check if all symmetric functions in equation F are satisfied.

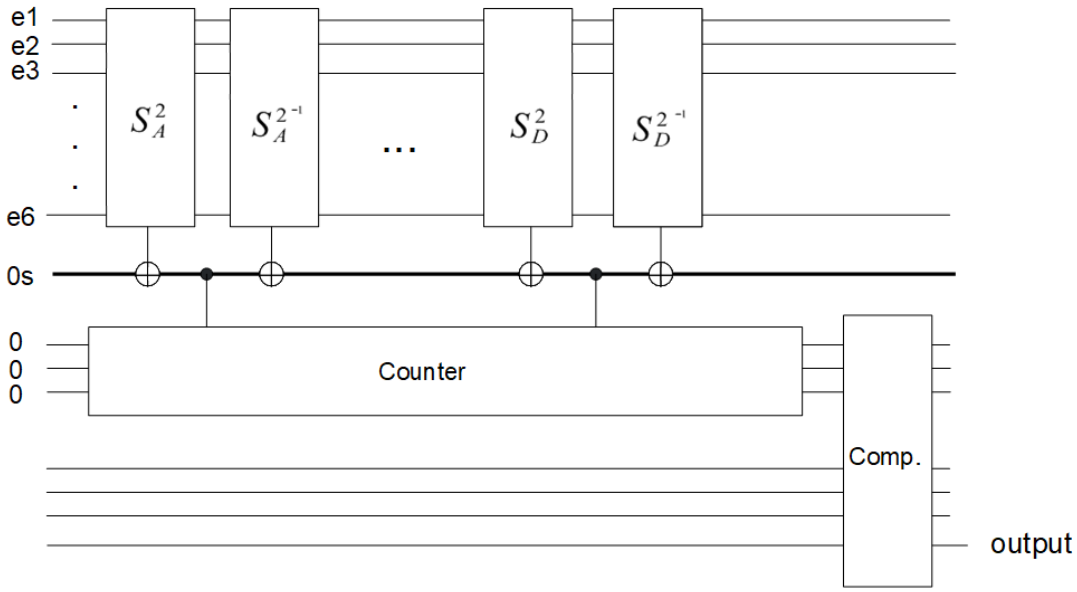


Figure 4.6 Quantum oracle for example 4.1

4.3 Hypercube Partitioning Problem

In the previous section, based on the subgraph the index of symmetric function could be different in the constraint block, with a small modification, our oracle can be changed for searching generalized hypercube graph.

Our oracle is designed for finding all partitions of an arbitrary undirected graph to regular graphs. Different from the graph partitioning in the previous section, for hypercube partitioning, the degree of every node in a hypercube subgraph is not limited to 2. The index of our symmetric function should be defined by the category of the subgraph.

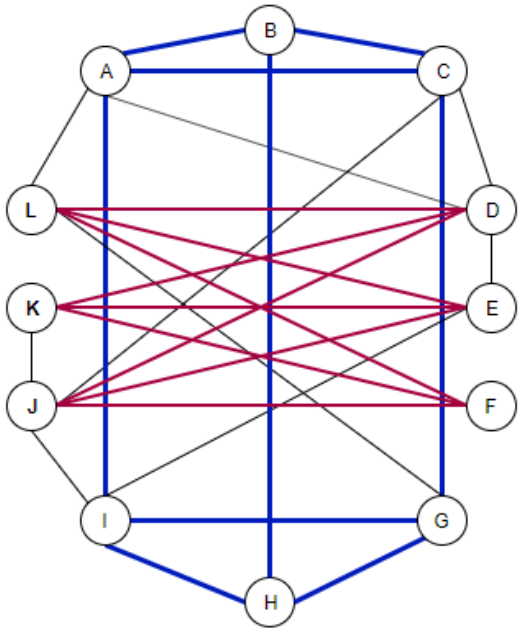


Figure 4.7 Example graph for hypercube partitioning

In Fig. 4.7 the minimum degree of this graph is 3, so we choose S^3 as our symmetric function to build the oracle. Since the size of this example is larger than the previous one, here only list the symmetric function of vertices A, B, D, since vertex D is the only vertex has degree 6 in the graph, the functions of vertex A and B are the most used structures in the oracle. The lattice diagrams of these three symmetric functions are the templates used in our oracle, by switching the input edges, the lattice diagram can be applied to the vertex with the same number of degree. For example, vertex A and C has the same degree, after we created the lattice diagram of vertex A, switching this lattice's input edges (e1, e2, e3, e4, e5) to (e4, e7, e8, e9, e10), the output of this new lattice is $F_C(e4,e7,e8,e9,e10) = S^3(e4,e7,e8,e9,e10)$. The mirror circuit in the oracle can also to be created using this trick.

$$\begin{aligned} &F_A(e1,e2,e3,e4,e5) = S^3(e1,e2,e3,e4,e5) \\ &= \overline{e1}\,\overline{e2}e3e4e5 \oplus \overline{e1}e2\overline{e3}e4e5 \oplus \overline{e1}e2e3\overline{e4}e5 \\ &\quad \oplus \overline{e1}e2e3e4\overline{e5} \oplus e1\overline{e2}\,\overline{e3}e4e5 \oplus e1\overline{e2}e3\overline{e4}e5 \\ &\quad \oplus e1\overline{e2}e3e4\overline{e5} \oplus e1e2\overline{e3}\,\overline{e4}e5 \oplus e1e2\overline{e3}e4\overline{e5} \\ &\quad \oplus e1e2e3\overline{e4}\,\overline{e5} \end{aligned} \tag{4.6}$$

$$F_B(e5,e6,e7) = S^3(e5,e6,e7) = e5e6e7 \tag{4.7}$$

$$
\begin{aligned}
&F_C(e4,e7,e8,e9,e10) = S^3(e4,e7,e8,e9,e10)\\
&= \overline{e4}\,\overline{e7}e8e9e10 \oplus \overline{e4}e7\overline{e8}e9e10 \oplus \overline{e4}e7e8\overline{e9}e10\\
&\quad \oplus \overline{e4}e7e8e9\overline{e10} \oplus e4\overline{e7}\,\overline{e8}e9e10 \oplus e4\overline{e7}e8\overline{e9}e10\\
&\quad \oplus e4\overline{e7}e8e9\overline{e10} \oplus e4e7\overline{e8}\,\overline{e9}e10 \oplus e4e7\overline{e8}e9\overline{e10}\\
&\quad \oplus e4e7e8\overline{e9}\,\overline{e10}
\end{aligned}
\tag{4.8}
$$

$$
\begin{aligned}
&F_D(e3,e10,e11,e12,e13,e14) = S^3(e3,e10,e11,e12,e13,e14)\\
&= \overline{e3}\,\overline{e10}\,\overline{e11}e12e13e14 \oplus \overline{e3}\,\overline{e10}e11\overline{e12}e13e14 \oplus \overline{e3}\,\overline{e10}e11e12\overline{e13}e14\\
&\quad \oplus \overline{e3}\,\overline{e10}e11e12e13\overline{e14} \oplus \overline{e3}e10\overline{e11}\,\overline{e12}e13e14 \oplus \overline{e3}e10\overline{e11}e12\overline{e13}e14\\
&\quad \oplus \overline{e3}e10\overline{e11}e12e13\overline{e14} \oplus \overline{e3}e10e11\overline{e12}\,\overline{e13}e14 \oplus \overline{e3}e10e11\overline{e12}e13\overline{e14}\\
&\quad \oplus \overline{e3}e10e11e12\overline{e13}\,\overline{e14} \oplus e3\overline{e10}\,\overline{e11}\,\overline{e12}e13e14 \oplus e3\overline{e10}\,\overline{e11}e12\overline{e13}e14\\
&\quad \oplus e3\overline{e10}\,\overline{e11}e12e13\overline{e14} \oplus e3\overline{e10}e11\overline{e12}\,\overline{e13}e14 \oplus e3\overline{e10}e11\overline{e12}e13\overline{e14}\\
&\quad \oplus e3\overline{e10}e11e12\overline{e13}\,\overline{e14} \oplus e3e10\overline{e11}\,\overline{e12}\,\overline{e13}e14 \oplus e3e10\overline{e11}\,\overline{e12}e13\overline{e14}\\
&\quad \oplus e3e10\overline{e11}e12\overline{e13}\,\overline{e14} \oplus e3e10e11\overline{e12}\,\overline{e13}\,\overline{e14}
\end{aligned}
\tag{4.9}
$$

$$
\begin{aligned}
&F_E(e11,e15,e16,e17,e18) = S^3(e11,e15,e16,e17,e18)\\
&= \overline{e11}\,\overline{e15}e16e17e18 \oplus \overline{e11}e15\overline{e16}e17e18 \oplus \overline{e11}e15e16\overline{e17}e18\\
&\quad \oplus \overline{e11}e15e16e17\overline{e18} \oplus e11\overline{e15}\,\overline{e16}e17e18 \oplus e11\overline{e15}e16\overline{e17}e18\\
&\quad \oplus e11\overline{e15}e16e17\overline{e18} \oplus e11e15\overline{e16}\,\overline{e17}e18 \oplus e11e15\overline{e16}e17\overline{e18}\\
&\quad \oplus e11e15e16\overline{e17}\,\overline{e18}
\end{aligned}
\tag{4.10}
$$

$$F_F(e19,e20,e21) = S^3(e19,e20,e21)$$
$$= e19e20e21 \tag{4.11}$$

$$F_G(e9,e22,e23,e24) = S^3(e9,e22,e23,e24)$$
$$= \overline{e9}e22e23e24 \oplus e9\overline{e22}e23e24 \oplus e9e22\overline{e23}e24 \tag{4.12}$$
$$\oplus e9e22e23\overline{e24}$$

$$F_H(e6,e24,e25) = S^3(e6,e24,e25)$$
$$= e6e24e25 \tag{4.13}$$

$$F_I(e2,e18,e23,e25,e26) = S^3(e2,e18,e23,e25,e26)$$
$$= \overline{e2}\,\overline{e18}e23e25e26 \oplus \overline{e2}e18\overline{e23}e25e26 \oplus \overline{e2}e18e23\overline{e25}e26$$
$$\oplus \overline{e2}e18e23e25\overline{e26} \oplus e2\overline{e18}\,\overline{e23}e25e26 \oplus e2\overline{e18}e23\overline{e25}e26$$
$$\oplus e2\overline{e18}e23e25\overline{e26} \oplus e2e18\overline{e23}\,\overline{e25}e26 \oplus e2e18\overline{e23}e25\overline{e26} \tag{4.14}$$
$$\oplus e2e18e23\overline{e25}\,\overline{e26}$$

$$
\begin{aligned}
&F_J(e8,e27,e17,e21,e26,e27) = S^3(e8,e27,e17,e21,e26,e27)\\
&= \overline{e8}\,\overline{e27}\,\overline{e17}e21e26e27 \oplus \overline{e8}\,\overline{e27}e17\overline{e21}e26e27 \oplus \overline{e8}\,\overline{e27}e17e21\overline{e26}e27\\
&\quad \oplus \overline{e8}\,\overline{e27}e17e21e26\overline{e27} \oplus \overline{e8}e27\overline{e17}\,\overline{e21}e26e27 \oplus \overline{e8}e27\overline{e17}e21\overline{e26}e27\\
&\quad \oplus \overline{e8}e27\overline{e17}e21e26\overline{e27} \oplus \overline{e8}e27e17\overline{e21}\,\overline{e26}e27 \oplus \overline{e8}e27e17\overline{e21}e26\overline{e27}\\
&\quad \oplus \overline{e8}e27e17e21\overline{e26}\,\overline{e27} \oplus e8\overline{e27}\,\overline{e17}\,\overline{e21}e26e27 \oplus e8\overline{e27}\,\overline{e17}e21\overline{e26}e27\\
&\quad \oplus e8\overline{e27}\,\overline{e17}e21e26\overline{e27} \oplus e8\overline{e27}e17\overline{e21}\,\overline{e26}e27 \oplus e8\overline{e27}e17\overline{e21}e26\overline{e27}\\
&\quad \oplus e8\overline{e27}e17e21\overline{e26}\,\overline{e27} \oplus e8e27\overline{e17}\,\overline{e21}\,\overline{e26}e27 \oplus e8e27\overline{e17}\,\overline{e21}e26\overline{e27}\\
&\quad \oplus e8e27\overline{e17}e21\overline{e26}\,\overline{e27} \oplus e8e27e17\overline{e21}\,\overline{e26}\,\overline{e27}
\end{aligned}
\tag{4.15}
$$

$$
\begin{aligned}
&F_K(e13,e16,e20,e27) = S^3(e13,e16,e20,e27)\\
&= \overline{e13}e16e20e27 \oplus e13\overline{e16}e20e27 \oplus e13e16\overline{e20}e27\\
&\quad \oplus e13e16e20\overline{e27}
\end{aligned}
\tag{4.16}
$$

$$
\begin{aligned}
&F_L(e1,e12,e15,e19,e22) = S^3(e1,e12,e15,e19,e22)\\
&= \overline{e1}\,\overline{e12}e15e19e22 \oplus \overline{e1}e12\overline{e15}e19e22 \oplus \overline{e1}e12e15\overline{e19}e22\\
&\quad \oplus \overline{e1}e12e15e19\overline{e22} \oplus e1\overline{e12}\,\overline{e15}e19e22 \oplus e1\overline{e12}e15\overline{e19}e22\\
&\quad \oplus e1\overline{e12}e15e19\overline{e22} \oplus e1e12\overline{e15}\,\overline{e19}e22 \oplus e1e12\overline{e15}e19\overline{e22}\\
&\quad \oplus e1e12e15\overline{e19}\,\overline{e22}
\end{aligned}
\tag{4.17}
$$

Under this constraint, quantum simulation result finds partition {{A, B, C, I, H, G}, {D, E, F, L, K, J}}. The set {A, B, C, I, H, G} here is a generalized hypercube $K_2 \times K_3$ showed with blue edges in Fig.4.7, another set {D,E,F,L,K,J} with red edges is a K(3,3). In this

example, S^3 is just for demonstration, during the real problem, the degree can be changed to find particular hypercube graphs or symmetric graphs in an arbitrary undirected graph. Papers [39,40] investigates more general cases of the relationship between Hypercube graphs and symmetric graphs.

Table 4.2 The number of symmetric graphs related to the degree of vertex and number of vertices. [40]

n	degree													total
	0	1	2	3	4	5	6	7	8	9	10	11	12	
1	1													1
2	1	1												2
3	1		1											2
4	1	1	1	1										4
5	1		1		1									3
6	1	1	2	2	1	1								8
7	1		1		1		1							4
8	1	1	2	3	3	2	1	1						14
9	1		2		3		2		1					9
10	1	1	2	3	4	4	3	2	1	1				22
11	1		1		2		2		1		1			8
12	1	1	4	7	11	13	13	11	7	4	1	1		74
13	1		1		3		4		3		1		1	14
14	1	1	2	3	6	6	9	9	6	6	3	2	1	56
15	1		3		8		12		12		8		3	48
16	1	1	3	7	16	27	40	48	48	40	27	16	7	286

In Tab. 4.2 some patterns can be easily found. For example, the numbers 1 on the diagonal of this table correspond to clique graphs which can be easily verified. The graphs in column with label 2 correspond to graphs with degree=2. If the number of vertices *n* is a prime then there is only one related symmetric graph, and this graph is a cycle [40]. For those non-prime numbers, those graphs may contain subgraphs, for n=6, one of the solutions is a cycle with six nodes, another solution is built with two cycles of three nodes each. Similarly,

generalized hypercube graphs can be found. For example, n=6 degree =3 is a hypercube of $\{0, 1\} \times \{0, 1, 2\}$ which is shown in Fig. 4.2(c). But there is another symmetric graph with the same number of vertices and degree, this graph is K(3,3), unfortunately, our quantum oracle cannot distinguish these two graphs, we need to do that step on a classical computer. Since hypercube graph is generated by Cartesian product of clique graph, so its minimum chromatic number should be the largest clique in the hypercube [39]. By comparing the chromatic number of the graphs we can find hypercube from other symmetric graphs. As we see the chromatic number of K(3,3) is 2 while the chromatic number of generalized hypercube $\{0, 1\} \times \{0, 1, 2\}$ is 3. Similarly, we can distinguish generalized hypercubes among symmetric graphs for other values of number of nodes and degree of edges.

Besides the complete hypercube, our oracle can be extended to find partial hypercubes as well. In partial hypercube partitioning, we find all subsets of nodes that are sub-hypercubes (cliques) but we allow nodes that do not belong to any sub-hypercube.

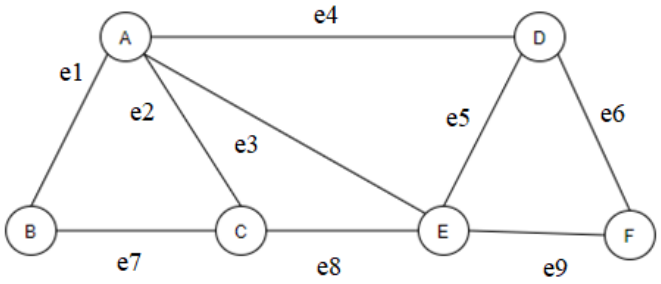


Figure 4.8 Example graph for partial hypercube

Thus, for the graph in Fig. 4.8 all individual subgraph partitions: {{A, B, C}, {D, E, F}}, {{A, C, D, E}, {B}, {F}}, {{A, B, C, E}, {D, F}} and many others will be effective solutions. The small modification of oracle is to allow the subsets of nodes. Thus, change can be implemented by adding the index of this node's symmetric function, for example, if we set the symmetric function of node B and F to S0,1, node A, C, D and E to S2.

$$S_A^2(e1,e2,e3,e4) = \overline{e1}\,\overline{e2}e3e4 \oplus \overline{e1}e2\overline{e3}e4 \oplus \overline{e1}e2e3\overline{e4} \oplus e1\overline{e2}\,\overline{e3}e4 \oplus e1\overline{e2}e3\overline{e4} \oplus e1e2\overline{e3}\,\overline{e4} \tag{4.18}$$

$$S_B^{0,1}(e1,e7) = \overline{e1}\,\overline{e7} \oplus e1\overline{e7} \oplus \overline{e1}e7 \tag{4.19}$$

$$S_C^2(e2,e3,e7,e8) = \overline{e2}\,\overline{e3}e7e8 \oplus \overline{e2}e3\overline{e7}e8 \oplus \overline{e2}e3e7\overline{e8} \oplus e2\overline{e3}\,\overline{e7}e8 \oplus e2\overline{e3}e7\overline{e8} \oplus e2e3\overline{e7}\,\overline{e8} \tag{4.20}$$

$$S_D^2(e4,e5,e6) = \overline{e4}e5e6 \oplus e4\overline{e5}e6 \oplus e4e5\overline{e6} \tag{4.21}$$

$$S_E^2(e3,e5,e8,e9) = \overline{e3}\,\overline{e5}e8e9 \oplus \overline{e3}e5\overline{e8}e9 \oplus \overline{e3}e5e8\overline{e9} \oplus e3\overline{e5}\,\overline{e8}e9 \oplus e3\overline{e5}e8\overline{e9} \oplus e3e5\overline{e8}\,\overline{e9} \tag{4.22}$$

$$S_F^{0,1}(e6,e9) = \overline{e6}\,\overline{e9} \oplus \overline{e6}e9 \oplus e6\overline{e9} \tag{4.23}$$

With this added index 0, When searching the hypercube, our oracle would consider nodes B and F could be a scatter node in a possible solution. Then the subgraph with nodes {A, C, D, E} can be found. The partial partitioning extends the ability our methodology in finding symmetric subgraphs in general graphs.

4.4 Quantum Simulators and Tools

There are two quantum simulators used in our experiments: Quipper, and Qiskit.

Quipper is an embedded functional programing language for quantum computation, developed by a group of researchers at Dalhousie University. It uses Haskell as the host language, with its data types, combinators, and a library of Haskell functions. Quipper uses an extended model of quantum computation. It enables both quantum and classical wires and operations in a circuit. Although Quipper is a good platform for simulating quantum circuits, it has also some drawbacks. One of the drawbacks is that its code is not portable and is difficult to debug, because the compiler presents the compilation errors in the host language, not the embedded language. Another drawback is that Haskell lacks some features that are useful in quantum programming: linear type and dependent type. Quipper was developed as a part of a funded project. It did not have good support after that project was finished. The last update for this package was sent out in 2019.

Besides Quipper, there are many other quantum simulators developed for researchers, Microsoft's Q#, IBM's Qiskit, and Intel's IQS. IQS is recently announced by Intel, it is still not open for public access. Q# is similar to Quipper, it is embedded with Microsoft's own functional language F#, compared to Quipper, Q# has better support, but it also has drawbacks like Quipper because of the property of embedded language. We have tried Q# with some of our oracles, but it does not have good support for gate-level simulation, and it is difficult to fix errors in the low-level quantum simulation. Then we move to Qiskit. Qiskit is an open-source quantum computing framework, it is based on Python. Qiskit

contains four libraries: Terra, Aqua, Aer, Ignis. Terra covers all low-level quantum computing like gate design, Aqua, and Aer provides quantum simulation and emulation from algorithmic-level to gate-level, Ignis includes constructors related to quantum hardware characterization, verification, and correction. Taken together Qiskit can provide the most comprehensive software solution for quantum computing. In our experiment, we mainly use Aer which can provide a gate-level simulation.

Besides the quantum simulators, we also use SAT solver(miniSAT) and reversible circuit synthesizer(RevKit) to verify and evaluate our results.

4.5 Result Analysis and Quantum Cost Estimation

The quantum oracle used in this chapter mainly consist of the following three parts: the block representing graph, the counter, and the equality comparator. The block representing graph is composed of various symmetric function blocks realized in quantum circuit by lattice diagrams, one for each vertex of the graph. The quantum cost of the oracle is estimated by adding the quantum costs of these three parts for an arbitrary graph. Because of the linear layout [18] of the lattice diagram, the size of the symmetric function block is proportional to the number of its input variables. As we discussed in Chapter 3 the number of Davio gates needed for a single symmetric function block, and the quantum cost of Davio gate is equal to CNOT gate. Thus, the cost of one symmetric function block S_x^2 can

be estimated as $\frac{5(d^2-d)}{2}$, where *d* is the number of edges connected to the vertex *x*.

Because we need a mirror circuit for every vertex, the cost of the circuit for one vertex is $5d^2 - 5d$. The total cost of the block representing the graph is the sum of the costs of circuits for all single vertices. Let us denote the number of edges incident to vertex *i* by d_i. Therefore, the cost of the block representing the graph is the following:

$$\sum_{i=1}^{V} 5(d_i^2 - d_i) \tag{4.24}$$

where *V* is the number of vertices in the graph. The counter is realized with multiple increment gates. A well-known implementation of increment gates is built with multiple-input Toffoli gates and an inverter. Reference [18] proved that the quantum cost of an m-qubit Toffoli gate is $2^{m+1} - 3$ The cost of a single increment gate in our oracle is:

$$2^{\log_2 V + 1} + 3\log_2 V - 1 \tag{4.25}$$

Because this gate requires $log_2 V$ qubits. The number of repetitions of the increment gates is *V*, and the total cost of the counter block is $V\left(2^{\log_2 V + 1} + 3\log_2 V - 1\right)$. The equality comparator block is built with two CNOT gates and one *N* input Toffoli gate, where *N* is the size of the input numbers to the comparator. *N* is $\log_2 V$ in our oracle, so

the cost of the comparator block is $2^{\log_2 V+1}+5\log_2 V-3$. The total cost of the oracle is the sum of the above three costs:

$$\mathrm{V}(2^{log_2 V+1}+3log_2 V-1)+5log_2 V+2^{log_2 V+1}-3+\sum_{i=1}^{V}(20d_i-10) \tag{4.26}$$

The qubits needed for this oracle are divided into two groups. One group represents the search space with superpositions, and the other is the ancilla qubits initialized with constants. The qubits needed for the search space are the number of edges in the graph. For the ancilla qubits, the counter and comparator blocks need $2\log_2 V+2$ in total. The ancilla qubits for the symmetric blocks are based on the maximum number of edges connected to the vertex in the graph.

To evaluate our quantum oracle, we use RevKit as a comparison, which is a well-known reversible circuit synthesis toolkit. It is an open-source package for reversible circuit design and synthesis, for the synthesis part, it provides a variety of methods like BDD, KFDD, transform-based, heuristic synthesis. For the work of this chapter we used the BDD based method for comparison. Here we only compare the symmetric function part, because other blocks in our oracle like the comparator are not using Lattice diagrams. The cost of those parts is the same for different synthesis methods. The oracle function used in Tab. 4.3 is from example 4.1.

Table 4.3 Comparisons of different synthesis methods

Synthesis Method	Number of quantum gates	Ancilla qubits
Revkit(BDD-based)	142	1
Shannon Lattice Method	50	22
Davio Lattice Method	35	9

The results in Tab. 4.3 do not involve the consideration of the mirror circuit, because it is easy to get the result by doubling the number of quantum gates. The results present the advantage of area efficiency of our oracle, to realize the same function, our oracle cost fewer gates and ancilla lines than the BDD-based method.

4.6 Summary of Chapter 4

Two graph theory problems were formulated at the beginning of this chapter. Then we transformed those graph problems into Boolean satisfiability problems (SAT problems). A methodology of building quantum oracle to solve those SAT problems was introduced and presented in detail by examples. At the end of this chapter, the quantum complexity of our oracle was discussed, and the quantum costs of our quantum oracle with different realizations are compared, those results were based on Quipper and Revkit.

Chapter 5: Hybrid Quantum/Classical Algorithm to Minimize Switching Functions based on Graph Partitions

In this chapter we introduce a new methodology to build a hybrid quantum oracle to solve the minimization of switching functions, this is based on my previously discussed quantum oracle in chapter 4. This hybrid oracle is discussed in detail with two examples.

5.1 Introduction

Minimization of Boolean functions is a classical problem in logic synthesis of VLSI circuit design, it has many applications like symbolic minimization of logic function and Boolean decomposition.

There are many logic minimization tools like Espresso and MINI, they are based on heuristic iterative improvement techniques, in general, those tools can only give suboptimal results and no clues for how far to get the global minimum. My method involves transferring this problem into a graph partition problem and using quantum searching to find the global minimum result for arbitrary Boolean functions. Unfortunately, large size NP-hard problems must be solved to find the exact solution. Often the methods require also to generate astronomic numbers of prime implicants. Solving these problems with classical

computers, even parallel computers, seems to not lead to interesting results and even not much has been published in recent years on these topics. However, future quantum computers give a promise. With the fast development of quantum circuits, several researchers focus on creating quantum algorithms for problems in graph theory. Although now only small problems can be solved, future quantum computers will be able to achieve *"quantum advantage"*. This gives promise to the work presented here that is not practical at the moment.

Our methodology is a hybrid quantum-classical algorithm, because the quantum computer can only take qubits as the input data. We need to use the classical computer to prepare the input data for the quantum computer and for collecting the results.

Definition 1 (Cube)

Let x_i be a Boolean variable, A Boolean function with n inputs is defined as:

$$f(x_1, x_2, ..., x_n): P_1 \times P_2 \times ... \times P_n \rightarrow B \tag{5.1}$$

where $P_i = \{0, 1\}$ $B = \{0, 1\}$, Let $S_i \subseteq P_i$, then $x_i^{S_i} = 1$ $if\ x_i \in S_i$ otherwise it is 0. $x_i^{S_i}$ is called a literal of variable x_i. Boolean product of literals is called a product term or a cube.

Definition 2 (Majority degree)

In the compatibility graph G(E, V), the verities have the same degree can be grouped into a set. For the set which contains the most number of vertices, its degree is called the majority degree.

Definition 3 (Implicant)

A product implicant of a logic function is a product term such that if the term is true then so is the function.

A prime implicant is an implicant that is not fully contained within any other implicants.

We assume that the readers are familiar with basic operations of the cube such as union, intersection, disjoint sharp. A formal definition of those terms can be found in [26].

In this section, we are focusing on Disjoint Sum of Product (DSOP) which is one of the operations not commonly used to minimize the Boolean switching functions. A DSOP realization of a Boolean function can be represented as a hypercube graph in which a realization with *disjoint products of literals* corresponds to *disjoint partitioning* to sub-hypercubes. This can be extended to *Sum of Products* (SOP) [20,21] by removing literals from each disjoint cube thus transforming it to a prime implicant.

5.2 Solving DSOP/SOP and Minimization Problems for Boolean Functions using Partial Hypercube Partitioning

We present a small example of partial hypercube partitioning in Chapter 4. In this chapter, we apply our oracle to solve the DSOP/SOP minimization problems. Normally, SOP minimization is reduced to finding the set of all prime implicants (primes) and next solving the set-covering problem to cover all true minterms with set of primes of the lowest cost. We follow the approach [19] to find DSOP first. For instance, in one variant our hybrid algorithm solves the DSOP minimization by finding partitions to large product implicants first and follows with partitions to smaller products. The result is not optimal but we obtain the quadratic speedup to the quantum component of this problem. DSOP can be transformed to SOP equations by enlarging each product implicant to the cheapest prime implicant [19].

DSOP minimization. All minterms included in a product implicant are pairwise compatible [19] so the nodes of these minterms are all pairwise connected by edges in the graph (a clique, a sub-hypergraph). For instance, for a Boolean function specified by the set of minterms 0000, 0001, 0100, 0101, 0111, 0110, 1111, 1110 the minterms 0000, 0001, 0100, 0101 create a clique or a 3-Regular subgraph that is disjoint from the other 3-Regular subgraph 0111, 0110, 1111, 1110. This complete partition is a disjoint partition (clique

covering) leading to a DSOP solution $\bar{a}c + bc$ of this function. This is also an optimal SOP, as products $\bar{a}c$ and bc are disjoint. In another DSOP variant the subgraph {0100, 0101, 0111, 0110} is found which corresponds to prime $\bar{a}b$ to be next used in covering. The detailed presentation of similar algorithms can be found in [19], for which in our recent work presented here we created quantum oracles for the Grover algorithm.

SOP minimization. Every product implicants found from the DSOP found is individually extended to the largest prime for SOP using the method from [19]. In rare cases, but only in unspecified Boolean functions, minterms in a clique can be pairwise compatible but not compatible as a group thus they do not create a product implicant [19]. In this case, a special transformation is done [19] to create a SOP or a three-level circuit is synthesized.

Our hybrid algorithm contains three parts: **preprocessing, quantum search, and postprocessing**. The first part is the data preprocessing at the classical computer, the second part uses our quantum oracle to performing Grover quantum searching as a subroutine, the last part is collecting and transforming the result from the second part, this part is running on a classical computer.

Preprocessing (Classical Computer)

The input of our algorithm is an arbitrary Boolean function, for the input Boolean function, firstly, my algorithm would transfer this function into a compatibility graph.

The input function is read in format with *On/Off* set, every implicant in *On* set becomes a node in the compatibility graph, then perform union operator to generate edges in the graph.

algorithm 1 Generating compatibility graph

input: $On - set$: The set contains all true implicants.

input: $Off - set$: The set contains all false implicants.

output: G: Compatibility graph

```
1: for All implicants from On − set do
2:     Perform union operation for every two implicants
3:     if The result of two implicants doesn't contains a term in Off − set then
4:         Create an edge between those two implicants on compatibility graph
5:     end if
6: end for
```

The complexity of this part is based on the number of input variables, for the function of *m* true minterms, the complexity of the preprocessing part is $O(m^2)$. After the compatibility graph is generated, the following pieces of information are needed for the next steps: the relation of edges and nodes, degree of every node.

Quantum searching (Quantum Computer)

In this part, the main idea is similar to the concept in Chapter 4, we transformed the minimization problem into a partial hypercube partition problem.

For the partial partition, the symmetric functions in the oracle need two indices (a,b). The initial value of variable *a* is the majority degree of the compatibility graph. If the number is 0, then *a* would be assigned to the degree of the second majority node. If the majority degree is not unique, then the index *a* can be chosen from either group of nodes. The second index *b* is initialized to be *0*, this index aims to identify the nodes and their connected edges which are not included in the result of the current searching procedure and keep them from being removed by the algorithm.

To find the best solution, both indices can be changed during the multiple calls of the Grover algorithm. After the result is returned in the first searching, it would be saved in a classical computer, then the classical computer removes this solution from searching space by disabling the related edges at the input of the quantum algorithm. In the next round, the classical computer modifies the first index *a* in (*a*,*0*), and runs our oracle with the reduced searching space. If no result is found, then index a is changed to the degree of the second majority node and this procedure is repeated until all edges are removed.

Postprocessing (Classical Computer)

The results of each searching round are saved in the classical computer. When the whole searching process is finished, the DSOP/SOP of this function can be derived by performing a supercube calculation on sets of nodes. Because of the binary encoding, the result of

quantum searching is in a format of vector like (10101110111), in the result, 1 means this edge is selected, 0 means not selected. When the result of the final round is received, the classical algorithm transfers the information about edges into nodes, and then performs a union operation on those nodes to get the implicants of DSOP/SOP.

Additional details can be found in the following examples.

Example 5.1

Given is a Boolean function: $F(a,b,c,d) = \sum(1,5,6,7,11,12,13,15)$, its K-map is presented in Fig. 5.1.

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

Figure 5.1 Karnaugh map for function F(a, b, c, d)

Based on the input function F(a, b, c, d), a compatibility graph is created by a classical computer, every true minterm in F(a, b, c, d) is a node in this graph, minterm $\bar{a}\,\bar{b}\,\bar{c}\,d$ is node n1 in Fig. 5.2, the detailed node information is in Tab. 1. Next a supercube operation [19] is executed for every two minterms. If the supercube of two minterms doesn't contain any false minterm, then create an edge between the two nodes that correspond to these minterms. For example, the union result of node n2 and n7 is bd, the term bd doesn't contain any false minterms in the original function, so there is an edge between these 2 nodes. The union result of n1 and n3 is $\bar{a}d$, this term contains false minterms $\bar{a}\,\bar{b}cd,\quad \bar{a}\,\bar{b}c\,\bar{d}$, so n1 and n3 are not connected.

Table 5.1 Node and edge connection for Fig.5.2

Node (related minterms)	Edges connected to this node
n1($\bar{a}\ \bar{b}\ \bar{c}d$)	e1
n2($\bar{a}\ b\ \bar{c}d$)	e1,e2,e4,e5
n3($\bar{a}\ b\ cd$)	e2,e3,e5,e7

n4($\bar{a}\ b\ c\bar{d}$)	e3
n5($ab\bar{c}\ \bar{d}$)	e8
n6($ab\bar{c}d$)	e4,e6,e8,e9
n7($abcd$)	e6,e7,e9,e10
n8($a\bar{b}cd$)	e10

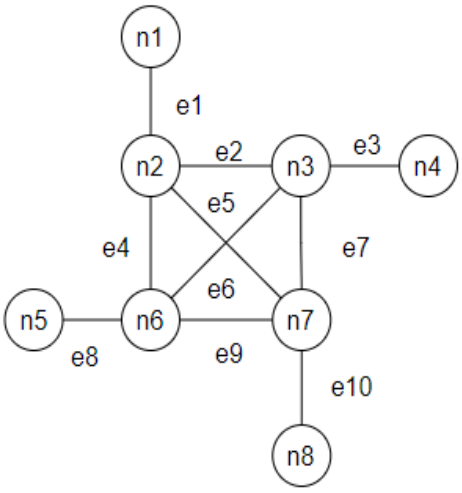


Figure 5.2 Compatibility graph for function F(a, b, c, d)

After the compatibility graph and the required information is created, we can move to the quantum part. The majority degree of this example are 1 and 4, as we previously discussed, the index could be 1 or 4. If we choose 1, then the symmetric function would be $S^{0,1}$, the first searching would return (1010000101), edges: e1,e3,e8, and e10 are selected.

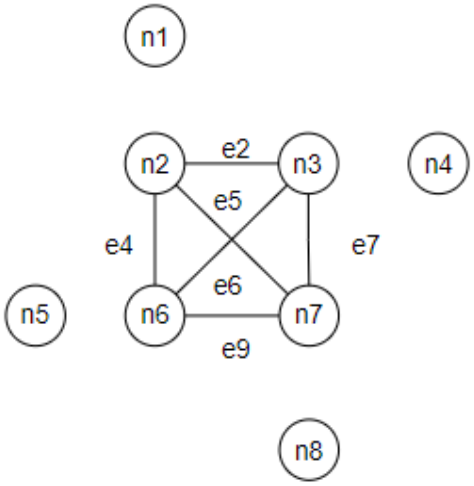


Figure 5.3 Reduced graph for function F(a, b, c, d)

In the next round, those edges are disabled at the input space, and the symmetric functions are changed into $S^{0,3}$, the rest of the edges are founded in this round. The result of the first round is (1010000101), transformed implicants of this vector are: $\bar{a}\bar{c}d, \bar{a}bc, ab\bar{c}, acd$. The result of the second round is (0101111010), transformed implicants of this vector are: $bd, \bar{a}\bar{b}\bar{c}d, ab\bar{c}\bar{d}, a\bar{b}cd, \bar{a}bc\bar{d}$. The final DSOP of function F(a, b, c, d) are:

$$F(a,b,c,d) = bd + \bar{a}\,\bar{b}\,\bar{c}d + ab\bar{c}\,\bar{d} + a\bar{b}cd + \bar{a}bc\bar{d} \quad (5.2)$$

or

$$F(a,b,c,d) = \bar{a}\,\bar{c}d + \bar{a}bc + ab\bar{c} + acd \quad (5.3)$$

If we choose indices (4,0) as an initial value to star our search, the result is the same, but there is a little difference in the procedure, because there is no satisfied result for $S^{0,4}$ in the first search, because we can not only choose nodes which degree is 4, meanwhile keep the rest nodes at 0 degree. If the degree equals to 4 is satisfied, that means the degree of nodes: n1, n4, n5, and n8 is 1. Then the whole symmetric function is not satisfied. At this point, we have to reduce the indices to (3,0) and keep the procedure moving forward.

We would present another example to illustrates how our method handles a case in which there are multiple solutions during the search.

Example 5.2

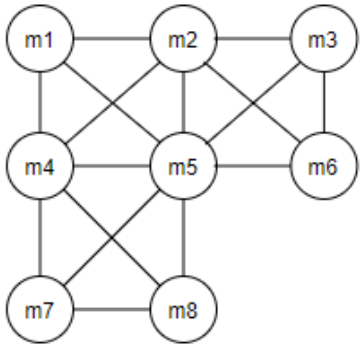
This example, to minimize function G(a, b, c, d) from Fig. 5.4 (a). Because the degree of the majority nodes is 3, the hybrid algorithm starts with symmetric function $S^{0,3}$. Under this constraint, there are multiple solutions that can be found, the edges between nodes m1, m2, m4, m5 are selected as a possible solution for illustration. The results of the first search is ($\bar{a}\,\bar{c}, a\bar{b}\,\bar{c}d, ab\bar{c}d, \bar{a}bc\bar{d}, \bar{a}bcd$).

Fig. 5.4 (c) is the reduced graph after removing the results of the first run, the degree of the majority nodes is still 3. After applying the constraint with symmetric function $S^{0,3}$, there is no result found. This is the case that index *i* needs to be changed to the degree of the second majority node, which is 2. With this modification, multiple results can be found, the same as in the first search. In the example, the quantum computer finds the edges, then classical computer transfers edges to nodes (m2, m3, m6) and (m4, m7, m8) for instance, and the result is: ($\bar{c}d, \bar{a}b\ \bar{a}\bar{b}\bar{c}\bar{d}, a\bar{b}c\bar{d}$).

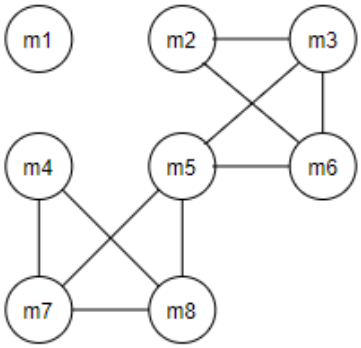
Fig. 5.4 (d) is the reduced graph after removing the edges related to nodes (m2, m3, m6) and (m4, m7, m8). Applying our algorithm with Fig. 5.4 (d), the indices are (1,0). The algorithm keeps the indices as (1,0) until all product implicants are found. As the final result, the optimal SOP is found: $G(a,b,c,d) = \bar{a}\bar{c} + \bar{c}d + \bar{a}b$.

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

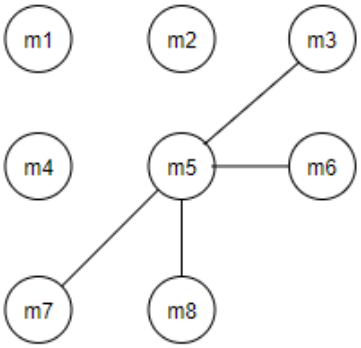
(a)



(b)



(c)



(d)

Figure 5.4 (a)Karnaugh map of Boolean Function G(a,b,c,d). (b) Compatibility graph of G(a,b,c,d). (c) Reduced graph after the first search. (d) Reduced graph after the second search.

5.3 Summary of Chapter 5

The hybrid algorithm presented above illustrates how several abstract decision and optimization problems can be reduced to graph theory problems base on symmetric functions. These problems include graph coloring, graph covering, maximum cliques, shortest path, longest path, Traveling Salesman, and domination. Similar methods can be applied to minimization of ESOP and factorized ESOP expressions [20,24]. In these problems the concept of compatibility of certain Boolean functions is fundamental and serves to define various partitioning problems to symmetric functions, such as those presented in sections 1 to 5. Edges are created for pairs of compatible nodes. In addition, please note that many interesting and practical problems can be also reduced to some of these graph theory problems [20,25]. For instance, a sudoku puzzle can be reduced to a graph coloring problem. We believe there are other fascinating problems in Graph Theory and Topology that would get more efficient solutions with the power of quantum computing.

Chapter 6 Designing Quantum Oracles for Logic Games

When we are formulating those graph theory problems in the previous chapter, we also get some by-products during the group discussion. We investigate our methodology of building the quantum oracle and the hybrid quantum algorithm with the River crossing puzzle (Missionaries and Cannibals problem), Maximum independent set problem.

6.1 Quantum Oracles for River Crossing Puzzle

The river Crossing puzzle is a very old and famous planning problem, the earliest known problem can be traced by Alcuin's manuscript in the 9th century [41], and this problem has many variations, like wolf, goat, cabbage, human puzzle, Missionaries and Cannibals problem. With the given constraints like the number of passengers, the maximum capacity of items on a boat, this puzzle can be formulated as a searching problem, given a state space which is encoded by every item in this problem, the difficulty of the problem is how to find the safe path in the state space without violating any constraint condition. These puzzles could become NP-Hard with more constraints [41] and can be solved by transferring them to graph problems [42]. During we were formulating our problems, we noticed the missionaries and cannibals puzzle and found interesting symmetry property inside this puzzle.

The description of missionaries and cannibals puzzle is that there are three missionaries and three cannibals who want to cross a river from left bank to right. A boat is available and it only holds two peoples at a time, this boat can be operated by any combination of missionaries and cannibals by any number (one or two) and it can't move automatically. If the number of missionaries at either bank of the river, or during the transportation, is less than cannibals at any time, the cannibals will eat those missionaries. The goal of this puzzle is to find a safe sequence to move all people to the other side of the river.

This puzzle is used as an example to introducing classical searching algorithms like depth-first or breadth-first, so we ask ourselves can we use a quantum searching algorithm to solve this puzzle?

Before we apply Grover's searching algorithm to solve this puzzle, we need to build an oracle function. To formulate this puzzle, we conclude two constraints:

(1) **Bank safety constraint**: The number of the missionary should be always greater than the cannibal's number. This constraint will keep all items in both banks in a safe condition.

(2) **Valid movement constraint**: The boat can't move by itself, it must be operated by one passenger. This constraint will keep all movement between two banks are satisfying the valid condition.

6.1.1 Modeling the Constraints of River Crossing Puzzle

For building a quantum oracle, we need to formulate these two constraints into Boolean logic function. Assuming the number of missionaries and cannibals is equal, and it is *n*. Missionary and cannibal are denoted by literal M and C, where $M, C \in \{0, 1\}$. C_n denotes the *n*-th cannibal at bank one, $\overline{C_n}$ denotes the *n*-th cannibal at bank two.

Example 6.1

In the example, we choose *n*=3 to present the constraints in the Boolean logic expression. For three missionaries and three cannibals, the bank safety constraint can be concluded as following cases:

1. Three missionaries and one cannibal at the same bank.

2. Two missionaries and two cannibals at the same bank.

3. Three missionaries and two cannibals at the same bank.

4. Three missionaries and three cannibals at the same bank.

5. Three missionaries and three cannibals at the different banks.

Let us consider the unsafety cases:

1. Three cannibals and one missionary at the same bank.

2. Three cannibals and two missionaries at the same bank.

3. Two cannibals and one missionary at the same bank.

Each case needs to be transformed into a Boolean logic expression, the unsafety condition has fewer expressions. Considering the size of the oracle, the unsafety condition is a better option.

Take the first cases in unsafety cases to demonstrate the transformation of Boolean logic expression. Because the statement doesn't mention the cannibals and missionaries stay at bank 1 or bank 2, we need to consider both banks.

For the case "Three cannibals and one missionary at the same bank", its Boolean logic expression:

$$\begin{aligned} &C_1C_2C_3\left(\overline{M_1}\,\overline{M_2}M_3+\overline{M_1}M_2\,\overline{M_3}+M_1\,\overline{M_2}\,\overline{M_3}\right)\ + \\ &\overline{C_1}\,\overline{C_2}\,\overline{C_3}\left(\overline{M_1}M_2M_3+M_1\,\overline{M_2}M_3+M_1M_2\,\overline{M_3}\right) \end{aligned} \tag{6.1}$$

Transforming into Boolean symmetric function:

$$S^3(C_1,C_2,C_3)S^1(M_1,M_2,M_3)\ \ +S^0(C_1,C_2,C_3)S^2(M_1,M_2,M_3) \tag{6.2}$$

The Boolean logic expressions for the rest cases in unsafety condition are:

$$\begin{aligned} &C_1C_2C_3\left(\overline{M_1}M_2M_3+M_1\,\overline{M_2}M_3+M_1M_2\,\overline{M_3}\right)\ + \\ &\overline{C_1}\,\overline{C_2}\,\overline{C_3}\left(\overline{M_1}\,\overline{M_2}M_3+\overline{M_1}M_2\,\overline{M_3}+M_1\,\overline{M_2}\,\overline{M_3}\right) \end{aligned} \tag{6.3}$$

And

$$\left(C_1 C_2 \overline{C_3} + C_1 \overline{C_2} C_3 + C_1 C_2 \overline{C_3}\right)\left(\overline{M_1}\,\overline{M_2} M_3 + \overline{M_1} M_2\,\overline{M_3} + M_1\,\overline{M_2}\,\overline{M_3}\right) + \left(\overline{C_1}\,\overline{C_2} C_3 + \overline{C_1} C_2\,\overline{C_3} + C_1\,\overline{C_2}\,\overline{C_3}\right)\left(\overline{M_1} M_2 M_3 + M_1\,\overline{M_2} M_3 + M_1 M_2\,\overline{M_3}\right) \quad (6.4)$$

The final expression in Symmetric Function format is:

$$\begin{aligned} &S^3(C_1,C_2,C_3)S^1(M_1,M_2,M_3) + S^0(C_1,C_2,C_3)S^2(M_1,M_2,M_3) + \\ &S^3(C_1,C_2,C_3)S^2(M_1,M_2,M_3) + S^0(C_1,C_2,C_3)S^1(M_1,M_2,M_3) + \\ &S^2(C_1,C_2,C_3)S^1(M_1,M_2,M_3) + S^1(C_1,C_2,C_3)S^2(M_1,M_2,M_3) \\ &= S^{0,2,3}(C_1,C_2,C_3)S^1(M_1,M_2,M_3) + S^{0,1,3}(C_1,C_2,C_3)S^2(M_1,M_2,M_3) \end{aligned} \quad (6.5)$$

Equation 6.5 can be simplified by an XOR gate as follows:

$$S^{0,2,3}(C_1,C_2,C_3) = 1 \oplus S^1(C_1,C_2,C_3) \quad (6.7)$$

the simplified equation is:

$$\overline{S^1(C_1,C_2,C_3)} \oplus S^1(M_1,M_2,M_3) + \overline{S^2(C_1,C_2,C_3)} \oplus S^2(M_1,M_2,M_3) \quad (6.8)$$

Next step, for the valid movement constraint, the Boolean logic expression needs to check the value of this object at the current state is different from the value at the previous state. The notation C_n is extended to C_n^m, where *m* is the current state of *n-th* cannibal, the next state denoted as C_n^{m+1}. A valid movement between two banks means there should only be one or two literals is different from the value of its previous state, in Boolean logic

expression it is: $\sum_{1}^{n} C_i^m \oplus C_i^{m+1} \in [1,2]$ and $\sum_{1}^{n} M_i^m \oplus M_i^{m+1} \in [1,2]$. We use literal Z_i to represent the XOR result of two states.

$$Z_i = \begin{cases} M_j^m \oplus M_j^{m+1} & \text{if } i \text{ is even number} \\ C_j^m \oplus C_j^{m+1} & \text{if } i \text{ is odd number} \end{cases}, \; i,j = 1,2...,\; n \tag{6.9}$$

Continued with the example of three missionaries and three cannibals,

$$\begin{aligned} &C_1^m \oplus C_1^{m+1} = Z_1, C_2^m \oplus C_2^{m+1} = Z_3, C_3^m \oplus C_3^{m+1} = Z_5, \\ &M_1^m \oplus M_1^{m+1} = Z_2,\; M_2^m \oplus M_2^{m+1} = Z_4,\; M_3^m \oplus M_3^{m+1} = Z_6 \end{aligned} \tag{6.10~15}$$

Then the valid movement constraint of this example can be easily written into symmetric function because it only counts the number of positive literals for *Z*. Its symmetric function is: $S^{1,2}(Z_1,Z_2,Z_3,Z_4,Z_5,Z_6)$

Combining those two constraints, we could get the symmetric Boolean function of this oracle:

$$\begin{aligned} &(1 \oplus S^1(C_1,C_2,C_3) \oplus S^1(M_1,M_2,M_3) + 1 \oplus S^2(C_1,C_2,C_3) \oplus S^2(M_1,M_2,M_3)) \\ &\cdot S^{1,2}(Z_1,Z_2,Z_3,Z_4,Z_5,Z_6) \end{aligned} \tag{6.16}$$

6.1.2 Results Analysis

This puzzle is different from those problems in Chapters 4 and 5, it is a sequential decision problem [43]. The searching space of this puzzle for quantum Grover's algorithm is all

states (all possible movement) in the puzzle, and any of a safe state is a solution (target) in Grover's searching procedure. The solutions of problems in Chapter 4 and 5 are independent of each other, but for the sequential problem, the current state is based on the previous state, to find the solution of this type of problems, that means we need to find all of the states which are satisfied with the constraints in the searching space. For the example with three missionaries and cannibals, there are 32 states (containing the begin and end state, all missionaries and cannibals are at the same bank), and 13 safe states, this is verified by miniSAT based on our equations. For quantum simulation, we didn't run the exhaust simulation to find all of the solution states, since it cost too much time for Grover's algorithm in the multiple target searching problems.

The performance of Grover's searching algorithm in solving this type of problem is not comparable with a classical computer now, paper [44] introduced the capability and limitation of most quantum algorithms, even Google proved a dramatic advantage of quantum computing [45] comparing with the classical computer in a specific problem, but quantum computing still has significant drawbacks in some area. How to take the advantage of quantum computing and find an efficient methodology to create hybrid classical-quantum algorithms, is still a question for many quantum computing researchers, and this is also one of the motivations of this dissertation.

6.2 Quantum Oracle for Maximum Independent Set Problem

The maximum independent set (MIS) problem is a basic graph optimization problem, but it has many applications in topology. [46]

Definition 1 (Maximum independent set) [48]

Given an undirected graph G = (V,E) a subset of nodes $S \subseteq V$ is an independent set if and only if there is no edge in E between any two nodes in S. A subset of nodes S is a clique if every pair of nodes in S have an edge between them in G.

Based on this Definition 1, the MIS problem is the following: given a graph G(V,E) find an independent set in G of maximum cardinality. This problem has many classical algorithms to solve this problem like greedy [51], linear-programming [50], and random selection algorithm [49].

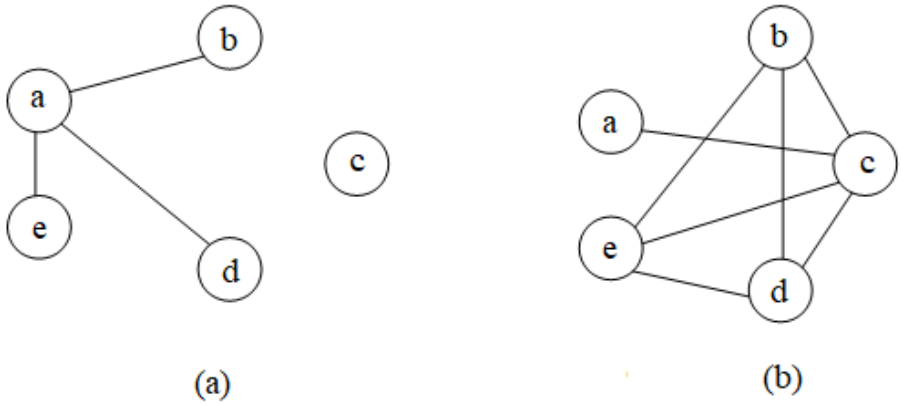


Figure 6.1 (a) Example graph with 5 vertices and 3 edges. (b) complement graph of G, denoted as G`

An efficient way to find the maximum clique is to consider its complement graph [48], our oracle is designed based on this idea. For example, the maximum clique in G` is {b, c, d, e}, it is the maximum independent set of G.

The idea of this oracle is from combinatory logic, for the input graph $G(V,E,\psi)$, ψ is the incidence function that associate with each edge of G, if e is an edge in graph G, $e \in E$, $\psi(e) = uv$, $u,v \in V$, the vertices *u* and *v* are called the ends of *e*.[47] Next our oracle needs to create a POS, each sum term in the POS is generated from ψ, $\psi(e) = uv \Rightarrow u + v$. Next, our oracle needs to change the POS to SOP, then count the number of literals for every product in SOP expression, find the smallest product,

denoted as set S. The complement set of S is the maximum independent set of G. The details of our oracle are shown in the following example.

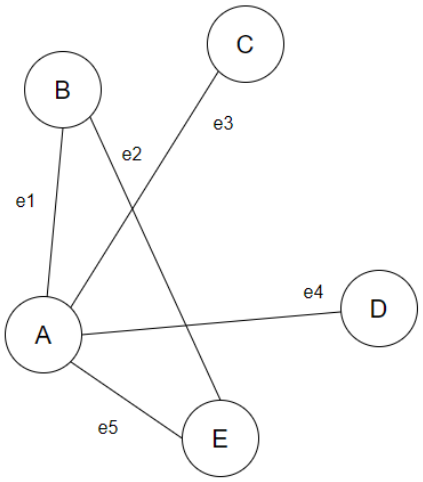


Figure 6.2 Example graph G with five vertices and five edges

Example 6.2

Given graph $G(V,E,\psi)$, incidence functions for every edge in graph G: $\psi(e1)=AB,\psi(e2)=BE,\psi(e3)=AC,\psi(e4)=AD,\psi(e5)=AE$. To model this graph, we need to generate a set of literals $S_G=\{a,b,c,d,e\}$, which related to the nodes in graph G. Graph G can be written into a Boolean expression:

$$G = (a+b)(a+c)(a+d)(a+e)(b+e) \tag{6.17}$$

By definition of the independent set, if two vertices are connected, then these two vertices can not be in the same independent set. Every satisfied minterm of equation 6.17 is a set of clique in the complement graph of G. To find the maximum independent set, we just need to find the smallest size of product term in equation 6.17, and its complement set is the maximum independent set of graph G. The smallest product of equation 6.17 is *ab* and *ae*, then the maximum independent set of this problem is *cde* and *bcd*.

Our quantum oracle to solve this problem contains two constraint blocks: positive-literal block and function satisfiability block.

The function of a positive-literal block is to constraint the number of positive literal in the solution, since the satisfied minterms only contain positive literals, any result that contains negative literal is unacceptable. And we also need to find the smallest term for the independent set. This block contains a quantum accumulator and equality comparator. The quantum accumulator is to counting numbers of positive literal at the input side. The comparator checks the result from the accumulator and comparing it with the user's threshold to pursue the smallest minterm.

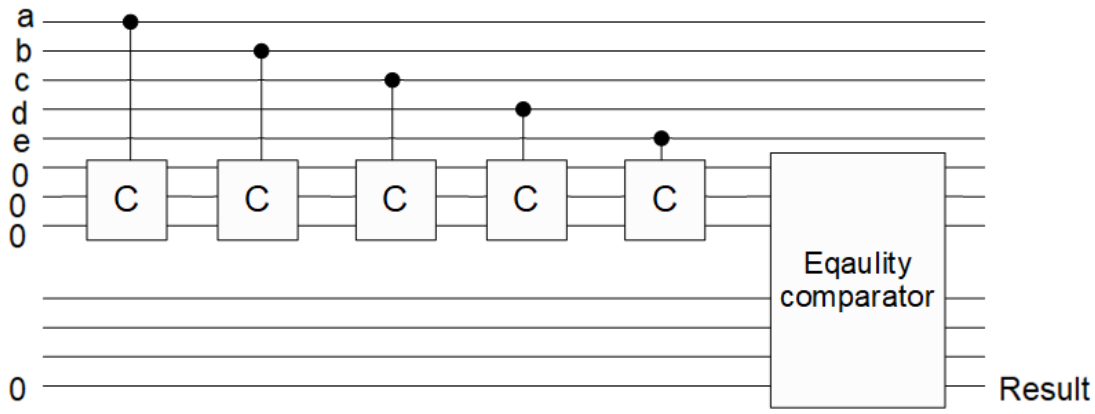


Figure 6.3 Block diagram of the positive-literal block for example 6.2

The function satisfiability block is similar to the SAT example in chapter 2, it is a problem-specific circuit that depends on the POS equation.

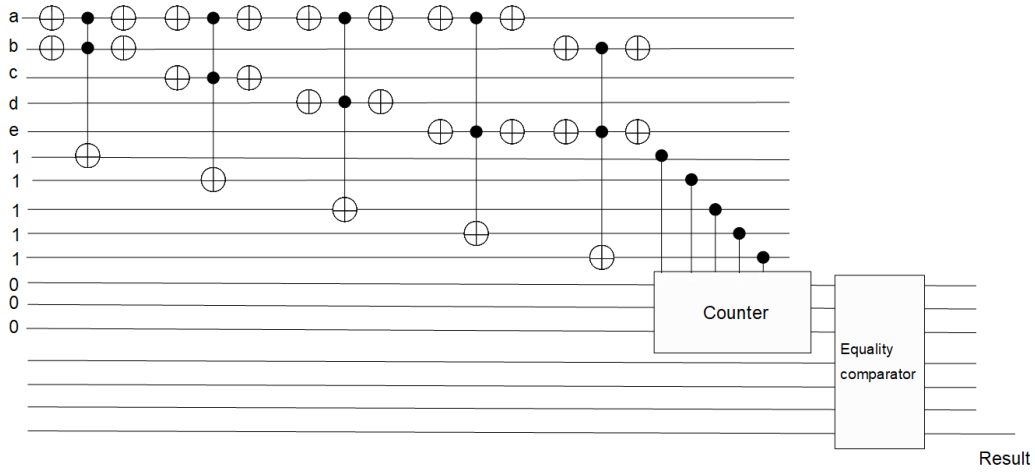


Figure 6.4 Block diagram for example 6.2

6.3 Summary of Chapter 6

Two quantum oracles of logic puzzle problems are presented in this chapter. The river crossing puzzle discussed the limitation of Grover's algorithm in sequential problems, this pitfall is caused by multiple targets in the searching space, it is an open problem in Grover's algorithm. Some papers [52,53,54] provide quantum walk as an alternative solution for multiple target searching problems. The oracle of MIS problem inherits the same methodology in chapters 4 and 5, this is a general idea in our oracle design methodology to solve constraint satisfaction problem (CSPs).

Chapter 7: Conclusion and Future Work

7.1 Conclusion

In the research work, a quantum algorithm based on the Grover searching algorithm was developed. We used our methodology and Boolean symmetric functions to design different quantum oracles to solve graph-theory-related problems. We explored our oracle design methodology on two graph theory problems (i.e., cycle detection and hypercube partition) and extended the solution to the hypercube partition problem to a quantum algorithm to solve the Boolean function minimization problem. Current computers still cannot give exact optimal solutions to the Boolean function minimization problem because the solution requires astronomic numbers of prime or product implicants. The methodology used to design the quantum oracle in this dissertation has many applications in graph-theory-related problems and a significant impact on various optimization problems.

As far as we know, there are no quantum algorithms for graph partitioning, as defined here. There are also no classical algorithms for the problems formulated and solved in Chapter 4. There are no quantum algorithms that can be used to solve classical DSOP and SOP minimization problems, as presented in Chapter 5. These problems, like clique covering and similar, are all NP-hard. The presented methods will become practical with the appearance of quantum computers that can handle more qubits than current technology.

The usefulness of these algorithms for noisy intermediate-scale quantum (NISQ) era computing should be also studied.

7.1.1 Contributions

This research has resulted in the following accomplishments:

- The invention of a new quantum oracle for searching for cycles in a graph
- The invention of a new quantum oracle to partition the Hamiltonian cycle in a graph
- The creation of a new methodology for designing a quantum oracle based on graph theory to solve the Boolean minimization problem
- The realization of several graph-theory-related quantum oracles and simulations in Qiskit and Quipper
- The realization of two quantum oracles that can solve the Boolean minimization problem in Qiskit

7.2 Future Work

The research in this dissertation can be extended in the following directions:

- Extending the Boolean minimization oracle to a multivalued logic minimization oracle
- Using the flexibility of the symmetric function to explore additional graph-theory-related problems, like graph coloring, TSP, and vertex covering
- Applying our oracle design methodology to additional combinatory problems

- Optimizing the inner connections of quantum lattice structures to reduce the quantum cost of producing lattice diagrams

References

[1] Nielsen, M. A., and Chuang, Isaac L. (2000). Quantum computation and quantum information (10th anniversary edition.). Cambridge University Press.

[2] L. K. Grover, “A fast quantum mechanical algorithm for database search,” arXiv:quant-ph/9605043, May 1996, Accessed: Jun. 27, 2019. M. Chrzanowska-Jeske and A. Mishchenko, “Synthesis for Regularity using Decision Diagrams,” in IEEE International Symposium on Circuits and Systems, pages 4721–4724. 2005

[3] Yanofsky, Noson S, and Mannucci, Mirco A. (2008). *Quantum Computing for Computer Scientists*. Cambridge University Press.

[4] D. Shah, M. A. Perkowski, "Synthesis of quantum arrays with low quantum costs from Kronecker Functional Lattice Diagrams" , IEEE Congress on Evolutionary Computation, 2010. Pages 24 -28

[5] M. Lukac, D. Shah, M. Perkowski, and M. Kameyama, “Synthesis of Quantum Arrays from Kronecker Functional Lattice Diagrams,” IEICE Trans. Inf. Syst., vol. E97.D, no. 9, pages 2262–2269, 2014.

[6] M. Chrzanowska-Jeske and A. Mishchenko, “Synthesis for Regularity using Decision Diagrams,” in IEEE International Symposium on Circuits and Systems, pages 4721–4724. 2005

[7] S.B. Akers, “A rectangular logic array”, Trans. IEEE Computer, VOL. C-21, pages 848-857, 1972.

[8] T. Sasao, "A new expansion of symmetric functions and their application to nondisjoint functional decompositions for LUT-type FPGAs", Proc. Int. Workshop Logic Synthesis, pages 105-110, 2000.

[9] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis", The Kluwer International

Series in Engineering and Computer Science, Vol. 2, Boston, MA: Kluwer Academic Publishers, 1984.

[10] M. A. Perkowski, D. Shah, M. Kameyama, “Synthesis of quantum circuits in Linear Nearest Neighbor model using Positive Davio Lattices”, FACTA UNIVERSITATIS (NIS) SER.: ELEC. ENERG. vol. 24, no. 1, April 2011, 71-87.

[11] V. P. Suprun, D. A. Gorodecky, “Matrix Method of Polynomial Expansion of Symmetric Boolean Functions”, Automatic Control and Computer Sciences, vol. 47, no. 1, pages 1–6, 2013

[12] Godsil, C. and Royle, G. Algebraic Graph Theory. New York: Springer-Verlag, 2001

[13] M. A. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, “Lattice diagrams using reed-muller logic”, in IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, pages 85–102, 1997.

[14] L.K. Grover. “A fast quantum mechanical algorithm for database search”. In Proc. 28th Annual ACM Symposium on Theory of Computing, pages 212–219,1996.

[15] Y. Li, E. Tsai, M. A. Perkowski, and X. Song, “Grover-based Ashenhurst-Curtis decomposition using quantum language quipper,” Quantum Information & Computation, vol. 19, pp. 35–66, 2018.

[16] W. H. Mills, “Some Complete Cycles on the n-Cube,” Proc. Amer. Math. Soc., vol. 14, no. 4, pp. 640–643, 1963

[17] F. Harary, J. P. Hayes, and H.-J. Wu, “A survey of the theory of hypercube graphs,” Comput. Math. Appl., vol. 15, no. 4, pp. 277–289, 1988, doi: 10.1016/0898-1221(88)90213-1.

[18] Maslov, D., and Dueck, G. W. (2003). Improved Quantum Cost for n-bit Toffoli Gates. Electronics Letters, 39(25), 1790. https://doi.org/10.1049/el:20031202

[19] M.J. Ciesielski, S. Yang, and M.A. Perkowski, “Multiple-Valued Boolean Minimization Based on Graph Coloring”, in Proc. 1989 IEEE Int. Conf. Comp. Design, USA, pp.265-265, 1989.

[20] A. Mishchenko, and M.A. Perkowski, “Fast Heuristic Minimization of Exclusive-Sums-of-Products”, in Proc. Proc. Reed-Muller’2001, Japan, pp. 213-215 .2001.

[21] N. Song and M.A. Perkowski, “Minimization of Exclusive sum-of-products expressions for multiple-valued input, incompletely specified functions”. IEEE Trans. CAD, Vol. 15, (4), pp. 385-395. 1996.

[22] E. Chow, H. Madan, J. Peterson, D. Grunwald, and D. Reed, “Hyperswitch network for the hypercube computer,” in Proc. 15th Intern. Symp. Comp. Arch. pp. 90–99, 1988.

[23] R. Babbush et al., “Encoding Electronic Spectra in Quantum Circuits with Linear T Complexity,” Phys. Rev. X, vol. 8, no. 4, pp. 041-015, Oct. 2018.

[24] M.A. Perkowski, A. Mishchenko, M. Chrzanowska-Jeske, "Generalized Inclusive Forms — New Canonical Reed-Muller Forms Including Minimum ESOPs" VLSI Design Vol. 1, pp. 13-21, 2002.

[25] P. Gao, Y.Li and M.A. Perkowski, Realization of Quantum Oracles using Symmetries of Boolean Functions, Quantum Inf. Comput. vol. 20 (5&6), pp. 418-448, 2020.

[26] Rudell, R.L, and Sangiovanni-Vincentelli, A. (1987). Multiple-Valued Minimization for PLA Optimization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 6(5), 727–750.

[27] Aspect, Alain, Dalibard, Jean, and Roger, Gérard. (1982). Experimental Test of Bell's Inequalities Using Time- Varying Analyzers. Physical Review Letters, 49(25), 1804–1807.

[28] Müller, K A, and Bednorz, J G. (1987). The Discovery of a Class of High-Temperature Superconductors. Science (American Association for the Advancement of Science), 237(4819), 1133-1139.

[29] Deutsch, David. (1985). Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer. Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, 400(1818), 97-117.

[30] Harrow, Aram W, Hassidim, Avinatan, and Lloyd, Seth. (2009). Quantum algorithm for linear systems of equations. Physical Review Letters, 103(15), 150502.

[31] Grover, Lov K. (1997). Quantum Mechanics Helps in Searching for a Needle in a Haystack. Physical Review Letters, 79(2), 325-328.

[32] Shor, P.W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. Proceedings 35th Annual Symposium on Foundations of Computer Science, 124-134.

[33] Rebentrost, Patrick, Mohseni, Masoud, and Lloyd, Seth. (2014). Quantum support vector machine for big data classification. Physical Review Letters, 113(13), 130503.

[34] J., Abhijith, Adetokunbo Adedoyin, John Ambrosiano, Petr Anisimov, Andreas Bärtschi .etc. 2020. “Quantum Algorithm Implementations for Beginners.” ArXiv:1804.03719 [Quant-Ph].

[35] Frisk Kockum, Anton. (2014). Quantum Optics with Artificial Atoms.

[36] U. Kalay, M. A. Perkowski, D. V. Hall, B. Steinbach, and S. A.Shahjahan, “Rectangle Covering Factorization of ESOPs Into ScanBased Levelized Circuits with Universal Test Set,” 2007.

[37] R. K. Brayton. Factoring Logic Functions. IBM J. Res.Develop., 31(2),Mar. 1987.

[38] B. Schaeffer, L. Tran, A. Gronquist, M. Perkowski, and P. Kerntopf, “Synthesis of Reversible Circuits Based on Products of Exclusive OR Sums,” in 2013 IEEE 43rd International Symposium on MultipleValued Logic, 2013, pp. 35–40, doi: 10.1109/ISMVL.2013.54.

[39] Praeger, Cheryl E., and Ming-Yao Xu. 1989. “A Characterization of a Class of Symmetric Graphs of Twice Prime Valency.” European Journal of Combinatorics 10(1):91–102.

[40] McKay, B. D. and Royle, G. F. "The Transitive Graphs with at Most 26 Vertices." Ars Combin. 30, 161-176, 1990.

[41] Ito, Hiro, Langerman, Stefan, and Yoshida, Yuichi. (2015). Generalized River Crossing Problems. *Theory of Computing Systems, 56*(2), 418-435.

[42] Csorba, P, Hurkens, C.A.J, and Woeginger, G.J. (2012). The Alcuin Number of a Graph and Its Connections to the Vertex Cover Number. SIAM Review, 54(1), 141-154.

[43] Beard, Randal W. (2018). Decision Making Under Uncertainty: Theory and Application. IEEE Control Systems, 38(6), 114-115.

[44] Montanaro, Ashley. (2016). Quantum algorithms: An overview. *Npj Quantum Information, 2*(1), 15023.

[45] Arute, Frank, Arya, Kunal, Babbush, Ryan, Bacon, Dave, Bardin, Joseph C, Barends, Rami, . . . Martinis, John M. (2019). Quantum supremacy using a programmable superconducting processor. Nature (London), 574(7779), 505-510.

[46] Lefschetz, S. (1975). *Applications of algebraic topology : Graphs and networks : The Picard-Lefschetz theory and Feynman integrals* (Applied mathematical sciences (Springer-Verlag New York Inc.) ; v. 16). New York: Springer-Verlag.

[47] Bondy, J., and Murty, U. S. R. (1976). *Graph theory with applications*. New York: Elsevier Science Publishing.

[48] West, D. (1996). *Introduction to graph theory*. Upper Saddle River, NJ: Prentice Hall.

[49] Luby, M. (1986). A simple parallel algorithm for the maximal independent set problem. SIAM Journal on Computing, 15(4), 1036-1053.

[50] Blelloch, Guy, Fineman, Jeremy, and Shun, Julian. (2012). Greedy sequential maximal independent set and matching are parallel on average. *Proceedings of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 308-317.

[51] Colombi, Marco, Mansini, Renata, and Savelsbergh, Martin. (2017). The generalized independent set problem: Polyhedral analysis and solution approaches. *European Journal of Operational Research, 260*(1), 41-55.

[52] Ambainis, Andris. (2010). Quantum Search with Variable Times. *Theory of Computing Systems, 47*(3), 786-807.

[53] Brun, Todd A, Carteret, H. A, and Ambainis, Andris. (2003). Quantum random walks with decoherent coins. *Physical Review. A, Atomic, Molecular, and Optical Physics, 67*(3), Physical review. A, Atomic, molecular, and optical physics, 2003-03, Vol.67 (3).

[54] Ambainis, Andris. 2004. "Quantum Walks and Their Algorithmic Applications." ArXiv:Quant-Ph/0403120.

[55] Dirac, P. A. M. (1939). A new notation for quantum mechanics. *Mathematical Proceedings of the Cambridge Philosophical Society, 35*(3), 416-418.

[56] Bryant, R. E. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers, C-35*(8), 677-691.

[57] Menon, P.R, Ahuja, H, and Harihara, M. (1994). Redundancy identification and removal in combinational circuits. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 13*(5), 646-651.

[58] Chris. Durr, M. Heiligman, P. Hoyer, M. Mhalla. (2006). Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6), 1310-1328

[59] Chris. Durr, and Peter, Hoyer, Peter. (1996). A Quantum Algorithm for Finding the Minimum. *Arxiv*.

[60] Vidya Raj, C., and M. S. Shivakumar. (2007). Applying Quantum Algorithm to Speed Up the Solution of Hamiltonian Cycle Problems. in *Intelligent Information Processing III. Vol. 228*,. 53–61

[61] Rudolph, T. (1996). Quantum computing Hamiltonian cycles. *Arxiv*

[62] Burger, John Robert. (2005). Quantum Algorithm Processors to Reveal Hamiltonian Cycles. *Arxiv*

[63] Dajian Zhang, Dianmin Tong, Yao Lu, Guilu Long. (2015). An Alternative Adiabatic Quantum Algorithm for the Hamiltonian Cycle Problem. *Communications in Theoretical Physics*, 63(5), 554-558.

[64] Ge, Yimin, and Dunjko, Vedran. (2019). A hybrid algorithm framework for small quantum computers with application to finding Hamiltonian cycles. *Arxiv*.

[65] Mahasinghe, Anuradha, Hua, Richard, Dinneen, Michael, and Goyal, Rajni. (2019). Solving the Hamiltonian Cycle Problem using a Quantum Computer. *Proceedings of the Australasian Computer Science Week Multiconference*, 1–9.

[66] Eppstein, David. (2007). The Traveling Salesman Problem for Cubic Graphs. *Journal of Graph Algorithms and Applications*, *11*(1), 61–81.

[67] Xiao, Mingyu, and Nagamochi, Hiroshi. (2016). An Exact Algorithm for TSP in Degree-3 Graphs Via Circuit Procedure and Amortization on Connectivity Structure. *Algorithmica*, 74(2), 713–741.

[68] Xiao, Mingyu, and Nagamochi, Hiroshi. (2016). An Improved Exact Algorithm for TSP in Graphs of Maximum Degree 4. *Theory of Computing Systems*, 58(2), 241–272.

[69] Moylett, Dominic J, Linden, Noah, and Montanaro, Ashley. (2016). Quantum speedup of the Travelling Salesman Problem for bounded-degree graphs. *Arxiv*

[70] Srinivasan, Karthik, Satyajit, Saipriya, Behera, Bikash K, and Panigrahi, Prasanta K. (2018). Efficient quantum algorithm for solving travelling salesman problem: An IBM quantum experience. *Arxiv*

[71] Ushijima-Mwesigwa, Hayato, Negre, Christian F. A, and Mniszewski, Susan M. (2017). Graph Partitioning using Quantum Annealing on the D-Wave System. *Arxiv*

[72] Cocchi, Eugenio, Tignone, Edoardo, and Vodola, Davide. (2021). Graph Partitioning into Hamiltonian Subgraphs on a Quantum Annealer. *Arxiv*

[73] Bojić, Alan. (2012). Quantum Algorithm for Finding a Maximum Clique in an Undirected Graph. *Journal of Information and Organizational Sciences*, 36(2), 91.

[74] Sanyal, Arpita, Saha, Amit, Saha, Debasri, Saha, Banani, and Chakrabarti, Amlan. (2020). Circuit Design for Clique Problem and Its Implementation on Quantum Computer. *Arxiv*

[75] Li, Xi, Wu, Mingyou, and Chen, Hanwu. (2019). Algorithm for Finding the Maximum Clique Based on Continuous Time Quantum Walk. *Arxiv*

[76] Pelofske, Elijah, Hahn, Georg, and Djidjev, Hristo N. (2019). Solving large Maximum Clique problems on a quantum annealer. *Arxiv*

[77] Chapuis, Guillaume, Djidjev, Hristo N, Hahn, Georg, and Rizk, Guillaume. (2018). Finding Maximum Cliques on the D-Wave Quantum Annealer. *Arxiv*

[78] Montanaro, Ashley. (2015). Quantum walk speedup of backtracking algorithms. *Arxiv*

[79] Bian, Zhengbing, Chudak, Fabian, Israel, Robert Brian, Lackey, Brad, Macready, William G, and Roy, Aidan. (2016). Mapping Constrained Optimization Problems to Quantum Annealing with Application to Fault Diagnosis. Frontiers in ICT, 3.

[80] Mesman, Koen, Al-Ars, Zaid, and Möller, Matthias. (2021). QPack: Quantum Approximate Optimization Algorithms as universal benchmark for quantum computers. *Arxiv*.

[81] Xu, Hong, Sun, kexuan., Koenig, Sven., Hen, Itay., and Kumar, (2020). Hybrid Quantum-Classical Algorithms for Solving the Weighted CSP. *Proceedings of the International Symposium on Artificial Intelligence and Mathematics*.