

4-15-2024

Quantum Search Algorithms for Constraint Satisfaction and Optimization Problems Using Grover's Search and Quantum Walk Algorithms with Advanced Oracle Design

Abdirahman Sheikh Hassan Alasow
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds



Part of the [Electrical and Computer Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Alasow, Abdirahman Sheikh Hassan, "Quantum Search Algorithms for Constraint Satisfaction and Optimization Problems Using Grover's Search and Quantum Walk Algorithms with Advanced Oracle Design" (2024). *Dissertations and Theses*. Paper 6606.

<https://doi.org/10.15760/etd.3738>

This Dissertation is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

Quantum Search Algorithms for Constraint Satisfaction and Optimization Problems
Using Grover's Search and Quantum Walk Algorithms with Advanced Oracle Design

by

Abdirahman Sheikh Hassan Alasow

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

Doctor of Philosophy
in
Electrical and Computer Engineering

Dissertation Committee:
Marek Perkowski, Chair
Xiaoyu Song
John Acken
Steven Bleiler

Portland State University
2024

© 2024 Abdirahman Sheikh Hassan Alasow

Abstract

The field of quantum computing has emerged as a powerful tool for solving and optimizing combinatorial optimization problems. To solve many real-world problems with many variables and possible solutions for constraint satisfaction and optimization problems, the required number of qubits of scalable hardware for quantum computing is the bottleneck in the current generation of quantum computers. In this dissertation, we will demonstrate advanced, scalable building blocks for the quantum search algorithms that have been implemented in Grover's search algorithm and the quantum walk algorithm. The scalable building blocks are used to reduce the required number of qubits in the design. The proposed architecture effectively scales and optimizes the number of qubits needed to solve large problems with a limited number of qubits. Thus, scaling and optimizing the number of qubits that can be accommodated in quantum algorithm design directly reflect on performance. Also, accuracy is a key performance metric related to how accurately one can measure quantum states.

The search space of quantum search algorithms is traditionally created by using the Hadamard operator to create superposition. However, creating superpositions for problems that do not need all superposition states decreases the accuracy of the measured states. We present an efficient quantum circuit design that the user has control over to create the subspace superposition states for the search space as needed. Using only the subspace states as superposition states of the search space will increase the rate of correct solutions.

In this dissertation, we will present the implementation of practical problems for Grover's search algorithm and quantum walk algorithm in logic design, logic puzzles, and machine learning problems such as SAT, MAX-SAT, XOR-SAT, and like SAT problems in EDA, and mining frequent patterns for association rule mining.

Dedication

This dissertation is dedicated to

My beloved parents, with deepest gratitude for their love and encouragement. For remembering me always in their prayers and for reminding me to "seek knowledge from the cradle to the grave."

My lovely wife, who had devoted her life to supporting me by any means during my journey on this dissertation. For your unconditional love and care for our beloved kids, for bearing with me, and for giving me time to completely focus on my studies after coming from a full-time job. Without your constant support, love, prayer, and encouragement to strive for excellence, I would not have reached the end of this journey.

My son Mohamed and my daughter Munira, who joined us during my work on this dissertation. For being my source of happiness and pleasure that enabled me to persevere and finish the doctorate program.

My brothers and sisters for their encouragement and praying.

Acknowledgments

Studying for a doctorate program while working a full-time job and being a father was not an easy task. However, with the grace of God and his assistance, I was able to handle and complete my PhD program.

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Marek Perkowski, for believing in me. I truly appreciate his guidance, support, and encouragement throughout the years that have enriched my scientific skills and thinking, not only as an engineer but also philosophically. I am grateful for being available in the late evening meeting to accommodate my availability for deep technical discussions. I am also thankful for his understanding and patience with me. Sometimes, I needed more attention to my full-time job or my family. He helped me not only with my PhD program but also provided me with invaluable life advice in both academic and non-academic life that I will continue to cherish for the rest of my life.

I would like to acknowledge Dr. Xiaoyu Song, who guided me to analyze my work in a detailed manner, step by step, to make it more understandable to a large audience. Taking all his classes during my master's time was fundamental knowledge that enabled me to apply it to quantum computing.

I would like to acknowledge Dr. John Acken for his guidance while I was writing the dissertation and preparing presentations for both the proposal and the final defense.

I would like to thank Dr. Steven Bleiler for his advice on making my dissertation clear, concise, and correct.

I thank all my advisory committee members for their time, patience, and valuable feedback and for always being available for me when I needed it.

With all my heart, I thank my wife, who has been by my side and never lets me lose sight of achieving my goals. Her sacrifice, efforts, endless support, and affection have been the primary source of competing to pass the valleys of this journey. I will forever be indebted to you.

Table of Contents

| | |
|--|-------------|
| Abstract..... | i |
| Dedication | iii |
| Acknowledgments | iv |
| List of Tables..... | xiii |
| List of Figures | xv |
| 1 Introduction..... | 1 |
| 1.1 Objective of the Research..... | 2 |
| 1.2 Dissertation Outline..... | 4 |
| 2 Introduction of Quantum Computing | 6 |
| 2.1 Introduction of linear algebra for quantum computing | 7 |
| 2.1.1 Inner Product (dot product)..... | 8 |
| 2.1.2 Outer Product | 9 |
| 2.1.3 Tensor Product | 9 |
| 2.1.4 Orthogonal vs Orthonormal | 9 |
| 2.1.5 Eigenvalues and Eigenvectors..... | 11 |
| 2.1.6 Unitary Operator..... | 11 |
| 2.2 Quantum Gates..... | 12 |
| 2.2.1 Pauli gates | 13 |

| | | |
|----------|--|-----------|
| 2.2.2 | Hadamard Gate..... | 15 |
| 2.2.3 | Controlled NOT Gate | 16 |
| 2.2.4 | Controlled-Z Gate..... | 17 |
| 2.2.5 | Toffoli Gate..... | 17 |
| 2.2.6 | SWAP Gate | 18 |
| 2.2.7 | Fredkin Gate..... | 18 |
| 2.3 | Overview of the history of quantum algorithms | 19 |
| 2.4 | Quantum Computer Hardware and Software..... | 24 |
| 2.4.1 | Quantum Annealing..... | 24 |
| 2.4.2 | Gate-based Quantum Computing | 25 |
| 2.4.3 | Superconducting Qubits..... | 26 |
| 2.4.4 | Trapped Ions..... | 27 |
| 2.4.5 | Photonics..... | 28 |
| 2.4.6 | Silicon Quantum Dots / Silicon Spins | 28 |
| 3 | Grover’s Search Algorithm | 31 |
| 3.1 | Finding an element in array using Grover’s algorithm | 31 |
| 3.2 | Quantum Oracle | 33 |
| 3.3 | Schematic View | 36 |
| 3.4 | Matrix Representation | 38 |
| 3.5 | Geometric Representation | 43 |

| | | |
|----------|--|----|
| 3.6 | Algebraic Representation..... | 48 |
| 3.7 | Type of oracles and their implementation | 52 |
| 3.8 | Applications of Grover’s Algorithm | 76 |
| 3.8.1 | Cryptography..... | 77 |
| 3.8.2 | Quantum machine learning | 77 |
| 3.8.3 | K-medians | 78 |
| 3.8.4 | K-means | 78 |
| 3.8.5 | Reinforcement Learning | 78 |
| 3.8.6 | Constraint satisfaction and optimization problems | 79 |
| 4 | Quantum Algorithm for Variant Satisfiability and Maximum Satisfiability | |
| | 80 | |
| 4.1 | Introduction..... | 82 |
| 4.1.1 | Satisfiability | 82 |
| 4.1.2 | Maximum Satisfiability | 83 |
| 4.2 | Related Work | 85 |
| 4.2.1 | Maximum Satisfiability Applications..... | 85 |
| 4.2.2 | Classical Algorithm for Maximum Satisfiability Problem | 88 |
| 4.2.3 | Quantum Algorithms for Maximum Satisfiability Problem | 88 |
| 4.3 | Definitions and Preliminaries | 90 |
| 4.3.1 | Nth Root of NOT gate | 91 |
| 4.3.2 | Controlled-Nth Root of NOT Gate..... | 92 |

| | | |
|-------|--|-----|
| 4.3.3 | Quantum Cost..... | 93 |
| 4.3.4 | Peres Gate | 94 |
| 4.4 | Quantum Algorithm for Maximum Satisfiability | 96 |
| 4.4.1 | Quantum Counter | 98 |
| 4.4.2 | Traditional Oracle for Satisfiability Boolean function | 100 |
| 4.4.3 | Construction of a Quantum Oracle for MAX-SAT..... | 102 |
| 4.4.4 | Verifying an Unsatisfiable Function..... | 105 |
| 4.5 | Calculation of Quantum Cost..... | 110 |
| 4.5.1 | Calculation of Quantum Counter Size..... | 110 |
| 4.5.2 | Quantum Cost Calculation for Quantum Counter..... | 112 |
| 4.6 | Variants of SAT Oracles Using Quantum Counter..... | 113 |
| 4.6.1 | Oracle for SOPs..... | 114 |
| 4.6.2 | Oracle for Product of SOPs (POSOP SAT)..... | 114 |
| 4.6.3 | Oracle for Exclusive-or-Sum-of-Products (ESOP) | 115 |
| 4.7 | OR Satisfiability Problems for Electronic Design Automation..... | 116 |
| 4.8 | XOR SAT | 120 |
| 4.8.1 | Classical XOR SAT | 122 |
| 4.8.2 | Translating XOR clauses to CNF clauses..... | 122 |
| 4.8.3 | Gaussian Elimination of XOR constraints | 123 |
| 4.8.4 | Quantum Oracle for XOR-SAT | 126 |
| 4.8.5 | Quantum Oracle for CNF-XOR SAT..... | 130 |

| | | |
|----------|---|------------|
| 5 | Quantum Algorithm for Mining Frequent Patterns for Association Rule Mining | 133 |
| 5.1 | Introduction..... | 134 |
| 5.2 | Related Works..... | 139 |
| 5.2.1 | Classical Algorithms for Association Rule Mining..... | 139 |
| 5.2.2 | Quantum Algorithms for Association Rule Mining | 146 |
| 5.3 | Quantum Oracle Design for ARM | 148 |
| 5.3.1 | Dicke States..... | 150 |
| 5.3.2 | Quantum Comparator | 156 |
| 6 | Quantum Random Walk..... | 166 |
| 6.1 | Limitation of Grover’s search algorithm..... | 166 |
| 6.1.1 | Search on Grid..... | 166 |
| 6.1.2 | Search on element distinctness..... | 167 |
| 6.1.3 | Search on Hypercube..... | 167 |
| 6.2 | Random walk on the line | 168 |
| 6.3 | Quantum Random walk on the line..... | 169 |
| 6.4 | Random walk on the graph | 172 |
| 6.5 | Quantum Random walk on the graph..... | 175 |
| 6.6 | Quantum Walk Algorithm for MAX-SAT | 179 |

| | | |
|----------|---|------------|
| 6.7 | Quantum Walk Algorithm for Mining Frequent Patterns of Association Rule Mining | 182 |
| 7 | Quantum Complexity Theory | 187 |
| 7.1 | Classical Complexity | 187 |
| 7.2 | Classical Complexity Classes | 189 |
| 7.3 | Quantum Complexity Classes | 192 |
| 7.4 | Practical Complexity Analysis | 194 |
| 8 | Methodology for Advanced Quantum Oracle Design | 203 |
| 8.1 | Covering Problems | 204 |
| 8.1.1 | Related work | 206 |
| 8.1.2 | Classical Algorithms for Covering Problems | 208 |
| 8.1.3 | Quantum Algorithm for Covering Problem | 209 |
| 8.1.4 | Finding All Prime Implicants for the Exact Minimum Covering of a SOP Circuit | 212 |
| 8.1.5 | Finding the Minimum Covering for an Implication Graph | 216 |
| 8.1.6 | Finding Minimum Cost Constraint | 222 |
| 8.1.7 | Minimization of Incompletely Specified Finite State Machines | 225 |
| 8.2 | Quantum oracles based on arithmetic for constraint satisfaction and optimization problems | 233 |
| 8.2.1 | Classical Half and Full Adders | 234 |

| | | |
|----------|--------------------------------------|------------|
| 8.2.2 | Quantum Half and Full Adders | 235 |
| 8.2.3 | Comparison analysis | 238 |
| 8.2.4 | Quantum Wallace Multiplier..... | 241 |
| 8.3 | Logic Puzzles | 245 |
| 8.3.1 | Quantum binary sudoku..... | 246 |
| 9 | Summary and Conclusions | 254 |
| 9.1 | Conclusion | 254 |
| 9.2 | Achievements..... | 257 |
| 9.3 | Publications..... | 261 |
| 9.4 | Future Work..... | 261 |
| | References | 263 |

List of Tables

| | |
|---|-----|
| Table 2.1 Some practical applications for quantum algorithms..... | 23 |
| Table 2.2 Some quantum software frameworks..... | 30 |
| Table 3.1 Truth table $f = x_1 + x_2x_3$ | 66 |
| Table 4.1 Analysis of 3-qubit quantum counter block from Figure 4.7 | 99 |
| Table 4.2 Karnaugh map of POS for the Boolean function $f(a, b, c) = a + b + ca + b + cb + c$ | 101 |
| Table 4.3 Quantum counter size; total qubits for counter..... | 111 |
| Table 4.4 XOR clause to CNF clauses | 123 |
| Table 4.5 Truth table for XOR SAT function $f = (ab \oplus ac)(abc \oplus bc)$ | 130 |
| Table 5.1 Transaction with Items. | 141 |
| Table 5.2 Binary Matrix Corresponds from Table 5.1. | 148 |
| Table 5.3 Truth Table of one-bit comparator..... | 157 |
| Table 6.1 Probability of classical random walk on integer point of a line. | 169 |
| Table 6.2 Probability of quantum random walk on integer points on a line..... | 172 |
| Table 6.3 Swap operation for 11100 node to reach all other connected nodes..... | 184 |
| Table 6.4 SWAP operation for all nodes | 184 |
| Table 7.1 Asymptotic notation..... | 189 |
| Table 7.2 Common equation for time complexity | 189 |
| Table 7.3 P and NP complexity classes | 191 |
| Table 7.4 General formula comparison for total number of gates and qubits | 198 |
| Table 7.5 Comparison of quantum circuit cost | 201 |

| | |
|--|-----|
| Table 8.1 Detailed operations for Figure 8.21 | 236 |
| Table 8.2 Comparison analysis for quantum circuit..... | 238 |
| Table 8.3 Binary sudoku 4×4 | 247 |

List of Figures

| | |
|--|----|
| Figure 2.1 Operator transformation..... | 12 |
| Figure 2.2 Bloch sphere [8]..... | 13 |
| Figure 2.3 Symbol of single qubit quantum gates of X, Y, Z and Hadamard gates..... | 16 |
| Figure 2.4 Controlled NOT (CNOT) unitary matrix, gate symbol, and the truth table..... | 17 |
| Figure 2.5 Controlled-Z unitary matrix, gate symbol, and the truth table..... | 17 |
| Figure 2.6 Toffoli unitary matrix, gate symbol, and the truth table..... | 18 |
| Figure 2.7 SWAP unitary matrix and SWAP gate symbol..... | 18 |
| Figure 2.8 Controlled SWAP unitary matrix and gate symbol..... | 19 |
| Figure 2.9 Overview of the history of quantum algorithms [36]..... | 20 |
| Figure 2.10 Superconducting qubits [10]..... | 27 |
| Figure 2.11 Trapped Ions [10]..... | 28 |
| Figure 2.12 Silicon quantum dot qubits [10]..... | 29 |
| Figure 3.1 Classical oracle as a set of tentative solutions and marked element solution shown in blue..... | 33 |
| Figure 3.2 Quantum Boolean oracle in a schematic view..... | 34 |
| Figure 3.3 Schematic circuit for Grover’s algorithm [48]..... | 37 |
| Figure 3.4 Detailed view for the standard Grover’s algorithm [51] with Boolean Oracle | 37 |
| Figure 3.5 Simplified Grover iteration..... | 41 |
| Figure 3.6 Arbitrary state $ \psi\rangle$ is in between $ \omega\rangle$ and $ \alpha\rangle$, where the amplitude for all $ N\rangle$ are equal..... | 45 |

| | |
|--|----|
| Figure 3.7 Oracle reflection Uf is applied on $ \psi\rangle$ where the amplitude of the $ \omega\rangle$ becomes negative and the average overall amplitude is decreased. | 46 |
| Figure 3.8 Diffusion reflection D is applied on $Uf \psi\rangle$ where the $ \omega\rangle$ amplitude becomes positive and higher than the rest of the other amplitude..... | 46 |
| Figure 3.9 Euler's Formula..... | 48 |
| Figure 3.10 Rotation vector r from point (x_0, y_0) for angle θ to (x_1, y_1) for angle $\theta + \phi$ | 49 |
| Figure 3.11 Grover schematic for Boolean oracle with diffusor operator from [266] | 54 |
| Figure 3.12 Grover schematic for Boolean oracle with modified diffusor operator from [261]..... | 54 |
| Figure 3.13 Phase oracle in Grover algorithm. | 55 |
| Figure 3.14 Oracle design of $f = x_1 + x_2x_3$ | 58 |
| Figure 3.15 Oracle design $f = x_1 + x_2x_3$ with diffusor operator from [266] | 59 |
| Figure 3.16 Oracle $f = x_1 + x_2x_3$ with modified diffusor operator from [261] | 67 |
| Figure 4.1 Gate symbol: NOT, CNOT, 3-qubit Toffoli gates..... | 91 |
| Figure 4.2 Some symbols for quantum gates of Controlled-nth root of NOT gate and their inverse (\dagger) dagger or conjugate..... | 93 |
| Figure 4.3 3-bit Toffoli gate represented as controlled- V/V^\dagger and CNOT gates..... | 94 |
| Figure 4.4 (I) Three-bit Peres gate and its representation using controlled- V/V^\dagger | 94 |
| Figure 4.5 A Peres gate realized on five qubits..... | 95 |
| Figure 4.6 Generalized Peres gate realized on n qubits. | 96 |
| Figure 4.7 Three-qubit quantum counter. | 98 |

| | |
|--|-----|
| Figure 4.8 Convert sum term to product term using De Morgan’s law..... | 100 |
| Figure 4.9 Traditional oracle for Multiple input Toffoli gate used as global AND gate $f = a + b + ca + b + cb + c$ | 101 |
| Figure 4.10 Improved version of the part the oracle $f = a + b + ca + b + cb + c$ | 102 |
| Figure 4.11 Improved and optimized version of the part the oracle $f = a + b + ca + b + cb + c$ | 103 |
| Figure 4.12 Improved complete oracle using the quantum counter..... | 104 |
| Figure 4.13 Oracle from Figure 4.12 is used twice inside the Grover’s algorithm..... | 104 |
| Figure 4.14 Measurement of the Boolean variables and the outcome of function from Figure 4.13 | 105 |
| Figure 4.15 Oracle with counter $f a, b = (a + b)(a + b)(a + b)(a + b)$ | 106 |
| Figure 4.16 Oracle with a “counter circuit” and with a “threshold with comparator” circuit. | 108 |
| Figure 4.17 MAX-SAT verification such that 00, 01, 10, 11 for ab has always MAX-SAT terms equal to 3 for $f a, b = (a + b)(a + b)(a + b)(a + b)$ | 109 |
| Figure 4.18 Oracle from Figure 4.15 applied Grover’s algorithm. | 110 |
| Figure 4.19 Measurement from Figure 4.18 | 110 |
| Figure 4.20 Comparison of required numbers of ancilla qubits for our oracle and the traditional oracle..... | 112 |
| Figure 4.21 Calculation of the quantum cost for the 3-bit counter. | 113 |
| Figure 4.22 Part of the product of SOP oracle that realizes SOP function $f = ab + bc + ac$ | 114 |

| | |
|--|-----|
| Figure 4.23 Realization of the Oracle for $f = (ab + ac)(abc + bc)$, with POSOP SAT. | 115 |
| Figure 4.24 Realization of Oracle $f = ab \oplus bc \oplus ac$ for ESOP SAT realized in Grover's Algorithm. | 116 |
| Figure 4.25 XOR clauses solved by Gaussian Elimination | 125 |
| Figure 4.26 Classical oracle design for XOR SAT function $f = (ab \oplus ac)(abc \oplus bc)$. | 127 |
| Figure 4.27 Quantum oracle for XOR SAT $f = (ab \oplus ac)(abc \oplus bc)$ with quantum counter block. | 128 |
| Figure 4.28 Grover's algorithm applied XOR SAT oracle in Figure 4.27 | 128 |
| Figure 4.29 Measurement of the Boolean variables from Figure 4.28. | 129 |
| Figure 4.30 Quantum oracle for CNF-XOR SAT function $f a, b, c = a + b + ca + b + cb + c(ab \oplus ac)(abc \oplus bc)$. | 130 |
| Figure 4.31 Grover's algorithm applied CNF-XOR SAT oracle form Figure 4.30. | 131 |
| Figure 4.32 Measurement of the Boolean variables of function from Figure 4.31. | 131 |
| Figure 5.1 Association rule mining phases. | 138 |
| Figure 5.2 Generating maximum frequent itemsets using the Apriori Algorithm. | 143 |
| Figure 5.3 Construction of $SCSn, l$ of a two-qubit gate [184] | 155 |
| Figure 5.4 Construction of $SCSn, l$ of a three-qubit gate [184] | 155 |
| Figure 5.5 Dicke state $ D_{35}$ circuit, where ln gates are shorthand for Y -rotation $Ry2\cos - 1ln$ [184]. | 156 |
| Figure 5.6 Four-controlled qubits of a quantum counter. | 157 |
| Figure 5.7 One-qubit comparator (a) equal or greater than. (b) not less than. | 158 |

| | |
|--|-----|
| Figure 5.8 Four-bit quantum comparator built using IBM Qiskit simulator. | 159 |
| Figure 5.9 Full quantum oracle circuit for $abd + abe + ade + bde + abd + abe +$ $ade + abd + acd \geq 2$ | 160 |
| Figure 5.10 Full quantum algorithm circuit design for the associate rule mining using the diffusor in Figure 3.11 | 161 |
| Figure 5.11 Histogram of measured value from the Figure 5.10 | 162 |
| Figure 5.12 Proposed algorithm design for associate rule mining with diffusor in Figure 3.11. | 164 |
| Figure 5.13 Histogram to compare number of qubits in our design (blue) and other designs from [182, 183] (orange). | 164 |
| Figure 6.1 Integer points of a line. | 168 |
| Figure 6.2 undirected graph of five nodes. | 174 |
| Figure 6.3 Quantum walk algorithm based on SKW design. | 177 |
| Figure 6.4 Three-dimensional hypercube. | 179 |
| Figure 6.5 Direction oracle consists of: Grover coin for coin operator and shift operator. | 180 |
| Figure 6.6 Oracle circuit for $fa, b, c = a + b + ca + b + cb + c$ | 180 |
| Figure 6.7 Quantum walk algorithm for solving $fa, b, c = a + b + ca + b + cb + c$.. | 181 |
| Figure 6.8 Measurement of the Boolean variables of $fa, b, c = a + b + ca + b + cb + c$ | 181 |
| Figure 6.9 Johnson graph nk , where $k = 3$ for 3-itemset of $n = 5$ for $abcde$ | 183 |
| Figure 6.10 Oracle with coin and multi-controlled SWAP gates for the shift operators. | 185 |

| | |
|--|-----|
| Figure 7.1 Main classical complexity classes. | 191 |
| Figure 7.2 Quantum complexity circuit schematic. | 193 |
| Figure 7.3 relationship P, NP and BQP complexities [200] | 194 |
| Figure 8.1 (a) Truth table in form of a Karnaugh Map for all prime implicants of a SOP circuit. (b) Covering table for SOP function from (a). | 213 |
| Figure 8.2 Quantum oracle for $f = A \cdot B \cdot C \cdot DA + EB + ED + EC + E$ | 215 |
| Figure 8.3 Grover's algorithm with 4 iterations using the oracle circuit from Figure 8.2 | 215 |
| Figure 8.4 Measurement of the Boolean variables from Figure 8.3..... | 216 |
| Figure 8.5 Implication graph for the set 1,2,3,4,5,6..... | 217 |
| Figure 8.6 Covering-closure table based on the implication graph from Figure 8.5 | 218 |
| Figure 8.7 Quantum oracle for solving the Implication Graph Problem for $A + DA + C + EBA + D + EA + BC(C + D)(E + B)$ | 220 |
| Figure 8.8 Grover's algorithm with 2 iterations using the oracle circuit..... | 221 |
| Figure 8.9 Measurement of the Boolean variables from $A + DA + C + EBA + D + EA + BC(C + D)(E + B)$ | 221 |
| Figure 8.10 Quantum oracle for the Binate Covering Problem with $x_0 + x_3(x_2 + x_3) (x_1 + x_2)(x_1 + x_2 + x_3)$ | 223 |
| Figure 8.11 Grover's algorithm with 2 iterations using the oracle circuit..... | 224 |
| Figure 8.12 Measurement of the Boolean variables from Binate Covering Problem $x_0 + x_3(x_2 + x_3) (x_1 + x_2)(x_1 + x_2 + x_3)$ | 224 |

| | |
|---|-----|
| Figure 8.13 (a) Truth table of FSM (b) FSM triangular table generated based on the table from (a). | 227 |
| Figure 8.14 (a) Compatibility graph (b) Incompatibility graph. | 228 |
| Figure 8.15 Covering-Closure table for the FSM..... | 229 |
| Figure 8.16 FSM Oracle Design with counter circuit and threshold with comparator. .. | 231 |
| Figure 8.17 Steps to create an exactly minimized deterministic FSM using binate covering problem. (a) the table created directly from the solution to the covering-closure problem; (b) a non-deterministic automaton created from the table in (a); (c) one deterministic automaton in (b) | 232 |
| Figure 8.18 Half adder logic circuit with truth table. | 234 |
| Figure 8.19 Full adder logic circuit with truth table..... | 234 |
| Figure 8.20 Ripple carry adder proposed in [247] | 236 |
| Figure 8.21 Quantum circuit design for full adder proposed in [250]..... | 236 |
| Figure 8.22 3-bit ripple carry adder..... | 237 |
| Figure 8.23 Quantum circuit for half adder in [251] | 238 |
| Figure 8.24 Multiplication 4-bit number $x_3x_2x_1x_0$ and $y_3y_2y_1y_0$ | 239 |
| Figure 8.25 First stage of partial products. | 240 |
| Figure 8.26 Second stage of partial products. | 240 |
| Figure 8.27 Final stage of partial products. | 241 |
| Figure 8.28 Quantum circuit to generate the partial products in the first stage. | 242 |
| Figure 8.29 Schematic for quantum circuit design of Wallace tree multiplier. | 243 |
| Figure 8.30 Quantum oracle schematic for solving a and b | 243 |

| | |
|---|-----|
| Figure 8.31 Oracle circuit for $x_0 \oplus x_1 + x_1 \oplus x_2$ | 248 |
| Figure 8.32 Oracle balance circuit for row $x_0x_1x_2x_3$ | 249 |
| Figure 8.33 Oracle balance circuit for all rows and columns. | 250 |
| Figure 8.34 Oracle circuit for term $x_0 \oplus x_4 + x_1 \oplus x_5 + x_2 \oplus x_6 + x_3 \oplus x_7$ | 251 |
| Figure 8.35 Oracle circuit for 4×4 binary sudoku. | 252 |

1 INTRODUCTION

We live in a digitalized era where computer power for complex problems in science and engineering revealed the limitations of classical computers, even supercomputers. For instance, combinatorial optimization problems are computationally expensive in classical computers. Combinatorial optimization problems are widely studied in fundamental academic research and in solving real-life problems. The combinatorial optimization problems involve finding an arrangement of items, permutations, and combinations that optimize a desired goal from many possible solutions. In the case of an exhaustive search, it is necessary to iterate through all arrangements to identify the optimal solution depending on the desired goal in classical computers. As the number of items increases, the more possible solutions increase exponentially, and the more difficult it is to find the optimal solution. The quantum computer leverages quantum mechanical properties to solve certain problems faster than classical computers.

Over the past decade, the field of quantum computing has emerged as a powerful approach to solving constraint stratification and optimization problems. Quantum computing can solve some NP-hard problems in time with a significant speedup over classical computers due to superposition, entanglement, and quantum parallelism. Some of these NP-hard problems are in combinatorial optimization problems which can apply quantum search algorithms, such as Grover's search algorithm and quantum walk algorithm. Combinatorial optimization problems typically contain many items, and the goal is to find the optimal solutions depending on the desired goal. Those items and their computations are represented as qubits in quantum algorithms. The challenge in quantum

computing is how to utilize these qubits, which require an efficient quantum algorithm design.

The race into a new era of quantum computers began as more applications were developed. Thus, a quantum performance benchmark is needed to evaluate the performance of quantum computers. According to IBM [1], the performance of the quantum computers is measured based on three metrics: scale, quality, and speed. Scale is the number of qubits in the design. Quality is related to the depth of the quantum circuits, which depends on the technology that implements the quantum circuits. Speed is related to the cost of the quantum circuit, which is measured by the number of primitive circuits in the design. In addition to these three metrics, accuracy is added to the performance metric [2]. Accuracy is related to how accurately one can measure the quantum states.

1.1 Objective of the Research

To solve many real-world problems with many variables and many possible solutions, the required number of qubits of scalable hardware for quantum computing is the bottleneck in current generation of quantum computers. In this dissertation, we will demonstrate advanced scalable building blocks for the quantum search algorithms that have been implemented in Grover's search algorithm and quantum walk algorithm. The scalable building blocks are used to reduce the required number of qubits in the design. Scaling the required number of qubits in design is critical in the current generation of quantum computers because the number of qubits determines the degree of computational complexity. The more qubits a quantum processor holds, the more complex and valuable the quantum circuits can be designed [3]. Thus, to scale and optimize the number of

qubits that can accommodate in quantum algorithm design is directly reflected on the performance. Increasing qubit count can be used as a resource to improve quality and speed [1].

In quantum search algorithms, qubits are used to represent the parameters of a given problem and also to represent the computation of the problem. Computation qubits are known as ancilla qubits. The challenge in the quantum search algorithms is how to design the computation qubits so that the number of qubits in design is small, so large problems do not consume more qubits. My proposed architectures effectively scale and optimize the number of variable qubits and computational qubits needed to solve large problems with a limited total number of qubits.

The search space of quantum search algorithms is created by using the Hadamard operator to create superposition states. However, creating superposition states for problems that do not need all superposition states decreases the accuracy of the measured states. There are many problem representations, such as symmetric Boolean functions, frequent pattern, and element distinctness, that search space for them is a subspace of the superposition space from the Hadamard operator. I propose an efficient quantum circuit design that the user has control to create the subspace superposition states for the search space as needed. Using only the subspace states as a superposition of search space for quantum search algorithm will increase the rate of the correct solutions.

1.2 Dissertation Outline

The rest of this dissertation discusses quantum search algorithms for constraint satisfaction and optimization problems using Grover's search and Quantum Walk algorithms with advanced oracle design. The dissertation is organized as follows.

Chapter 2 is an introduction to quantum computing that covers the basics of linear algebra for quantum computing, quantum gates, an overview of the history of quantum algorithms, quantum hardware technologies, and software.

Chapter 3 is dedicated to a detailed description of Grover's search algorithm in different formats, such as matrix representation, geometric representation, and algebraic representation. Also, we discuss types of oracles and their implementations with examples that are explained step by step. Finally, we discuss applications of Grover's search algorithm.

Chapter 4 discusses a quantum algorithm for variant satisfiability and maximum satisfiability. This chapter presents our innovative advanced quantum oracle designs for SAT and MAX-SAT problems. Also, we present variants of SAT oracle formats such as circuits based on SOP, POS, ESOP, and XOR logic types. We provided a full implementation of our oracle design for MAX-SAT problems in the QISKIT simulator.

Chapter 5 presents a quantum algorithm for mining frequent patterns for association rule mining. In this chapter, we design a new version of Grover's algorithm using different blocks, such as the Dicke states, to create the superposition states, quantum counter, and quantum comparator blocks that are for frequent use in my methodology.

Chapter 6 first studies the quantum random walk algorithm, and then we present mining frequent patterns for association rule mining as a practical application to the quantum walk using our methodology of designing the shift operator design. Quantum complexity theory and the connection between quantum and classical complexity are covered in Chapter 7.

Chapter 8 presents methodologies for various problems using our advanced quantum circuit design. These problems include constraint satisfaction and optimization problems, such as covering problems with practical applications and logic puzzles. We also present a quantum oracle circuit design for arithmetic for constraint satisfaction and optimization problems. Finally, the conclusions of the dissertation, my achievements, my publications, and future work are presented in Chapter 9.

2 INTRODUCTION OF QUANTUM COMPUTING

Classical computing is based on classical bits, in which data are presented as binary 0 and 1 states. Classical computing operates based on digital logical gates such as NOT, AND, OR, and XOR gates. The data are processed sequentially, and the output is deterministic. Classical computing power increases as the number of transistors increases. Moore's Law [4] predicts that the number of transistors on a chip will approximately double every two years. Based on scaling the transistor density, high performance, and more energy efficiency are also achieved [5]. Information technology overall and especially artificial intelligence are continuing to drive rapid technology innovation, such that classical computing, including supercomputers, has limitations to handle the immense computational task required for complex problems and the workloads of artificial intelligence. A new era in information technology will begin as Moore's Law gradually ends [6, 7]. The two main emerging technologies beyond classical computing are quantum computing and brain-inspired computing [6].

Quantum computing is a promising technology that pushes boundaries to solve complex computational problems much faster than in the case of classical computing. Quantum computing is based on quantum mechanical principles to manipulate data. The data are represented as a quantum bit, "qubit" in which 0, 1, and all superposition states with both 0 and 1 can exist simultaneously. The operation of quantum computing is performed using quantum gates, and data are processed in parallel, which lets quantum computing solve problems faster than in classical computing. Quantum computing has certain distinctive properties, such as superposition and entanglement. Superposition

refers to the ability of qubit to exist in multiple states 0 and 1 at once. The superposition principle allows quantum parallelism such that the quantum computer is able to process a vast number of operations in parallel. Entanglement is a state when two or more qubits are entangled, so an action performed on one of them can immediately affect the other, no matter how far apart they are. When one qubit of entangled pair is measured then the other qubit of the pair is revealed. To understand quantum computing, we need to address some essential mathematics for quantum computing.

2.1 Introduction of linear algebra for quantum computing

We first introduce some preliminary and basic concepts about quantum computing such as linear algebra for quantum computing [47] to understand better the operations on quantum gates. Also, we explain the main quantum gates and their operations [48]. The quantum state is represented as a vector and quantum gate is represented as matrix. The vector in quantum mechanics is represented by Dirac notation or *bra-ket* notation. If a is a column vector, then this is called a ket vector, which is denoted by $|a\rangle$. If a is a row vector, then this is called a bra vector, which is denoted by $\langle a|$ and equals the conjugate transpose of $|a\rangle$.

$$|a\rangle = \begin{bmatrix} a_0 \\ \cdot \\ a_n \end{bmatrix}, \quad \langle a| = [a_0^\dagger, \dots, a_n^\dagger]$$

Quantum logic system consists of two basis states that are labeled as $|0\rangle$ and $|1\rangle$.

These two basis states are identified by column vector:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Based on the basis states, we can write an arbitrary state:

$$|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$$

Where α and β are complex coefficients. If we measured $|0\rangle$ in $|\varphi\rangle$ then we got a probability $|\alpha|^2$, and $|1\rangle$ with probability $|\beta|^2$ that satisfy $|\alpha|^2 + |\beta|^2 = 1$. The quantum measurement of the qubit provides the classical information of the qubits based on the observed qubits in the arbitrary state.

2.1.1 Inner Product (dot product)

The inner product is a generalization of the dot product which returns a scalar. The inner product of two vectors $\langle a|$ a bra (row vector) and $|b\rangle$ a ket (column vector) is denoted as

$\langle a|b\rangle$ where $\langle a| = [a_0^\dagger, a_1^\dagger, \dots, a_n^\dagger]$ and $|b\rangle = \begin{bmatrix} b_0 \\ \dots \\ b_n \end{bmatrix}$. The inner product is:

$$\langle a|b\rangle = a_0^\dagger b_0 + a_1^\dagger b_1 + \dots + a_n^\dagger b_n$$

The inner product of two orthogonal vectors is always zero. For example, if we have the orthogonal vectors $|0\rangle$ and $|1\rangle$, then the inner product is:

$$\langle 1|0\rangle = [0,1] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0 * 1 + 1 * 0 = 0$$

The inner product of two equal vectors is always one:

$$\langle 1|1\rangle = [0,1] \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1 \quad \langle 0|0\rangle = [1,0] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$$

2.1.2 Outer Product

The outer product returns a matrix. Outer product of two vector $|a\rangle\langle b|$, where $\langle b|$ is equal the conjugate transpose of $|b\rangle$.

$$|a\rangle\langle b| = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \begin{bmatrix} b_0^\dagger & b_1^\dagger & \dots & b_n^\dagger \end{bmatrix} = \begin{bmatrix} a_0 b_0^\dagger & a_0 b_1^\dagger & \dots & a_0 b_n^\dagger \\ a_1 b_0^\dagger & a_1 b_1^\dagger & & \vdots \\ \vdots & \ddots & & \vdots \\ a_n b_0^\dagger & & \dots & a_n b_n^\dagger \end{bmatrix}$$

2.1.3 Tensor Product

Tensor products are multiplications of vector spaces together. Tensor product doesn't require taking one of the vector's conjugate transposes like the outer product. Every element in the first vector is multiplied with all elements of the second vector. If $|a\rangle =$

$\begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$ and $|b\rangle = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$ then the tensor product $|a\rangle\otimes|b\rangle$ is:

$$|a\rangle\otimes|b\rangle = \begin{bmatrix} a_0 \times \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \\ a_1 \times \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_0 b_0 \\ a_0 b_1 \\ a_1 b_0 \\ a_1 b_1 \end{bmatrix}$$

2.1.4 Orthogonal vs Orthonormal

Two vectors are orthogonal if their inner product (dot product) is zero. Also, in geometric view, both vectors are perpendicular to each other. For example, $|\omega\rangle = [1, 0]$ and $|\nu\rangle = [0, 1]$ are orthogonal with respect to inner product.

$$\langle\omega|\nu\rangle = [1,0] \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0$$

We define the norm of vector $|\nu\rangle$ by

$$\| |v\rangle \| = \sqrt{\langle v|v\rangle}$$

A unit vector is a vector $|v\rangle$ such $\| |v\rangle \| = 1$. We also say that $|v\rangle$ is normalized if $\| |v\rangle \| = 1$. Another way of understanding the norm is a magnitude (size) of a vector by using the Euclidean norm.

$$\| |v\rangle \| = \sqrt{\sum_{i=1}^n v_i^2}$$

Example: $|v\rangle = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$ then $\| |v\rangle \| = \sqrt{1 + 4} = \sqrt{5}$. It is convenient to talk of

normalization of a vector by dividing by its norm; thus $\frac{|v\rangle}{\| |v\rangle \|}$ is the normalized form of $|v\rangle$.

A set of vectors is orthonormal if each vector is a unit vector and distinct vectors in the set are orthogonal. For instance: $|v\rangle$ and $|\omega\rangle$ are orthonormal if $\langle \omega|v\rangle = 0$ (orthogonal) and $\| |\omega\rangle \| = \| |v\rangle \| = 1$ (unit vector). For example, two vectors $|\omega\rangle = [1, 0]$ and $|v\rangle = [0, 1]$ then:

$$\langle \omega|v\rangle = [1, 0] \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0$$

$$\| |v\rangle \| = \sqrt{0 + 1} = 1$$

$$\| |\omega\rangle \| = \sqrt{1 + 0} = 1$$

2.1.5 Eigenvalues and Eigenvectors

A square matrix A , eigenvector v is a vector that corresponds to the matrix A and satisfies the following equation:

$$Av = \lambda v \implies (A - \lambda I)v = 0$$

Where λ is scalar value called an eigenvalue of A and v is called an eigenvector of A . To determine the eigenvector of matrix A , first we need to solve the eigenvalues by using a characteristic equation, which is called the determinant (symbol $||$ or denoted $\det A$):

$$|A - \lambda I| = 0$$

Let say $A = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$ then the determinant of A is:

$$\det A = \left| \begin{bmatrix} a & b \\ c & d \end{bmatrix} \right| = ad - bc$$

Inverse of matrix A^{-1}

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \implies A^{-1} = \frac{1}{\det A} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

The eigenvalues and eigenvectors are fundamental properties in quantum computing, such that all measurements are calculated based on the eigenvalues of the operators to observe the real classical values.

2.1.6 Unitary Operator

Unitary operator is a very important property in which all quantum gates and circuits in quantum computing are unitary. A unitary operator U does not change the norm of a vector (or, here, a norm of a quantum state). So, if state $|\psi\rangle$ is normalized, then state

$U|\psi\rangle$ is normalized too. In general, unitary operator, technically is any operator that does not change the length (inner product) of the vectors or functions that it operates on. A unitary operator (which is represented as a unitary matrix) is an operator that can change either the coordinates or the state itself. The requirement is that it does not change the magnitude (norm) of the state. As a matrix, its inverse is the same as the transpose of the complex conjugate of the matrix. If \mathbf{A} is unitary matrix, then $\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger\mathbf{A} = \mathbf{I}$, and $\mathbf{A}\mathbf{A}^T = \mathbf{A}^T\mathbf{A} = \mathbf{I}$ and hence $\mathbf{A}^{-1} = \mathbf{A}^T$.

Operator is a transformation on a quantum state. Operator O maps one state vector (quantum state), $|\psi\rangle$, into another $|\psi'\rangle$. In Figure 2.1, $O|\psi\rangle = |\psi'\rangle$, operator O is applied on the quantum state $|\psi\rangle$ that some action takes place, and a new state is created $|\psi'\rangle$. The operators are quantum gates such as any rotations, and quantum circuits on any number of qubits.

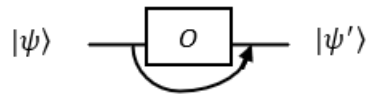


Figure 2.1 Operator transformation.

2.2 Quantum Gates

A quantum gate is simply an operator that acts on qubits and changes qubits between states, such operators will be represented by unitary matrices. Mathematically, a quantum gate with n qubit input can be represented as a $2^n \times 2^n$ unitary matrix. Quantum gates are the analogues to classical logic gates for classical computers. Quantum gates are reversible such that their inverse operation recovers the original state of the qubits. When

quantum gate is controlled, it is known as a controlled gate. Controlled gate acts on n qubits such that some of n qubits are controlling the operation of the gate. When the controlled qubits are active then the quantum gate performs an operation of other qubits. We describe a few common quantum gates such as Pauli gates, Hadamard gate and controlled NOT gate. The quantum state can be represented as a unit sphere known as the Bloch sphere. The Pauli X, Y, and Z gates can be easily viewed as geometric representation on Bloch sphere in Figure 2.2. More details about quantum gates and their operations can be found in [48].

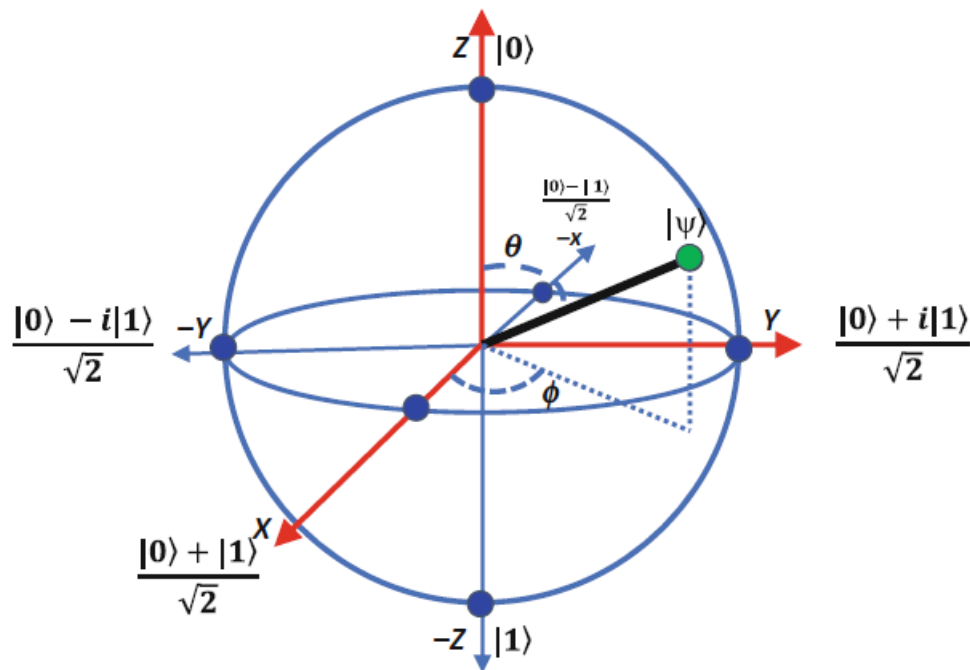


Figure 2.2 Bloch sphere [8]

2.2.1 Pauli gates

Pauli gates are X, Y and Z gates. These gates can be represented by 2 by 2 matrices or rotations through π radians around the X, Y and Z axis respectively. Pauli X gate is

single qubit gate that rotates π around x-axis. X gate is known as a NOT gate, that is equivalent of the classical invert logic gate. X gate can be represented as a unitary Pauli-X matrix:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = |0\rangle\langle 1| + |1\rangle\langle 0|$$

If X gate applied on $|0\rangle$ state, then it flips into $|1\rangle$ state and if X gate is applied on $|1\rangle$ state, then it flips into $|0\rangle$ state.

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

Pauli Y gate is single qubit gate that rotates π around y-axis of the Bloch sphere. The rotation on the y-axis is imaginary space. Y gate is X gate multiplied by i and phase flips when Y gate is applied on $|1\rangle$ state. Thus, Y gate is bit and phase flip.

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = i|1\rangle\langle 0| - |0\rangle\langle 1|$$

$$Y|0\rangle = i|1\rangle$$

$$Y|1\rangle = -i|0\rangle$$

Pauli Z gate is a single qubit gate that rotates π around z-axis of the Bloch sphere. Z gate flips the phase of the $|1\rangle$ state. Z gate is called phase gate shift or zero phase shift gate.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = |0\rangle\langle 0| - |1\rangle\langle 1|$$

$$Z|0\rangle = |0\rangle$$

$$Z|1\rangle = -|1\rangle$$

2.2.2 Hadamard Gate

One of the main properties of the quantum system is superposition. There are many ways to create superposition, but the most well-known method is using the Hadamard gate operation. The Hadamard gate creates uniform superposition of the two basis states that the measurement of which will have equal probability to become 1 or 0. The Hadamard gate is a single qubit gate that rotates π between x and z -axes. The Hadamard gate rotation maps the x -axis to z -axis and vice versa and inverts the y -axis. The unitary matrix of Hadamard gate is:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle$$

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle$$

$H^{\otimes n} = H \otimes H \otimes \dots \otimes H$ repeated n times

$$H^{\otimes n}|0\rangle^{\otimes n} = H|0\rangle \otimes H|0\rangle \otimes \dots \otimes H|0\rangle$$

$$= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \dots \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$= \frac{1}{\sqrt{2^n}}(|00 \dots 00\rangle + |00 \dots 01\rangle + |00 \dots 10\rangle + \dots + |11 \dots 10\rangle + |11 \dots 11\rangle)$$

$$= \frac{1}{\sqrt{2^n}} (|0\rangle + |1\rangle + |2\rangle + |3\rangle + \dots + |2^n - 2\rangle + |2^n - 1\rangle) = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$

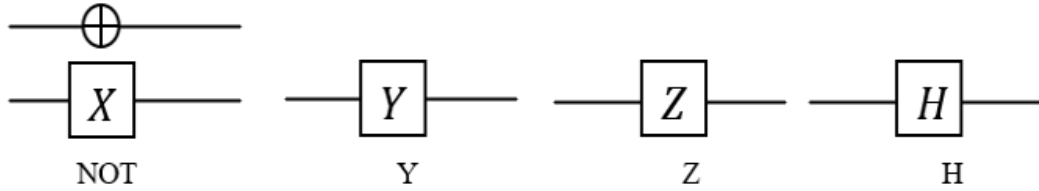


Figure 2.3 Symbol of single qubit quantum gates of X, Y, Z and Hadamard gates.

The quantum gates such as Pauli gates and Hadamard gate are single qubit gates as can be seen in Figure 2.3. However, with two or more controlled qubits, the X, Y, and Z gates can be controlled gates. The operation of the controlled gates is performed only when the control qubits are active.

2.2.3 Controlled NOT Gate

Controlled NOT (CNOT) gate is X gate controlled by another qubit. The CNOT gate consist of two qubits that one qubit is the control qubit, and the other one is the target qubit. The target qubit flips when the control qubit is $|1\rangle$ and the target qubit is unchanged when the control qubit is $|0\rangle$. The CNOT gate can be represented as unitary matrix as can be seen in Figure 2.4. The \oplus symbol is NOT gate that the operation is applied on the target qubit when the control qubit is one. For instance, $|q\rangle$ is the control qubit and $|a\rangle$ the target qubit in Figure 2.4. The unitary matrix, symbol of the gate and the truth table of CNOT gate can be seen in Figure 2.4.

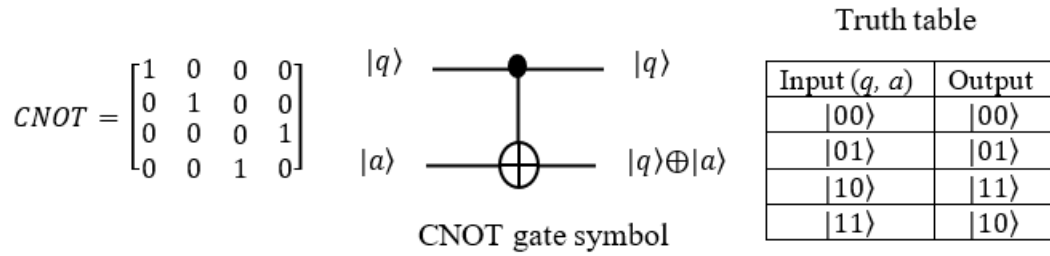


Figure 2.4 Controlled NOT (CNOT) unitary matrix, gate symbol, and the truth table.

2.2.4 Controlled-Z Gate

Controlled-Z (CZ) gate is two qubit gate that flips the phase of the target qubit only when the target qubit is $|1\rangle$ state and the control qubit is $|1\rangle$. As can be seen in Figure 2.5, the phase of the target is flipped only when both $|q\rangle$ and $|a\rangle$ are $|1\rangle$.

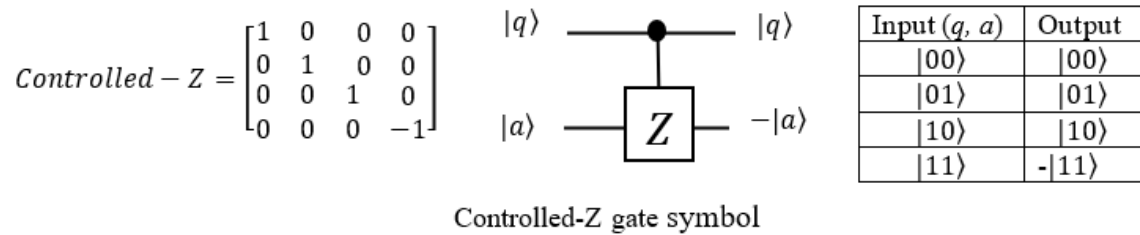


Figure 2.5 Controlled-Z unitary matrix, gate symbol, and the truth table.

2.2.5 Toffoli Gate

Toffoli gate consists of two control qubits and one target qubit. The Toffoli gate is also known as the controlled-controlled-NOT gate. The target qubit is flipped when the two control qubits are both $|1\rangle$, otherwise the target unchanged if either of the control qubits is $|0\rangle$. For instance, when both the control qubits $|ab\rangle$ are 1 then the target qubit $|c\rangle$ is flipped as can be seen truth table in Figure 2.6.

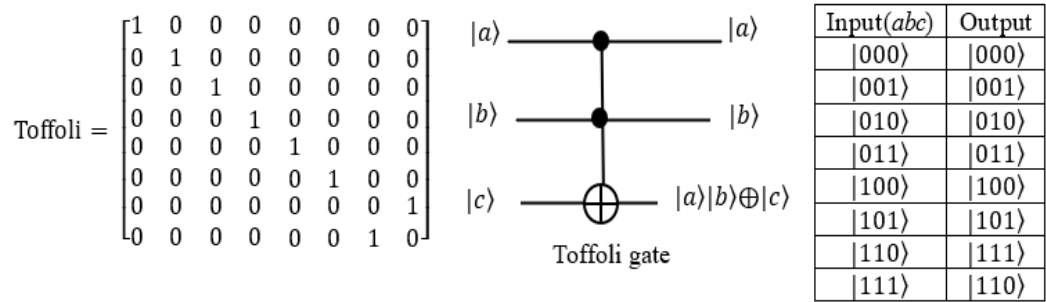


Figure 2.6 Toffoli unitary matrix, gate symbol, and the truth table.

The Toffoli gate that consist of n -controlled qubits and one target qubit is known as multi-control Toffoli gate or n -qubit Toffoli gate when n is more than two qubits. When all the n qubits are $|1\rangle$, then target qubit is flipped.

2.2.6 SWAP Gate

SWAP gate is a two qubits gate such the two qubits are swapped. The SWAP gate can be represented as a unitary matrix as can be seen in Figure 2.7.

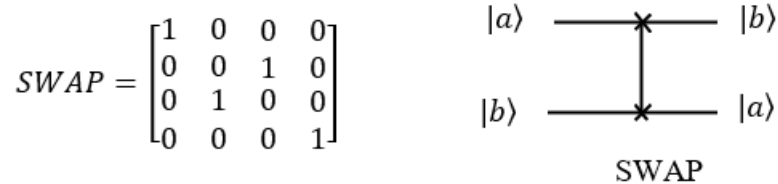


Figure 2.7 SWAP unitary matrix and SWAP gate symbol.

2.2.7 Fredkin Gate

Fredkin gate is also known as a controlled SWAP gate. Fredkin gate is three qubit gate and SWAP operation is applied two target qubits when the controlled qubit is $|1\rangle$. The controlled SWAP gate can be represented as a unitary matrix as can be seen in Figure 2.8.

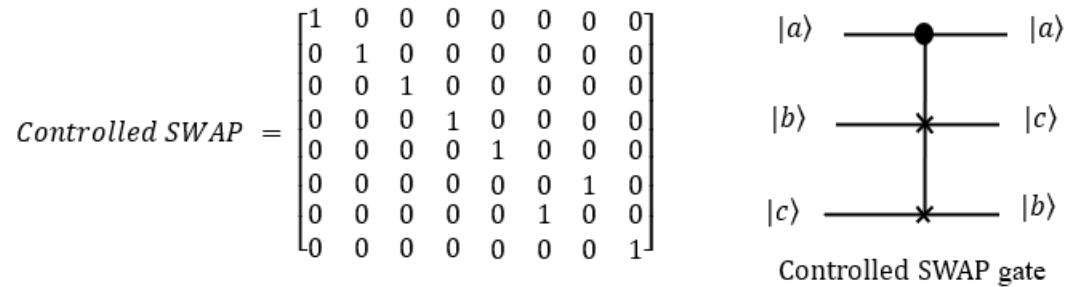


Figure 2.8 Controlled SWAP unitary matrix and gate symbol.

The field of quantum computing is an emerging technology where engineers, computer scientists, and physicists are developing two main parts: Discovering quantum algorithms that are faster than classical algorithms and building actual hardware for quantum computers. We introduce the significant milestones of quantum algorithms and quantum hardware technologies.

2.3 Overview of the history of quantum algorithms

In 1980 Paul Benioff [37] described a theoretical basis for quantum mechanical Hamiltonian models of computers. Also, in 1980 Yuri Manin [38] proposed an idea of quantum computing. In 1981 at MIT, Richard Feynman proposed the idea of using quantum computers to simulate quantum systems that are too complex to do so using classical computers. From there, the spark of quantum computing started.

Many popular quantum algorithms have been proposed since the 1980s, which can be divided into stages [36] as can be seen in Figure 2.9.

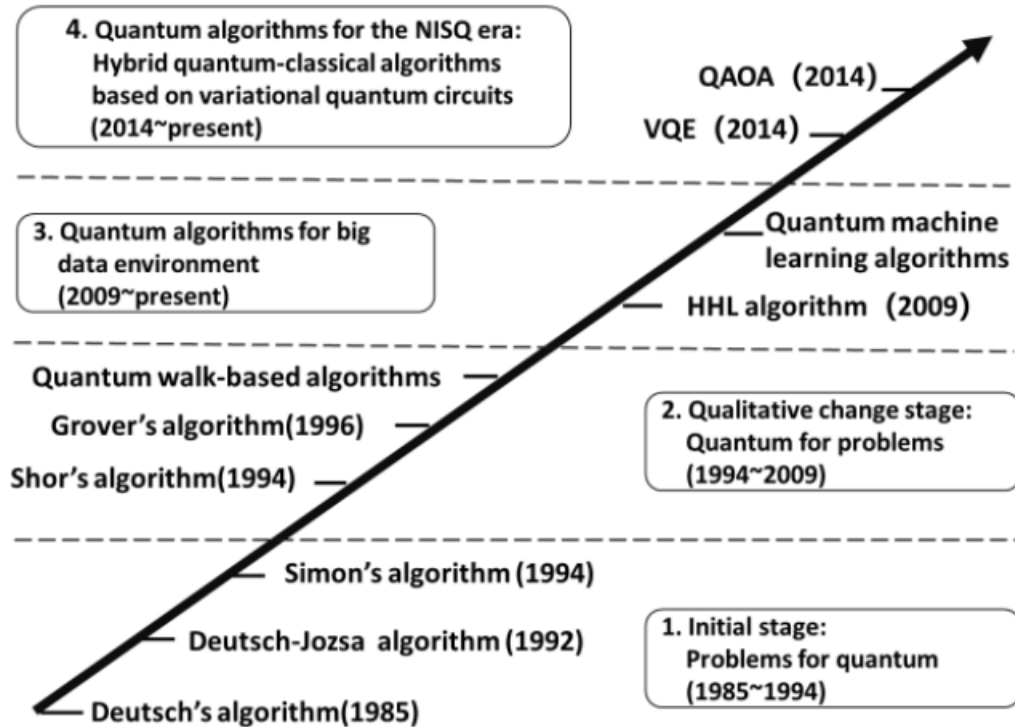


Figure 2.9 Overview of the history of quantum algorithms [36]

Stage 1: The initial stage of quantum algorithms was mathematical models that showed the quantum computer is better than the classical computer. In 1985 David Deutsch proposed a quantum Turing machine [27] to model quantum computation. This concept helped spark a lot of interest in creating quantum algorithms to take advantage of quantum physics properties. Later in 1992, David Deutsch and Richard Jozsa proposed the Deutsch-Jozsa algorithm [28]. The Deutsch-Jozsa algorithm was the first principle of quantum algorithm that performs better than the classical algorithm for this particular problem. In the Deutsch-Jozsa algorithm, we are given an oracle of Boolean function that takes an input of a bit string and returns 0 or 1. A constant of Boolean function returns either 0 or 1 for any input, and a balanced function returns half 0's and the other half return 1's. In a classical algorithm, we need to check half of the inputs plus one ($2^{n-1} +$

1) queries to determine if the function is constant or balanced. In the Deutsch-Jozsa algorithm, because of superposition, only one query determines if the function is constant or balanced. In 1994, Daniel Simon introduced the first example of a quantum algorithm (Simon's algorithm) [29] that quantum computers can provide an exponential speedup over classical computers for solving a specific problem. Simon's algorithm maps an input of bit strings to a unique output of bit strings.

Stage 2: In 1994, Peter Shor developed a quantum algorithm [30] based on Quantum Fourier Transform that makes it possible for quantum computers to factorize large integers exponentially faster than the best classical algorithm on classical computers. Shor's algorithm breaks the security of the RSA public-key cryptosystem, which is made of two large prime numbers. Finding these large prime numbers on which internet security is built allows for decrypting the messages. In 1996, Lov Grover proposed a quantum search algorithm that gives a quadratic speedup over the classical algorithm. Grover's Algorithm [31] searches an unordered array of elements to find a particular element with a given property. Grover's algorithm gives a quadratic speed up. Grover's algorithm has many applications in search and optimization problems. In 2003, a quantum random-walk search algorithm [32] was proposed, which is a more general type of quantum search algorithm. Quantum walk for graph takes steps in directions determined by coin and shift operators, a quantum oracle circuit [32].

Stage 3: In 2009, Aram Harrow, Avinatan Hassidim, and Seth Lloyd developed a new design of a quantum machine learning algorithm known as the HHL algorithm [33]. The HHL algorithm solves a linear system of equations with many machine learning

applications. Later on, following the HHL algorithm, many other quantum machine learning algorithms were proposed.

Stage 4: The current practical quantum computers struggle greatly with noise and decoherence, known as Noisy Intermediate-Scale Quantum (NISQ) machines. Hybrid quantum-classical algorithms were proposed to mitigate the challenges in NISQ. In 2014, Peruzzo et al. [34] proposed the first variational quantum algorithm known as the Variational Quantum Eigensolver (VQE). Also, The Quantum Approximate Optimization Algorithm (QAOA) [35] was proposed for the quantum annealing system. Hybrid quantum-classical algorithms use classical algorithms to offload some optimization calculations.

For the last decade, few new quantum algorithms have been introduced compared to the late 20th century, when the most well-known algorithms were discovered. However, there has still been significant progress in the quantum computing field. Why is the introduction of new quantum algorithms at a slow pace compared to classical algorithms? There are possible factors that contribute to the slowdown in the introduction of new quantum algorithms. Our method of developing new quantum algorithms faster than classical algorithms requires an understanding of quantum complexity and quantum mechanics that is different from our intuition of classical complexity [268]. A huge gap exists between the current available quantum hardware and the number of qubits needed for practical applications [267]. For the last decade, the race in the quantum computing research community has been to close the gap, such as by developing practical quantum hardware. Current efforts are focused in two directions. First, more applications for

current quantum algorithms in various domains (machine learning, chemistry, cryptography, communication, and finance) have been developed. The second is designing a practical quantum computer hardware system that is scalable and fault tolerant. In addition, the framework of programming tools and simulations for quantum algorithms in classical computers has also been developed.

The research trend for quantum algorithms is hybrid quantum-classical algorithms that combine both quantum and classical algorithms. This method shows promising results for speeding up some existing classical algorithms, such as using quantum algorithms as basic subroutines to leverage the computation strengths of quantum algorithms. These hybrid algorithms and the development of quantum algorithms from classical algorithms may not be recognized as new quantum algorithms. Table 2.1 provides some applications for quantum algorithms developed in the last decade.

Table 2.1 Some practical applications for quantum algorithms.

| Application | Title | Year | Refence |
|------------------|---|------|---------|
| Cryptography | Multiparty quantum key agreement based on quantum search algorithm. | 2017 | [269] |
| | Applying Grover’s algorithm to AES: quantum resource estimates. | 2016 | [270] |
| | Quantum attacks without superposition queries: the offline Simon’s algorithm. | 2019 | [271] |
| | An efficient quantum computing technique for cracking RSA using Shor’s algorithm. | 2020 | [272] |
| | Quantum cryptography based on the Deutsch-Jozsa algorithm. | 2017 | [273] |
| Machine learning | Quantum algorithm for linear regression. | 2017 | [274] |
| | Quantum-assisted Gaussian process regression. | 2019 | [275] |
| | Quantum regularized least squares. | 2023 | [276] |
| | A quantum deep convolutional neural network for image recognition. | 2020 | [277] |
| | Quantum generative adversarial learning. | 2018 | [278] |
| | Classical artificial neural network training using quantum walks as a search procedure. | 2021 | [279] |
| Chemistry | Quantum algorithms for quantum chemistry and quantum materials science. | 2020 | [280] |

| | | | |
|---------------|---|------|-------|
| | Quantum chemistry in the age of quantum computing. | 2019 | [281] |
| Communication | Quantum search algorithms for wireless communications. | 2018 | [282] |
| | Quantum machine learning for 6G communication networks: State-of-the-art and vision for the future. | 2019 | [283] |
| | A secure information transmission protocol for healthcare cyber based on quantum image expansion and Grover search algorithm. | 2022 | [284] |
| Finance | Quantum computing for finance: State-of-the-art and future prospects. | 2020 | [285] |
| | Quantum computing for finance. | 2023 | [286] |

2.4 Quantum Computer Hardware and Software

Quantum computing hardware technology can be divided into two main categories [39]:

- First: Gate-based quantum computing, also known as gate modal or universal quantum computer.
- Second: Quantum annealing based on analog quantum computers.

2.4.1 Quantum Annealing

Quantum annealing [40, 45] is an analog quantum computer mainly used to solve optimization problems. The problems are encoded into Quadratic Unconstrained Binary Optimization (QUBO), which is compatible with the quantum annealer architecture. The quantum annealer manipulates the qubits as analog values. Quantum annealing is less noise-sensitive than gate-based quantum computing [41]. However, to solve the real-world application problems in quantum annealing hardware, the problems should be expressed in the form of QUBO model rather than in quantum gates [43]. Most quantum algorithms are designed in terms of quantum gates that make a problem easier to design in the same domain as a classical computer. A pioneer in quantum annealing architecture

is D-Wave Inc. [42], which developed quantum machines and offers compatible software called Ocean [44].

2.4.2 Gate-based Quantum Computing

Gate-based quantum computing [39], or universal quantum computers, is based on quantum logical gates to perform computations. The sequence of gates is called a quantum circuit. The universal quantum computer is based on quantum circuits that can solve a broader range of problems using quantum algorithms. Universal quantum computers use quantum gates that are the counterparts of the classical logic gates of the classical computer.

Qubit is the basic unit of quantum information. First, qubits are encoded using the quantum states of physical objects such as the spin of electrons, photons, or other objects. Second, the quantum gates perform an operation on the qubits. The quantum gate consists of different probes such as electric fields, microwaves, laser beams, and other types of technology. There are main metrics to evaluate the design of these technologies such as:

1. Gate speed: how long the gate takes to perform a single operation.
2. Coherence time (lifetime): how long the quantum state exists.
3. Gate fidelity: related to the error rate introduced during gate operation. The error in a quantum system is related to noise and the environment surrounding the qubit.

Many types of quantum technology were proposed [9, 10], such as superconducting qubits, trapped ions, photonics, and silicon quantum dots. The physical qubit

implementation is the heart of the quantum race, and multiple companies are investigating different technologies.

2.4.3 Superconducting Qubits

Superconducting qubits [11, 12] are built using superconducting circuits. The superconducting circuit is based on a resistance-free current oscillation circuit that contains an inductor and capacitor in series as can be seen in Figure 2.10. The superconducting circuit, based on the phenomenon of superconductivity, operates at low temperatures and conducts electricity with zero resistance without energy loss. A microwave signal controls the superconducting circuit, and the microwave resonator excites the current into superposition states. Technology-wise, superconducting qubits can be built using the existing semiconductor technology. Gate operations on superconducting qubits are faster, and this is critical because practical quantum computers would probably need millions of logical gates. Superconducting qubits have some challenges, such as decoherence (short lifetime), sensitivity to noise, large footprint, and requiring very low temperatures to operate appropriately. There are several major companies developing superconducting qubits, such as IBM [13], Google [14], Intel [15], and others (Oxford Quantum Circuits, Rigetti Computing, and Quantum Circuits Inc).

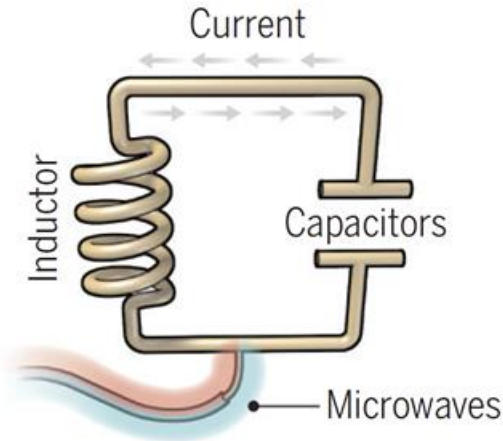


Figure 2.10 Superconducting qubits [10]

2.4.4 Trapped Ions

Ions are atoms that gained or lost an electron. Such atoms are electrically charged and trapped in an electric or magnetic field as can be seen in Figure 2.11. The trapped ions [16, 17] are controlled using lasers in a high vacuum environment that puts them in superposition states. Trapped ions have longer coherence time than the superconducting qubits; thus, trapped ions are more stable with high-quality qubits. The required cooling temperature is not as critical as for superconducting qubits. There are some main challenges for trapped ions such that trapped ions are slow gate operation. Technology-wise, trapped ions technology is not mature yet because many laser and optical systems are required. The frequency of the lasers needs to be tuned to get precise frequencies. Companies that build trapped ions quantum computer include IonQ [18], Quantinuum, and Honeywell [19].

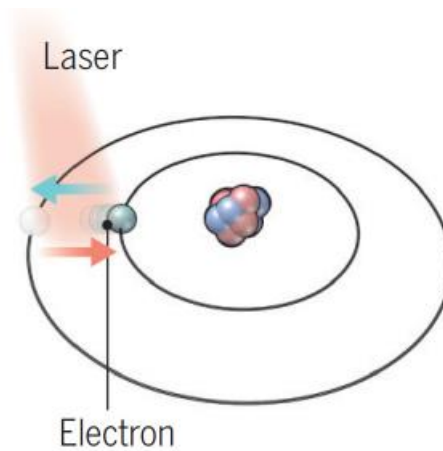


Figure 2.11 Trapped Ions [10]

2.4.5 Photonics

Photonic qubits use particles of light (photons) as a source to process quantum information. Photonic qubits [20] are implemented using phase shifters, beam splitters, and photon detectors. Photonic technology has some key advantages, such as very fast gate operation, a small footprint that can be built on existing semiconductor fabs, no decoherence, thus being more stable, and requiring no cooling and therefore can operate at a room temperature. On the other side, there are some challenges, such as that photonic qubits are more prone to noise due to photon loss, and photons do not interact with each other, making it more difficult to implement multi-qubit gates. Xanadu [21] and PsiQuantum [22] are the leading companies working on photonic quantum computers.

2.4.6 Silicon Quantum Dots / Silicon Spins

Silicon quantum dot qubits [23] are silicon-based qubits that artificial atoms are made of by adding an electron to a pure silicon atom. The electrons are confined in silicon quantum dots, where quantum dots are tiny silicon-based structures. In Figure 2.12, a microwave is used to control the spin of the electron, which is a quantum state; thus, the

silicon quantum dot is also known as a silicon spin quantum dots. Intel [24] leads this domain by leveraging existing semiconductor manufacturing technology to build silicon-based qubits. Still, silicon-based qubits need cooling like superconducting but have a longer coherence time [25]. Silicon qubits demonstrate fast gate speed and strong gate fidelity. However, the silicon qubit still has challenges, such as interference or crosstalk from atoms on the silicon chips.

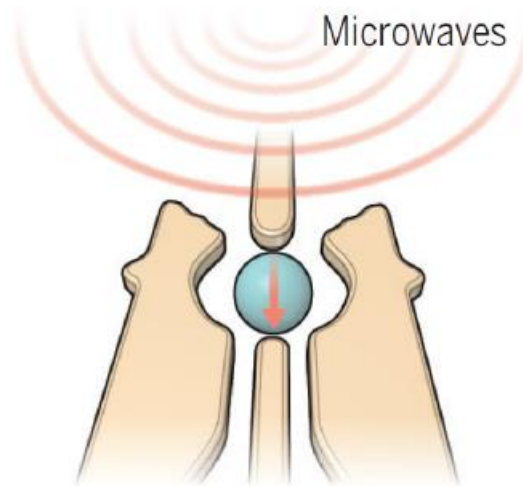


Figure 2.12 Silicon quantum dot qubits [10]

There are other architecture designs of the quantum computer [9, 26], such as topological qubits, neutral atoms, nitrogen-vacancy centers, and nuclear magnetic resonance.

Many applications and classical algorithms are developed as quantum algorithms. Quantum programming languages to describe those algorithms and software frameworks to compile the quantum operations and circuits are required in order to keep the progress of developing many applications for the current quantum algorithms. Although the

availability of quantum computer devices is limited to the public and research community, many quantum simulation frameworks are developed on classical computers. Table 2.2 shows some quantum frameworks and programming languages from leading quantum technology companies.

Table 2.2 Some quantum software frameworks

| Name | Description |
|-------------------|--|
| IBM Qiskit | Python based open-source software. It can be tested to access in IBM's real quantum device with small number of qubits. |
| Google Cirq | Python based open-source framework. |
| Microsoft Q# | Q# is standalone which domain-specific language for quantum programming. It's part of the Quantum Development Kit (QDK) which also support Qiskit and Cirq |
| Intel Quantum SDK | quantum Software Development Kit (SDK) based on C++ |

Based on the availability of these programming languages, many real-world application problems are applied to quantum search algorithms. We use IBM Qiskit to simulate the problems presented in this dissertation. Also, the quantum circuit we designed is compatible with gate-based quantum computing technologies. Let's first discuss Grover's search algorithm and the quantum walk algorithm.

3 GROVER'S SEARCH ALGORITHM

Grover's search algorithm [31] is a quantum search algorithm that searches an unordered array of N elements to find a particular element with a given property. In classical computations, in the worst case, this search takes N queries (tests and evaluations of the classical oracle). In the average case, a particular element will be found in $N/2$ queries. Grover's algorithm can find elements in \sqrt{N} queries. Grover's algorithm speeds up a classical unordered search algorithm of complexity $O(N)$ to $O(\sqrt{N})$ in the space of N objects. Hence, Grover's algorithm gives a quadratic speedup, but please note that this speedup is only for classical algorithms that are based on exhaustive unordered search. My presentation of Grover's algorithms is based on [46, 47, 48, 49, 50].

3.1 Finding an element in array using Grover's algorithm

We first introduce Grover's search algorithm in a simple way, as a computer science version or in simple math formulas, without going into detail about quantum notations. This is easily understood as the number of elements in an array instead of using explanations based on rotations and angles. More details about Grover's algorithm in the computer science version can be found in [46]. Grover's search algorithm consists of two primary operations, which we define in a simplified way:

1. Phase inversion: change the sign of the desired element a_{x^*} to a negative sign in a given array.
2. Inversion about the mean: calculate the average about the mean μ of every element a_x in the array. This operation would amplify the desired element.

The mean μ is the average of given data set $\{a_0, a_1, \dots, a_{N-1}\}$ of the array N

$$\mu = \frac{1}{N} \sum_{x=0}^{N-1} a_x$$

The inversion about the mean is calculated for every element in the data set such that:

$$a_{x(new)} = (\mu - a_{x(old)}) + \mu$$

$$a_{x(new)} = 2\mu - a_{x(old)}$$

Example: Given an array of elements $\{8, 8, 8, 8, 8\}$, apply the two operations of Grover's algorithm. First, we perform phase inversion. Let's assume our desired element a_{x^*} is the third element of the five numbers.

$$\{8, 8, -8, 8, 8\}$$

Second, calculate the mean:

$$\mu = \frac{4 \times 8 - 8}{5} = 4.8$$

Third, calculate the inversion about the mean of the desired element (-8).

$$a_{x^*(new)} = 2\mu - a_{x^*(old)} = 2 \times 4.8 - (-8) = 17.6$$

Fourth, calculate the inversion about the mean of the other elements (8, 8, 8, 8):

$$a_{x(new)} = 2\mu - a_{x(old)} = 2 \times 4.8 - 8 = 1.6$$

Thus, we got new values:

$$\{1.6, 1.6, 17.6, 1.6, 1.6\}$$

As can be seen, the third element, 17.6, is larger than the rest of the elements. This means, after performing the phase inversion and inversion about the mean, the desired element has a higher probability than the rest of the other elements.

In the remaining of this chapter, we provide a detailed explanation for Grover’s search algorithm in mathematical analysis in different ways [46, 47, 48, 49, 50], such as the algebraic analysis based on the concepts of a matrix, eigenvalue, eigenvector, and the geometric representation based on the concept of rotations, as well as the comprehensive implementation step-by-step in phase oracle and Boolean oracle. The important concepts of phase oracle versus Boolean oracle will be introduced next.

3.2 Quantum Oracle

Grover’s search algorithm starts by applying a quantum oracle and then performing amplitude amplification. An oracle is a black box operation that takes an input and gives an output, a yes/no decision. The classical oracle function is defined as a Boolean function $f(x)$ that takes a tentative solution x of the search problem of N elements as can be seen in Figure 3.1. If x is equal to ω , which is called the marked element or a solution, then $f(x) = 1$; If x is not a solution, then $f(x) = 0$.

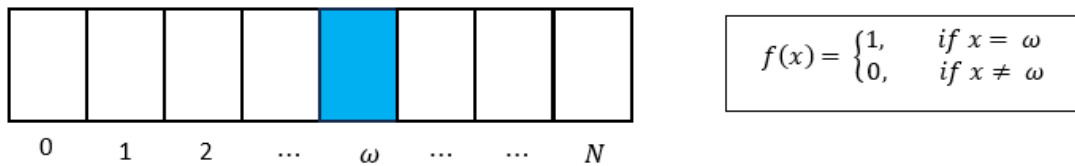


Figure 3.1 Classical oracle as a set of tentative solutions and marked element solution shown in blue.

A Boolean quantum oracle is realized as a binary reversible circuit that is used in quantum algorithms for the estimation of the value of the Boolean function realized in it.

The quantum oracle also has to replicate all input variables on the respective output qubits. If the function of the oracle is not reversible, we use ancilla qubits to make the function reversible. If the oracle uses ancilla qubits initialized to $|0\rangle$, this oracle has to return $|0\rangle$ for every ancilla qubit.

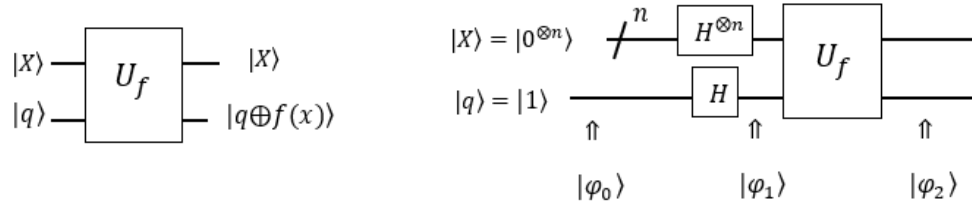


Figure 3.2 Quantum Boolean oracle in a schematic view

The quantum oracle is a unitary operator U_f such that:

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

where X is the value in search space, q is a single qubit called the oracle workspace or the oracle qubit, and \oplus is the EXOR operator (also called the addition modulo 2). Let derive the operation of the quantum oracle given in Figure 3.2. First, initialize the first qubit with $|0^{\otimes n}\rangle$ and the second qubit with $|1\rangle$:

$$|\varphi_0\rangle = |0^{\otimes n}\rangle|1\rangle$$

Apply the Hadamard gate $H^{\otimes n}$ on the initial state n qubit of $|0^{\otimes n}\rangle$, the input qubits and H on the oracle qubit $|0\rangle$:

$$|X\rangle = H^{\otimes n}|0^{\otimes n}\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$$

$$|q\rangle = H|1\rangle = \frac{1}{\sqrt{2}}[|0\rangle - |1\rangle]$$

$$|\varphi_1\rangle = H^{\otimes n}|0^{\otimes n}\rangle H|0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |X\rangle \left(\frac{1}{\sqrt{2}}[|0\rangle - |1\rangle] \right)$$

Apply the quantum oracle operation such that $|x\rangle|q\rangle \xrightarrow{U_f} |x\rangle|q \oplus f(x)\rangle$

$$\begin{aligned} |\varphi_2\rangle &= \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \{0,1\}^n} |X\rangle ([|0\rangle \oplus f(x) - |1\rangle \oplus f(x)]) \\ &= \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \{0,1\}^n} |X\rangle ([f(x) - |1\rangle \oplus f(x)]) \end{aligned}$$

$$|\varphi_2\rangle = \begin{cases} \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \{0,1\}^n} |X\rangle ([0 - |1\rangle]), & \text{if } f(x) = 0 \\ \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \{0,1\}^n} |X\rangle ([1 - 0]), & \text{if } f(x) = 1 \end{cases}$$

$$|\varphi_2\rangle = \begin{cases} \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \{0,1\}^n} |X\rangle ((-1)^{f(x)} [0 - |1\rangle]), & \text{if } f(x) = 0 \\ \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \{0,1\}^n} |X\rangle ((-1)^{f(x)} [0 - 1]), & \text{if } f(x) = 1 \end{cases}$$

$$|\varphi_2\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \{0,1\}^n} |X\rangle ((-1)^{f(x)} [0 - |1\rangle])$$

$$|\varphi_2\rangle = (-1)^{f(x)} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |X\rangle \left(\frac{1}{\sqrt{2}}[|0\rangle - |1\rangle] \right)$$

$$|\varphi_2\rangle = (-1)^{f(x)}|X\rangle|q\rangle$$

Oracle qubit $|q\rangle$ is always unchanged, but if the oracle recognizes an element as the solution, the phase of the desired state $|X\rangle$ is changed by -1 if $f(x) = 1$. This phase change is called phase inversion. Thus, a simplified formula for the quantum oracle can be written as:

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

3.3 Schematic View

In general, the schematic view of Grover's search algorithm consists of the Hadamard vector and Grover Loop operators. As shown in Figure 3.3, the n qubits in the superposition state result from applying a vector of Hadamard gates to the initial state $|0\rangle^n$. Next, we applied a repeated operator G , called the Grover's Loop. After the iteration $O(\sqrt{N})$ times of the Grover's Loop operator, the output is measured for all input qubits. Oracle can use an arbitrary number of ancilla qubits, but all these qubits must be returned to value $|0\rangle$ inside the oracle. The Grover's Loop G is a quantum subroutine that can be broken into two steps, as shown in Figure 3.4:

1. Apply the Oracle U_f . This step is called the phase inversion.
2. Perform amplitude amplification for desired states, known as inversion about the mean or diffusion operator. This step is broken into three sub steps:

- a. Apply the Hadamard transform $H^{\otimes n}$ ($H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$)

- b. Perform the condition phase shift, also known as a zero-state phase shift, in which all states receive a phase shift of -1 except for the zero state $|0\rangle$.
- c. Apply the Hadamard transform $H^{\otimes n}$

A measurement is applied after performing several repetitions of Grover Loop, the oracle and the amplitude amplification.

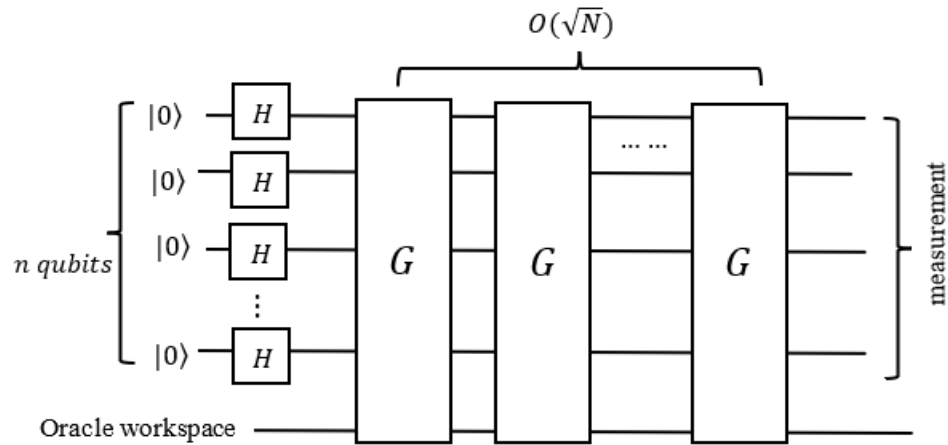


Figure 3.3 Schematic circuit for Grover's algorithm [48]

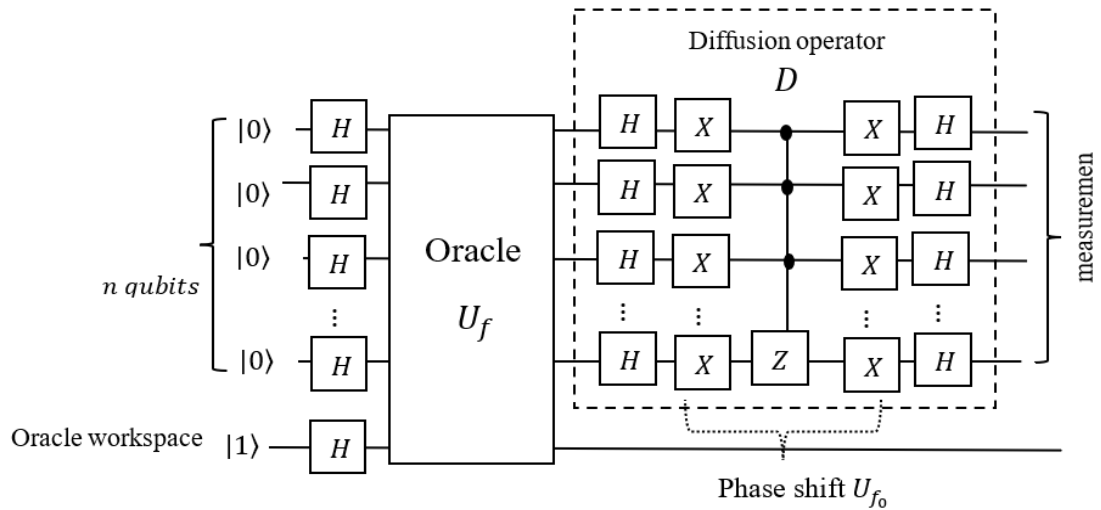


Figure 3.4 Detailed view for the standard Grover's algorithm [51] with Boolean Oracle

The phase inversion of the desired state is not sufficient to get a high probability that it cannot separate from unmarked states. When we apply a measurement of a marked state, the superposition would collapse to any one of the basic states with the same probability. We need one more trick to boost the phase of marked states, called amplitude amplification (also known as inversion about the mean, diffusion operator, or conditional phase shift).

The quantum oracle U_f is a unitary operation such that:

$$U_f|\omega\rangle = -|\omega\rangle, \quad \forall x = \omega$$

$$U_f|x\rangle = |x\rangle, \quad \forall x \neq \omega$$

Where ω is desired state. We can write U_f as a unitary operator such that:

$$U_f = I - 2|\omega\rangle\langle\omega|$$

This can be proven using the inner product:

$$|x\rangle \equiv |\omega\rangle \Rightarrow U_f|\omega\rangle = (I - 2|\omega\rangle\langle\omega|)|\omega\rangle = |\omega\rangle - 2|\omega\rangle\langle\omega|\omega\rangle = |\omega\rangle - 2|\omega\rangle = -|\omega\rangle$$

$$|x\rangle \not\equiv |\omega\rangle \Rightarrow U_f|x\rangle = (I - 2|\omega\rangle\langle\omega|)|x\rangle = |x\rangle - 2|\omega\rangle\langle\omega|x\rangle = |x\rangle$$

3.4 Matrix Representation

The oracle is a diagonal matrix, where the entry corresponding to the marked (solution) item will have a negative phase.

$$U_f = \begin{bmatrix} (-1)^{f(x)} & 0 & \dots & 0 \\ 0 & (-1)^{f(x)} & \dots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \dots & (-1)^{f(x)} \end{bmatrix}$$

Conditional phase shift (U_{f_0}): inverts the phase of all states orthogonal to the $|0\rangle$ state, meaning all signs are flipped except the $|0\rangle$ state. Conditional phase shift is also known as zero state phase shift:

$$U_{f_0}|0\rangle^{\otimes n} \rightarrow |0\rangle^{\otimes n}$$

$$U_{f_0}|x\rangle \rightarrow -|x\rangle, \forall x \neq 00 \dots 0$$

We can write U_{f_0} as unitary:

$$U_{f_0} = 2|0\rangle\langle 0| - I$$

We can proof the U_{f_0} operation:

$$|x\rangle \equiv |0\rangle \rightarrow U_{f_0}|0\rangle = (2|0\rangle\langle 0| - I)|0\rangle = 2|0\rangle\langle 0|0\rangle - |0\rangle = 2|0\rangle - |0\rangle = |0\rangle$$

$$|x\rangle \not\equiv |0\rangle \rightarrow U_{f_0}|x\rangle = (2|0\rangle\langle 0| - I)|x\rangle = 2|0\rangle\langle 0|x\rangle - |x\rangle = -|x\rangle$$

The unitary matrix representation of U_{f_0} :

$$U_{f_0} = 2|0\rangle\langle 0| - I = \begin{bmatrix} 2 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ 0 & 0 & 1 & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \vdots & \dots & \ddots & \vdots \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

The U_{f_0} is a diagonal matrix where all entries have a negative phase except the first entry. The matrix for the inversion about the mean can be driven using the diffusion operator D :

$$D = H^{\otimes n} U_{f_0} H^{\otimes n}$$

$$D = H^{\otimes n} \left[\begin{array}{cccc} 2 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & 0 \end{array} \right] - I H^{\otimes n} = H^{\otimes n} \left[\begin{array}{cccc} 2 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & 0 \end{array} \right] H^{\otimes n} - H^{\otimes n} I H^{\otimes n}$$

$$= H^{\otimes n} \left[\begin{array}{cccc} 2 & 2 & 2 & 2 \\ \frac{2}{\sqrt{N}} & \frac{2}{\sqrt{N}} & \frac{2}{\sqrt{N}} & \frac{2}{\sqrt{N}} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \vdots & \dots & \ddots & \vdots \\ 0 & 0 & 0 & 0 \end{array} \right] - I = \left[\begin{array}{cccc} \frac{2}{N} & \frac{2}{N} & \frac{2}{N} & \frac{2}{N} \\ 2 & 2 & 2 & 2 \\ \frac{2}{N} & \frac{2}{N} & \frac{2}{N} & \frac{2}{N} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \dots & \ddots & \vdots \\ 2 & 2 & 2 & 2 \\ \frac{2}{N} & \frac{2}{N} & \frac{2}{N} & \frac{2}{N} \end{array} \right] - I$$

$$D = \left[\begin{array}{cccc} \frac{2}{N} - 1 & \frac{2}{N} & \frac{2}{N} & \frac{2}{N} \\ \frac{2}{N} & \frac{2}{N} - 1 & \frac{2}{N} & \frac{2}{N} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \dots & \ddots & \vdots \\ \frac{2}{N} & \frac{2}{N} & \frac{2}{N} & \frac{2}{N} - 1 \end{array} \right]$$

$$D_{i,j} = \begin{cases} \frac{2}{N}, & \text{if } i \neq j \\ \frac{2}{N} - 1, & \text{if } i = j \end{cases}$$

if D operation is applied to a general state $|\varphi\rangle$ such that:

$$|\varphi\rangle = \sum_{k=0}^{N-1} a_k |k\rangle$$

Then we get:

$$D|\varphi\rangle = \begin{bmatrix} \frac{2}{N} - 1 & \frac{2}{N} & \frac{2}{N} & \frac{2}{N} \\ \frac{2}{N} & \frac{2}{N} - 1 & \frac{2}{N} & \frac{2}{N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{2}{N} & \frac{2}{N} & \frac{2}{N} & \frac{2}{N} - 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \\ \vdots \\ a_{N-1} \end{bmatrix} |k\rangle = \left[\frac{2}{N} \sum_{x=0}^{N-1} a_x - \sum_{k=0}^{N-1} a_k \right] |k\rangle$$

Recall:

$$\mu = \frac{1}{N} \sum_{x=0}^{N-1} a_x$$

Where μ is the mean value of a_k , then:

$$D|\varphi\rangle = \left[2\mu - \sum_{k=0}^{N-1} a_k \right] |k\rangle = \sum_{k=0}^{N-1} [2\mu - a_k] |k\rangle$$

We can summarize the Grover iteration, as can be seen in Figure 3.5, as follows:

$$G = U_f H^{\otimes n} U_{f_0} H^{\otimes n} = U_f D$$

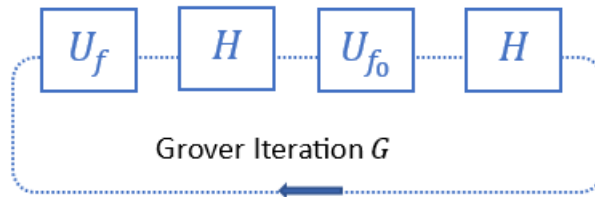


Figure 3.5 Simplified Grover iteration.

Let's assume an arbitrary superposition state $|\psi\rangle$ to represent the Grover iteration.

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

$$D = HU_{f_0}H = H(2|0\rangle\langle 0| - I)H = 2H|0\rangle\langle 0|H - H IH = 2|\psi\rangle\langle\psi| - I$$

$$G = U_f(2|\psi\rangle\langle\psi| - I)$$

Example matrix representation: suppose we are given 2 qubits such that $N = 4$ and marked element is $f(2) = 1$, and the remaining values $f(0) = f(1) = f(3) = 0$.

Here the details of the solution:

The initial state is $|\psi_0\rangle = |00\rangle$

Then apply the Hadamard operation.

$$|\psi_1\rangle = H^{\otimes 2}|00\rangle = (H|0\rangle)^{\otimes 2} = \left[\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\right]^{\otimes 2} = \left[\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix}\right]^{\otimes 2} = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

All states have equal amplitude of $\frac{1}{2}$. The oracle is diagonal matrix where the entry that corresponds to the marked item $f(2)$ has a negative phase.

$$U_f = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Apply oracle U_f on $|\psi_1\rangle$:

$$|\psi_2\rangle = U_f|\psi_1\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$

Now apply the inversion about the mean

$$D = H^{\otimes n} U_{f_0} H^{\otimes n} = \begin{bmatrix} \frac{2}{N} - 1 & \frac{2}{N} & \frac{2}{N} & \frac{2}{N} \\ \frac{2}{N} & \frac{2}{N} - 1 & \frac{2}{N} & \frac{2}{N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{2}{N} & \frac{2}{N} & \frac{2}{N} & \frac{2}{N} - 1 \end{bmatrix}$$

$$D_{i,j} = \begin{cases} \frac{2}{N} - \frac{2}{4} = \frac{1}{2}, & \text{if } i \neq j \\ \frac{2}{N} - 1 = \frac{2}{4} - 1 = -\frac{1}{2}, & \text{if } i = j \end{cases} = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$

$$|\psi_3\rangle = D|\psi_2\rangle = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

After applying the oracle and inversion about the mean, the amplitude of the marked element 2 is increased from $\frac{1}{2}$ to 1 and the amplitude of other elements 0, 1, and 3 is decreased from $\frac{1}{2}$ to 0. If we apply measurement on the $|\psi_3\rangle$, then after the measurement the marked element of 2 has a probability of 1.

3.5 Geometric Representation

We can prove Grover's algorithm's correctness in terms of geometric representation such that the arbitrary state $|\psi\rangle$ rotates towards the target state $|\omega\rangle$ after performing oracle and

diffusion operations. Grover iteration can be represented as a rotation in a two-dimensional space spanned between vector $|\alpha\rangle$ for non-solution states and vector $|\omega\rangle$ for solution states to the search problem. This two-dimensional space is included in a many dimensional spaces. The $|\alpha\rangle$ is perpendicular to $|\omega\rangle$.

$$|\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_{x \neq \omega} |x\rangle$$

$$|\omega\rangle = \frac{1}{\sqrt{M}} \sum_x |x\rangle$$

where N is the number of elements in the search space, and M is the number of solutions, with $1 \leq M \leq N$. Suppose an arbitrary state $|\psi\rangle$ is a superposition state that resides in the plane spanned by $|\alpha\rangle$ and $|\omega\rangle$.

$$\begin{aligned} |\psi\rangle &= \sqrt{\frac{N-M}{N}} |\alpha\rangle + \sqrt{\frac{M}{N}} |\omega\rangle = \sqrt{\frac{N-M}{N}} \left(\frac{1}{\sqrt{N-M}} \sum_{x \neq \omega} |x\rangle \right) + \sqrt{\frac{M}{N}} \left(\frac{1}{\sqrt{M}} \sum_x |x\rangle \right) \\ &= \frac{1}{\sqrt{N}} \sum_x |x\rangle \end{aligned}$$

$|\psi\rangle$ is the sum of states from 0 to $N - 1$. Let's describe the Grover operation as a geometric operation. Step 1: Prepare $|\psi\rangle$ an arbitrary state. Let assume $\frac{\theta}{2}$ is the angle between $|\psi\rangle$ and $|\alpha\rangle$ as can be seen in Figure 3.6.

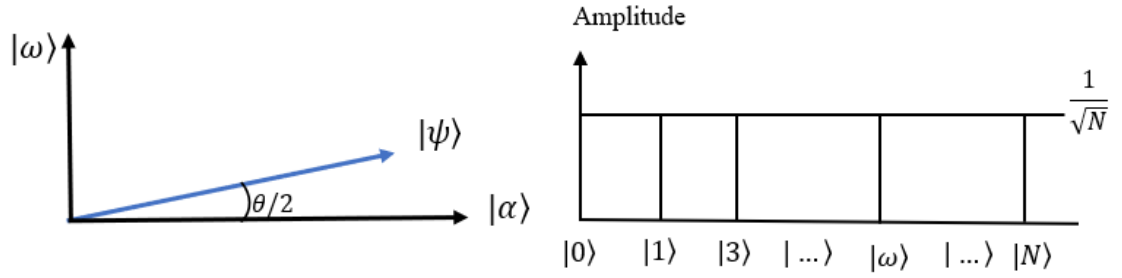


Figure 3.6 Arbitrary state $|\psi\rangle$ is in between $|\omega\rangle$ and $|\alpha\rangle$, where the amplitude for all $|N\rangle$ are equal.

From Figure 3.6, we can derive the value of θ as follows:

$$|\psi\rangle = \sqrt{\frac{N-M}{N}}|\alpha\rangle + \sqrt{\frac{M}{N}}|\omega\rangle = \cos\frac{\theta}{2}|\alpha\rangle + \sin\frac{\theta}{2}|\omega\rangle$$

$$\sin\frac{\theta}{2} = \sqrt{\frac{M}{N}}$$

$$\theta = 2\arcsin\sqrt{\frac{M}{N}}$$

Step 2: Apply the oracle reflection U_f to $|\psi\rangle$. The oracle operation U_f is a reflection of the state $|\alpha\rangle$, which is the vector (line) perpendicular to $|\omega\rangle$.

$$U_f|\psi\rangle = U_f(\cos\frac{\theta}{2}|\alpha\rangle + \sin\frac{\theta}{2}|\omega\rangle) = \cos\frac{\theta}{2}|\alpha\rangle - \sin\frac{\theta}{2}|\omega\rangle$$

The amplitude of the $|\omega\rangle$ state becomes a negative as can be seen in Figure 3.7. Thus, the average overall amplitude is decreased, which is less than $\frac{1}{\sqrt{N}}$. The reflection angle is

$\frac{\theta}{2}$ from $|\alpha\rangle$.

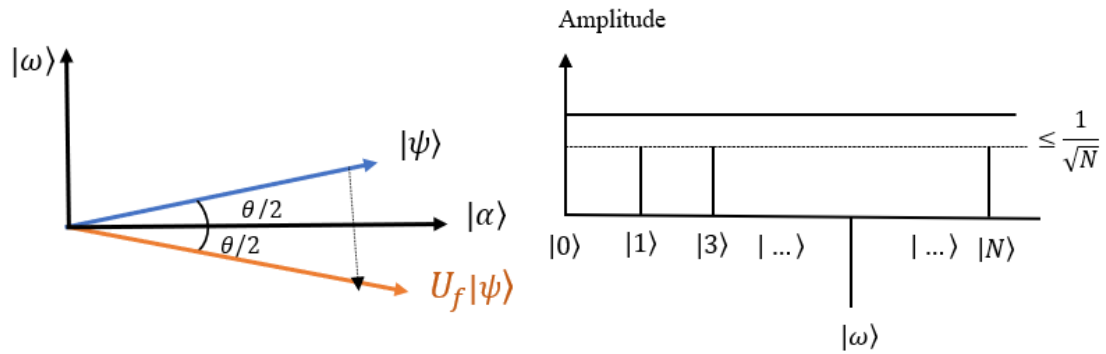


Figure 3.7 Oracle reflection U_f is applied on $|\psi\rangle$ where the amplitude of the $|\omega\rangle$ becomes negative and the average overall amplitude is decreased.

Step 3: Apply diffusion reflection D on $U_f|\psi\rangle$, which reflects about $|\psi\rangle$ as can be seen in Figure 3.8. This reflection on $U_f|\psi\rangle$ maps to $DU_f|\psi\rangle$ state. The reflection angle is θ from $|\psi\rangle$. This reflection amplifies the amplitude of $|\omega\rangle$ and shrinks the non-solution states.

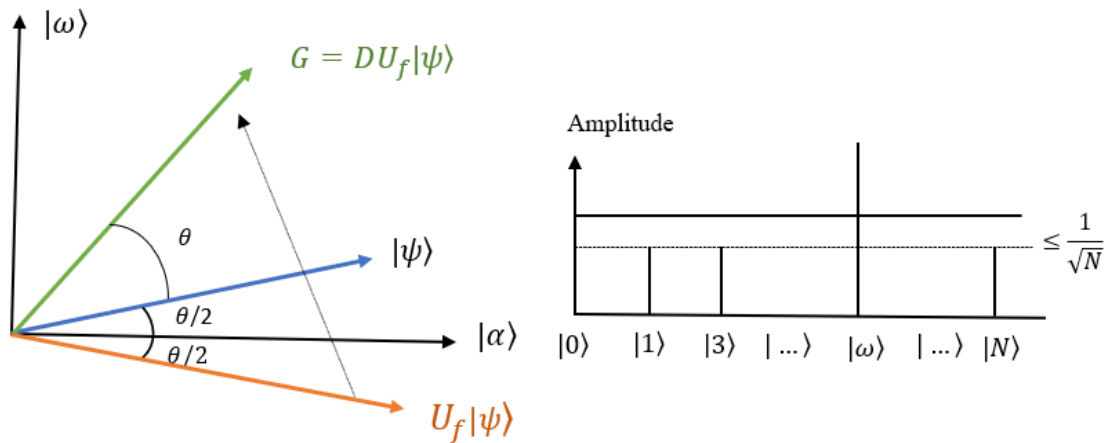


Figure 3.8 Diffusion reflection D is applied on $U_f|\psi\rangle$ where the $|\omega\rangle$ amplitude becomes positive and higher than the rest of the other amplitude.

As can be seen after applying two reflections DU_f , the $|\psi\rangle$ state rotates closer to solution state $|\omega\rangle$. If we apply these reflections $O(\sqrt{N})$ times, then $|\psi\rangle$ would be closer to

the solution, and when we measure the amplitude of the $|\omega\rangle$ than the solution state would have much higher probability than for the non-solution states. Let us write the Grover operator G as a rotation of angle:

$$G = DU_f|\psi\rangle = \sin\frac{3\theta}{2}|\omega\rangle + \cos\frac{3\theta}{2}|\alpha\rangle$$

in general, k rotations(iterations)

$$G^k|\psi\rangle = \sin\left(\frac{2k+1}{2}\theta\right)|\omega\rangle + \cos\left(\frac{2k+1}{2}\theta\right)|\alpha\rangle$$

$$\theta_k = \frac{2k+1}{2}\theta$$

$$G^k|\psi\rangle = \sin\theta_k|\omega\rangle + \cos\theta_k|\alpha\rangle$$

In order to have a high probability of obtaining the solution $|\omega\rangle$, choose k such that:

$\sin\left(\frac{2k+1}{2}\theta\right) \approx 1$, which means that:

$$k\theta + \frac{\theta}{2} \approx \frac{\pi}{2} \Rightarrow k \approx \frac{\pi}{2\theta} - \frac{1}{2} \approx \frac{\pi}{4\arcsin\sqrt{\frac{M}{N}}} - \frac{1}{2} \approx \frac{\pi}{4}\sqrt{\frac{N}{M}} = O\left(\sqrt{\frac{N}{M}}\right)$$

In general, it is required to have the optimal number of iterations k such that:

$$k \leq \left\lceil \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rceil$$

after k iterations, the final measurement will result in state $|\omega\rangle$ with the maximum probability. This probability satisfies the formula below:

$$p(\omega) \geq 1 - \sin^2 \frac{\theta}{2} = 1 - \frac{M}{N}$$

As we can see, the Grover iteration is a product of two reflections DU_f , which is rotation through twice the θ angle per reflection of G . Also, we can see that the initial state $|\psi\rangle$ is closer to the solution $|\omega\rangle$. How many iterations are required? This will keep rotating $O(\sqrt{N})$ to find a solution. For multiple solutions M , $O(\sqrt{N/M})$ rotations will be needed to find a single of these solutions.

3.6 Algebraic Representation

Grover iteration can be represented as a rotation matrix. As can be seen in Figure 3.8, the Grover operation DU_f is a rotation by an angle θ in the direction from $|\psi\rangle$ to $|\omega\rangle$. The Grover rotation can be represented as a rotation matrix:

$$G = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Grover rotation matrix can be proved from Euler's formula. In Figure 3.9, let's convert a complex number to a matrix representation using Euler's formula.

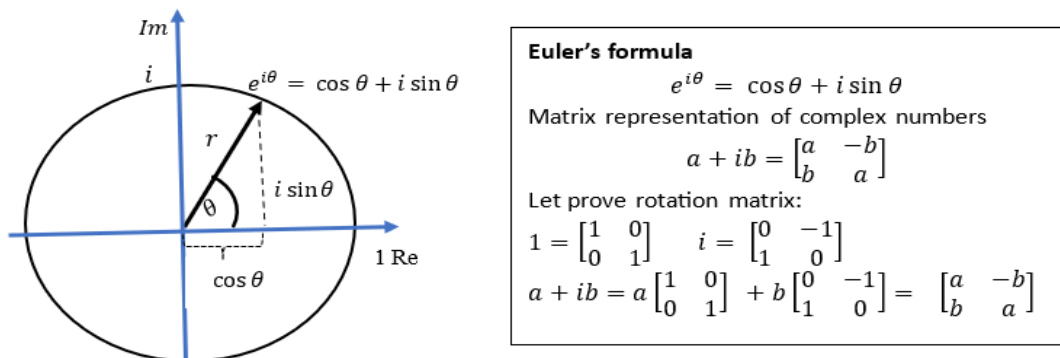


Figure 3.9 Euler's Formula

Proof using the rotation matrix:

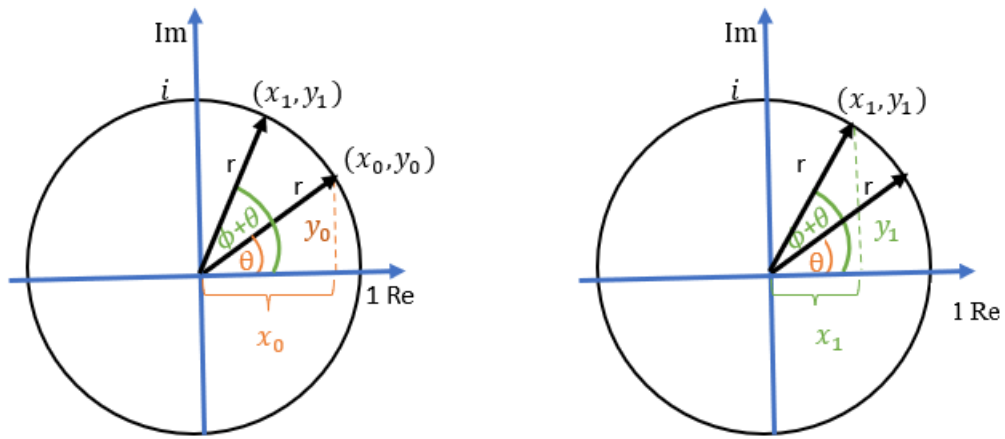


Figure 3.10 Rotation vector r from point (x_0, y_0) for angle θ to (x_1, y_1) for angle $\theta + \phi$

From Figure 3.10, we are rotating vector r from point (x_0, y_0) to (x_1, y_1) while keeping the length of r the same for both points but changing the angle. Angle ϕ is called the rotation angle. Let us derive the rotation matrix by using the trigonometry sum formulas:

$$\cos \theta = \frac{x_0}{r} \Rightarrow x_0 = r \cos \theta ; \quad \sin \theta = \frac{y_0}{r} \Rightarrow y_0 = r \sin \theta$$

$$\begin{aligned} x_1 &= r \cos(\theta + \phi) = r(\cos \theta \cos \phi - \sin \theta \sin \phi) = r \cos \theta \cos \phi - r \sin \theta \sin \phi \\ &= x_0 \cos \phi - y_0 \sin \phi \end{aligned}$$

$$\begin{aligned} y_1 &= r \sin(\theta + \phi) = r(\sin \theta \cos \phi + \cos \theta \sin \phi) = r \sin \theta \cos \phi + r \cos \theta \sin \phi \\ &= y_0 \cos \phi + x_0 \sin \phi \end{aligned}$$

$$x_1 = x_0 \cos \phi - y_0 \sin \phi$$

$$y_1 = y_0 \cos \phi + x_0 \sin \phi$$

These two equations can be written in a matrix form:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

The rotation matrix is:

$$R_\phi = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}$$

Thus, the Grover rotation matrix is the same as the rotation matrix. Let drive the eigenvalues and eigenvectors of the rotation matrix. Consider the eigenvalue problem:

$$R_\phi \vec{v} = \lambda \vec{v} = (R_\phi - \lambda I) \vec{v} = 0$$

Where λ is called an eigenvalue of R_ϕ and \vec{v} is called an eigenvector of R_ϕ . To find the eigenvalue, we calculate the determinant as follows:

$$\det(R_\phi - \lambda I) = 0 \implies \det \begin{bmatrix} \cos \phi - \lambda & -\sin \phi \\ \sin \phi & \cos \phi - \lambda \end{bmatrix} = 0$$

Which yields:

$$(\cos \phi - \lambda)^2 + \sin^2 \phi = 0$$

$$\lambda^2 - 2\lambda \cos \phi + \cos^2 \phi + \sin^2 \phi = \lambda^2 - 2\lambda \cos \phi + 1 = 0$$

$$\lambda = \cos \phi \pm \sqrt{\cos^2 \phi - 1} = \cos \phi \pm i \sin \phi = e^{\pm i\phi}$$

So, from the Grover rotation matrix, we proved that the Grover operator has a corresponding eigenvalue $e^{\pm i\phi}$. The eigenvalue $e^{\pm i\phi}$ is used to derive the quantum

counting formulas and quantum walk formulas. A simple calculation shows that two independent eigenvectors for this matrix are:

$$\begin{bmatrix} i \\ 1 \end{bmatrix}, \begin{bmatrix} -i \\ 1 \end{bmatrix}$$

Proof

$$(R_\phi - \lambda I)\vec{v} = 0 \rightarrow \begin{bmatrix} \cos \phi - \lambda & -\sin \phi \\ \sin \phi & \cos \phi - \lambda \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

Let's first find the eigenvectors corresponding to the eigenvalue $\lambda = \cos \phi + i \sin \phi$

$$\begin{bmatrix} -i \sin \phi & -\sin \phi \\ \sin \phi & -i \sin \phi \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0 \rightarrow \begin{cases} -iv_1 \sin \phi - v_2 \sin \phi = 0 \\ v_1 \sin \phi - iv_2 \sin \phi = 0 \end{cases} \rightarrow \begin{cases} -iv_1 - v_2 = 0 \\ v_1 - iv_2 = 0 \end{cases}$$

If $v_2 = 1$ then $v_1 = i$

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} i \\ 1 \end{bmatrix}$$

Also, if $\lambda = \cos \phi - i \sin \phi$ then

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} -i \\ 1 \end{bmatrix}$$

The Grover operator has a corresponding eigenvalue $e^{\pm i\phi}$ with two independent eigenvectors $\begin{bmatrix} i \\ 1 \end{bmatrix}, \begin{bmatrix} -i \\ 1 \end{bmatrix}$. The eigenvectors are expressed in the $|\alpha\rangle$ and $|\omega\rangle$ vectors basis:

$$|\psi_+\rangle = \frac{1}{\sqrt{2}}|\alpha\rangle + \frac{i}{\sqrt{2}}|\omega\rangle$$

$$|\psi_-\rangle = \frac{1}{\sqrt{2}}|\alpha\rangle - \frac{i}{\sqrt{2}}|\omega\rangle$$

Using both equations for the eigenvectors:

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

$$|\psi\rangle = e^{i\phi} \frac{1}{\sqrt{2}} |\psi_+\rangle + e^{-i\phi} \frac{1}{\sqrt{2}} |\psi_-\rangle$$

$|\psi\rangle$ is an equally weighted superposition of eigenvectors for G having eigenvalues $e^{i\phi}$ and $e^{-i\phi}$.

Grover's search technique can be represented in different ways, such as a matrix, a geometric angle rotation, and an algebraic formulation with eigenvectors and eigenvalues. However, in all variations of representations, the Grover's algorithm consists of the following steps:

- Initialize a uniform superposition by applying the Hadamard gate.
- Apply an oracle to recognize the solution. This step is called phase inversion.
- Apply diffusion operation to amplify the amplitude of solution states. The oracle and diffusion operations are repeated $O(\sqrt{N})$ times.
- Apply measurement operators.

3.7 Type of oracles and their implementation

The oracle design is the main part of Grover's search algorithm, such that any real problem implementation needs a unique quantum oracle design. There are two types of oracles: Phase oracle and Boolean oracle [51].

Phase oracle: Recall the goal of the oracle, if the oracle recognizes an element as the solution, then the phase of the desired state is inverted. A phase oracle can be constructed by Z gate or a controlled- Z gate. If the Z gate is applied to the $|0\rangle$ state, then there is no change in the phase of the $|0\rangle$ state, but if the Z gate is applied to the $|1\rangle$ state, then the phase of the $|1\rangle$ state is inverted. The phase oracle applies a phase to mark the solution states. There is no need for ancilla qubit to store the result of oracle. Also, there is no need to mirror the phase oracle since it's using only the Z gate or a controlled- Z gate.

$$Z|0\rangle = |0\rangle$$

$$Z|1\rangle = -|1\rangle$$

Boolean oracle: the phase oracle is difficult to construct for the practical applications of Grover's search algorithm. In the Boolean oracle, if the function of the oracle is not reversible, we use ancilla qubits to make the function reversible. Also, the result yes/no of the Boolean oracle is stored in one ancilla qubit. The Boolean oracle can be constructed by NOT, CNOT, SWAP, and multi-Toffoli gates. After the final result is completed, the Boolean oracle needs to mirror for each computation to get the original input because of the reversibility principle.

In the literature, there are many different diffusor operator designs [32, 259-266] that are used for Boolean oracle. The oracle output is used as a control for the diffusor operator in [266] and similar to Grover's diffusor used in Quantum Walk algorithm in [32], as can be seen in Figure 3.11. The diffusor operator proposed in [261] is used with phase kickback so that the output of the oracle is applied as a control to an ancilla qubit

and then applied to the Z gate operation, as can be seen in Figure 3.12. We have verified the diffuser operator in Figure 3.11 by manual and QISKIT simulation. Also, we verified the diffuser operator proposed in [261] after we changed the multi-controlled Toffoli gate to a multi-controlled Z gate (Z gate is the native gate used for the diffuser of the phase oracle) in Figure 3.12.

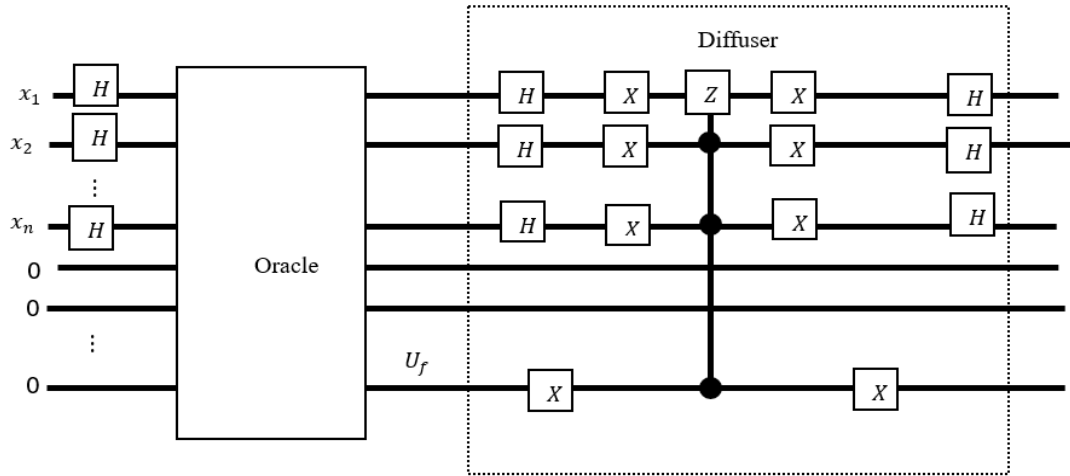


Figure 3.11 Grover schematic for Boolean oracle with diffuser operator from [266]

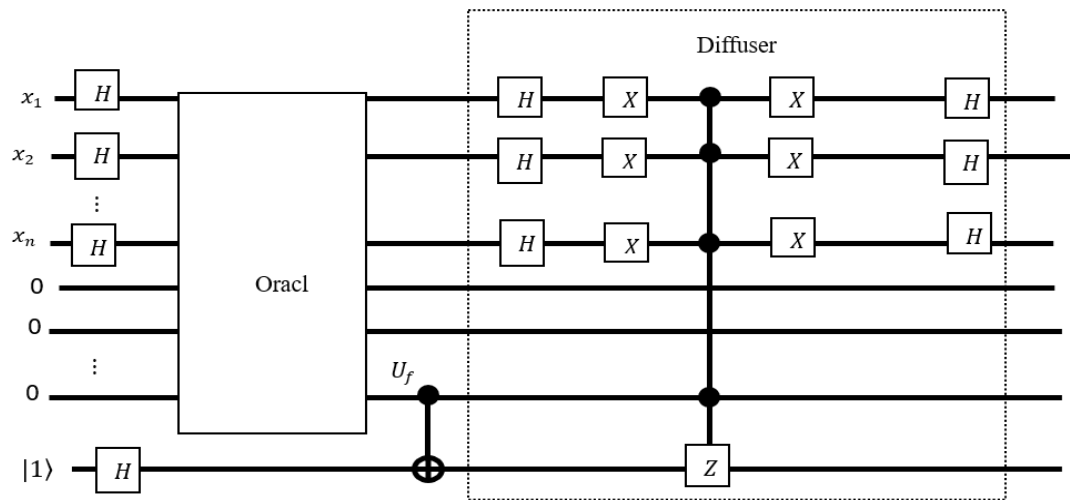


Figure 3.12 Grover schematic for Boolean oracle with modified diffuser operator from [261]

$|1\rangle$ state in Figure 3.12 is a base state that is not equal to 1. In the case of simulation or manual verification, it should initialize as a base state, not as zero applied with NOT gate.

Example of phase oracle: suppose we are given 3 qubits, and we need to recognize 010 and 110 as marked solutions from 000, 001, ..., 111.

The oracle in Figure 3.13 is a phase oracle to recognize the 010 and 110 with standard diffusion.

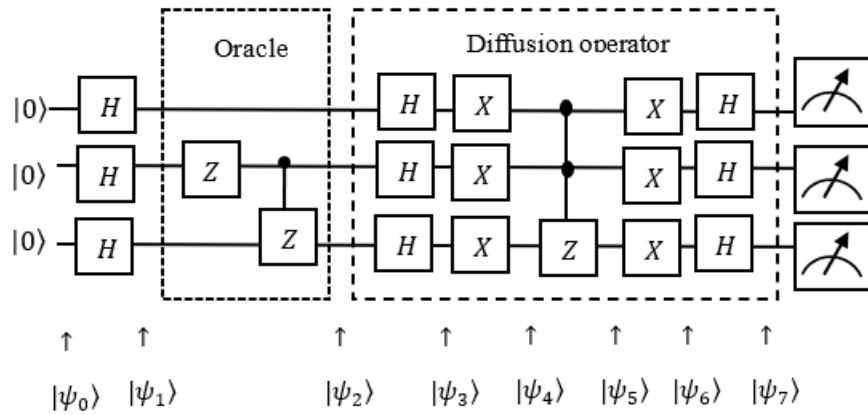


Figure 3.13 Phase oracle in Grover algorithm.

Let start to analyze step by step from ψ_0 to ψ_7 in Figure 3.13:

The initial state is $|\psi_0\rangle = |000\rangle$

Apply the Hadamard gate to create the superposition states.

$$\begin{aligned}
 |\psi_1\rangle &= H|0\rangle H|0\rangle H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\
 &= \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)
 \end{aligned}$$

Apply oracle operation such that Z and controlled- Z gates are applied 1 and 2 qubits on $|\psi_1\rangle$. This means flipping the sign for only $|010\rangle$ and $|110\rangle$ states.

$$|\psi_2\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle - |010\rangle + |011\rangle + |100\rangle + |101\rangle - |110\rangle + |111\rangle)$$

Apply the Hadamard gates to each state in $|\psi_2\rangle$

$$H^{\otimes 3}|000\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)$$

$$H^{\otimes 3}|001\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle)$$

$$\begin{aligned} -H^{\otimes 3}|010\rangle &= -\frac{1}{\sqrt{8}}(|000\rangle + |001\rangle - |010\rangle - |011\rangle + |100\rangle + |101\rangle - |110\rangle \\ &\quad - |111\rangle) \end{aligned}$$

$$H^{\otimes 3}|011\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle)$$

$$H^{\otimes 3}|100\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle - |100\rangle - |101\rangle - |110\rangle - |111\rangle)$$

$$H^{\otimes 3}|101\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle + |010\rangle - |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle)$$

$$\begin{aligned} -H^{\otimes 3}|110\rangle &= -\frac{1}{\sqrt{8}}(|000\rangle + |001\rangle - |010\rangle - |011\rangle - |100\rangle - |101\rangle + |110\rangle \\ &\quad + |111\rangle) \end{aligned}$$

$$H^{\otimes 3}|111\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle + |110\rangle - |111\rangle)$$

Adding all the states we get:

$$|\psi_3\rangle = \frac{1}{2}(|000\rangle - |001\rangle + |010\rangle + |011\rangle)$$

Applying the X gates to $|\psi_3\rangle$

$$|\psi_4\rangle = \frac{1}{2}(|111\rangle - |110\rangle + |101\rangle + |100\rangle)$$

Apply a 2-qubit controlled- Z gate between the 1, 2 (controls) qubits and qubit 3 (target).

This operation flips the sign of $|111\rangle$ state:

$$|\psi_5\rangle = \frac{1}{2}(-|111\rangle - |110\rangle + |101\rangle + |100\rangle)$$

Apply the X gates to $|\psi_5\rangle$

$$|\psi_6\rangle = \frac{1}{2}(-|000\rangle - |001\rangle + |010\rangle + |011\rangle)$$

Apply the Hadamard gates to $|\psi_6\rangle$

$$\begin{aligned} -H^{\otimes 3}|000\rangle &= -\frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle \\ &\quad + |111\rangle) \end{aligned}$$

$$\begin{aligned} -H^{\otimes 3}|001\rangle &= -\frac{1}{\sqrt{8}}(|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle \\ &\quad - |111\rangle) \end{aligned}$$

$$H^{\otimes 3}|010\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle - |010\rangle - |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle)$$

$$H^{\otimes 3}|011\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle)$$

Adding all the states we get:

$$|\psi_7\rangle = \frac{1}{\sqrt{2}}(-|010\rangle - |110\rangle)$$

Finally, the three qubits are measured to retrieve states $|010\rangle$ and $|110\rangle$. This example represents the case of two solutions. It is needed only to run one iteration for the Grover operator since there are two solution states out of eight possible states.

Example of a Boolean oracle: Suppose we are given a SAT Boolean function $f = x_1 + x_2x_3$. Let's design a traditional oracle circuit as can be seen in Figure 3.14 and apply it to Grover's search algorithm.

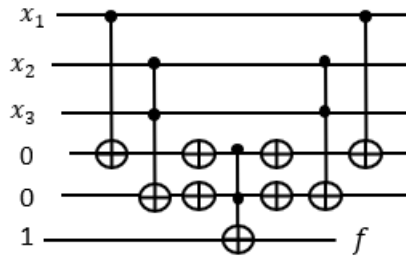


Figure 3.14 Oracle design of $f = x_1 + x_2x_3$

Let's solve without relying on the simulation that we start from φ_1 to φ_{10} step by step as it shows in Figure 3.15. In this example, we use the diffuser operator in Figure 3.11. Also, it can be used with the other diffuser in Figure 3.12. The initial value of $x_1x_2x_3 = 000$.

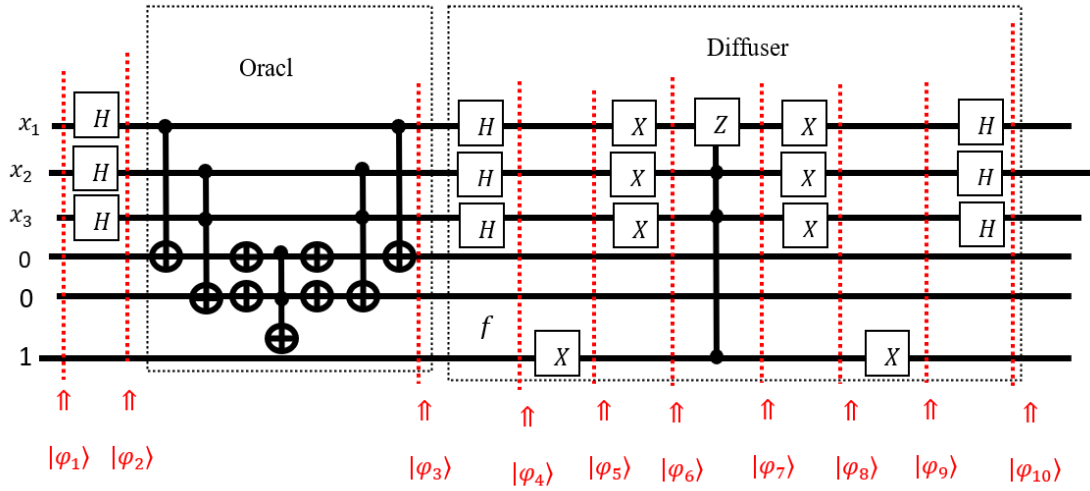


Figure 3.15 Oracle design $f = x_1 + x_2x_3$ with diffusor operator from [266]

$$|\varphi_1\rangle = |000001\rangle$$

$$|\varphi_2\rangle = H|0\rangle H|0\rangle H|0\rangle |001\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) |001\rangle$$

$$|\varphi_2\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) |001\rangle$$

$$|\varphi_3\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle) |000\rangle + \frac{1}{\sqrt{8}}(|011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) |001\rangle$$

The f value is 0 or 1 depending on the oracle result in $|\varphi_3\rangle$. This can be verified manually.

$$f = 0 \Rightarrow |000\rangle, |001\rangle, |010\rangle$$

$$|\varphi_3\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle) |000\rangle$$

$$f = 1 \Rightarrow |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle$$

$$|\varphi_3\rangle = \frac{1}{\sqrt{8}}(|011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)|001\rangle$$

Apply the Hadamard gate operation to each state.

$$H^{\otimes 3}|000\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)$$

$$H^{\otimes 3}|001\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle)$$

$$H^{\otimes 3}|010\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle - |010\rangle - |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle)$$

$$H^{\otimes 3}|011\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle)$$

$$H^{\otimes 3}|100\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle - |100\rangle - |101\rangle - |110\rangle - |111\rangle)$$

$$H^{\otimes 3}|101\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle + |010\rangle - |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle)$$

$$H^{\otimes 3}|110\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle - |010\rangle - |011\rangle - |100\rangle - |101\rangle + |110\rangle + |111\rangle)$$

$$H^{\otimes 3}|111\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle + |110\rangle - |111\rangle)$$

$f = 0$:

$|\varphi_4\rangle$ apply three Hadamard operation.

$$\begin{aligned}
|\varphi_4\rangle &= \frac{1}{\sqrt{8}}(H^{\otimes 3}|000\rangle + H^{\otimes 3}|001\rangle + H^{\otimes 3}|010\rangle)|000\rangle = \\
&\frac{1}{8}\{|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle \\
&\quad + |000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle \\
&\quad + |000\rangle + |001\rangle - |010\rangle - |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle\} |000\rangle \\
&= \frac{1}{8}\{3|000\rangle + |001\rangle + |010\rangle - |011\rangle + 3|100\rangle + |101\rangle + |110\rangle - |111\rangle\} |000\rangle
\end{aligned}$$

Apply the X gate to f the last qubit.

$$|\varphi_5\rangle = \frac{1}{8}\{3|000\rangle + |001\rangle + |010\rangle - |011\rangle + 3|100\rangle + |101\rangle + |110\rangle - |111\rangle\} |001\rangle$$

Apply three X gate operation:

$$|\varphi_6\rangle = \frac{1}{8}\{3|111\rangle + |110\rangle + |101\rangle - |100\rangle + 3|011\rangle + |010\rangle + |001\rangle - |000\rangle\} |001\rangle$$

Apply the Z gate operation that flips the sign of $|111\rangle$ state.

$$\begin{aligned}
|\varphi_7\rangle &= \frac{1}{8}\{-3|111\rangle + |110\rangle + |101\rangle - |100\rangle + 3|011\rangle + |010\rangle + |001\rangle \\
&\quad - |000\rangle\} |001\rangle
\end{aligned}$$

Apply three X gate operation:

$$\begin{aligned}
|\varphi_8\rangle &= \frac{1}{8}\{-3|000\rangle + |001\rangle + |010\rangle - |011\rangle + 3|100\rangle + |101\rangle + |110\rangle \\
&\quad - |111\rangle\} |001\rangle
\end{aligned}$$

Apply X gate on f the last qubit.

$$|\varphi_9\rangle = \frac{1}{8}\{-3|000\rangle + |001\rangle + |010\rangle - |011\rangle + 3|100\rangle + |101\rangle + |110\rangle - |111\rangle\}|000\rangle$$

Apply Hadamard gate operation.

$$|\varphi_9\rangle = \frac{1}{8}\{-3H^{\otimes 3}|000\rangle + H^{\otimes 3}|001\rangle + H^{\otimes 3}|010\rangle - H^{\otimes 3}|011\rangle + H^{\otimes 3}3|100\rangle + H^{\otimes 3}|101\rangle + H^{\otimes 3}|110\rangle - H^{\otimes 3}|111\rangle\}|000\rangle$$

$$|\varphi_{10}\rangle = \frac{1}{8\sqrt{8}}\{-3|000\rangle - 3|001\rangle - 3|010\rangle - 3|011\rangle - 3|100\rangle - 3|101\rangle - 3|110\rangle - 3|111\rangle$$

$$+|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle$$

$$+|000\rangle + |001\rangle - |010\rangle - |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle$$

$$-|000\rangle + |001\rangle + |010\rangle - |011\rangle - |100\rangle + |101\rangle + |110\rangle - |111\rangle$$

$$+3|000\rangle + 3|001\rangle + 3|010\rangle + 3|011\rangle - 3|100\rangle - 3|101\rangle - 3|110\rangle - 3|111\rangle$$

$$+|000\rangle - |001\rangle + |010\rangle - |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle$$

$$+|000\rangle + |001\rangle - |010\rangle - |011\rangle - |100\rangle - |101\rangle + |110\rangle + |111\rangle$$

$$-|000\rangle + |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle\}|000\rangle$$

$$|\varphi_{10}\rangle = \frac{1}{8\sqrt{8}}\{2|000\rangle + 2|001\rangle + 2|010\rangle - 6|011\rangle - 6|100\rangle - 6|101\rangle - 6|110\rangle -$$

$$6|111\rangle\}|000\rangle$$

Let's look the second case when $f = 1$ in $|\varphi_3\rangle$

$|\varphi_4\rangle$ apply Hadamard operation.

$$\begin{aligned}
 |\varphi_4\rangle &= \frac{1}{\sqrt{8}}(H^{\otimes 3}|011\rangle + H^{\otimes 3}|100\rangle + H^{\otimes 3}|101\rangle + H^{\otimes 3}|110\rangle + H^{\otimes 3}|111\rangle)|001\rangle \\
 &= \frac{1}{8}\{|000\rangle - |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle \\
 &\quad + |000\rangle + |001\rangle + |010\rangle + |011\rangle - |100\rangle - |101\rangle - |110\rangle - |111\rangle \\
 &\quad + |000\rangle - |001\rangle + |010\rangle - |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle \\
 &\quad + |000\rangle + |001\rangle - |010\rangle - |011\rangle - |100\rangle - |101\rangle + |110\rangle + |111\rangle \\
 &\quad + |000\rangle - |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle + |110\rangle - |111\rangle \\
 &\quad \}|001\rangle \\
 &= \frac{1}{8}\{6|000\rangle - |001\rangle - |010\rangle + |011\rangle - 3|100\rangle - |101\rangle - |110\rangle + |111\rangle\}|001\rangle
 \end{aligned}$$

Apply X gate in the last qubit for f

$$|\varphi_5\rangle = \frac{1}{8}\{6|000\rangle - |001\rangle - |010\rangle + |011\rangle - 3|100\rangle - |101\rangle - |110\rangle + |111\rangle\}|000\rangle$$

Apply three X gate operation:

$$|\varphi_6\rangle = \frac{1}{8}\{6|111\rangle - |110\rangle - |101\rangle + |100\rangle - 3|011\rangle - |010\rangle - |001\rangle + |000\rangle\}|000\rangle$$

Apply Z gate operation but the f value is 0 which controlled the Z gate, so no sign change for the $|111\rangle$

$$|\varphi_7\rangle = \frac{1}{8}\{6|111\rangle - |110\rangle - |101\rangle + |100\rangle - 3|011\rangle - |010\rangle - |001\rangle + |000\rangle\} |000\rangle$$

Apply three X gate operation:

$$|\varphi_8\rangle = \frac{1}{8}\{6|000\rangle - |001\rangle - |010\rangle + |011\rangle - 3|100\rangle - |101\rangle - |110\rangle + |111\rangle\} |000\rangle$$

Apply X gate on f the last qubit.

$$|\varphi_9\rangle = \frac{1}{8}\{6|000\rangle - |001\rangle - |010\rangle + |011\rangle - 3|100\rangle - |101\rangle - |110\rangle + |111\rangle\} |001\rangle$$

Apply the Hadamard gate operation.

$$\begin{aligned} |\varphi_{10}\rangle &= \frac{1}{8}\{H^{\otimes 3}6|000\rangle - H^{\otimes 3}|001\rangle - H^{\otimes 3}|010\rangle + H^{\otimes 3}|011\rangle - 3H^{\otimes 3}|100\rangle \\ &\quad - H^{\otimes 3}|101\rangle - H^{\otimes 3}|110\rangle + H^{\otimes 3}|111\rangle\} |001\rangle \\ &= \frac{1}{8\sqrt{8}}\{6|000\rangle + 6|001\rangle + 6|010\rangle + 6|011\rangle + 6|100\rangle + 6|101\rangle + 6|110\rangle + 6|111\rangle \\ &\quad - |000\rangle + |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle \\ &\quad - |000\rangle - |001\rangle + |010\rangle + |011\rangle - |100\rangle - |101\rangle + |110\rangle + |111\rangle \\ &\quad + |000\rangle - |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle \\ &\quad - 3|000\rangle - 3|001\rangle - 3|010\rangle - 3|011\rangle + 3|100\rangle + 3|101\rangle + 3|110\rangle + 3|111\rangle \\ &\quad - |000\rangle + |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle \\ &\quad - |000\rangle - |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle \\ &\quad + |000\rangle - |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle + |110\rangle - |111\rangle \} \end{aligned}$$

$\rangle|001\rangle$

$$|\varphi_{10}\rangle = \frac{1}{8\sqrt{8}}\{|000\rangle + |001\rangle + |010\rangle + 9|011\rangle + 9|100\rangle + 9|101\rangle + 9|110\rangle + 9|111\rangle\}|001\rangle$$

Finally, the measurement is applied to the input value for $x_1x_2x_3$ from both $|\varphi_{10}\rangle$ for the two cases $f = 0$ and $f = 1$

$$\frac{1}{8\sqrt{8}}\{2|000\rangle + 2|001\rangle + 2|010\rangle - 6|011\rangle - 6|100\rangle - 6|101\rangle - 6|110\rangle - 6|111\rangle\} +$$

$$\frac{1}{8\sqrt{8}}\{|000\rangle + |001\rangle + |010\rangle + 9|011\rangle + 9|100\rangle + 9|101\rangle + 9|110\rangle + 9|111\rangle\}$$

The easy way to understand the measurement is by adding the absolute value of the coefficient of each state and then apply squaring:

$$|000\rangle = \left(\left|\frac{2}{8\sqrt{8}}\right| + \left|\frac{1}{8\sqrt{8}}\right|\right)^2 = \frac{5}{512}$$

$$|001\rangle = \left(\left|\frac{2}{8\sqrt{8}}\right| + \left|\frac{1}{8\sqrt{8}}\right|\right)^2 = \frac{5}{512}$$

$$|010\rangle = \left(\left|\frac{2}{8\sqrt{8}}\right| + \left|\frac{1}{8\sqrt{8}}\right|\right)^2 = \frac{5}{512}$$

$$|011\rangle = \left(\left|-\frac{6}{8\sqrt{8}}\right| + \left|\frac{9}{8\sqrt{8}}\right|\right)^2 = \frac{117}{512}$$

$$|100\rangle = \left(\left|-\frac{6}{8\sqrt{8}}\right| + \left|\frac{9}{8\sqrt{8}}\right|\right)^2 = \frac{117}{512}$$

$$|101\rangle = \left(\left| -\frac{6}{8\sqrt{8}} \right| + \left| \frac{9}{8\sqrt{8}} \right| \right)^2 = \frac{117}{512}$$

$$|110\rangle = \left(\left| -\frac{6}{8\sqrt{8}} \right| + \left| \frac{9}{8\sqrt{8}} \right| \right)^2 = \frac{117}{512}$$

$$|111\rangle = \left(\left| -\frac{6}{8\sqrt{8}} \right| + \left| \frac{9}{8\sqrt{8}} \right| \right)^2 = \frac{117}{512}$$

As can be seen, these states $|011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle$ have a high probability, such that $f = 1$ from the oracle in $|\varphi_3\rangle$. Also, these state $|000\rangle, |001\rangle, |010\rangle$ have a low probability, such that $f = 0$ from the oracle in $|\varphi_3\rangle$. Thus, $|011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle$ states with the high probability are the solutions and can be verified by simple truth table in Table 3.1.

Table 3.1 Truth table $f = x_1 + x_2x_3$

| $x_1x_2x_3$ | $x_1 + x_2x_3$ |
|-------------|----------------|
| 000 | 0 |
| 001 | 0 |
| 010 | 0 |
| 011 | 1 |
| 100 | 1 |
| 101 | 1 |
| 110 | 1 |
| 111 | 1 |

Let solve $f = x_1 + x_2x_3$ with different diffusor as in Figure 3.16. The initial value of $x_1x_2x_3 = 000$

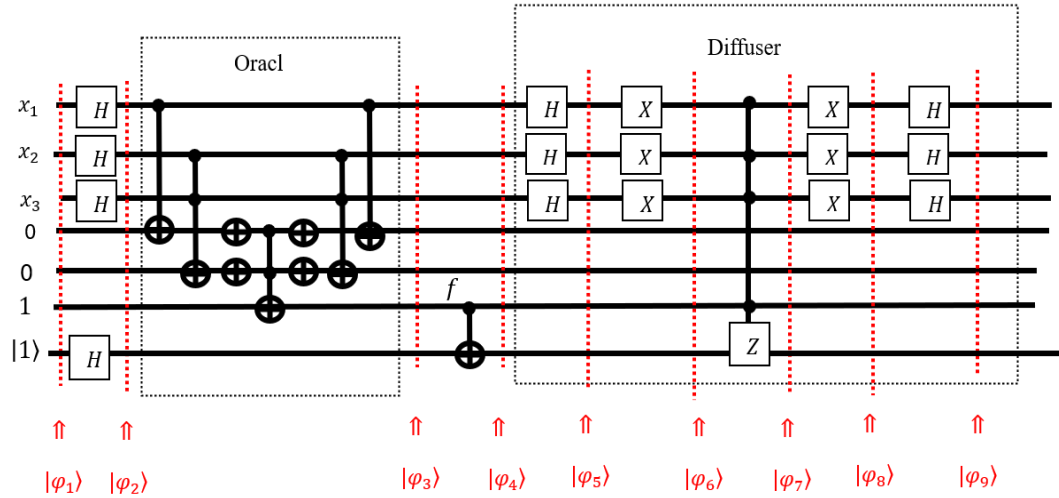


Figure 3.16 Oracle $f = x_1 + x_2x_3$ with modified diffusor operator from [261]

$$|\varphi_1\rangle = |000001\rangle$$

$|\varphi_2\rangle$ apply Hadamard gates.

$$|\varphi_2\rangle = H|0\rangle H|0\rangle H|0\rangle |001\rangle H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) |001\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$|\varphi_2\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) |001\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Apply the oracle which gives the following values.

$$|\varphi_3\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle) |000\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) + \frac{1}{\sqrt{8}}(|011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) |001\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

We get two values for f :

- $f = 0$ for these value $|000\rangle + |001\rangle + |010\rangle$
- $f = 1$ for these value $|011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle$

$|\varphi_4\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ is controlled by f value that $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ changes only when $f = 1$

$f = 0$

$$|\varphi_4\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle)|000\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$f = 1$

$$|\varphi_4\rangle = \frac{1}{\sqrt{8}}(|011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)|001\rangle \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)$$

Apply the Hadamard gate operation for each state.

$$H^{\otimes 3}|000\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)$$

$$H^{\otimes 3}|001\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle)$$

$$H^{\otimes 3}|010\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle - |010\rangle - |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle)$$

$$H^{\otimes 3}|011\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle)$$

$$H^{\otimes 3}|100\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle - |100\rangle - |101\rangle - |110\rangle - |111\rangle)$$

$$H^{\otimes 3}|101\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle + |010\rangle - |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle)$$

$$H^{\otimes 3}|110\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle - |010\rangle - |011\rangle - |100\rangle - |101\rangle + |110\rangle + |111\rangle)$$

$$H^{\otimes 3}|111\rangle = \frac{1}{\sqrt{8}}(|000\rangle - |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle + |110\rangle - |111\rangle)$$

$$f = 0$$

$$|\varphi_4\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle)|000\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$|\varphi_5\rangle$ apply three Hadamard gate operation.

$$\begin{aligned} |\varphi_5\rangle &= \frac{1}{8}\{(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) \\ &\quad + (|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle) \\ &\quad + (|000\rangle + |001\rangle - |010\rangle - |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle)\} \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\ |\varphi_5\rangle &= \frac{1}{8}\{3|000\rangle + |001\rangle + |010\rangle - |011\rangle + 3|100\rangle + |101\rangle + |110\rangle \\ &\quad - |111\rangle\} |000\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned}$$

$|\varphi_6\rangle$ apply three X gate operation.

$$\begin{aligned} |\varphi_6\rangle &= \frac{1}{8}\{3|111\rangle + |110\rangle + |101\rangle - |100\rangle + 3|011\rangle + |010\rangle + |001\rangle \\ &\quad - |000\rangle\} |000\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned}$$

$|\varphi_7\rangle$ is equal to $|\varphi_6\rangle$ because f is equal to 0; the least significant qubit in $|000\rangle$; therefore, Z gate is not performing any operation.

$$|\varphi_7\rangle = \frac{1}{8}\{3|111\rangle + |110\rangle + |101\rangle - |100\rangle + 3|011\rangle + |010\rangle + |001\rangle - |000\rangle\} |000\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$|\varphi_8\rangle$ apply three X gate operation.

$$|\varphi_8\rangle = \frac{1}{8}\{3|000\rangle + |001\rangle + |010\rangle - |011\rangle + 3|100\rangle + |101\rangle + |110\rangle - |111\rangle\} |000\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$|\varphi_9\rangle$ apply three Hadamard gate operation.

$$|\varphi_9\rangle = \frac{1}{8}\{3H^{\otimes 3}|000\rangle + H^{\otimes 3}|001\rangle + H^{\otimes 3}|010\rangle - H^{\otimes 3}|011\rangle + H^{\otimes 3}3|100\rangle + H^{\otimes 3}|101\rangle + H^{\otimes 3}|110\rangle - H^{\otimes 3}|111\rangle\} |000\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$|\varphi_9\rangle =$

$$\begin{aligned} &= \frac{1}{8\sqrt{8}}\{(3|000\rangle + 3|001\rangle + 3|010\rangle + 3|011\rangle + 3|100\rangle + 3|101\rangle + 3|110\rangle + 3|111\rangle) \\ &\quad + (|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle) \\ &\quad + (|000\rangle + |001\rangle - |010\rangle - |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle) \\ &\quad + (-|000\rangle + |001\rangle + |010\rangle - |011\rangle - |100\rangle + |101\rangle + |110\rangle - |111\rangle)\} \end{aligned}$$

$$\begin{aligned}
&+(3|000\rangle + 3|001\rangle + 3|010\rangle + 3|011\rangle - 3|100\rangle - 3|101\rangle - 3|110\rangle - 3|111\rangle) \\
&+(|000\rangle - |001\rangle + |010\rangle - |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle) \\
&+(|000\rangle + |001\rangle - |010\rangle - |011\rangle - |100\rangle - |101\rangle + |110\rangle + |111\rangle) \\
&+(-|000\rangle + |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle)\frac{1}{\sqrt{2}}(|0\rangle \\
&\quad - |1\rangle)
\end{aligned}$$

$$|\varphi_9\rangle = \frac{1}{8\sqrt{8}}\{8|000\rangle + 8|001\rangle + 8|010\rangle\}|000\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$|\varphi_9\rangle = \frac{1}{\sqrt{8}}\{|000\rangle + |001\rangle + |010\rangle\}|000\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$f = 1$

$$|\varphi_4\rangle = \frac{1}{\sqrt{8}}(|011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)|001\rangle\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)$$

$|\varphi_5\rangle$ apply three Hadamard operation.

$$\begin{aligned}
|\varphi_5\rangle &= \frac{1}{8}(|000\rangle - |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle) \\
&+(|000\rangle + |001\rangle + |010\rangle + |011\rangle - |100\rangle - |101\rangle - |110\rangle - |111\rangle) \\
&+(|000\rangle - |001\rangle + |010\rangle - |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle) \\
&+(|000\rangle + |001\rangle - |010\rangle - |011\rangle - |100\rangle - |101\rangle + |110\rangle + |111\rangle)
\end{aligned}$$

$$+(|000\rangle - |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle + |110\rangle - |111\rangle)|001\rangle \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)$$

$$|\varphi_5\rangle = \frac{1}{8}(5|000\rangle - |001\rangle - |010\rangle + |011\rangle - 3|100\rangle - |101\rangle - |110\rangle + |111\rangle)|001\rangle \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)$$

$|\varphi_6\rangle$ apply three X operation.

$$|\varphi_6\rangle = \frac{1}{8}(5|111\rangle - |110\rangle - |101\rangle + |100\rangle - 3|011\rangle - |010\rangle - |001\rangle + |000\rangle)|001\rangle \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)$$

In $|\varphi_7\rangle$, $\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)$ will apply Z gate operation only when the first three qubits are equal to $|111\rangle$ and always $f = 1$ in this case. The value of $\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)$ changes to $\frac{1}{\sqrt{2}}(-|1\rangle - |0\rangle)$ because Z gate only changes the phase (sign) of $|1\rangle$ state. Here the result will be:

$$|\varphi_7\rangle = \frac{1}{8}(5|111\rangle)|001\rangle \frac{1}{\sqrt{2}}(-|1\rangle - |0\rangle) + \frac{1}{8}(-|110\rangle - |101\rangle + |100\rangle - 3|011\rangle - |010\rangle - |001\rangle + |000\rangle)|001\rangle \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)$$

$|\varphi_8\rangle$ apply three X gate operation.

$$|\varphi_8\rangle = \frac{1}{8}(5|000\rangle)|001\rangle\frac{1}{\sqrt{2}}(-|1\rangle - |0\rangle) + \frac{1}{8}(-|001\rangle - |010\rangle + |011\rangle - 3|100\rangle \\ - |101\rangle - |110\rangle + |111\rangle)|001\rangle\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)$$

$|\varphi_9\rangle$ apply three Hadamard gate operation.

$$|\varphi_9\rangle = \left(\frac{5}{8\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle \right. \\ \left. + |111\rangle)|001\rangle\frac{1}{\sqrt{2}}(-|1\rangle - |0\rangle) \right) \\ + \\ \frac{1}{8\sqrt{8}}\{(-|000\rangle + |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle - |110\rangle + |111\rangle) \\ + (-|000\rangle - |001\rangle + |010\rangle + |011\rangle - |100\rangle - |101\rangle + |110\rangle + |111\rangle) \\ + (|000\rangle - |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle) \\ + (-3|000\rangle - 3|001\rangle - 3|010\rangle - 3|011\rangle + 3|100\rangle + 3|101\rangle + 3|110\rangle + 3|111\rangle) \\ + (-|000\rangle + |001\rangle - |010\rangle + |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle) \\ + (-|000\rangle - |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle) \\ + (|000\rangle - |001\rangle - |010\rangle + |011\rangle - |100\rangle + |101\rangle + |110\rangle - |111\rangle)\}|001\rangle\frac{1}{\sqrt{2}}(|1\rangle \\ - |0\rangle)$$

$$\begin{aligned}
|\varphi_9\rangle = & \left(\frac{5}{8\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle \right. \\
& \left. + |111\rangle) |001\rangle \frac{1}{\sqrt{2}}(-|1\rangle - |0\rangle) \right) \\
& + \left(\frac{1}{8\sqrt{8}}\{(-5|000\rangle - 5|001\rangle - 5|010\rangle + 3|011\rangle + 3|100\rangle + 3|101\rangle \right. \\
& \left. + 3|110\rangle + 3|111\rangle)\} |001\rangle \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) \right)
\end{aligned}$$

To simplify

$$\frac{1}{\sqrt{2}}(-|1\rangle - |0\rangle) = -\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) = -\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Thus, we need to apply phase kickback which flips the sign of the input value:

$$\begin{aligned}
|\varphi_9\rangle = & \left(\frac{5}{8\sqrt{8}}(-|000\rangle - |001\rangle - |010\rangle - |011\rangle - |100\rangle - |101\rangle - |110\rangle \right. \\
& \left. - |111\rangle) |001\rangle \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \right) \\
& + \left(\frac{1}{8\sqrt{8}}\{(5|000\rangle + 5|001\rangle + 5|010\rangle - 3|011\rangle - 3|100\rangle - 3|101\rangle \right. \\
& \left. - 3|110\rangle - 3|111\rangle)\} |001\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \right)
\end{aligned}$$

The final result is the combination of $f = 0$ and $f = 1$

$$f = 0 \Rightarrow$$

$$|\varphi_9\rangle = \frac{1}{\sqrt{8}}\{|000\rangle + |001\rangle + |010\rangle\} |000\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$f = 1 \Rightarrow$$

$$\begin{aligned} |\varphi_9\rangle = & \left(\frac{5}{8\sqrt{8}}(-|000\rangle - |001\rangle - |010\rangle - |011\rangle - |100\rangle - |101\rangle - |110\rangle \right. \\ & \left. - |111\rangle) |001\rangle \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \right) \\ & + \left(\frac{1}{8\sqrt{8}}\{(5|000\rangle + 5|001\rangle + 5|010\rangle - 3|011\rangle - 3|100\rangle - 3|101\rangle \right. \\ & \left. - 3|110\rangle - 3|111\rangle)\} |001\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \right) \end{aligned}$$

To find a coefficient for each state, because of phase kickback, we add coefficients and normalized for both $f = 0$ and $f = 1$ in $|\varphi_9\rangle$:

$$\begin{aligned} |000\rangle = |001\rangle = |010\rangle &= \frac{\frac{1}{\sqrt{8}} + \frac{5}{8\sqrt{8}} - \frac{5}{8\sqrt{8}}}{\sqrt{\left|\frac{1}{\sqrt{8}}\right|^2 + \left|\frac{5}{8\sqrt{8}}\right|^2 + \left|-\frac{5}{8\sqrt{8}}\right|^2}} = \frac{\frac{1}{\sqrt{8}}}{\sqrt{\frac{114}{512}}} = \frac{1}{\sqrt{8}} \times \sqrt{\frac{512}{114}} \\ &= 0.75 \end{aligned}$$

$$\begin{aligned}
|011\rangle = |100\rangle = |101\rangle = |110\rangle = |111\rangle &= \frac{-\frac{5}{8\sqrt{8}} - \frac{3}{8\sqrt{8}}}{\sqrt{\left|-\frac{5}{8\sqrt{8}}\right|^2 + \left|-\frac{3}{8\sqrt{8}}\right|^2}} = \frac{-\frac{1}{\sqrt{8}}}{\sqrt{\frac{34}{512}}} \\
&= -\frac{1}{\sqrt{8}} \times \sqrt{\frac{512}{34}} = -1.37
\end{aligned}$$

The final value will be:

$$\begin{aligned}
&0.75|000\rangle + 0.75|001\rangle + 0.75|010\rangle - 1.37|011\rangle - 1.37|100\rangle - 1.37|101\rangle \\
&\quad - 1.37|110\rangle - 1.37|111\rangle
\end{aligned}$$

The measurement of each state is square root of the corresponding coefficient for each state. Recall f has two values: $f = 0$ for non-solution states and $f = 1$ for solution states:

$$f = 0 \Rightarrow |000\rangle, |001\rangle, |010\rangle$$

$$f = 1 \Rightarrow |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle$$

As can be seen, the square root of the corresponding coefficient for each value of $f = 1$ is $|-1.37|^2$ while $f = 0$ is $|0.75|^2$. The solutions of the Boolean function $f = x_1 + x_2x_3$ are all values $|011\rangle, |100\rangle, |101\rangle, |110\rangle$, and $|111\rangle$ that have high probability. Also, these values $|000\rangle, |001\rangle$, and $|010\rangle$ are non-solutions with low probability.

3.8 Applications of Grover's Algorithm

The Grover's algorithm has many applications and is also used as a subroutine or a building block in other modern quantum algorithms. Database search is one of the most

significant applications in science and engineering and has many applications in real-world problems. In Grover's original paper, the algorithm was referred to as a database search algorithm; however, the fundamental concepts of Grover's algorithm are applicable in various fields such as cryptography, data mining, machine learning, and constraint satisfaction and optimization problems [55]. In theory, every discrete constraint satisfaction problem can be solved using Grover search algorithm. In addition, every discrete optimization problem can be reduced to several calls of Grover's algorithm.

3.8.1 Cryptography

Quantum computers can be used to break cryptographic schemes using Grover's algorithm. There are many different quantum oracle designs presented in [52, 53, 54, 56] for symmetric cryptography, that the block cipher can be implemented as a quantum oracle to recover the key from the given ciphertext. A block cipher is a method of encrypting plaintext (unencrypted data) with a secret key (also known as a cipher key) to produce ciphertext (encrypted data) using a cryptographic algorithm. Grover's algorithm can be used to identify the secret key by going through all possible binary keys, assuming that the block cipher was implemented on quantum computers. Grover's algorithm can be used to attack the Advanced Encryption Standard (AES) algorithm, which will accelerate the process of finding the encrypted key.

3.8.2 Quantum machine learning

Many machine algorithms classify data in high-dimensional vector spaces, which take polynomial time depending on the data size. Quantum machine learning algorithms [59,

60] can provide an exponential speed-up over classical algorithms, such that quantum machine learning can take logarithmic time to classify data in high-dimensional vector spaces. Many machine learning algorithms [58] are proposed to use Grover's algorithm as their subroutine.

3.8.3 K-medians

K-medians clustering based on Grover's algorithm was proposed [57]. The k-medians algorithm aims to partition a given dataset into k clusters by minimizing the distance between each data point belonging to a cluster and its closest cluster center of the actual point of the dataset. Grover's algorithm is used to locate the median in a cluster.

3.8.4 K-means

k-means clustering assigns a given dataset into k clusters, and each cluster has a virtual centroid center that corresponds to the average distance that belongs to each cluster. The goal is to minimize the average distance to the centroid of the cluster. The Grover's-based algorithm [58, 61] is used to determine the index of the cluster centroid and assign vectors to the closest centroids.

3.8.5 Reinforcement Learning

Reinforcement learning is a machine learning method in which an agent learns from a sequence of actions, it rewards desired behaviors, and punishes negative behaviors.

Grover's search algorithm for reinforcement learning [62, 63] is used to find sequences of actions that lead to high rewards by improving the decision rules. For each Grover iteration, the amplitude of the given state represents the reward value, such that the good actions are amplified.

There are other machine learning algorithms based on Grover's search algorithm, such as quantum support vector machines [64], quantum neural networks [65], and quantum K-Nearest-Neighbor [66], and there are still other potential machine learning algorithms.

3.8.6 Constraint satisfaction and optimization problems

Combinatorial optimization problems require exhaustive search, and many of them are NP-hard problems that can be improved at runtime by using Grover's search algorithm. The goal of constraint satisfaction is to find a set of solutions that satisfy a set of constraints, while optimization is to find the best solution from the possible solutions. There are many constraint satisfaction and optimization problems that can be solved using Grover's algorithm, such as graph coloring [67], maximum clique [68], maximum cut [69], and maximum independent set [70].

4 QUANTUM ALGORITHM FOR VARIANT SATISFIABILITY AND MAXIMUM SATISFIABILITY

Chapter 4

Quantum Algorithm for Variant Satisfiability and Maximum Satisfiability

Alasow, Abdirahman, and Marek Perkowski. "Quantum Algorithm for Maximum Satisfiability." In 2022 IEEE 52nd International Symposium on Multiple-Valued Logic (ISMVL), pp. 27-34. IEEE, 2022. <https://doi.org/10.1109/ISMVL52857.2022.00012>

Authors: Abdirahman Alasow, Marek Perkowski

Abdirahman Alasow:

Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Writing—original draft

Marek Perkowski:

Conceptualization, Methodology, Supervision, Visualization, Writing—review & editing.

Doi: 10.1109/ISMVL52857.2022.00012

Alasow, Abdirahman, Peter Jin, and Marek Perkowski. "Quantum Algorithm for Variant Maximum Satisfiability." *Entropy* 24, no. 11 (2022): 1615.

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

Authors: Abdirahman Alasow, Peter Jin, and Marek Perkowski

Author Contributions: Conceptualization, A.A. and P.J.; Data curation, A.A.; Formal analysis, A.A.; Investigation, A.A.; Methodology, A.A., P.J. and M.P.; Project administration, M.P.; Resources, A.A.; Software, A.A.; Supervision, M.P.; Validation, A.A. and M.P.; Visualization, A.A.; Writing—original draft, A.A. and P.J.; Writing—review & editing, A.A. and M.P. All authors have read and agreed to the published version of the manuscript.

Doi: 10.3390/e24111615

https://pdxscholar.library.pdx.edu/ece_fac/707/

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

4.1 Introduction

4.1.1 Satisfiability

The satisfiability (SAT) problem for a given Boolean function is the problem of determining if there exists a set of assignment values of input variables for the given Boolean function that evaluates this function to TRUE. Boolean or propositional-logic expressions are formed using operators AND, OR, EXOR, and NOT from input variables. Satisfiability expression (circuit) is often expressed as a product-of-sum (POS)

form. POS is a logical ANDs of OR terms, where each OR term is an inclusive sum of literals. For instance, the POS SAT function $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$ is satisfiable because when $c = 1$ and either a or b is equal to 1, then $f(a, b, c)$ evaluates to 1. Another example, $f(a, b) = (a + b)(\bar{a} + \bar{b})(\bar{a} + b)(a + \bar{b})$ is not satisfiable because no binary assignment of values for variables a and b , $f(a, b)$ would evaluate to 1.

Satisfiability problems have a wide range of applications, such as model checking in electronic design automation (EDA) [71], automatic test pattern generation (ATPG) [72], software and hardware verification [73], and circuit design [74]. Satisfiability problems also have many applications in Artificial Intelligence [75], robotics, and electronic design. Based on Cook's theorem [76], satisfiability is an NP-complete problem. Solving a satisfiability problem involving many variables and terms using traditional algorithms is computationally expensive.

4.1.2 Maximum Satisfiability

Maximum satisfiability (MAX-SAT) is an optimization version of the SAT problem. MAX-SAT finds the maximum number of constraints of a given Boolean function that are satisfied. Suppose a Boolean function in the POS form contains thousands of sum (OR) terms (also called clauses). The MAX-SAT problem is to examine the maximum number of terms that are satisfied. For example, $f(a, b, c, \dots, N) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c) \dots (\dots) = 1$. The function f is true for a binary assignment of values to variables a, b, c, \dots, N for which all terms are true. This is the SAT satisfiability. In

contrast, the goal of MAX-SAT is not only to find the decision satisfied/unsatisfied (yes/no) but also to provide the maximum number of terms (clauses) that are satisfied with the actual satisfying assignment values for the variables in case the formula is not SAT satisfiable. The MAX-SAT is considered to be an NP-hard problem [77].

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

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

There are three main variants of MAX-SATs [78,79]:

1. Weighted MAX-SAT: Each clause has an associated weight cost, and the objective is to maximize the sum of the weights of the satisfied clauses.
2. Partial MAX-SAT: Finds the assignment values for the variables that must be satisfied for all hard clauses and must be maximized on the soft clauses.
3. Weighted partial MAX-SAT is a combination of the partial and weighted MAX-SAT.

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

4.2 Related Work

4.2.1 Maximum Satisfiability Applications

There are many optimization problems and real-world applications that can be encoded to MAX-SAT. Some of the successful applications used for MAX-SAT are data analysis and machine learning, planning and scheduling, verification and security, bioinformatics, and combinatorial optimization [78]. We will briefly discuss some of these applications.

4.2.1.1 Data Analysis and Machine Learning

MAX-SAT has been used in many problems in Data Analysis, Artificial Intelligence (AI) and Machine Learning [80]. Correlation clustering is a well-studied problem in data analysis and AI in which data are divided into subgroups in a meaningful way.

Discovering an optimal way of making such a division is a computational challenge.

There are many approaches to find the optimal clustering, including a greedy local-search and approximation algorithms, which cannot find optimal clusterings. Solving exact formulations of the correlation clustering as MAX-SAT based approach leads to cost-optimal correlation clustering [81]. Bayesian Network Structure Learning (BNSL) is a computationally hard problem of finding a directed acyclic graph structure that optimally describes a given data structure. These problems use learning that can be based on probabilistic or exact inference methods. Using MAX-SAT as exact inference has been shown to yield a competitive approach to learning optimal bounded tree-width Bayesian network structures (BTW-BNSL) [82]. There are many other AI applications and data analysis approaches formulated as MAX-SAT, including causal structure discovery [83], and deriving interpretable classification rules [84].

4.2.1.2 Planning and Scheduling

MAX-SAT can be applied in linear temporal logic (LTL) specifications for robotic motion planning and control of autonomous systems. Suppose that we want to design a controller for a robotic museum guide; the robot has to give a tour of the exhibitions in a specific order, which constitutes the hard specification. Preferably, it also avoids certain locations, such as the staff's office, the library, or the passage when it is occupied. These preferences are encoded in the soft specifications [85]. This is an example of a partial MAX-SAT formulation. There are other planning problems that can be encoded as MAX-SAT for cost-optimal planning [86,87].

Scheduling problems are well-known problems that appear in various contexts, including health care, airlines, transportation services, and various financial and money transfer problems in organizations. These scheduling problems can be encoded as a weighted partial MAX-SAT problem [88].

4.2.1.3 Verification and Security

Functional verification tasks dominate the effort of contemporary VLSI and SoC design cycles. A major step of functional verification is design debugging, which determines the root cause of failed verification tasks such as simulation or equivalence checking. The MAX-SAT formulation is used as a pre-processing step to construct a highly optimized debugging framework [89–91]. One of the techniques for debugging both hardware and software is fault localization, where the goal is to pinpoint the localization of bugs. Fault localization is performed using the MAX-SAT approach to reduce and improve automation for error localization, which can speed up the debugging process [92,93].

MAX-SAT has many applications in security. Starting with solving the user authorization query problem [94], reconstructing AES key schedule images [95], detecting hardware Trojans [96], and malware detection [97].

4.2.1.4 Bioinformatics

MAX-SAT has many applications in the bioinformatics field, such as cancer therapy, finding the optimal set of drugs to fix or rectify the fault areas of the gene regulatory network [98], modeling biological networks and checking their consistency [99], finding the maximum similarity between RNA sequences [100] and finding the minimum-cardinality set of haplotypes that explains a given set of genotypes [101].

4.2.1.5 Combinatorial Optimization Problems

Combinatorial optimization problems are widely studied in fundamental academic research and in solving real-life problems. Many of these problems are NP-hard, where an exhaustive search is not tractable. For instance, MAX-SAT has been used to encode and solve such problems as the Max-Clique problem [102–104], given a group of vertices and edges. The maximal clique is the largest subset of vertices in which each point is directly connected to every other vertex in the subset.

Other applications within this domain that have been encoded into MAX-SAT consist of determining the Treewidth of a graph [105] and finding solutions for the maximum quartet consistency problem [106].

4.2.2 Classical Algorithm for Maximum Satisfiability Problem

There are many classical algorithms for solving MAX-SAT problems: exact algorithms, stochastic local search algorithms [107–109], evolutionary algorithms [110,111], and hybrids of local search and evolutionary algorithms [112,113]. Exact algorithms are often used for small or medium size problems that can be easily verified as satisfied or unsatisfied. The exact algorithms are based on the Davis–Putnam–Logemann–Loveland algorithm (DPLL) [114], an example being the Branch-and-Bound algorithm [115,116] which represents the search space of all possible value assignments to variables as a search tree. Branch-and-Bound explores the branch of the tree and creates new formulas with partial assignments in the internal nodes until the solution is found. The solution is stored in the leaf nodes, which are bound to prevent unnecessary branches. Large size problems use stochastic local search algorithms and evolutionary algorithms which can potentially provide a high-quality solution [112,117].

4.2.3 Quantum Algorithms for Maximum Satisfiability Problem

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

There is some active research to solve the SAT and MAX-SAT problems using the currently available quantum computers, especially the D-wave quantum annealer (QA) systems [118]. The SAT and MAX-SAT are encoded into Quadratic Unconstrained Binary Optimization (QUBO) compatible with the quantum annealer architecture. QUBO

is a mathematical class of problems expressed in binary variables as linear or pairwise quadratic terms, which may include constraints.

Practical MAX-SAT problems contain hundreds of variables and terms/clauses which cannot be handled by the currently available quantum computers. Thus, due to the limited number of qubits available, some algorithms suggested reducing the number of qubits. For instance, the quantum cooperative search algorithm for 3-SAT [119] proposed Grover's search algorithm combined with a classical algorithm that decreases the total number of variables by replacing some qubits with classical bits. However, still, the number of needed ancilla qubits is equal to the number of terms when applied to POS 3-SAT problems.

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

4.3 Definitions and Preliminaries

In this section, we will define some basic concepts related to quantum gates and quantum cost. A few useful gates are shown in Figure 4.1.

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

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

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

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

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

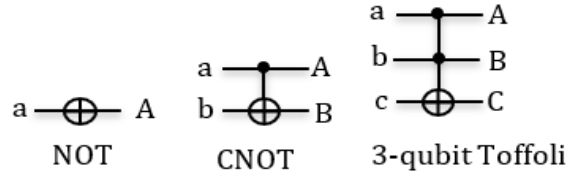


Figure 4.1 Gate symbol: NOT, CNOT, 3-qubit Toffoli gates.

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

4.3.1 Nth Root of NOT gate

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

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

Below given are notations and properties that will be used in the paper to design larger Peres gates:

- V gate = \sqrt{NOT} gate

- V^\dagger gate is inverse of V gate. Where V^\dagger is called V dagger or conjugate of V .
- $W = \sqrt{V} = \sqrt[4]{NOT}$
- $G = \sqrt{W} = \sqrt[8]{NOT}$
- $VV = NOT$
- $VV^\dagger = I$
- $WW = V$
- $GG = W$

4.3.2 Controlled-Nth Root of NOT Gate

The controlled-Nth root of NOT gate is a 2-qubit gate, where the first qubit is the control, and the second qubit is the target. When the control is one ($|1\rangle$) then the target qubit calculates the N -th root of NOT gate applied to its input value. Otherwise, with control $|0\rangle$ the target qubit is not changed. The matrix representation of controlled-Nth root of NOT gate is:

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

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

Figure 4.2 shows examples of various controlled- N th root of NOT gates that we will use in our design of large Peres gates used in counters.

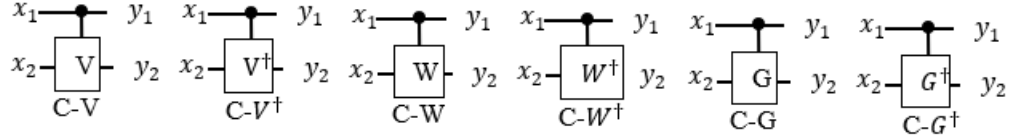


Figure 4.2 Some symbols for quantum gates of Controlled- n th root of NOT gate and their inverse (\dagger) dagger or conjugate.

4.3.3 Quantum Cost

Quantum cost of a quantum circuit is the number of elementary quantum gates used to build the circuit. The elementary quantum gates are primitive gates which are 1×1 and 2×2 reversible gates. The cost of the primitive gates is equal to 1; therefore, the quantum cost is just the number of primitive gates. For illustration, these are three elementary quantum gates that are used to calculate the quantum cost: NOT , controlled- n th root of NOT , and $CNOT$ gates where cost of each gate is equal to 1. (There are some more accurate characterizations of costs of primitive quantum gates [120] but for this paper we use the approximate costs defined as above.)

Toffoli gate could be built using controlled- n th root of NOT gate [121]. A 3-bit Toffoli gate from Figure 4.3 has two control qubits and one target qubit and is built from controlled V/V^\dagger gates and $CNOT$ gates. The quantum cost of the 3-bit Toffoli gate is 5. The generalized formula for quantum cost of m -control Toffoli gate [122] is equal to $2^{m+1} - 3$.

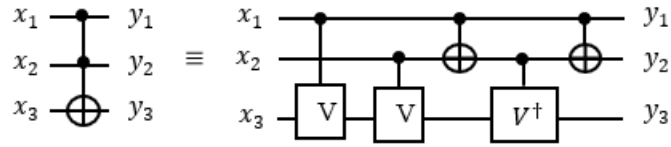


Figure 4.3 3-bit Toffoli gate represented as controlled- V/V^\dagger and CNOT gates.

4.3.4 Peres Gate

The Peres gate [123] can be characterized as a sequence of n -Toffoli followed by Feynman ($CNOT$) gates. For instance, a 3-bit Peres gate consists of a 3-bit Toffoli and a $CNOT$ gates (Figure 4.4I). When the 3-bit Toffoli and $CNOT$ gates are implemented separately using V/V^\dagger , the cost would be six (Figure 4.4II). However, the 3-bit Peres gate costs four (Figure 4.4III) because the adjacent $CNOT$ gates cancel each other. Thus, the Peres gates are used for quantum cost reduction of quantum circuits and for blocks of the iterative counter in this paper specifically.

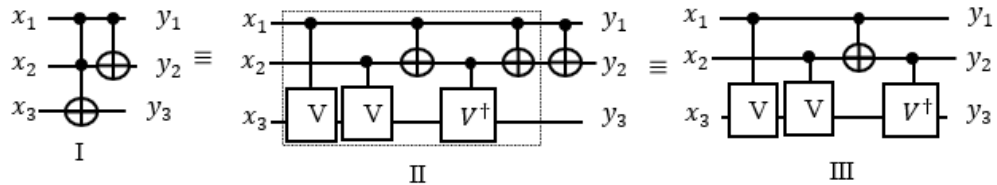


Figure 4.4 (I) Three-bit Peres gate and its representation using controlled- V/V^\dagger .

The Figure 4.4III:

- If x_1 is 1 and x_2 is equal to 0 or vice versa, then the transformation applied to x_3 and one of the V -gate will become active, and the other one will be inactive which behaves as the identity. Also, $CNOT$ will become active which produces 1 that will activate V^\dagger -gate, thus $VV^\dagger = I$.

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

In general, n -controlled Peres gate consists of $n - 1$ Toffoli and one $CNOT$ gate. Each n -qubit Peres gate can be built recursively using the $n - 1$ Peres gate block and a few additional controlled gates. The reader can appreciate this recursive way of building counter blocks of any size by analyzing Figure 4.5 in which a 4-controlled gate at the right uses the 3-controlled Peres gate in four upper qubits.

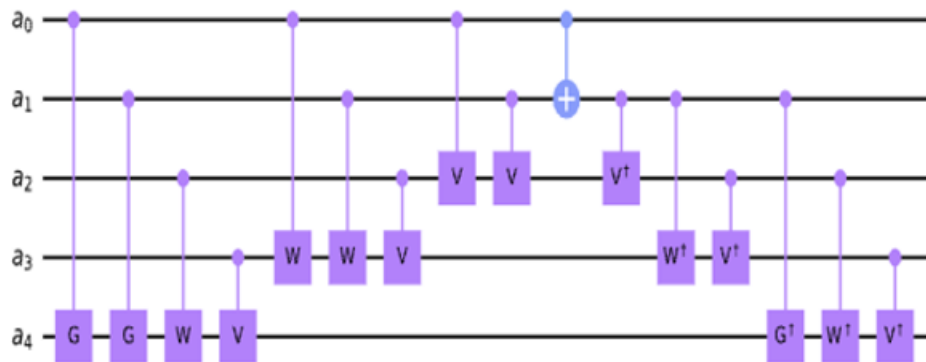


Figure 4.5 A Peres gate realized on five qubits.

As shown in Figure 4.5, the 5-qubit Peres gate uses the 4-qubit Peres gate as its sub-circuit. Figures 4.4 and 4.5 illustrate that the general formula for the quantum cost of m -controlled Peres gate [124] is equal to m^2 . For a larger design, the Peres gate can be designed as recursive blocks as shown in Figure 4.6.

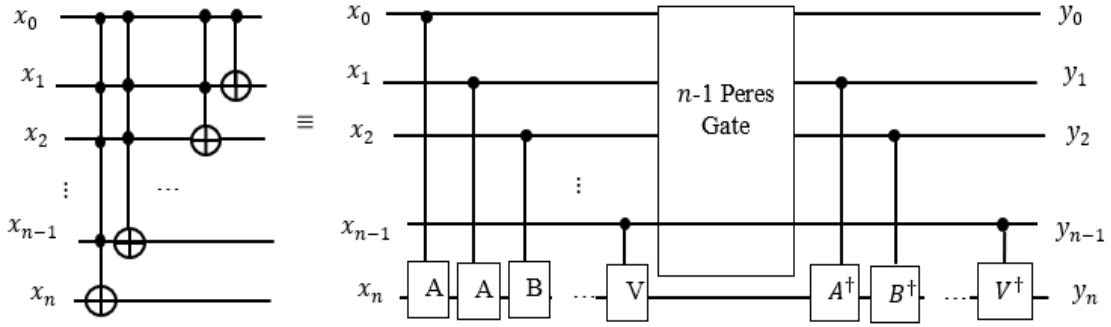


Figure 4.6 Generalized Peres gate realized on n qubits.

4.4 Quantum Algorithm for Maximum Satisfiability

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

gates. We connect one block of the iterative quantum counter after each Toffoli gate representing the OR term of the function POS formula. The satisfiability value of this term controls the block of the counter by activating this block or not. It then increments the count by 1 or 0, depending on the truth value of the OR term. Thus, our quantum counter counts the number of satisfied OR terms in the Boolean function implemented as a POS.

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

The MAX-SAT contains n variables from the given Boolean function which is used to represent the search space of $N = 2^n$ elements. To apply the MAX-SAT in Grover's algorithm, these N elements are applied in a superposition state which is the input to the oracle. If the oracle recognizes an element as the solution, then the phase of the marked state is inverted. The marked element is a true minterm of function f from the oracle. The true minterm is a product of all variables of function f that evaluates to $f = 1$. Thus,

Grover's algorithm can be used to solve the decision maximum satisfiability k -SAT for every value of k . To solve the optimization problem of finding MAX-SAT with maximum value of k the Grover's Algorithm has to be repeated.

4.4.1 Quantum Counter

As described in Section 3.3, the quantum counter block should be constructed from multiple-controlled Peres gates, where the first qubit of the Peres gate is applied a constant 1 with other variables combined, and the Peres gate is then turned into a quantum counter. (This qubit will be next taken from the OR term of the satisfiability formula to activate the counter block realized from Peres gates). For simplicity of explanation, we assume that the counter block is built from Toffoli and CNOT gates, as shown in Figure 4.7.

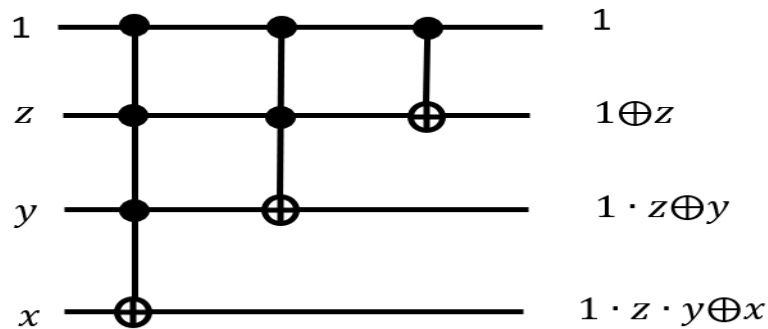


Figure 4.7 Three-qubit quantum counter.

As can be seen in Figure 4.7 and Table 4.1, z is the least significant qubit and x the most significant. The outputs of CNOT and two of the Toffoli gates are $1 \oplus z$, $1 \cdot z \oplus y$, and $1 \cdot z \cdot y \oplus x$, respectively. When $xyz = 000$, the first Toffoli gate outputs $1 \cdot z \cdot y \oplus x = 1 \cdot 0 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$ and the second $1 \cdot z \oplus y = 1 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$. The outputs of the qubits y and x are both zeros. The output of the qubit z is $1 \oplus z =$

$1 \oplus 0 = 1$. Hence the circuit incremented 000 by 1 to 001. Quantum counter circuit indeed outputs the value input+1.

Table 4.1 Analysis of 3-qubit quantum counter block from Figure 4.7

| input | | | Toffoli Gate Action | output | | |
|-------|-----|-----|--|--------|-----|-----|
| x | y | z | | x | y | z |
| 0 | 0 | 0 | $x = 1 \cdot 0 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$ $y = 1 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$ $z = 1 \oplus 0 = 1$ | 0 | 0 | 1 |
| 0 | 0 | 1 | $x = 1 \cdot 1 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$ $y = 1 \cdot 1 \oplus 0 = 1 \oplus 0 = 1$ $z = 1 \oplus 1 = 0$ | 0 | 1 | 0 |
| 0 | 1 | 0 | $x = 1 \cdot 0 \cdot 1 \oplus 0 = 0 \oplus 0 = 0$ $y = 1 \cdot 0 \oplus 1 = 0 \oplus 1 = 1$ $z = 1 \oplus 0 = 1$ | 0 | 1 | 1 |
| 0 | 1 | 1 | $x = 1 \cdot 1 \cdot 1 \oplus 0 = 1 \oplus 0 = 1$ $y = 1 \cdot 1 \oplus 1 = 1 \oplus 1 = 0$ $z = 1 \oplus 1 = 0$ | 1 | 0 | 0 |
| 1 | 0 | 0 | $x = 1 \cdot 0 \cdot 0 \oplus 1 = 0 \oplus 1 = 1$ $y = 1 \cdot 0 \oplus 0 = 0 \oplus 0 = 0$ $z = 1 \oplus 0 = 1$ | 1 | 0 | 1 |
| 1 | 0 | 1 | $x = 1 \cdot 1 \cdot 0 \oplus 1 = 0 \oplus 1 = 1$ $y = 1 \cdot 0 \oplus 1 = 0 \oplus 1 = 1$ $z = 1 \oplus 1 = 0$ | 1 | 1 | 0 |
| 1 | 1 | 0 | $x = 1 \cdot 0 \cdot 1 \oplus 1 = 0 \oplus 1 = 1$ $y = 1 \cdot 0 \oplus 1 = 0 \oplus 1 = 1$ $z = 1 \oplus 0 = 1$ | 1 | 1 | 1 |
| 1 | 1 | 1 | $x = 1 \cdot 1 \cdot 1 \oplus 1 = 1 \oplus 1 = 0$ $y = 1 \cdot 1 \oplus 1 = 1 \oplus 1 = 0$ $z = 1 \oplus 1 = 0$ | 0 | 0 | 0 |

If we connect the first control input of the quantum counter block to a circuit, then the output of the connected circuit (a term of the POS) will either activate or deactivate the counter. When the output of the connected circuit is equal to 1, the output of the counter block is incremented by 1. When the output of the circuit is equal to 0, the output of the counter block is unchanged.

4.4.2 Traditional Oracle for Satisfiability Boolean function

To build an OR term using a Toffoli gate, we use De Morgan's Law to convert the term into a product of the same variables $a + b + c = \overline{\overline{a + b + c}} = \overline{\overline{a} \cdot \overline{b} \cdot \overline{c}}$. With the XOR operation, $1 \oplus a = \overline{a}$. Hence $a + b + c = \overline{\overline{a} \cdot \overline{b} \cdot \overline{c}} = 1 \oplus \overline{\overline{a} \cdot \overline{b} \cdot \overline{c}}$. The corresponding quantum circuit using a Toffoli gate is shown in Figure 4.8.

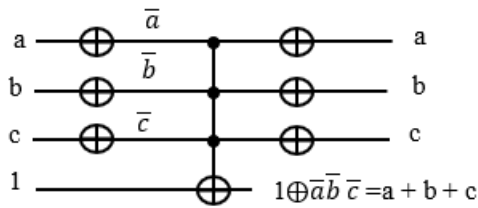


Figure 4.8 Convert sum term to product term using De Morgan's law.

Suppose we have a Boolean function $f(a, b, c) = (a + b + \overline{c})(\overline{a} + \overline{b} + c)(b + c)$ from Karnaugh map in Table 4.2. As one can see in Table 4.2, there are four ones in the Karnaugh Map, which means the solution of the Boolean variables in binaries are $(abc = 010, 011, 111, 101)$, which are satisfied for the Boolean function. Every true minterm in the Karnaugh map from Table 4.2 is a marked element and potential solution to the Grover's algorithm. However, in one run of Grover's search algorithm, only one solution is found.

We build a quantum oracle for the Grover's Loop using Toffoli gates, in which the XOR gate is controlled by the product of variables. We need to first convert the Sum expressions into Products using De Morgan's Law.

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

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

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

After building each term with the corresponding product expression, each with an assigned ancilla qubit for the output, we need to put the terms together as the product of the OR terms for the entire function $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$. Since $xyz \oplus 0 = xyz$, we use another Toffoli gate controlled by the product of the OR terms XORed with 0. The schematic of the entire circuit for $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$ is shown in Figure 4.9.

Table 4.2 Karnaugh map of POS for the Boolean function $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$

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

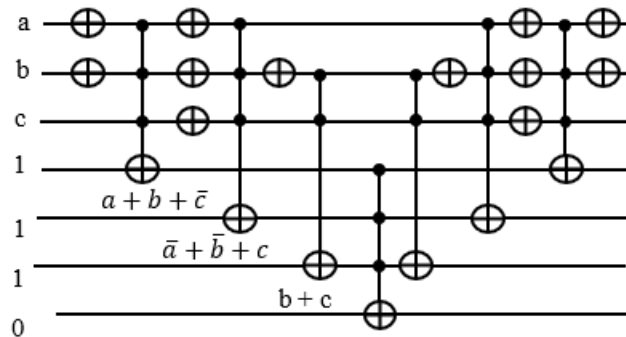


Figure 4.9 Traditional oracle for Multiple input Toffoli gate used as global AND gate $f =$

$$(a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$$

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

Let us define n number for variables and t number for terms then the number of qubits q needed for the oracle is: $q = n + t + 1$. Where 1 is for the OR terms XORed with 0. Notice that we need three ancilla qubits, which is equal to the number of terms. For a function involving thousands of terms, we would need an equal number of ancilla qubits.

4.4.3 Construction of a Quantum Oracle for MAX-SAT

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

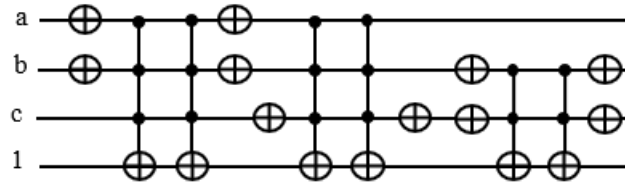


Figure 4.10 Improved version of the part the oracle $f = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$.

The target output of each Toffoli gate realizing an OR term is used to activate the counter block corresponding to it. In Figure 4.10, notice that there are two NOT gates adjacent to each other, canceling each other out. Hence, we can remove those gates from our circuit.

There are eight NOT and six Toffoli gates in this design in Figure 4.11 as opposed to 12 NOT and 7 Toffoli gates in the traditional design in Figure 4.9. We need ancilla qubits

in the traditional design because we need the outputs from the Toffoli gates recorded in the ancilla qubits for counting the number of satisfied terms. By sending the satisfaction result for each term to the quantum counter, we are able to reset the output line back to 1.

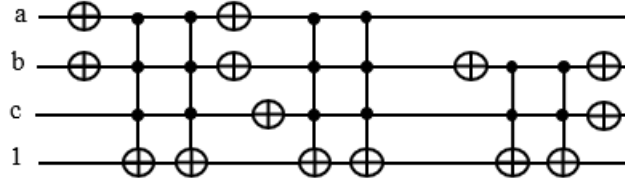


Figure 4.11 Improved and optimized version of the part the oracle $f = (a + b + c)(\bar{a} + \bar{b} + c)(b + c)$

The count for the number of satisfied terms is output on the xy qubits. In this case, we have three satisfied terms and want to have three as the output expressed as 11 which are expressed as $xy \oplus out_0 = xy \oplus 0$ on a Toffoli gate. If the Boolean function f is satisfied, then the outcome out_0 should be 1. The entire oracle with the function and the iterative counter is shown in Figure 4.12. We applied this oracle in the Grover search

algorithms for $R = 2$ iterations from this formula: $R \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil$ where $M = 4$ is the number of solutions in our problem from Table 4.2, and $N = 8$ is the number of all search space elements (cells of the Karnaugh map from Table 4.2). In general, the value of M is calculated using Quantum Counting algorithm [48], but an unsolved problem, the value of M , is taken as 1 to run the Grover iterations R .

In Figure 4.13, we run the circuit on the ‘qasm_simulator’ from QISKIT for 1024 shots (independent runs to obtain high precision probability) for which the circuit produces the correct answers. We measured a_0, a_1, a_2 and out_0 in Figure 4.13 where a_0, a_1, a_2 correspond to the Boolean variables, a, b, c , respectively in Figure 4.12. As can be

seen in Figure 4.14, it illustrates the QISKIT [125] output graphics for the simulated circuit. The measured values with high probability are 1010, 1101, 1110, and 1111, where the most significant qubit is out_0 which is 1, and the least three significant qubits 010, 101, 110, 111 are all satisfied values for the Boolean function. These solutions correspond to the true minterms from Table 4.2. For the unsatisfied, the measured values with low probability are 0000, 0001, 0011, and 0100, where the most significant qubit is out_0 which is 0, and the least three significant qubits 000, 001, 011, 100 are all unsatisfied values for the Boolean function.

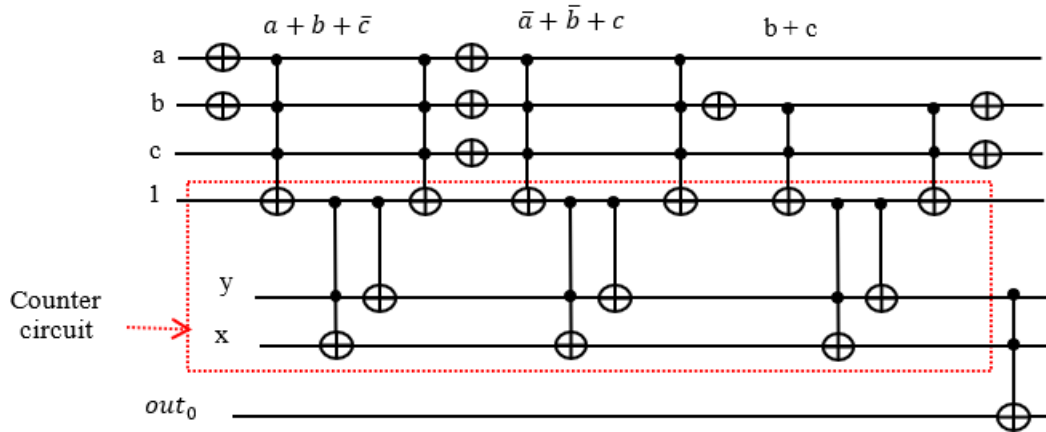


Figure 4.12 Improved complete oracle using the quantum counter.

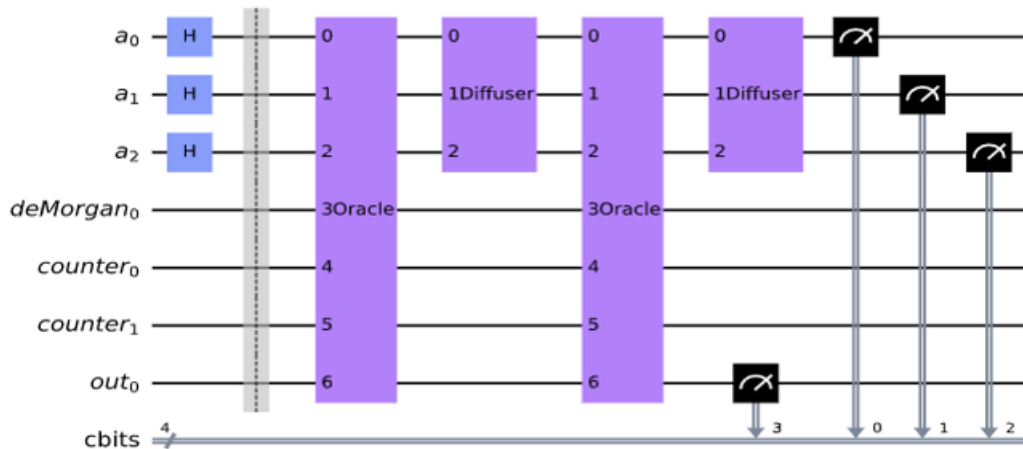


Figure 4.13 Oracle from Figure 4.12 is used twice inside the Grover's algorithm.

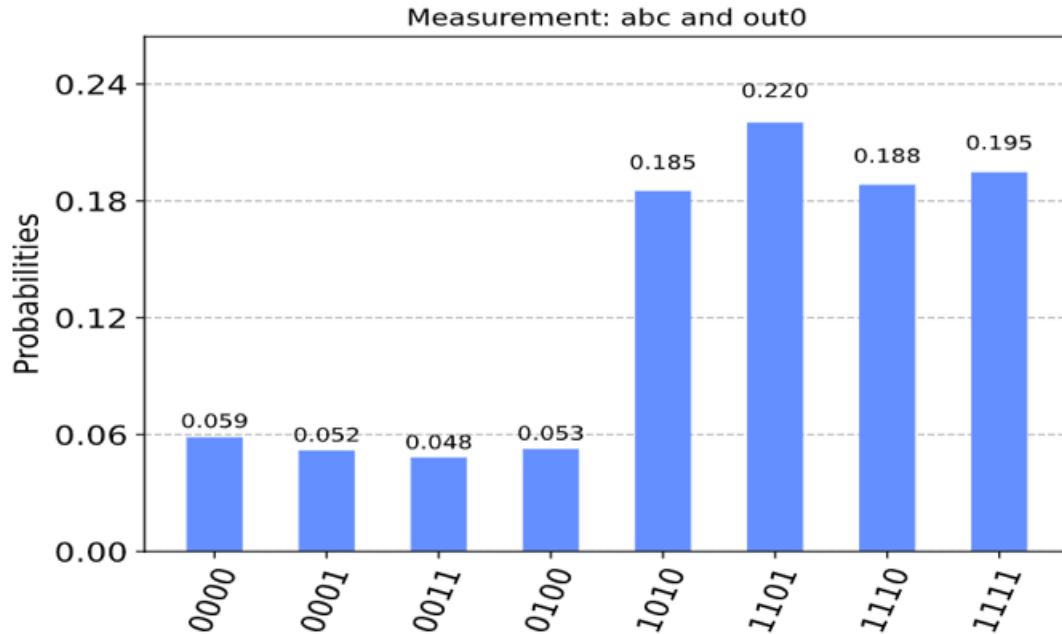


Figure 4.14 Measurement of the Boolean variables and the outcome of function from Figure 4.13

As can be seen in Figure 4.14, the four values 000, 001, 011, and 100 have some value with less probability because of noise created by the simulator. However, we verified the solutions by applying the number of iterations R , and the output from the simulation with high probability 010, 101, 110, and 111 matches the theoretical values, which can be verified manually. We also applied different shots to test, and the results were closely similar, with a high probability for all satisfying values.

4.4.4 Verifying an Unsatisfiable Function

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

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

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

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

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

The four qubits (1, z, y, x) in block (A) realize the counter, which can count from 0 to 7. We need the last qubit with out_0 ancilla bit to produce 1 when all terms are satisfied for Grover's algorithm. Since this function has four terms, to check satisfiability which is the last qubit should be 1, we need to add two NOT gates in the block (B) which makes the last qubit to produce 1 if the Boolean function is satisfied. The function $f(a, b)$ from Figure 4.15 is not satisfiable, so comparing to a value of 4 in the last gate would not generate any correct solution. Grover's algorithm will give a few random values that can be verified on the satisfiability formula outside Grover's Algorithm using function $f(a, b)$. Therefore, we remove the two NOT gates in block (B) to get the maximum satisfied terms of the function.

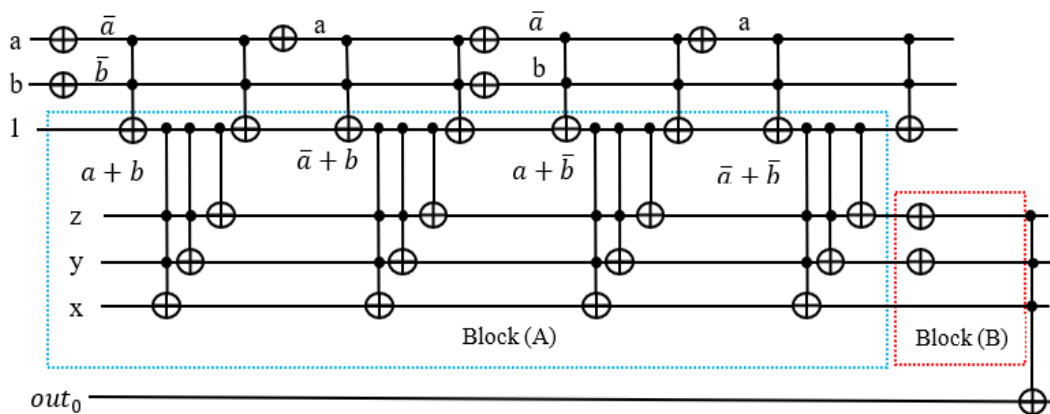


Figure 4.15 Oracle with counter $f(a, b) = (a + b)(\overline{a} + b)(a + \overline{b})(\overline{a} + \overline{b})$

In a more general case in Figure 4.16, we repeat the Grover Algorithm with tuning values of thresholds until equal to counter value xyz . The comparator $G = H$ compares the output from the counter with the threshold value given as constant values n_1, n_1 , and n_3 . For instance, $f(a, b) = (a + b)(\bar{a} + b)(a + \bar{b})(\bar{a} + \bar{b})$ has 4 terms, we tune the threshold value from 4, 3, 2, and 1 until the condition is met. The value of the counter where the condition is met is the MAX-SAT value. If the condition is met, the ancilla qubit out_0 will be flipped. It changes the quantum phase of the solution so that the elements that satisfy all constraints are marked. This method of the threshold with comparator is useful to check when the exact number of terms (constraints) are known, which can be checked whether the threshold is equal to the counter value. For instance, if there are 10 constraints in a given function, but it should satisfy a minimum seven constraints, then set the threshold to seven and check if the counter equals to seven. There are applications based on the method of the threshold with a comparator, such as finding the minimum set of support [126].

Every binary vector $|a, b\rangle$ of a solution can be verified by running outside of the Grover Algorithm, as can be seen in Figure 4.17 in which the maximum number of satisfied terms is 3 out of 4. We applied one Grover's Loop iteration for this oracle to get the MAX-SAT. In Figure 4.18, we run the circuit on the 'qasm_simulator' from QISKIT for 1024 shots.

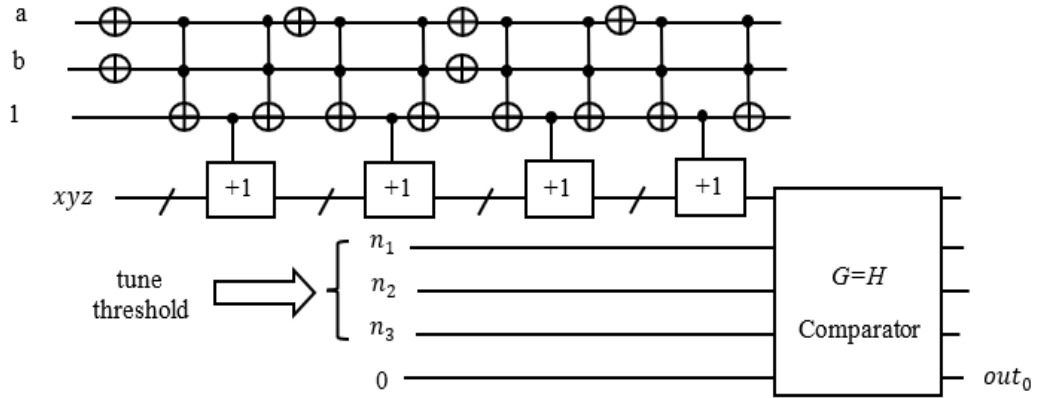


Figure 4.16 Oracle with a “counter circuit” and with a “threshold with comparator” circuit.

In Figure 4.18, we measured the Boolean variables, counter, and output. In Figure 4.19, the most significant qubit out_0 always is 0, which means the Boolean function is not satisfied because there are no such binary values for the least two significant qubits 00, 01, 10, and 11, which would satisfy the Boolean function. However, the novelty of our design is that the counter qubits give the maximum numbers of satisfied terms in the Boolean function. The counter qubits are the second, third, and fourth qubits from the most significant qubit, which in this case is 011.

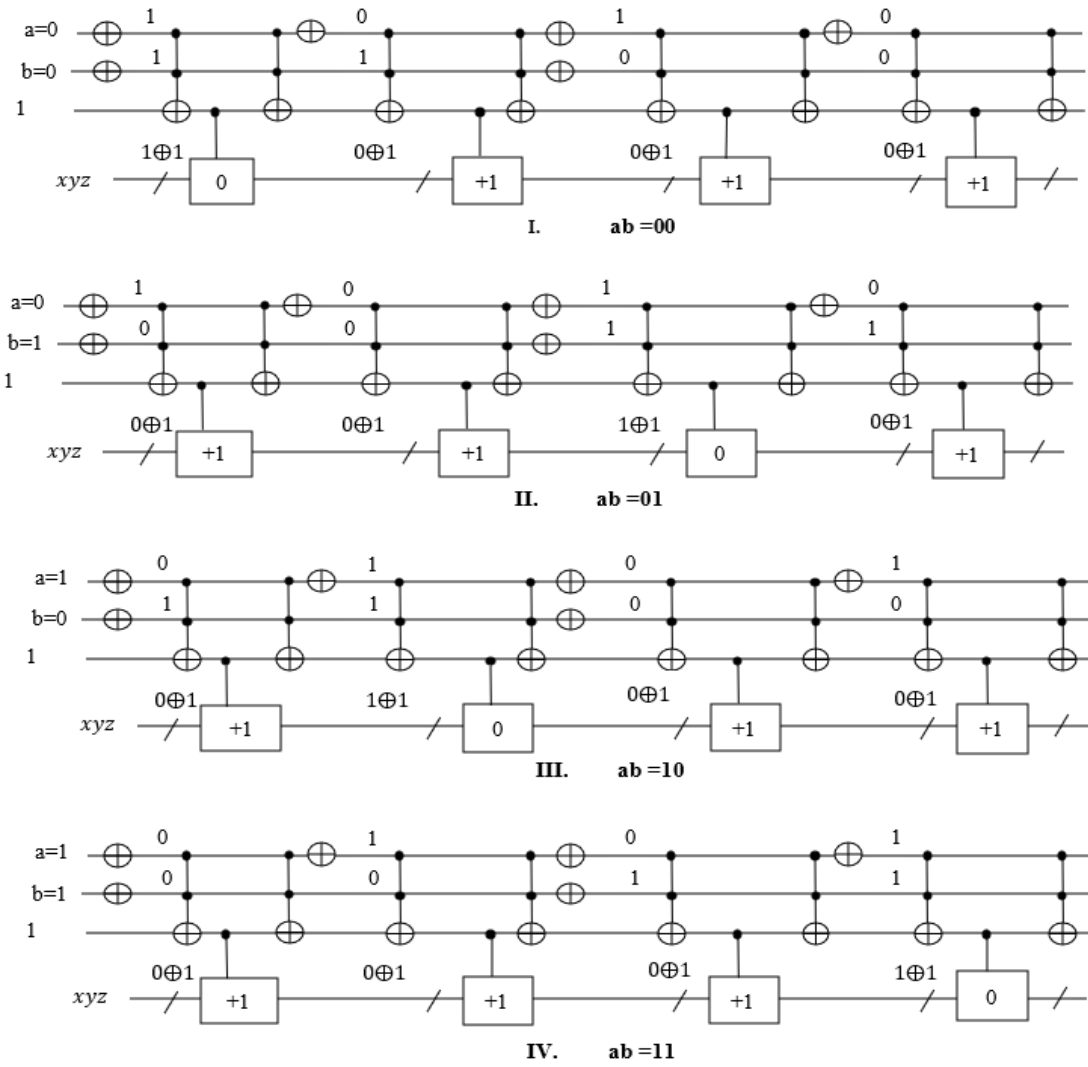


Figure 4.17 MAX-SAT verification such that 00, 01, 10, 11 for ab has always MAX-SAT terms equal

$$\text{to } 3 \text{ for } f(a, b) = (a + b)(\bar{a} + b)(a + \bar{b})(\bar{a} + \bar{b}).$$

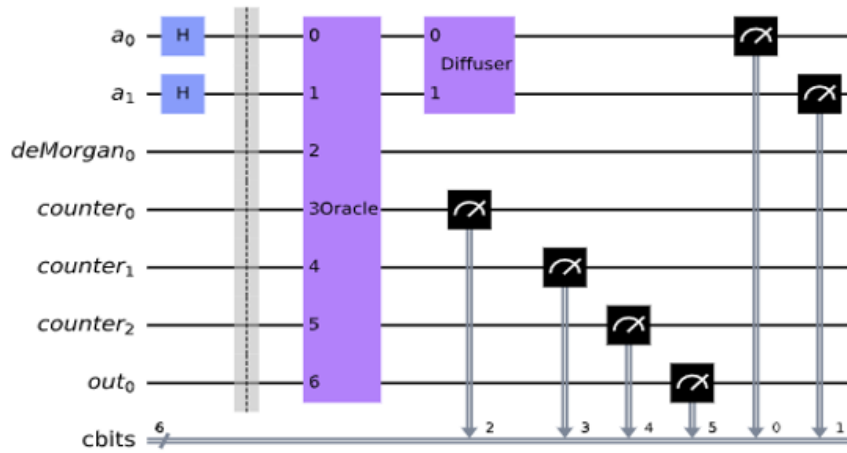


Figure 4.18 Oracle from Figure 4.15 applied Grover's algorithm.

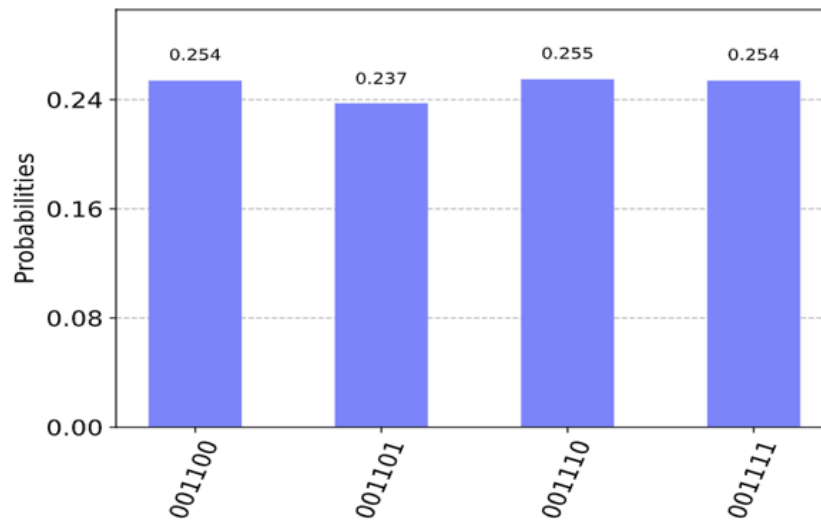


Figure 4.19 Measurement from Figure 4.18

4.5 Calculation of Quantum Cost

4.5.1 Calculation of Quantum Counter Size

Table 4.3 shows the required number of qubits for the quantum counter, in which each term is not required for one ancilla qubit, but many terms require a few ancilla qubits.

Table 4.3 Quantum counter size; total qubits for counter

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

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

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

As shown in Figure 4.20, for instance, if there are 100,000 terms, then the number of required ancilla qubits in traditional oracle is 100,000, but in our design, the quantum counter requires only $\lceil \log_2 T \rceil + 1 = 18$ ancilla qubits. Using the quantum counter, each term is not required for one ancilla qubit, but many terms are required for a few ancilla qubits.

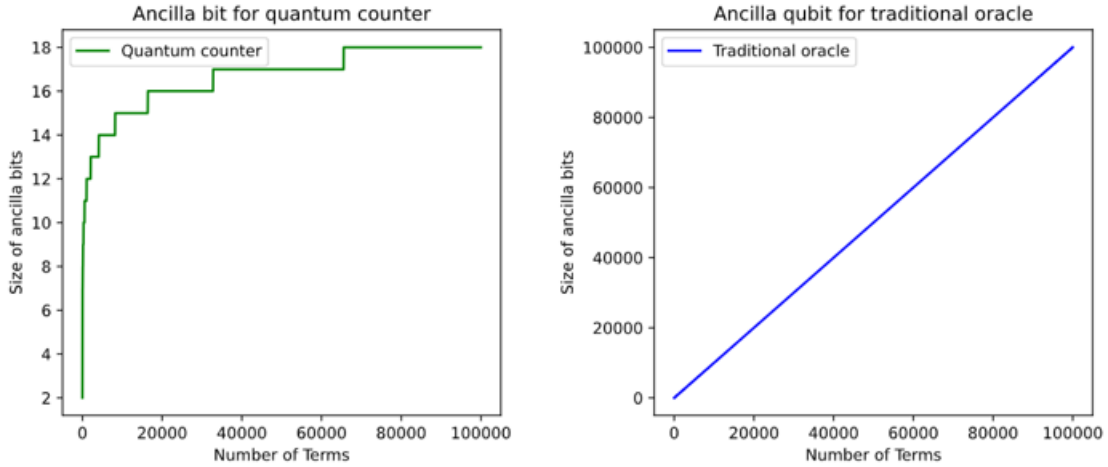


Figure 4.20 Comparison of required numbers of ancilla qubits for our oracle and the traditional oracle.

4.5.2 Quantum Cost Calculation for Quantum Counter

Each term in the Boolean function is represented as n -bit Toffoli gate, and the satisfiability result is passed down to the counter. We need as many counter blocks as there are terms in the given POS Boolean function. The counter can be built from Toffoli gates or Peres gates. It is important to have low cost quantum circuits for this high demand n -bit Toffoli gates. Since the Peres gate is a low-cost quantum circuit, we replaced the Toffoli gates with Peres gates for cost reduction [122]. The formula of quantum cost for m -controlled bits of Peres gate is m^2 and for Toffoli gate is $2^{m+1} - 3$.

In Figure 4.21 a three-qubit counter (3-control qubits) consists of three Toffoli gates which are 3-control, 2-control, and 1-control (CNOT) gates. for each of these Toffoli gates, the quantum cost is calculated separately: $(2^{3+1} - 3) + (2^{2+1} - 3) + (2^{1+1} - 3) = 28 - 9 = 19$. Four-qubit counter consists of four Toffoli gates, and the quantum cost is also calculated separately: $(2^{4+1} - 3) + (2^{3+1} - 3) + (2^{2+1} - 3) + (2^{1+1} - 3) = 60 - 12 = 48$.

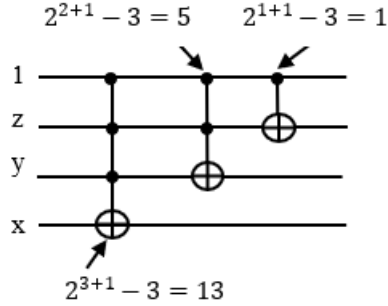


Figure 4.21 Calculation of the quantum cost for the 3-bit counter.

Thus, we can derive a general formula for the quantum cost of m -bit quantum counter using the Toffoli gate: $2^{m+2} - 4 - 3m$

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

$$\text{Peres cost} = T * m^2 \tag{4.1}$$

$$\text{Toffoli cost} = T * (2^{m+2} - 4 - 3m) \tag{4.2}$$

Based on these two Formulas (4.1) and (4.2), the Toffoli gate has a higher quantum cost than Peres gate. Thus, we used in our design the Peres gates. As we mentioned before, our final counter uses Peres gates, so we built our oracle using the Peres gate, and it is mapping to the n th root of NOT gates which leads to low quantum cost. The recursive design method from Peres gate was used.

4.6 Variants of SAT Oracles Using Quantum Counter

Following our preliminary work [127], in this section, we discuss some other applications of the quantum counter in variants of satisfiability problems, such as the product of SOPs SAT.

4.6.1 Oracle for SOPs

MAX-SAT can be solved for a Product of any function. In particular, this can be a Product of SOPs. The SOP functions can be implemented with a counter by summing the digits of the counter at the end, using De Morgan's rule. Each product term is simply a Toffoli gate, and the counter can be checked in a similar way to a regular sum term.

Figure 4.22 presents an example circuit for the function $ab + b\bar{c} + \bar{a}\bar{c}$.

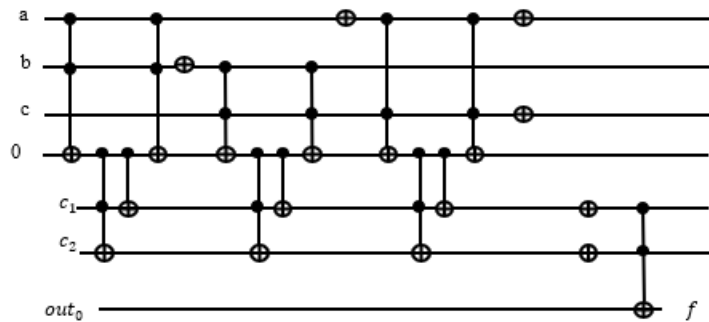


Figure 4.22 Part of the product of SOP oracle that realizes SOP function $f = ab + b\bar{c} + \bar{a}\bar{c}$.

4.6.2 Oracle for Product of SOPs (POSOP SAT)

POSOP functions consist of products of SOP functions. We were not able to find any references to this form of SAT. However, we can take advantage of the fact that every term must be true in a product for the product to be true, and thus we can check against a counter value of the number of terms in order to construct the oracle for POSOP. For example, Figure 4.23 presents the circuit for function $(\bar{a}\bar{b} + \bar{a}c)(abc + \bar{b}\bar{c})$.

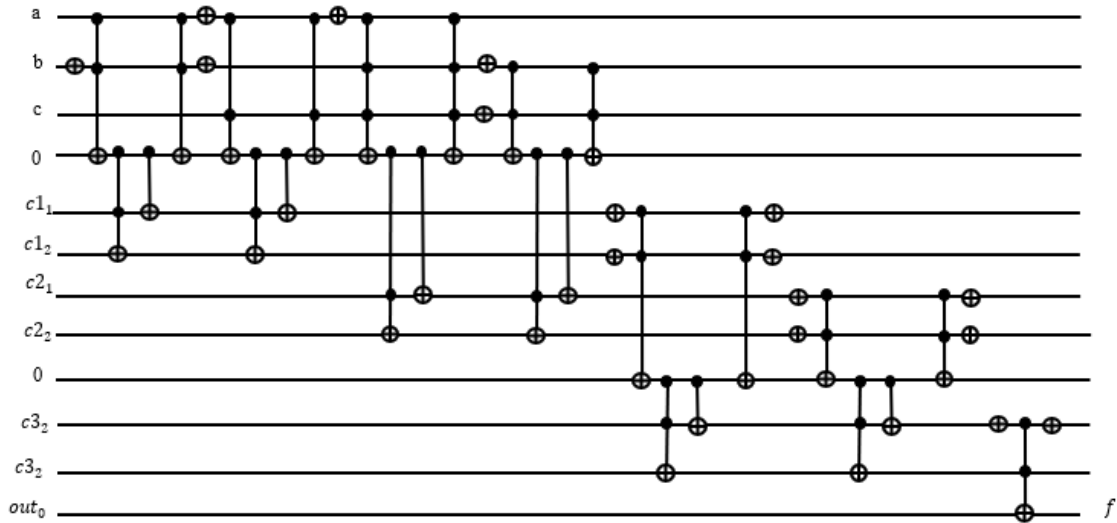


Figure 4.23 Realization of the Oracle for $f = (a\bar{b} + \bar{a}c)(abc + \bar{b}\bar{c})$, with POSOP SAT.

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

4.6.3 Oracle for Exclusive-or-Sum-of-Products (ESOP)

An Exclusive-or-Sum-of-Products (ESOP) form is an exclusive sum (using the ‘ \oplus ’) operator of product terms. There is not much published on ESOP SAT except for [128], although this is an interesting subject for research. Grover’s Oracle can be trivially applied to ESOP SAT, a problem that has also not been discussed yet in the literature on the subject. The advantage of ESOP SAT over OR SAT presented in the previous section is that ESOP SAT can be realized without the need for a large AND gate or a counter. Since every product in the EXOR sum can be implemented as a Toffoli gate, SAT with ESOP can be formulated with just the input qubits and one output qubit. For example,

given a function such as $ab \oplus b\bar{c} \oplus \bar{a}\bar{c}$, we can implement Grover's Oracle, as shown in Figure 4.24.

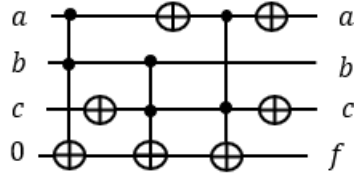


Figure 4.24 Realization of Oracle $f = ab \oplus b\bar{c} \oplus \bar{a}\bar{c}$ for ESOP SAT realized in Grover's Algorithm.

4.7 OR Satisfiability Problems for Electronic Design Automation

In this section, we will show that many EDA (Electronic Design Automation) problems can be reduced to SAT and MAX-SAT. In the most general case, the Satisfiability Decision Function problem is formulated as an arbitrary binary-valued-input, binary-output, and single-output function. For instance, a product of sums of literals, (the literals are variables negated or not), EXOR of products of literals, and product of sums of products of literals. These functions are created by transforming some natural language or mathematical decision problems, such as, for instance, cryptographic puzzles. The question is to find out for which values of variables the formula for SAT or MAX-SAT is satisfied. In some problems, one has to find all solutions; in some other problems we look for just one solution or only some solutions. For all these variants, we have some freedom to modify Grover's Algorithm, and/or call it several times with modified oracles [126].

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

literals, and each literal being a Boolean variable or its negation. We are interested in the following problems.

Problem 7.1 (Satisfiability): Answer Yes if there exists a product of literals that satisfies all terms or No if such a product does not exist. Give the solution as a set of literals.

Problem 7.2 (Optimization of the Generalized Petrick function): Find a product with the minimum number of literals that satisfies all terms or prove that such a product does not exist.

Problem 7.3 (Optimization of the Generalized Petrick function-nonnegated literal variant): Find such a product of literals that satisfies all terms and in which a minimum number of literals is not negated or prove that no such product exists. (The not negated literals will also be called positive literals). In particular, the Petrick Function is positive unate, which it means has only positive literals.

Problem 7.4 (MAX-SAT): Find such set of literals that satisfies the maximum number of terms.

Problem 7.5 (Tautology Checking): Verify whether a function is a Sum of Product Form is a Boolean tautology. Function F is a tautology (all input combinations are 1) when its negation \bar{F} is not satisfiable (all combinations are 0).

In some variants of these problems, depending on a particular application, we can look for all solutions, all optimal solutions, some optimal solutions, or for a single optimal solution. The central role of Problem 7.1 is well-established in computer science.

All NP-complete combinational decision problems are equivalent to the Satisfiability Problem [129]. Many reductions of practically important problems to other above problems were shown, including problems from VLSI Design Automation, especially in logic design and state machine design. SAT and MAX-SAT also have many applications in logistics, scheduling, AI, and robotics. Ashenhurst/Curtis Decomposition of Boolean functions can be done in an algorithm that repeatedly applies Satisfiability [130]. Generalized Ashenhurst/Curtis Decomposition was also realized by building a complex oracle for Grover's Algorithm based on the mathematics of Partition Calculus [131]. These SAT-like problem formulations are also of fundamental importance in many algorithms for Boolean minimization, factorization, and multi-level design. The set covering problem is reduced to the minimization of Petrick Function. The reductions of many practically important NP-hard combinatorial optimization problems can also be found in the literature. For instance, the minimization of the Sum of Products Boolean functions can be reduced to the Covering Problem [132] and Covering Problem can be further reduced to the Petrick Function Optimization Problem (PFOP) [133]. Many other problems, like test minimization, can also be reduced to the Covering Problem [132,134]. The problems of Partial Satisfiability and its applications are discussed by K. Lieberherr [135]. Many other reductions to the formulated above problems are discussed in [129,136]. Paper [137] discusses the reduction of three-level NAND circuits, TANT, to the covering-closure problem solved similarly to SAT. A similar problem of the synthesis of networks from negative gates uses the same reduction [138]. A design automation system [139] was created, in which many problems were first reduced to the few selected

“generic” combinatorial optimization problems. These problems include some of the problems listed above.

The problem of minimization of Finite State Machines includes: (1) the Maximum Clique Problem and (2) the problem of finding the minimum closed and a complete subgraph of a graph (Closure/Covering Problem) [137]. The first of these problems, (1), can be reduced to the Petrick Function Optimization Problem (PFOP). The problem of optimum output phase optimization of PLA [140] can be reduced to PFOP. The second problem, (2), can be reduced to the Generalized Petrick Function Optimization Problem (GPFOP), introduced above and illustrated below. Many other problems, like AND/OR graph searching [141], were reduced to the Closure/Covering Problem.

A number of problems (including Boolean minimization [142], layout compaction, and minimization of the number of registers in hardware compilation can be reduced to the Minimal Graph Coloring Problem. Regular layout problems can be reduced to SAT [143]. The Minimal Graph Coloring can be reduced to the Problem of Finding the Maximum Independent Sets, and next the Covering Problem (Maghout algorithm). The Problem of Finding the Maximum Independent Sets can be reduced to PFOP. The PFOP is a particular case of the GPFOP. The role and importance of Tautology and conversion methods from SOP to POS and vice versa in logic design are well known. These problems can also be solved using the SAT.

Concluding on OR SAT. In theory, every NP problem can be polynomially reduced to SAT and also to OR 3-SAT. But this is not practical. Many problems can be reduced to

graph coloring or maximum clique problems that can be in turn reduced to satisfiability problems.

As we see now, many problems can be solved with quadratic speedup using future quantum computers. A hybrid classical/quantum computer based on Grover tuned to solve variants of SAT problems of various types would be a tremendous asset to all these problems [126].

4.8 XOR SAT

Recent satisfiability solvers are extended to handle XOR clauses as an input language rather than only using CNF clauses, the native input of SAT solvers. XOR and CNF clauses are handled separately in different data structures. The XOR clauses are encoded to CNF clauses and performed Gaussian Elimination to accept the SAT solvers. These techniques revealed the inefficiency and performance issues of the classical SAT solvers. The XOR clauses are well-known in cryptographic problems, which are one area of the research to test and use in SAT solvers to enhance confidentiality and authenticity. The challenges in cryptography are even more difficult when it comes to post-quantum cryptography, which motivated the need for a quantum algorithm to handle the XOR clause in SAT solvers even more efficiently. We proposed a uniform quantum oracle design for quantum SAT solvers that can adapt all these forms of SAT formats, like XOR clauses, CNF clauses, and CNF-XOR clauses. Our proposed oracle design can be applied to both XOR SAT and MAX-XOR SAT problems.

XOR operator \oplus in Boolean logic performs a logical operation. If the input values are different, then the output is 1; else if the input values are same, then the output is 0. The

XOR operator has many applications throughout engineering and computer science, such as applications in encryption, error detecting, error correcting codes, machine learning [162], data structure [160, 161], and binary optimization [159], and others. Exclusive-or (XOR) constraint, also known as parity constraint, is a Boolean constraint of a series of XOR operations (\oplus) over a set of literals $x_1 \oplus x_2 \dots \oplus x_n = \varepsilon$ where x_1, \dots, x_n and $\varepsilon \in \{0,1\}$, and \oplus is modulo two arithmetic. A literal is a variable or negation of a variable. In classical SAT, a constraint is a disjunction of a set of literals. In XOR SAT it is a XOR of literals. In general, it can be an arbitrary Boolean function that describes some condition that must be satisfied. Note that constraint, clause, and term are interchangeable words in this paper. ESOP, Exclusive-or-Sum-of-Products, is an exclusive sum (using the ' \oplus ') operator of product terms such as $ab \oplus b\bar{c} \oplus \bar{a}\bar{c}$. The Product of ESOPs function such that $(\bar{a}\bar{b} \oplus \bar{a}c)(\bar{a}bc \oplus \bar{b}\bar{c})$, is called Product of ESOPs Satisfiability. XOR-SAT is a Product of ESOPs Satisfiability [153].

Satisfiability solvers are programs that employ highly optimized mathematical algorithms to decide whether a set of constraints has a solution. Most SAT solvers natively take the Boolean logic formula as input as the Conjunctive Normal Form (CNF) and return the decision. While the standard theoretical notation for SAT input formula is the CNF-SAT based, there are many real-world problems that are not in the CNF formula. There are many real-world problems that are easily formulated as XOR constraints, such as model counting [148, 151] and cryptanalysis [152]. For instance, many cryptographic problems are represented as XOR operation that is common in cryptography. CNF-XOR SAT was proposed [144] to extend SAT solvers to support

XOR operation as input language to the SAT solvers. CryptoMiniSat [145] is a SAT solver designed to accept a mixture of CNF and XOR-clauses as its input language.

4.8.1 Classical XOR SAT

The XOR-SAT problem is the problem of deciding whether there is a satisfying truth assignment for a given XOR formula. Most SAT solvers that are based on Davis-Putnam-Logemann-Loveland (DPLL) algorithm [158] only understand the conjunctive Normal Form (CNF) such that each clause contains ‘or’ operator. The constraints are presented to the SAT solvers as a product of sum (POS) ‘and of ors’ form. However, XOR-SAT problem uses the XOR ‘exclusive or’ instead of the usual ‘or’. There are pre- and in-processing techniques [148] that the SAT solver needs to process the XOR clauses. The two main techniques are: encoding XOR clauses into standard CNF clauses and performing Gaussian Elimination for XOR clauses.

4.8.2 Translating XOR clauses to CNF clauses

It is widely believed that XOR based constraints are empirically hard [146, 147, 153] for SAT solvers. When the XOR constraints are encoded to CNF representation, the number of constraints grows exponentially in size. An XOR clause that includes k variables is equivalently represented by 2^{k-1} CNF clauses with k variables in each CNF clause. This is because the XOR constraint’s Karnaugh table contains 2^{k-1} minterms, and hence needs 2^{k-1} clauses to describe in CNF. XOR clause can be converted to equivalent CNF clauses by listing the truth assignments from XOR clause. As can be seen in the Table 4.4, the XOR clause $(a \oplus b \oplus c)$ corresponds to CNF clauses $(a + b + c)(a + \bar{b} + \bar{c})(\bar{a} + b + \bar{c})(\bar{a} + \bar{b} + c)$. This encoded operation yields a large amount of CNF clauses which

means expensive overhead for SAT solvers. It is well known that the performance of SAT solvers depends on the width of the XOR clauses [149].

Table 4.4 XOR clause to CNF clauses

| a | b | c | $a \oplus b \oplus c$ | CNF |
|-----|-----|-----|-----------------------|-------------------------|
| 0 | 0 | 0 | 0 | $a + b + c$ |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 1 | |
| 0 | 1 | 1 | 0 | $a + \bar{b} + \bar{c}$ |
| 1 | 0 | 0 | 1 | |
| 1 | 0 | 1 | 0 | $\bar{a} + b + \bar{c}$ |
| 1 | 1 | 0 | 0 | $\bar{a} + \bar{b} + c$ |
| 1 | 1 | 1 | 1 | |

4.8.3 Gaussian Elimination of XOR constraints

Let n a set of XOR clauses over k variables $\mathbf{x} = \{x_1, x_2, \dots, x_k\}$ such that each XOR clause can be expressed as a linear equation as matrix form of $\mathbf{Ax} = \mathbf{b}$, where \mathbf{A} is an $n \times k$ matrix, \mathbf{b} is an $n \times 1$ constant vector from set $\{0,1\}$ and \mathbf{x} is variables from XOR clause. Gaussian Elimination is a classical algorithm for solving systems of linear equations that expressed as matrix form of $\mathbf{Ax} = \mathbf{b}$. The proposed CNF-XOR SAT solver in [144] combines Gaussian Elimination with DPPL-based algorithms. XOR constraints can be solved individually in polynomial time [150] by using Gaussian Elimination since each XOR-constraint is a linear equation. A system of n XOR constraints on k variables, Gaussian Elimination solves with time complexity of $O(kn^2)$ [154]. To solve XOR constraints in Gaussian Elimination, first, XOR constraints should be in the standard form in which all variables appear as positive form. A system of series of XOR clauses can be formulated as a system of linear equations, where each clause is a linear equation. If each

equation has a solution, then the series of XOR clauses are satisfied. For instance, $(x_1 \oplus x_3 \oplus x_4)(\bar{x}_1 \oplus x_2 \oplus x_3)$ is satisfied only if both clauses are satisfied, such as:

$$x_1 \oplus x_3 \oplus x_4 \equiv 1$$

$$\bar{x}_1 \oplus x_2 \oplus x_3 \equiv 1 \Rightarrow x_1 \oplus x_2 \oplus x_3 \oplus 1 \equiv 1 \Rightarrow x_1 \oplus x_2 \oplus x_3 \equiv 0$$

Note that the XOR operator is associative and commutative. As can be seen, each linear equation can be built as a matrix form of $\mathbf{Ax} = \mathbf{b}$. Such a system of linear equations is easily solvable using Gaussian Elimination. To perform Gaussian Elimination, XOR clauses are expressed in matrix form such that XOR clauses are represented as rows and variables are represented as columns. For instance, given four XOR clause $c_1: (x_1 \oplus x_3 \oplus x_4)$, $c_2: (\bar{x}_1 \oplus x_2 \oplus x_3)$, $c_3: (x_2 \oplus x_3 \oplus x_4)$, $c_4: (x_1 \oplus \bar{x}_2 \oplus \bar{x}_3 \oplus x_4)$. To solve these XOR clauses in Gaussian Elimination, first, the XOR clauses can be written in standard form as a linear equation.

$$x_1 \oplus x_3 \oplus x_4 \equiv 1$$

$$\bar{x}_1 \oplus x_2 \oplus x_3 \equiv 1 \Rightarrow x_1 \oplus x_2 \oplus x_3 \oplus 1 \equiv 1 \Rightarrow x_1 \oplus x_2 \oplus x_3 = 0$$

$$x_2 \oplus x_3 \oplus x_4 \equiv 1$$

$$x_1 \oplus \bar{x}_2 \oplus \bar{x}_3 \oplus x_4 \equiv 1 \Rightarrow x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus 1 \oplus 1 \equiv 1 \Rightarrow x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 1$$

These linear equations can be represented as matrix form as in Figure 4.25.

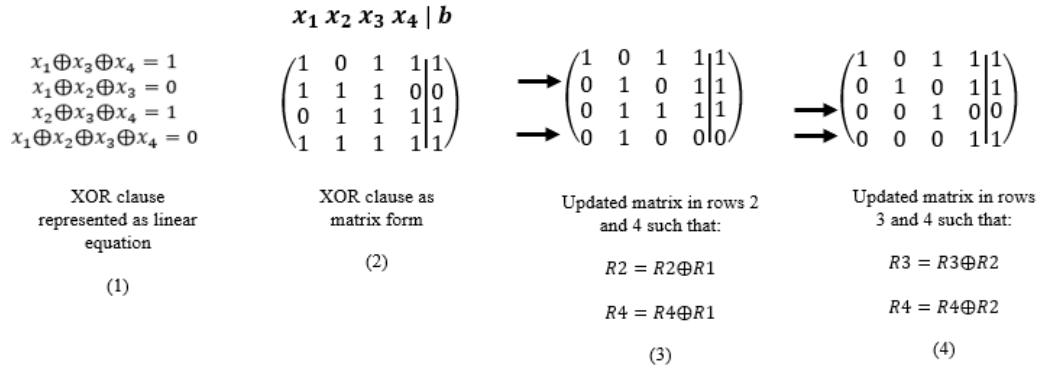


Figure 4.25 XOR clauses solved by Gaussian Elimination

The standard method of Gaussian Elimination applies row reduction until the lower triangular matrix becomes 0. In Figure 4.25(4), $x_4 = 1$, $x_3 = 0$, $x_2 \oplus x_4 = 1 \Rightarrow x_2 \oplus 1 = 1 \Rightarrow x_2 = 0$ and $x_1 \oplus x_3 \oplus x_4 = 1 \Rightarrow x_1 \oplus 0 \oplus 1 \Rightarrow x_1 \oplus 1 = 1 \Rightarrow x_1 = 0$. So, the solution for $(x_1, x_2, x_3, x_4) = (0, 0, 0, 1)$. Also, the XOR clauses can be solved by substitution method by following these four rules: $x \oplus 1 = \bar{x}$, and $x \oplus 0 = x$, and $x \oplus \bar{x} = 1$, and $x \oplus x = 0$.

CryptoMiniSat [145] SAT solver XOR clauses encoded into CNF then performed recovery steps for the XOR clauses to perform Gaussian Elimination. These in-processing steps to keep XOR and CNF clauses separately in different data structures had a significant overhead to process. Although various enhancements [154 -157] were proposed to increase the performance of the CNF-XOR SAT solvers, there are still some bottlenecks in the runtime of the SAT solvers. Performing XOR translations to CNF form and using Gaussian Elimination is very time and memory consuming.

We get motivated in terms of computation time and memory; quantum computing has the privilege of overcoming these challenges compared to classical computing. Also, in

the era of post-quantum cryptography, we would need powerful quantum SAT solvers in the cryptography environment to test the hardness of the problems such that they would be able to withstand attackers equipped with quantum computers. We propose a uniform quantum oracle design for Grover's search algorithm that can solve XOR-SAT, mixed CNF-XOR SAT, and CNF-SAT, the native SAT. To our best knowledge, this first quantum XOR SAT design was proposed using Grover's search algorithm. For instance, for the POS SAT function $f(a, b) = (a + b)(\bar{a} + \bar{b})(\bar{a} + b)(a + \bar{b})$ to build a quantum oracle, we need first to convert the OR terms into Products using De Morgan's Law. On the other hand, a XOR-SAT function $f(a, b, c) = (a\bar{b} \oplus \bar{a}c)(\bar{a}bc \oplus \bar{b}\bar{c})$ there is no need De Morgan's law because each clause is AND formula of ESOP, which is simple to build the circuit with corresponding Toffoli gates. We use a quantum counter to count the number of satisfied clauses. We present a uniform novel quantum oracle design to handle CNF-SAT, XOR-SAT, and CNF-XOR SAT. We assign a quantum counter block for each clause, and the output of each clause is used as the first qubit of the quantum counter.

4.8.4 Quantum Oracle for XOR-SAT

Suppose we have a Boolean function as of XOR SAT, a series of XOR clauses that are a conjunction (and) of a set of clauses, where each clause is an exclusive-or disjunction (xor) of n variables. For instance, given a product of ESOPs function $f(a, b, c) = (a\bar{b} \oplus \bar{a}c)(\bar{a}bc \oplus \bar{b}\bar{c})$ we want to solve MAX-XOR SAT such that to find the values of the Boolean variables for all possible solutions that make the function $f(a, b, c)$ true. To build the quantum oracle, we do not need to convert into Products using De Morgan's

Law since each product of the variables can be represented as Toffoli gate. Each ESOP clause needs one ancilla qubit to save the result, and then all products of ESOP clauses need one Toffoli gate. Then we set the input qubits back to the original by applying Toffoli and NOT gates when appropriate, as can be seen in Figure 4.26.

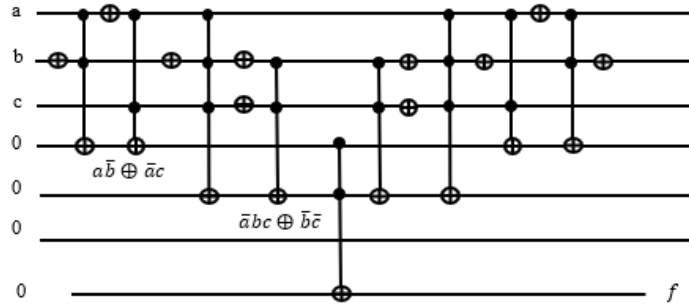


Figure 4.26 Classical oracle design for XOR SAT function $f = (a\bar{b} \oplus \bar{a}c)(\bar{a}bc \oplus \bar{b}\bar{c})$.

If we have many clauses, we will need many ancilla qubits, as many as clauses in function, and then one ancilla qubit for all products of ESOP. We propose a quantum counter block that each clause is connected to the first qubit of the counter block, which counts the number of clauses such that if a clause is true, then the counter is added by 1 to the previous accumulated value. The output of the counter is connected to out_0 . If all clauses are satisfied, then the output of the quantum counter is equal to the number of clauses in the given function.

In Figure 4.27, if the two clauses are satisfied, the binary output of the counter circuit of c_2c_1 is 10. We need the last qubit with out_0 ancilla bit to produce 1 when all clauses are satisfied for Grover's algorithm. We need to add a NOT gate in the output circuit and

mirror it, which makes the last qubit out_0 to produce 1 if the Boolean function f is satisfied. Then the oracle recognized the solution.

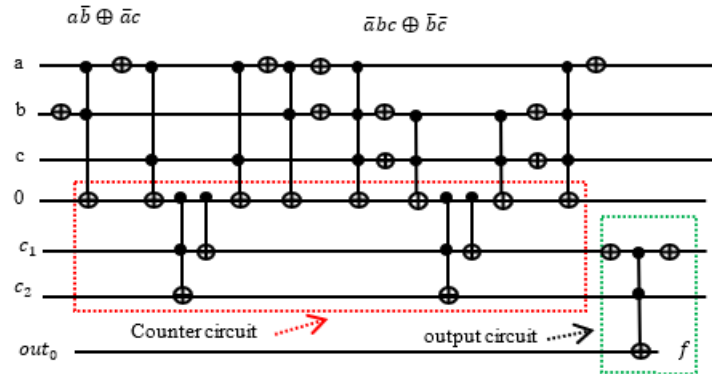


Figure 4.27 Quantum oracle for XOR SAT $f = (a\bar{b} \oplus \bar{a}c)(\bar{a}bc \oplus \bar{b}\bar{c})$ with quantum counter block.

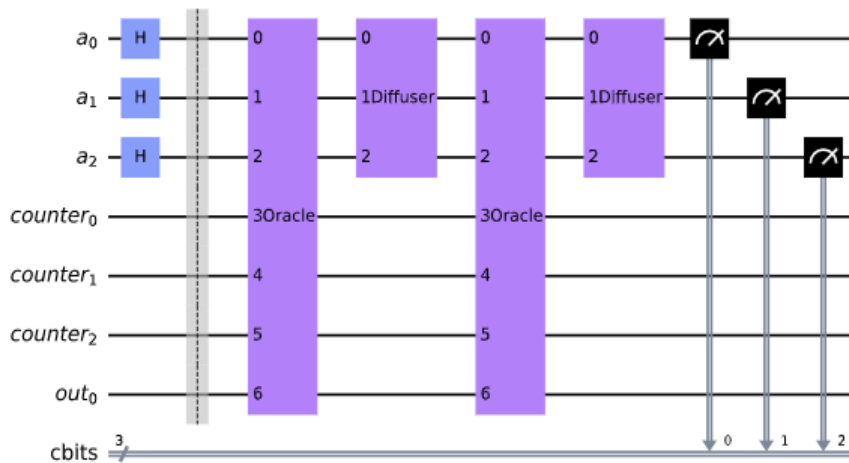


Figure 4.28 Grover's algorithm applied XOR SAT oracle in Figure 4.27

In Figure 4.28, we applied the oracle circuit in Figure 4.27 in Grover's search

algorithm for iterations $R = 2$ from this formula: $R \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil$ where $N = 2^3 = 8$ is the

number of all search space elements since there are 3 variables for a, b, c . $M = 2$ is the

number of solutions. In general, the value of M is calculated using the Quantum Counting

algorithm [48], but for unsolvable problems, the value of M is taken as 1 to run the Grover iterations R . We run the circuit on the ‘qasm_simulator’ from QISKIT for 1024 shots (independent runs to get high precision probability) and the circuit produces the correct answers (verified in Table 4.5). We measured a_0 , a_1 , and a_2 in Figure 4.28 where a_0 , a_1 , a_2 correspond to the Boolean variables c , b , a respectively in Figure 4.27.

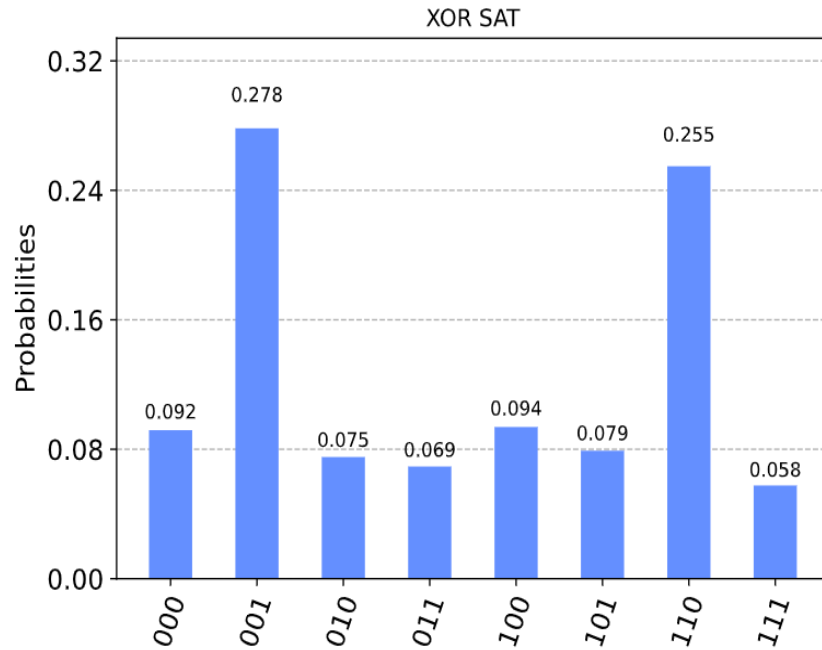


Figure 4.29 Measurement of the Boolean variables from Figure 4.28

As can be seen in Figure 4.29, the diagram illustrates the QISKIT [125] output graphics for the simulated circuit. The measured values $a_2a_1a_0$ with high probability are 110, 001. These are the only two values $cba = 110$ and $cba = 001$ for which the Boolean function $f(a, b, c) = (a\bar{b} \oplus \bar{a}c)(\bar{a}bc \oplus \bar{b}\bar{c})$ is satisfied. The two values of the solution can be verified without using Grover’s search algorithm, as can be seen Table 4.5.

Table 4.5 Truth table for XOR SAT function $f = (a\bar{b} \oplus \bar{a}c)(\bar{a}bc \oplus \bar{b}\bar{c})$

| cba | $a\bar{b} \oplus \bar{a}c$ | $\bar{a}bc \oplus \bar{b}\bar{c}$ | $f = (a\bar{b} \oplus \bar{a}c)(\bar{a}bc \oplus \bar{b}\bar{c})$ |
|-------|----------------------------|-----------------------------------|---|
| 000 | $0 \oplus 0 = 0$ | $0 \oplus 1 = 1$ | 0 |
| 001 | $1 \oplus 0 = 1$ | $0 \oplus 1 = 1$ | 1 |
| 010 | $0 \oplus 0 = 0$ | $0 \oplus 0 = 0$ | 0 |
| 011 | $0 \oplus 0 = 0$ | $0 \oplus 0 = 0$ | 0 |
| 100 | $0 \oplus 1 = 1$ | $0 \oplus 0 = 0$ | 0 |
| 101 | $1 \oplus 0 = 1$ | $0 \oplus 0 = 0$ | 0 |
| 110 | $0 \oplus 1 = 1$ | $1 \oplus 0 = 1$ | 1 |
| 111 | $0 \oplus 0 = 0$ | $0 \oplus 0 = 1$ | 0 |

4.8.5 Quantum Oracle for CNF-XOR SAT

Suppose we have mixed CNF clauses and XOR clauses $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)(a\bar{b} \oplus \bar{a}c)(\bar{a}bc \oplus \bar{b}\bar{c})$. As we solved the previous example in Quantum CNF SAT Section, the POS clauses convert to the Sum expressions into Products using De Morgan's Law. On the other hand, the XOR clauses are used without any conversion.

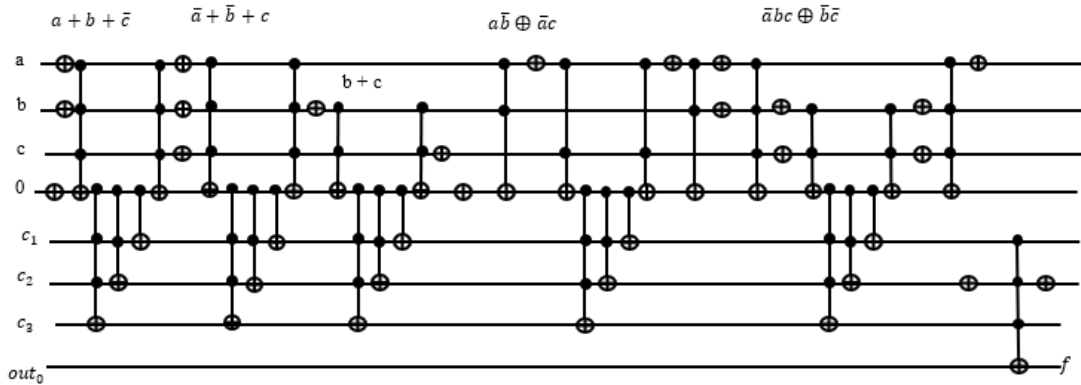


Figure 4.30 Quantum oracle for CNF-XOR SAT function $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)(a\bar{b} \oplus \bar{a}c)(\bar{a}bc \oplus \bar{b}\bar{c})$.

In Figure 4.30, if the five clauses are satisfied, the binary output of the counter circuit of $c_3c_2c_1$ is 101. We need the last qubit with out_0 ancilla bit to produce 1 when all clauses are satisfied for Grover's algorithm. We need to add a NOT gate in the output

circuit and mirror it, which makes the last qubit out_0 to produce one if the Boolean function f is satisfied.

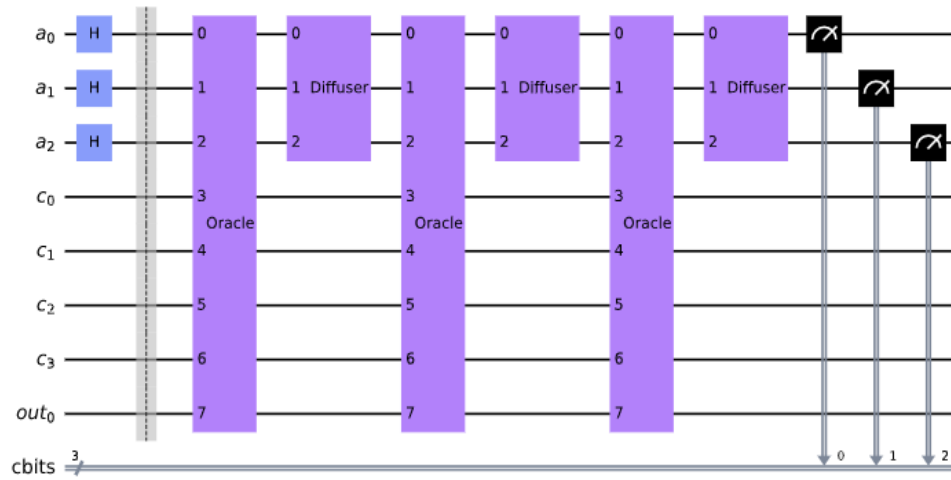


Figure 4.31 Grover's algorithm applied CNF-XOR SAT oracle from Figure 4.30

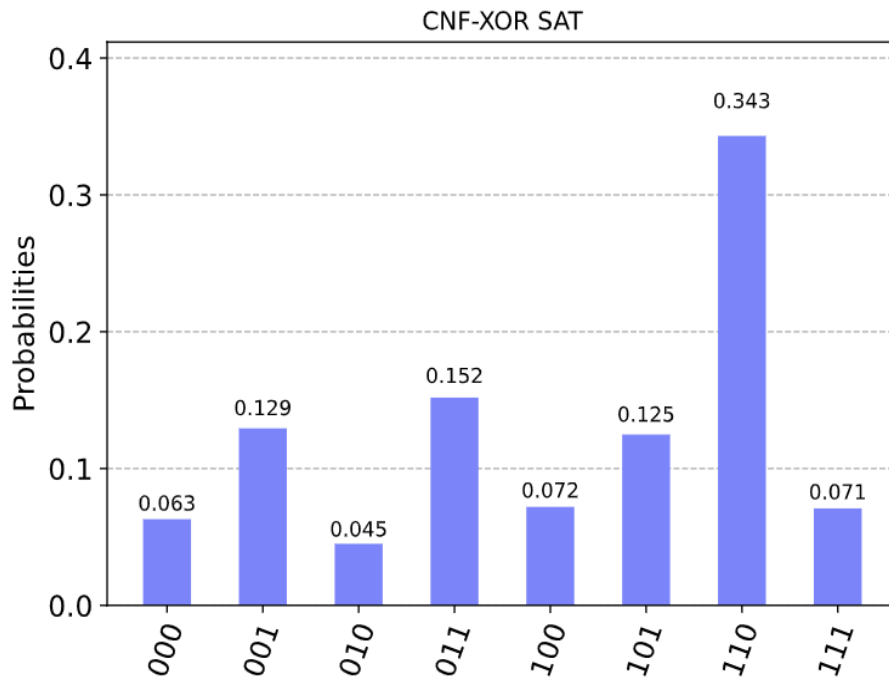


Figure 4.32 Measurement of the Boolean variables of function from Figure 4.31.

In Figure 4.31, we applied the oracle circuit in Figure 4.30 in Grover's search algorithm for iterations $R = 3$ from this formula: $R \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil$ where $N = 2^3 = 8$ is the number of all search space elements since there are 3 variables for a, b, c . $M = 1$ is the number of solutions. As we found in the previous two examples, XOR SAT example gives two solutions 110 and 001 for $a_2 a_1 a_0$ and CNF SAT example gives 010, 101, 110, 111 for $a_2 a_1 a_0$. So, we built the oracle for CNF-XOR SAT from those examples and gives the command value 110, which satisfies for $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)(a\bar{b} \oplus \bar{a}c)(\bar{a}bc \oplus \bar{b}\bar{c})$. We run the circuit on the 'qasm_simulator'. We measured a_0, a_1 , and a_2 in Figure 4.31 where a_0, a_1, a_2 correspond to the Boolean variables a, b, c respectively. As can be seen in Figure 4.32, the diagram illustrates the QISKIT output graphics for the simulated circuit. The measured values $a_2 a_1 a_0$ with high probability is 110. This is the only value $cba = 110$ which the Boolean function $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)(a\bar{b} \oplus \bar{a}c)(\bar{a}bc \oplus \bar{b}\bar{c})$ is satisfied.

All these different propositional formulas with one design without factoring XOR clauses into CNF clauses show promising performance improvement that quantum SAT solvers can achieve, especially in post-quantum cryptography. Comparatively, the classical XOR SAT and CNF-XOR SAT are required to factor XOR formulas into CNF formulas which can take a substantial amount of memory and time.

5 QUANTUM ALGORITHM FOR MINING FREQUENT PATTERNS FOR ASSOCIATION RULE MINING

Chapter 5

Quantum Algorithm for Mining Frequent Patterns for Association Rule Mining

Alasow, Abdirahman, and Marek Perkowski. "Quantum Algorithm for Mining Frequent Patterns for Association Rule Mining." *Journal of Quantum Information Science* 13, no. 1 (2023): 1-23. <https://doi.org/10.4236/jqis.2023.131001>

Authors: Abdirahman Alasow, Marek Perkowski

Abdirahman Alasow:

Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Writing—original draft

Marek Perkowski:

Conceptualization, Methodology, Supervision, Visualization, Writing—review & editing.

Doi: 10.4236/jqis.2023.131001

https://pdxscholar.library.pdx.edu/ece_fac/767/

Maximum frequent pattern generation from a large database of transactions and items for association rule mining is an important research topic in data mining. Association rule mining aims to discover interesting correlations, frequent patterns, associations, or causal structures between items hidden in a large database. By exploiting quantum computing, we propose an efficient quantum search algorithm design to discover the maximum frequent patterns. We modified Grover's search algorithm so that a subspace of arbitrary symmetric states is used instead of the whole search space. We presented a novel quantum oracle design that employs a quantum counter to count the maximum frequent items and a quantum comparator to check with a minimum support threshold. The proposed derived algorithm increases the rate of the correct solutions since the search is only in a subspace. Furthermore, our algorithm significantly scales and optimizes the required number of qubits in design, which directly reflected positively on the performance. Our proposed design can accommodate more transactions and items and still have a good performance with a small number of qubits.

5.1 Introduction

We live in a digitalized era where vast amounts of data are collected daily. Data mining is a process of extracting interesting information, which is called knowledge discovery (KDD) in a database. According to [163], data mining is a field related to machine learning, but data mining is more applied than machine learning. Data mining has a broader spectrum of applications in engineering, science, business, medical applications, and many other areas. Data mining is used to process raw data on a larger scale of data. There are various types of data mining techniques, such as association rules,

classification, and clustering [164]. Association rule mining (ARM) is one of the most important research topics in data mining. ARM aims to discover interesting correlations, frequent patterns, associations, or causal structures between items hidden in a large database. Association rule mining is a branch of unsupervised learning processes that discover hidden patterns in data in the form of easily recognizable rules. Association rule mining is often termed as market basket analysis which studies the buying behaviors of customers by searching for sets of items that are frequently purchased together. Association rule mining is widely used in the retail analysis of transactions [165, 166], recommendation engines [167, 168, 169], web mining [170, 171], medical diagnosis, bioinformatics [172], and other applications [173] in various areas.

- Retail analysis of transactions: The data from past transactions can be used to generate items that the customers most like to be purchased together. The retailer can then adjust the store layout, sales strategy, bundling prices, and inventory control to take advantage of the extracted rules that are generated from association rule mining.
- Recommendation engines: Recommendation systems such as entertainment, news, and social media can be designed using association rule mining to recommend the most interesting content based on the user's past behavior.
- Web mining: Web usage mining is used in e-commerce applications to get useful information about the behavior of customers. Association rule mining is applied to the data from past behavior of the customers, and then rules based on customer

preferences are generated. The developers can optimize and improve websites to personalize the web portals based on the association rules.

- **Medical diagnosis:** Association rule mining can be used to help diagnose patients. The symptoms of diseases and illnesses can be identified to find the conditional probability of the occurrence of illness using associate rule mining.
- **Bioinformatics:** There are many applications in bioinformatics problems, such as protein interaction networks, gene expression data, and others [172], that can be applied to association rule mining to identify biologically relevant patterns. These patterns can be translated into a biological context.

Association rule mining increases revenue by ensuring customer satisfaction based on customized web portals. as well as enhances medical treatment by relating the severity of the sickness and its symptoms.

Given a set of transactions in a database, the goal is to find the association rules that connect between itemsets. For example, $X \Rightarrow Y$ (X implies Y), which means the customer who buys the items in X also tends to buy the items in Y . The implication means the co-occurrence of X and Y items. If patterns within transaction data tell us that baby formula and diapers are usually purchased together in the same transaction, a retailer can take advantage of this association for bundle pricing, product placement, and even shelf space optimization within the store layout. Association rules are considered interesting and called strong if they satisfy the user-predefined thresholds, which are minimum support

(*minsup*) and minimum confidence (*minconf*) thresholds. These threshold values are pre-defined by users or domain experts.

The original association rule mining problem was first introduced in [174]. Let $I = \{I_1, I_2, \dots, I_M\}$ be the set of all items. Let T be a set of database transactions where $T = \{T_1, T_2, \dots, T_N\}$ and $T \subseteq I$. Each transaction is associated with an identifier, called a *TID*. A set of items is referred to as an itemset. An itemset that contains k items is called a k -itemset. An k -itemsets that occur frequently are called the frequent k -itemsets.

Association rules of the form $X \Rightarrow Y$ (X implies Y) are measured by the so-called support, which is the percentage of transactions that contain both X and Y , the union of itemsets X and Y . The support is taken to be the probability, $P(X \cup Y)$.

$$\text{support}(X \Rightarrow Y) = P(X \cup Y) = \frac{\text{number of transactions containing both } X \text{ and } Y}{\text{Total number of transactions}}$$

$$\text{support}(X) = \frac{\text{number of transactions containing } X}{\text{Total number of transactions}}$$

The support in the above equation is called *relative support*, where the frequency or occurrence of an item is called an *absolute support* or a *support count*. Another objective measure for association rules is *confidence*, which assesses the degree of certainty of the detected association [175]. The rule $X \Rightarrow Y$ has confidence, the percentage of transactions that containing X also contains Y . The confidence is taken to be the conditional probability, $P(Y|X)$.

$$\begin{aligned} \text{confidence } (X \Rightarrow Y) &= P(Y|X) = \frac{\text{number of transactions containing } X \text{ and } Y}{\text{number of transaction containing } X} \\ &= \frac{\text{support } (X \cup Y)}{\text{support } (X)} \end{aligned}$$

The main objective of ARM is to discover the itemsets that frequently appear in the transactions. The support (occurrence frequency) for each of these itemsets is generated from a number of candidate itemsets and not less than a pre-defined threshold [175, 176]. In Figure 5.1, association rule mining is decomposed into two phases: First, find out all the frequent itemsets such that each of these itemsets will occur at least as frequently as the pre-defined minimum support threshold. Those itemsets are called frequent or large itemsets. Second, generate association rules from those frequent itemsets with the constraints of minimum confidence threshold. There are many algorithms for mining frequent itemsets; the first phase of the association rule mining task. This first phase dominates the complexity of the whole process.

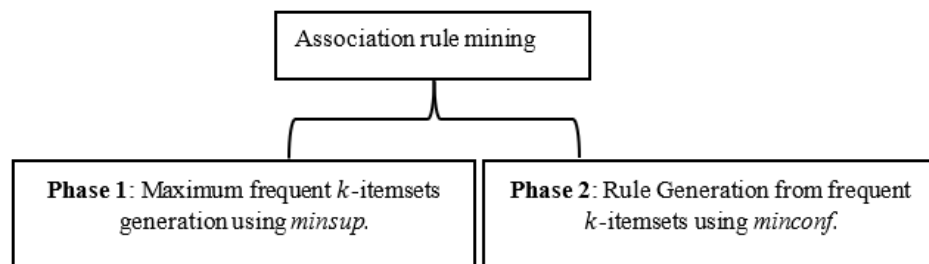


Figure 5.1 Association rule mining phases.

5.2 Related Works

5.2.1 Classical Algorithms for Association Rule Mining

The AIS (Agrawal, Imielinski, Swami) algorithm was the first algorithm proposed for mining association rules in [174]. In the AIS algorithm, the database was scanned many times to get maximum frequent or large itemsets. During the first pass, C_1 , the candidate 1-itemsets are generated, and the support count of each individual item was accumulated. From candidate 1-itemsets, F_1 frequent 1-itemsets are generated by eliminating itemsets whose support count less than the value of *minsup*. Candidate 2-itemsets are generated by extending frequent 1-itemsets with other items in the same transaction. During the second pass over the database, the support counts of those candidate 2-itemsets are accumulated and checked against the *minsup*. Similarly, those candidate $(k+1)$ -itemsets are generated by extending frequent k -itemsets with items in the same transaction. However, extending the itemsets that are not present in the previous pass results in unnecessarily generating and counting too many candidate itemsets that turn out to be small.

A number of ARM algorithms were proposed. Among these, the most well-known algorithm is the Apriori algorithm [176] that makes additional use of prune property to those candidates which have an infrequent subset before counting their supports. This optimization is possible because the support values of all subsets of a candidate are known in advance. The Apriori algorithm employs an iterative approach known as level-wise search, where k -itemsets are used to explore $(k+1)$ -itemsets. First, the set of candidate 1-itemsets is found by scanning the database to accumulate the count for each item and collecting those items that satisfy the minimum support threshold. The resulting

set is denoted by F_1 , the set of frequent 1-itemsets. Next, F_1 is used to find C_2 , the set of candidate 2-itemsets, which is used to find F_2 , and so on, until no more frequent k -itemsets can be found. The finding of each C_k requires one full scan of the database, which becomes the dilemma of performance in ARM when the transaction database is very large.

In the Apriori algorithm, finding of each C_k requires one full scan of the database. To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the Apriori property is used to reduce the search space. Apriori property is:

- If an itemset is frequent, then all its subsets must also be frequent.
- If an itemset is not frequent, then all its supersets cannot be frequent.

To construct candidate C_k combine frequent itemsets of size k . If $k = 1$, take all 1-itemset. If $k > 1$, join pairs of itemset that differ by just one item. For each generated candidate itemset ensure that all subsets of size k are frequent. The first pass of the Apriori algorithm simply counts item occurrences to determine the frequent 1-itemsets. A subsequent pass, say pass k , consists of two phases. First, the frequent itemsets F_{k-1} found in the $k - 1$ pass are used to generate the candidate itemsets C_k , using two step processes: self-join and prune.

- Self-join F_{k-1} : Generate set C_k by joining F_{k-1} itemsets that share the first $k - 1$ items.

- Prune: Remove from C_k the itemsets that contain a subset k -itemset that is not frequent

Second, the database is scanned and the support of candidates in C_k is counted.

Example: Let a store promote certain computer accessories, and the store manager bundles some accessories with other discounted accessories. The manager wants to know which accessories the customers pay for mostly when purchased together. Table 5.1 contains transactions with items.

Table 5.1 Transaction with Items.

| Transaction ID | Computer accessories items |
|----------------|-------------------------------------|
| T_1 | Mouse, Keyboard, Monitor, Headphone |
| T_2 | Webcam, Monitor |
| T_3 | Mouse, Keyboard, Monitor |
| T_4 | Printer |
| T_5 | Mouse, Keyboard, Headphone |
| T_6 | Webcam, Headphone |
| T_7 | Mouse, Monitor, Headphone |
| T_8 | Monitor, Headphone |
| T_9 | Mouse, Keyboard, Monitor |
| T_{10} | Mouse, Webcam, Monitor |

Let $minsup = 2$ and $mincof = 70\%$. We need first to generate a maximum frequent itemset. The transaction database in Table 5.1 is applied in the Apriori algorithm, as can be seen in Figure 5.2. The set of candidate 1-itemsets, C_1 , obtained and then scanned the database to count the support of each itemset. The set of frequent 1-itemsets, F_1 , whose support is equal to or greater than 2, the $minsup$ value, are generated from C_1 . Items in F_1 are joined to get candidate 2-itemsets, C_2 . The database is scanned, and the support of

each candidate itemset in C_2 is counted. The set of frequent 2-itemsets, F_2 , is determined based on the support count of each candidate 2-itemset in C_2 . From F_2 pairs of itemset are joined that differ by just one item to get candidate 3-itemsets, C_3 . The support of each itemset in C_3 is counted by scanning the database and F_3 is generated based on the minsup value. The candidate of 4-itemset, C_4 , is created by joining itemsets that differ by just two items in F_3 . The support count of C_4 itemset is less than the minsup value, so there is no frequent 4-itemsets, F_4 . Thus, in this case the maximum frequent itemset is F_3 . So, the maximum frequent itemset are: {Mouse, Keyboard, Monitor}, {Mouse, Keyboard, Headphone} and {Mouse, Monitor, Headphone} as can be seen in Figure 5.2.

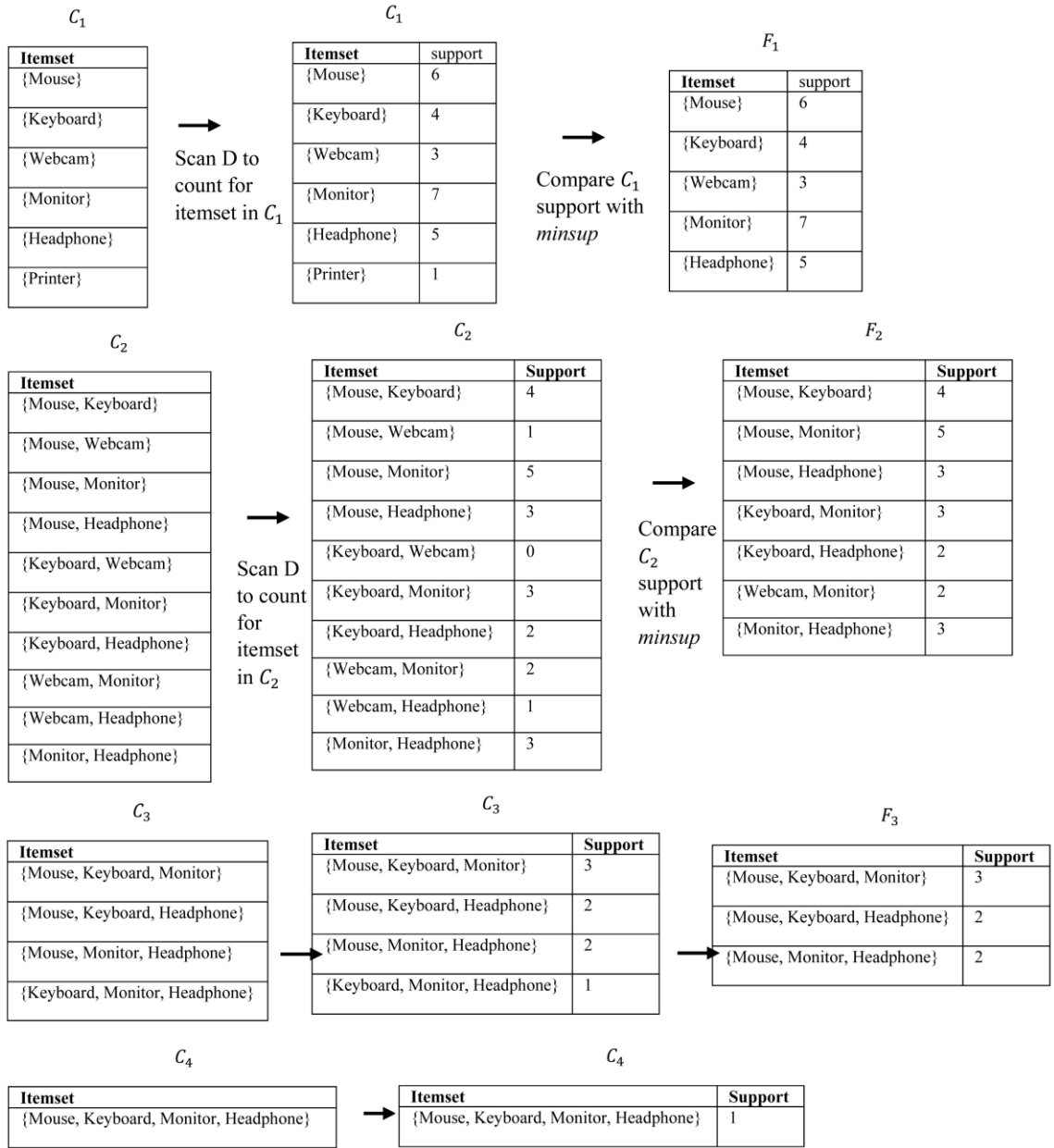


Figure 5.2 Generating maximum frequent itemsets using the Apriori Algorithm.

Let generate association rules based on $minconf = 70\%$ for {Mouse, Keyboard, Monitor}:

Rule 1: {Mouse, Keyboard} \rightarrow {Monitor}

$$\begin{aligned} & \text{confidence} (\{\text{Mouse, Keyboard}\} \rightarrow \{\text{Monitor}\}) \\ &= \frac{\text{support} \{\text{Mouse, Keyboard, Monitor}\}}{\text{support} \{\text{Mouse, Keyboard}\}} = \frac{3}{4} = 75\% \end{aligned}$$

Rule 2: $\{\text{Mouse, Monitor}\} \rightarrow \{\text{Keyboard}\}$

$$\begin{aligned} & \text{confidence} (\{\text{Mouse, Monitor}\} \rightarrow \{\text{Keyboard}\}) \\ &= \frac{\text{support} \{\text{Mouse, Keyboard, Monitor}\}}{\text{support} \{\text{Mouse, Monitor}\}} = \frac{3}{5} = 60\% \end{aligned}$$

Rule 3: $\{\text{Keyboard, Monitor}\} \rightarrow \{\text{Mouse}\}$

$$\begin{aligned} & \text{confidence} (\{\text{Keyboard, Monitor}\} \rightarrow \{\text{Mouse}\}) \\ &= \frac{\text{support} \{\text{Mouse, Keyboard, Monitor}\}}{\text{support} \{\text{Keyboard, Monitor}\}} = \frac{3}{3} = 100\% \end{aligned}$$

Rule 4: $\{\text{Mouse}\} \rightarrow \{\text{Keyboard, Monitor}\}$

$$\begin{aligned} & \text{confidence} (\{\text{Mouse}\} \rightarrow \{\text{Keyboard, Monitor}\}) \\ &= \frac{\text{support} \{\text{Mouse, Keyboard, Monitor}\}}{\text{support} \{\text{Mouse}\}} = \frac{3}{6} = 50\% \end{aligned}$$

Rule 5: $\{\text{Keyboard}\} \rightarrow \{\text{Mouse, Monitor}\}$

$$\begin{aligned} & \text{confidence} (\{\text{Keyboard}\} \rightarrow \{\text{Mouse, Monitor}\}) \\ &= \frac{\text{support} \{\text{Mouse, Keyboard, Monitor}\}}{\text{support} \{\text{Keyboard}\}} = \frac{3}{4} = 75\% \end{aligned}$$

Rule 6: $\{\text{Monitor}\} \rightarrow \{\text{Mouse, Keyboard}\}$

$$\begin{aligned} & \text{confidence} (\{\text{Monitor}\} \rightarrow \{\text{Mouse, Keyboard}\}) \\ &= \frac{\text{support} \{\text{Mouse, Keyboard, Monitor}\}}{\text{support} \{\text{Monitor}\}} = \frac{3}{7} = 42.9\% \end{aligned}$$

If the desired *minconf* is 70%, then all interested rules that satisfy the *minconf* are rules 1, 3, and 5. Based on rule 1, 75% of the customers who purchase {Mouse, Keyboard} also purchase a Monitor. Also, based on rule 3, 100% of the customers who purchase {Keyboard, Monitor} always purchase {Mouse}.

Although research related to the Apriori algorithm has grown in recent years [177], the drawback of the Apriori algorithm is the necessity of scanning the whole database many times. Based on the Apriori algorithm, many new algorithms were designed with some modifications or improvements. Generally, there were two approaches: one is to reduce the number of passes over the whole database or replace the whole database with only part of it based on the current frequent itemsets, and another approach is to explore different kinds of pruning techniques to make the number of candidate itemsets much smaller [178]. Direct hash and pruning (DHP) [179] and partitioning [180] are modifications of the Apriori algorithm. Another mining technique is to simultaneously mine both frequent and infrequent itemsets [181].

Associate rule mining aims to extract interesting correlations from a raw dataset that contains a huge number of database transactions and items in each transaction. Computing such a large scale of raw data in classical computing is very expensive. Leveraging the quantum search algorithm can solve such a problem significantly faster

than the classical algorithm. Grover's search algorithm gives a quadratic speedup compared to an exhaustive classical algorithm for the same problem.

5.2.2 Quantum Algorithms for Association Rule Mining

Quantum algorithms for association rule mining [182, 183] was proposed using a quantum counting algorithm [48]. The quantum circuit design that was proposed in [182, 183] presented experimental implementation for 2×2 (two transactions and two items). The experiment for 2×2 required 9 qubits. Let T , the number of the transaction and I , the number of the items, the required number of qubits is equal to $2TI + 1$ for each iteration to find the maximum frequent k -itemset. Also, for each iteration, the whole database is a search space.

To find the maximum frequent k -itemsets from n items, we are interested only in the subspace of states of Hamming weight k . We propose an efficient method that the search space is reduced from a 2^n -dimensional Hilbert space to an $\binom{n}{k}$ -dimensional subspace. Also, we presented an efficient quantum circuit design that the required number of qubits is reduced significantly compared with [182, 183]. We present a new quantum design method for association rule mining to generate the maximum frequent k -itemsets, which required fewer qubits, and the search space based only on the candidate k -itemset to discover the maximum frequent k -itemsets. We modified Grover's search algorithm [48] by employing the Dicke state [184] to create the superposition for k -itemset, quantum counter [185, 127] to count the frequent of k -itemset and quantum comparator to check the frequent k -itemset equal or greater than to *minsup* threshold. We start to transform the transaction database into a binary matrix such that the item value is '1' if the item is

present in a transaction and ‘0’ otherwise. The database is converted to binary matrix $A_{N \times M}$, where each row corresponds to transaction in T , each column corresponds to an item in the set of all items I , and N is the number of transactions, and M is the number of items. A Boolean function in the sum-of-product (SOP) form is generated from the binary matrix such that each row of the matrix corresponds to one product term of the SOP function expression. In the presented case, the term is a product of variables for all items that have a value of one.

The Apriori algorithm for associate rule mining, the maximum frequent k -itemset is required to scan the database for every 1-itemset, 2-itemset, etc. until to find the maximum k -itemset. The candidate itemset generated during an early iteration are generally of larger magnitude than the maximum frequent k -itemset, which likelihood can be found in later iterations. Therefore, the initial candidate itemset generation is the key issue in improving the performance of association rule mining [179]. In this case, we started the search from the maximum k -itemsets as the candidate to search the maximum frequent k -itemsets, which may be a better chance to be the actual choice, or close to the maximum, frequent k -itemsets rather than starting the search from the candidate 1-itemsets. Thus, the database size and the computation cost are reduced substantially. We choose only all the terms that have the maximum k -itemset from the SOP function. We built the quantum oracle design from the optimized SOP function that contained only the maximum k -itemset combined with the quantum counter and quantum comparator. We run an experiment and perform analysis using QISKIT, an IBM quantum simulator [125].

5.3 Quantum Oracle Design for ARM

First, let us create a binary matrix in Table 5.2 such that the item value is ‘1’ if the item is present in a transaction and ‘0’ otherwise. The binary matrix is based on items from Table 5.1.

Table 5.2 Binary Matrix Corresponds from Table 5.1.

| | a | b | c | d | e | f |
|----------------|-------|----------|--------|---------|-----------|---------|
| Transaction ID | Mouse | Keyboard | Webcam | Monitor | Headphone | Printer |
| T_1 | 1 | 1 | | 1 | 1 | |
| T_2 | | | 1 | 1 | | |
| T_3 | 1 | 1 | | 1 | | |
| T_4 | | | | | | 1 |
| T_5 | 1 | 1 | | | 1 | |
| T_6 | | | 1 | | 1 | |
| T_7 | 1 | | | 1 | 1 | |
| T_8 | | | | 1 | 1 | |
| T_9 | 1 | 1 | | 1 | | |
| T_{10} | 1 | | 1 | 1 | | |

To simplify, we change each item’s name to a letter so that we can build the SOP function. Let us observe that the SOP expressions used here are different from those applied in binary circuit synthesis. This is because here we can use repeated products or products included in other products, which can’t happen in SOP expressions used for binary circuit synthesis. Therefore, our expression compiled from Table 5.2 is as follows:

$$abde + cd + abd + f + abe + ce + ade + de + abd + acd$$

Please note that the above formula mixes Boolean notation and arithmetic addition operation. Let $minsup = 2$ and we need to find the maximum frequent 3-itemset. First, extract all itemsets equal to or greater than the 3-itemset.

$$abde + abd + abe + ade + abd + acd$$

Second, we decompose any itemset that is greater than 3-itemset. In this case, we have $abde$ which is decomposed into $abd + abe + ade + bde$ such that:

$$abd + abe + ade + bde + abd + abe + ade + abd + acd$$

Third, we build the quantum oracle for SOP function expression:

$$abd + abe + ade + bde + abd + abe + ade + abd + acd \geq 2 \quad (5.1)$$

Each term in the traditional quantum oracle design for the SOP function is a Toffoli gate, and the outcome is stored in an additional ancilla qubit. One large Toffoli gate that is controlled for all results is used to compute the output of the SOP function. The problem with the traditional quantum oracle architecture is that one additional ancilla qubit is needed for each term. We proposed that each term of the SOP function be connected to a quantum counter and then mirror the term back before computing the next term. In [175, 176], more information on this design is provided. Please observe that the presented above use of SOP formula is innovative and different than that used in logic function minimization. For instance, term abd is repeated several times which would be a nonsense in circuit design. Our goal here is however the counting of product terms and not minimization. This design trick for quantum oracles can be also used in other algorithms.

Grover's search algorithm is started to create the superposition by using Hadamard operator H . Hadamard operator of n qubits create $N = 2^n$ quantum states. Hadamard operator of $H^{\otimes n}$ applied $|0\rangle^{\otimes n}$:

$$H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \quad H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H^{\otimes 4}|0\rangle^{\otimes 4} = \frac{1}{\sqrt{16}}(0000 + 0001 + \dots + 1111)$$

For maximum frequent k -itemsets, all N states are not solutions, but the solution would be only in Hamming weight k -itemsets. Hamming weight in binary is the number of ones in the binary number. k -itemsets is Hamming weight of k -itemset. For instance, finding the maximum frequent 2-itemset from 4 items, the solution will be only in 6 states with Hamming weight 2 which are equal to 0011, 0101, 0110, 1001, 1010, 1100. So, using the Hadamard operator for four items creates 16 quantum states (0000, 0001, ..., 1111) which are not required to search from the possible solutions, but the possible solutions are only in 6 quantum states of Hamming weight 2.

5.3.1 Dicke States

The search space of search algorithms has a critical role in terms of performance in both classical and quantum computing. In Grover's search algorithm starts preparing superposition states for the search problems. The Hadamard operator is the traditional way to create 2^n superposition states for n qubits. There are many useful concepts that

can be used in real problems. These representation concepts include symmetric Boolean function [186], Johnson graph [187], and frequent patterns for associate rule mining that the possible solutions are in the form of Hamming weight, a pure symmetric state in $\binom{n}{k}$ states. For problems such as these, a proper quantum superposition state can be achieved using the Dicke state [184]. Finding the maximum frequent k -itemsets from n items, we are interested only in the subspace of states of Hamming weight k . The Dicke state $|D_k^n\rangle$ is an equal-weight superposition of all n -qubit states with Hamming Weight k (i.e., all strings of length n with exactly k ones over a binary). $|D_k^n\rangle$ creates $\binom{n}{k}$ symmetric states. Below we illustrate practical examples of the Dicke state.

$$|D_k^n\rangle = \binom{n}{k}^{-\frac{1}{2}} \sum_{x \in \{0,1\}^n, wt(x)=k} |X\rangle$$

Where $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ and $0 < k < n$. There are n qubits and $n - k$ of them are 0 and k are 1. For instance,

$$\begin{aligned} |D_2^4\rangle &= \frac{1}{\sqrt{6}} \sum_{x \in \{0,1\}^4, wt(x)=2} |X\rangle \\ &= \frac{1}{\sqrt{6}} (|0011\rangle + |0101\rangle + |0110\rangle + |1001\rangle + |1010\rangle + |1100\rangle) \end{aligned}$$

$|D_2^4\rangle$ is Dicke state of 4 qubits that has 6 symmetric states, and each state has 2 ones.

$$\begin{aligned} |D_3^5\rangle &= \sqrt{\frac{1}{10}} \{ |00111\rangle + |01011\rangle + |10011\rangle + |01101\rangle + |10101\rangle + |11001\rangle \\ &\quad + |01110\rangle + |10110\rangle + |11010\rangle + |11100\rangle \} \end{aligned}$$

$|D_3^5\rangle$ is Dicke state of 5 qubits that has 10 symmetric states, and each state has 3 ones.

Dicke state creates arbitrary symmetric pure states $|0\rangle^{\otimes n-k}|1\rangle^{\otimes k}$. Using the Dicke state increases the rate of the correct solutions since the search is only in subspace. The classical bitstring can be converted to the Dicke state by a recursive unitary operation called Split & Cyclic Shift (SCS) unitary $SCS_{n,k}$ [184]. To build $|D_k^n\rangle$ state, we start from $SCS_{n,k}$ where the original sequence is multiplied by a factor of $\sqrt{\frac{k}{n}}$ and then shift the first zero to the end and shift the whole sequence forward by multiplying by a factor of $\sqrt{\frac{n-k}{n}}$.

$$SCS_{n,k}: |0\rangle^{\otimes n-k}|1\rangle^{\otimes k} \rightarrow \sqrt{\frac{k}{n}}|0\rangle^{\otimes n-k}|1\rangle^{\otimes k} + \sqrt{\frac{n-k}{n}}|0\rangle^{\otimes n-k-1}|1\rangle^{\otimes k}|0\rangle$$

Dicke state given as input string $|0\rangle^{\otimes n-k}|1\rangle^{\otimes k}$ generates the entire Dicke states $|D_k^n\rangle$ by recursive unitary operation $SCS_{n,k}$.

$$SCS_{5,3}: |00111\rangle \rightarrow \sqrt{\frac{3}{5}}|00111\rangle + \sqrt{\frac{2}{5}}|01110\rangle$$

$$\sqrt{\frac{3}{5}}|00111\rangle \rightarrow \sqrt{\frac{3}{5}}\{|0011\rangle\} \otimes |1\rangle$$

$$\rightarrow \sqrt{\frac{3}{5}}\left\{\sqrt{\frac{2}{4}}|0011\rangle + \sqrt{\frac{2}{4}}|0110\rangle\right\} \otimes |1\rangle$$

$$\rightarrow \sqrt{\frac{3}{5}}\left\{\sqrt{\frac{2}{4}}\left[\sqrt{\frac{1}{3}}|001\rangle + \sqrt{\frac{2}{3}}|010\rangle\right] \otimes |1\rangle + \sqrt{\frac{2}{4}}\left[\sqrt{\frac{2}{3}}|011\rangle + \sqrt{\frac{1}{3}}|110\rangle\right] \otimes |0\rangle\right\} \otimes |1\rangle$$

$$\begin{aligned}
& \rightarrow \sqrt{\frac{3}{5}} \left\{ \sqrt{\frac{2}{4}} \left[\sqrt{\frac{1}{3}} |001\rangle + \sqrt{\frac{2}{3}} \left(\sqrt{\frac{1}{2}} |01\rangle + \sqrt{\frac{1}{2}} |10\rangle \right) \otimes |0\rangle \right] \otimes |1\rangle \right. \\
& \quad \left. + \sqrt{\frac{2}{4}} \left[\sqrt{\frac{2}{3}} \left(\sqrt{\frac{1}{2}} |01\rangle + \sqrt{\frac{1}{2}} |10\rangle \right) \otimes |1\rangle + \sqrt{\frac{1}{3}} |110\rangle \right] \otimes |0\rangle \right\} \otimes |1\rangle \\
& \rightarrow \sqrt{\frac{3}{5}} \left\{ \sqrt{\frac{2}{4}} \left[\sqrt{\frac{1}{3}} |001\rangle + \sqrt{\frac{1}{3}} (|010\rangle + |100\rangle) \right] \otimes |1\rangle \right. \\
& \quad \left. + \sqrt{\frac{2}{4}} \left[\sqrt{\frac{1}{3}} (|011\rangle + |101\rangle) + \sqrt{\frac{1}{3}} |110\rangle \right] \otimes |0\rangle \right\} \otimes |1\rangle \\
& \rightarrow \sqrt{\frac{3}{5}} \left\{ \sqrt{\frac{2}{4}} \left[\sqrt{\frac{1}{3}} |001\rangle + \sqrt{\frac{1}{3}} |010\rangle + \sqrt{\frac{1}{3}} |100\rangle \right] \otimes |1\rangle \right. \\
& \quad \left. + \sqrt{\frac{2}{4}} \left[\sqrt{\frac{1}{3}} (|011\rangle + |101\rangle) + \sqrt{\frac{1}{3}} |110\rangle \right] \otimes |0\rangle \right\} \otimes |1\rangle \\
& \rightarrow \sqrt{\frac{3}{5}} \left\{ \sqrt{\frac{2}{4}} \left[\sqrt{\frac{1}{3}} |0011\rangle + \sqrt{\frac{1}{3}} |0101\rangle + \sqrt{\frac{1}{3}} |1001\rangle \right] \right. \\
& \quad \left. + \sqrt{\frac{2}{4}} \left[\sqrt{\frac{1}{3}} |0110\rangle + \sqrt{\frac{1}{3}} |1010\rangle + \sqrt{\frac{1}{3}} |1100\rangle \right] \right\} \otimes |1\rangle \\
& \rightarrow \sqrt{\frac{1}{10}} \{ |00111\rangle + |01011\rangle + |10011\rangle + |01101\rangle + |10101\rangle + |11001\rangle \}
\end{aligned}$$

$$\begin{aligned}
\sqrt{\frac{2}{5}} |01110\rangle &\rightarrow \sqrt{\frac{2}{5}} \{|0111\rangle\} \otimes |0\rangle \\
&\rightarrow \sqrt{\frac{2}{5}} \left\{ \sqrt{\frac{3}{4}} |0111\rangle + \sqrt{\frac{1}{4}} |1110\rangle \right\} \otimes |0\rangle \\
&\rightarrow \sqrt{\frac{2}{5}} \left\{ \sqrt{\frac{3}{4}} \left[\sqrt{\frac{2}{3}} |011\rangle + \sqrt{\frac{1}{3}} |110\rangle \right] \otimes |1\rangle + \sqrt{\frac{1}{4}} |1110\rangle \right\} \otimes |0\rangle \\
&\rightarrow \sqrt{\frac{2}{5}} \left\{ \sqrt{\frac{3}{4}} \left[\sqrt{\frac{2}{3}} \left(\sqrt{\frac{1}{2}} |01\rangle + \sqrt{\frac{1}{2}} |10\rangle \right) \otimes |1\rangle + \sqrt{\frac{1}{3}} |110\rangle \right] \otimes |1\rangle + \sqrt{\frac{1}{4}} |1110\rangle \right\} \otimes |0\rangle \\
&\rightarrow \sqrt{\frac{2}{5}} \left\{ \sqrt{\frac{3}{4}} \left[\sqrt{\frac{1}{3}} |011\rangle + \sqrt{\frac{1}{3}} |101\rangle + \sqrt{\frac{1}{3}} |110\rangle \right] \otimes |1\rangle + \sqrt{\frac{1}{4}} |1110\rangle \right\} \otimes |0\rangle \\
&\rightarrow \sqrt{\frac{2}{5}} \left\{ \sqrt{\frac{1}{4}} |0111\rangle + \sqrt{\frac{1}{4}} |1011\rangle + \sqrt{\frac{1}{4}} |1101\rangle + \sqrt{\frac{1}{4}} |1110\rangle \right\} \otimes |0\rangle \\
&\rightarrow \sqrt{\frac{1}{10}} \{|01110\rangle + |10110\rangle + |11010\rangle + |11100\rangle\} \\
|00111\rangle &\rightarrow \sqrt{\frac{1}{10}} \{|00111\rangle + |01011\rangle + |10011\rangle + |01101\rangle + |10101\rangle + |11001\rangle \\
&\quad + |01110\rangle + |10110\rangle + |11010\rangle + |11100\rangle\}
\end{aligned}$$

The transformation or mapping of $SCS_{n,k}$ is constructed by 1-controlled and 2-controlled Y -rotation $R_y(2\theta)$ gate between two $CNOT$, where $R_y(2\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$. Dicke state is constructed recursively by smaller $|D_l^n\rangle$, where $l \leq k$.

$$|0\rangle^{\otimes n-k-1} |0\rangle^{\otimes k+1-l} |1\rangle^{\otimes l} \rightarrow \sqrt{\frac{l}{n}} |0\rangle^{\otimes n-k-1} |0\rangle^{\otimes k+1-l} |1\rangle^{\otimes l} + \sqrt{\frac{n-l}{n}} |0\rangle^{\otimes n-k-1} |0\rangle^{\otimes k-l} |1\rangle^{\otimes l} |0\rangle$$

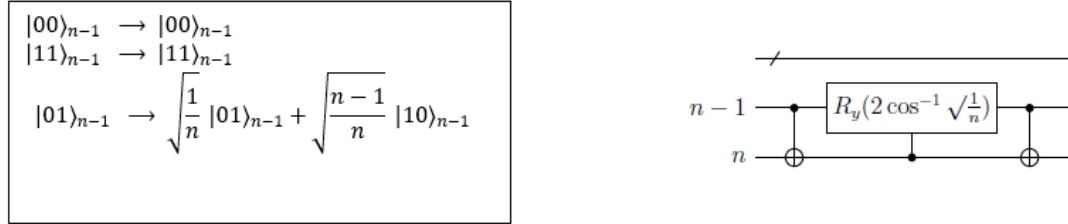


Figure 5.3 Construction of $SCS_{n,l}$ of a two-qubit gate [184]

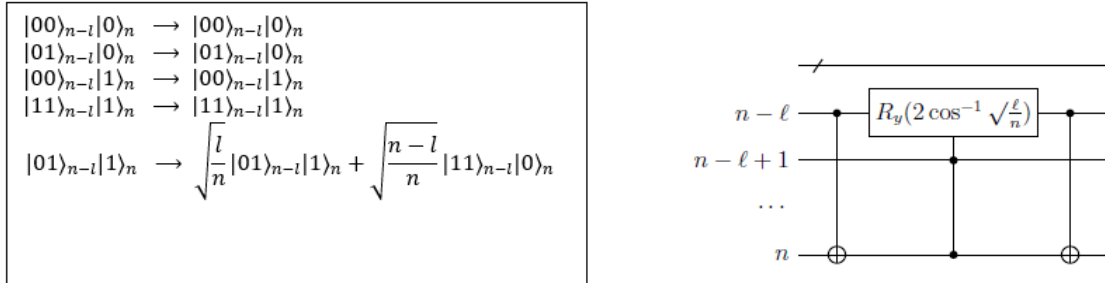


Figure 5.4 Construction of $SCS_{n,l}$ of a three-qubit gate [184]

In the Figure 5.3 and Figure 5.4 presented are the explicit constructions of $SCS_{n,k}$. To learn a more in-depth view of this construction the reader can refer to [184]. The R_y gate is a single qubit rotation gate through angle θ radian. The R_y gate rotation is around Y -axis by angle θ . If the controlled qubits for R_y gate are all active, then the original

sequence is multiplied by a factor of $\sqrt{\frac{l}{n}}$ and then shift the first zero to end and shift the whole sequence forward by multiplying by a factor of $\sqrt{\frac{n-l}{n}}$. If the controlled qubits for R_y gate are not all active, then the two $CNOT$ gates cancel each other.

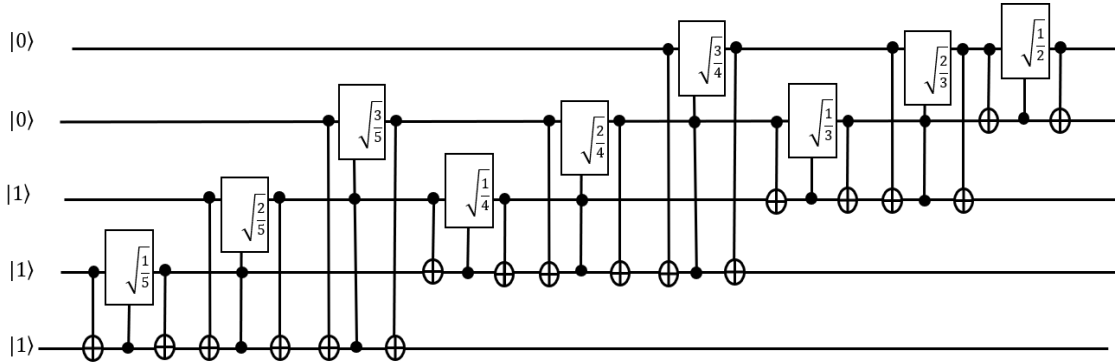


Figure 5.5 Dicke state $|D_3^5\rangle$ circuit, where $\sqrt{\frac{l}{n}}$ gates are shorthand for Y -rotation $R_y\left(2\cos^{-1}\sqrt{\frac{l}{n}}\right)$

[184].

The full circuit of Dicke state $|D_3^5\rangle$ is constructed recursively by $SCS_{n,k}$ as can be seen in Figure 5.5 which contains R_y gate in between two $CNOT$ gates. The R_y gates are controlled either one-qubit or two-qubits.

5.3.2 Quantum Comparator

Let's first discuss the quantum counter that will be connected to the quantum comparator. Since we have nine SOP terms in equation 5.1, we need 4-controlled qubits of a quantum counter, as can be seen in Figure 5.6, which count from zero to 15. The output of the quantum counter will be an input to the quantum comparator. In this problem, the first qubit (1) of the quantum counter will be the SOP term output, so we do not use one as

input to the first qubit of the counter. When the SOP term produces one, then the counter increments by one, and there is no change when the SOP term is zero.

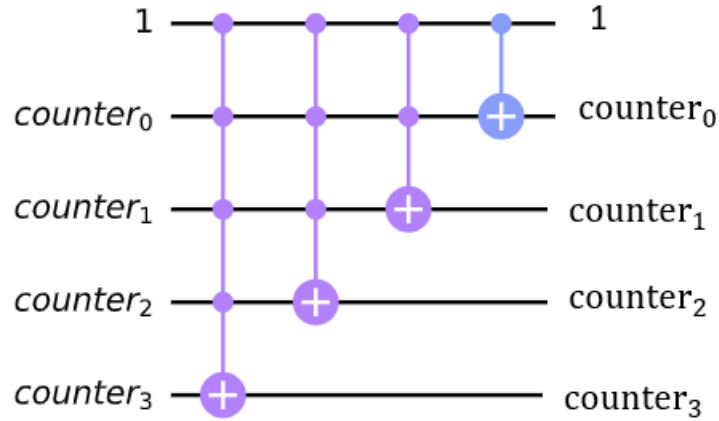


Figure 5.6 Four-controlled qubits of a quantum counter.

In Table 5.3, the truth table of one-bit comparator contains equal, greater than and less than.

Table 5.3 Truth Table of one-bit comparator.

| x_1y_1 | $x_1 = y_1 \Rightarrow \overline{x_1 \oplus y_1}$ | $x_1 > y_1 \Rightarrow x_1\overline{y_1}$ | $x_1 < y_1 \Rightarrow \overline{x_1}y_1$ |
|----------|---|---|---|
| 00 | 1 | 0 | 0 |
| 01 | 0 | 0 | 1 |
| 10 | 0 | 1 | 0 |
| 11 | 1 | 0 | 0 |

For 1-qubit comparator, to check equal ($\overline{x_1 \oplus y_1}$) or greater than ($x_1\overline{y_1}$), we add both values $\overline{x_1 \oplus y_1} + x_1\overline{y_1}$. To build this circuit requires 5 qubits, 2 Toffoli, 4 *NOT*, and 2 *CNOT* gates, as can be seen in Figure 5.7a. For more than 1-qubit comparator, the circuit would be huge in terms of the number of qubits and gates. To minimize the required qubits and gates, we use “not less than” such $\overline{\overline{x_1}y_1}$ which required only 3 qubits,

1 Toffoli, and 2 *NOT* gates as can be seen in Figure 5.7b. The output from the quantum counter would be compared with the *minsup* value. In this case we have a 4-bit quantum counter compared with $minsup = 2$ (0010). Every qubit from the quantum counter output is compared with every bit of the *minsup* value. For instance, let $x_4x_3x_2x_1$ quantum counter output and $y_4y_3y_2y_1$ *minsup* values, we build the quantum circuit for 4-bit comparator $(\overline{x_4y_4})(\overline{x_3y_3})(\overline{x_2y_2})(\overline{x_1y_1})$. In Figure 5.8 for 4-bit quantum comparator, we rename $x_4x_3x_2x_1$ to $counter_3counter_2counter_1counter_0$ and $y_4y_3y_2y_1$ to $comparator_3comparator_2comparator_1comparator_0$. The quantum comparator output is out_0 .

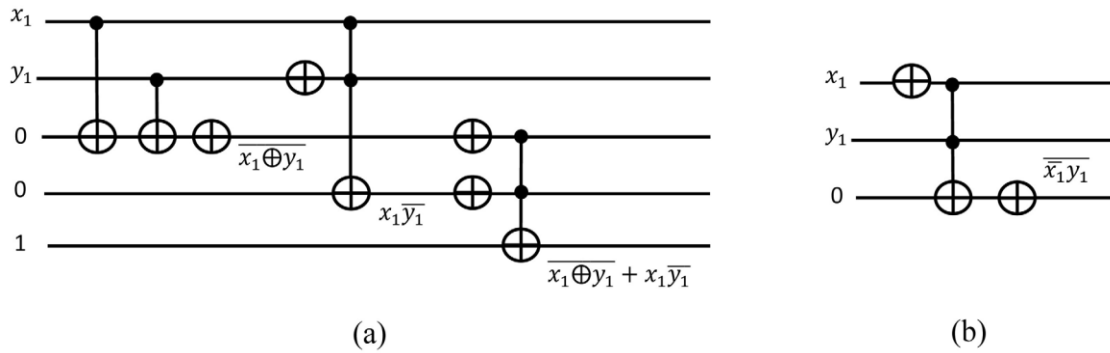


Figure 5.7 One-qubit comparator (a) equal or greater than. (b) not less than

In the Figure 5.8, the four-bit comparator compares the *minsup* value 0010 with the output of the quantum counter. Every n -qubit comparator requires $3n + 1$ qubits. The n -*ancilla_i* qubits are used to store the obtained value from the comparison.

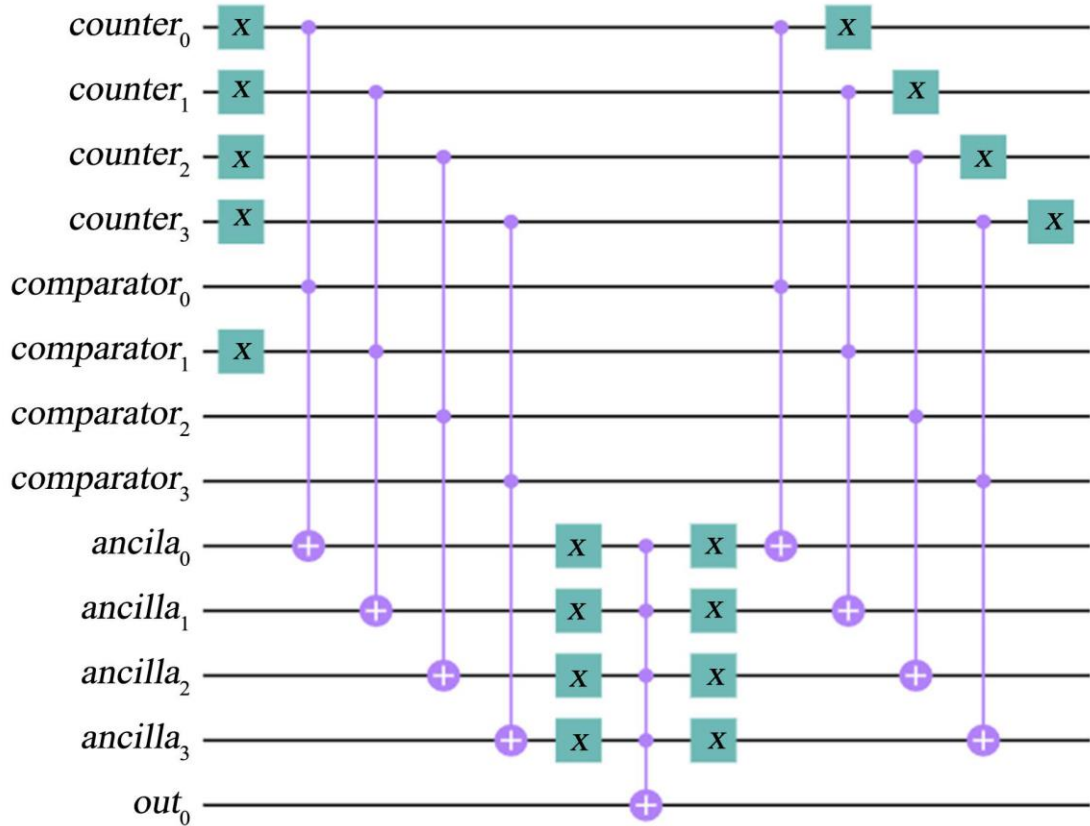


Figure 5.8 Four-bit quantum comparator built using IBM Qiskit simulator.

Figure 5.9 is the complete quantum oracle circuit design for associate rule mining.

The oracle circuit contains the circuit for each term in the SOP function expression connected with the quantum counter (Figure 5.6) for each term and the quantum comparator (Figure 5.8) connected with the quantum counter output. The $q_4q_3q_2q_1q_0$ represent the items and $control_counter_0$ is the control bit for the quantum counter which is output of SOP term. If the SOP term is one, then the counter value is incremented by 1. If the SOP term is zero, then the counter keeps its value. The number of required qubits for the quantum counter is equal to $\lceil \log_2 T \rceil + 1$, where the number of required qubits for the quantum comparator is $3\lceil \log_2 T \rceil + 1$. Note that if $\log_2 T$ is an integer value, then add 1 to the $\log_2 T$ value. The SOP function $abd + abe + ade +$

$bde + abd + abe + ade + abd + acd \geq 2$ contains 9 terms that requires $\lceil \log_2 9 \rceil = 4$ qubits for the quantum counter. The quantum comparator compares the output of the quantum counter 4-qubit with *minsup* value 4-qubit. The quantum comparator required four additional ancilla qubits for computation and one qubit for the output. In associate rule mining, we need the maximum frequent of *k*-itemset that is equal to or greater than to *minsup* = 2. In this case, 4-qubit from the quantum counter compared with 0010. The output is out_0 is equal to 1 when frequent of *k*-itemset equal to or greater than 2.

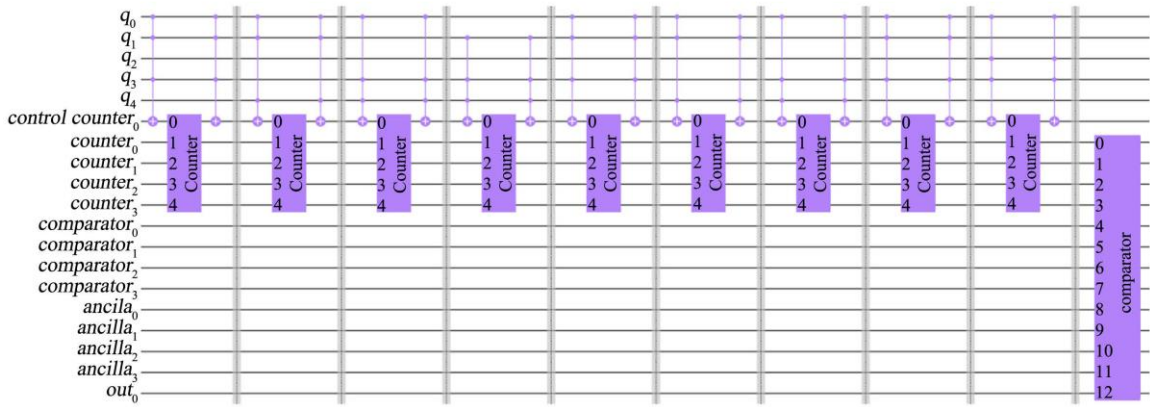


Figure 5.9 Full quantum oracle circuit for $abd + abe + ade + bde + abd + abe + ade + abd + acd \geq 2$

Let us observe that the SOP function above is not used as a logical function, but it is used as a pattern matching for our problem. This general idea can be used for other problems that require counting the matching patterns or counting the satisfied constraints. Some constraint satisfaction problems that can be formulated as such the SOP function above can refer to [126].

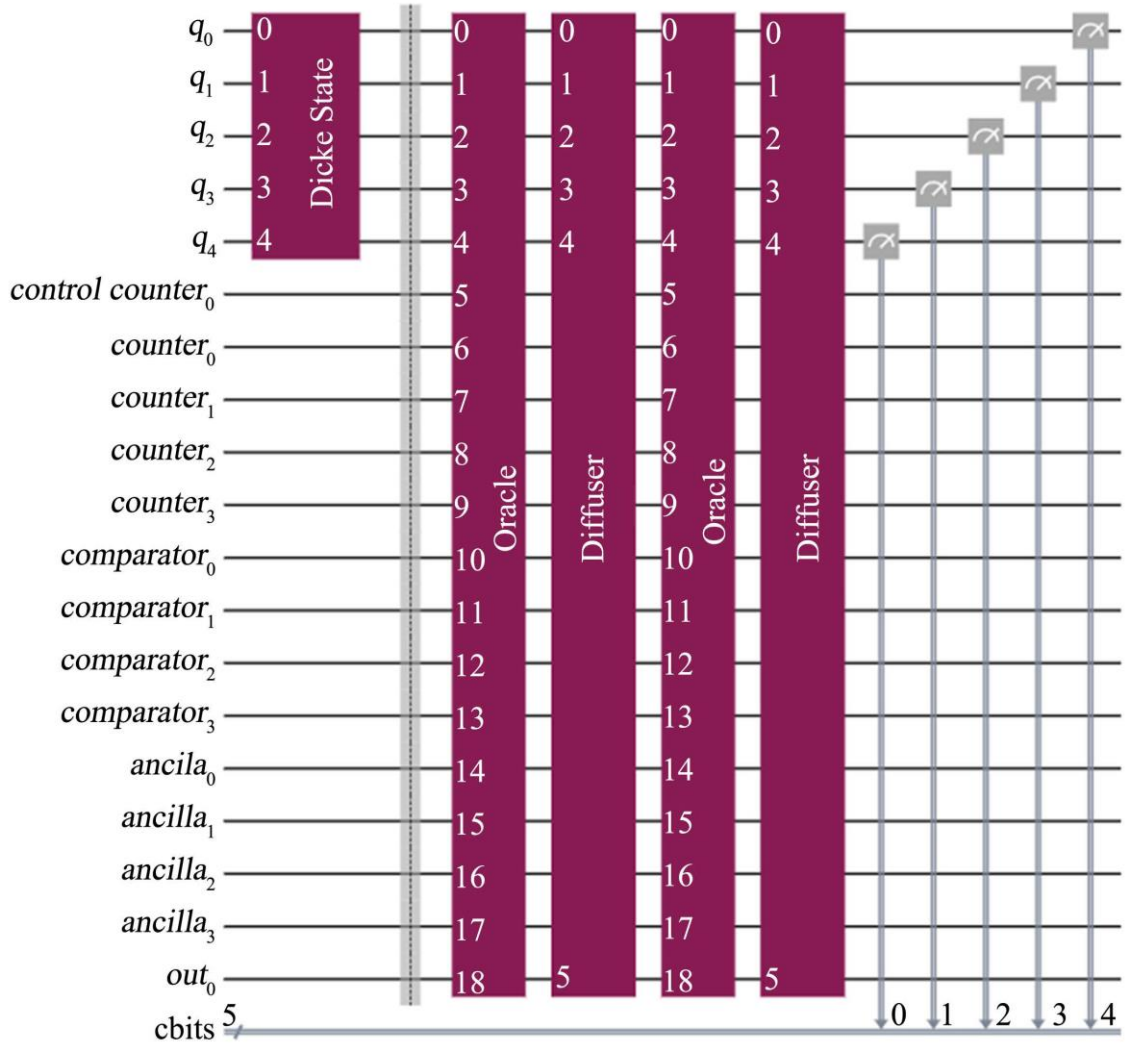


Figure 5.10 Full quantum algorithm circuit design for the associate rule mining using the diffuser in

Figure 3.11

Figure 5.10 is the complete quantum algorithm circuit design for the associate rule mining, which is a modification of Grover's search algorithm. In this simulation, we use the diffuser operator discussed in Figure 3.11. Also, the other diffuser from Figure 3.12 can be used. The Dicke state is used for superposition preparation, oracle, and diffuser to recognize the solution states. We applied this oracle in Grover's search algorithms for

$R = 2$ iterations from this formula: $R \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil$ where $M = 3$ is the number of solutions

in Figure 5.2 in the Apriori algorithm, and $N = 10$ is the number of all search space elements from the Dicke state of $|D_3^5\rangle$. We measured only $q_4 q_3 q_2 q_1 q_0$ for the items but for verification the measurement can be added to out_0 which is equal to 1 only if the k -itemset is greater than or equal to the $minsup$. As can be seen in Figure 5.11 the values with high probability are 10011, 11001 and 11010 for $abcde$, respectively based on this SOP function: $abd + abe + ade + bde + abd + abe + ade + abd + acd \geq 2$

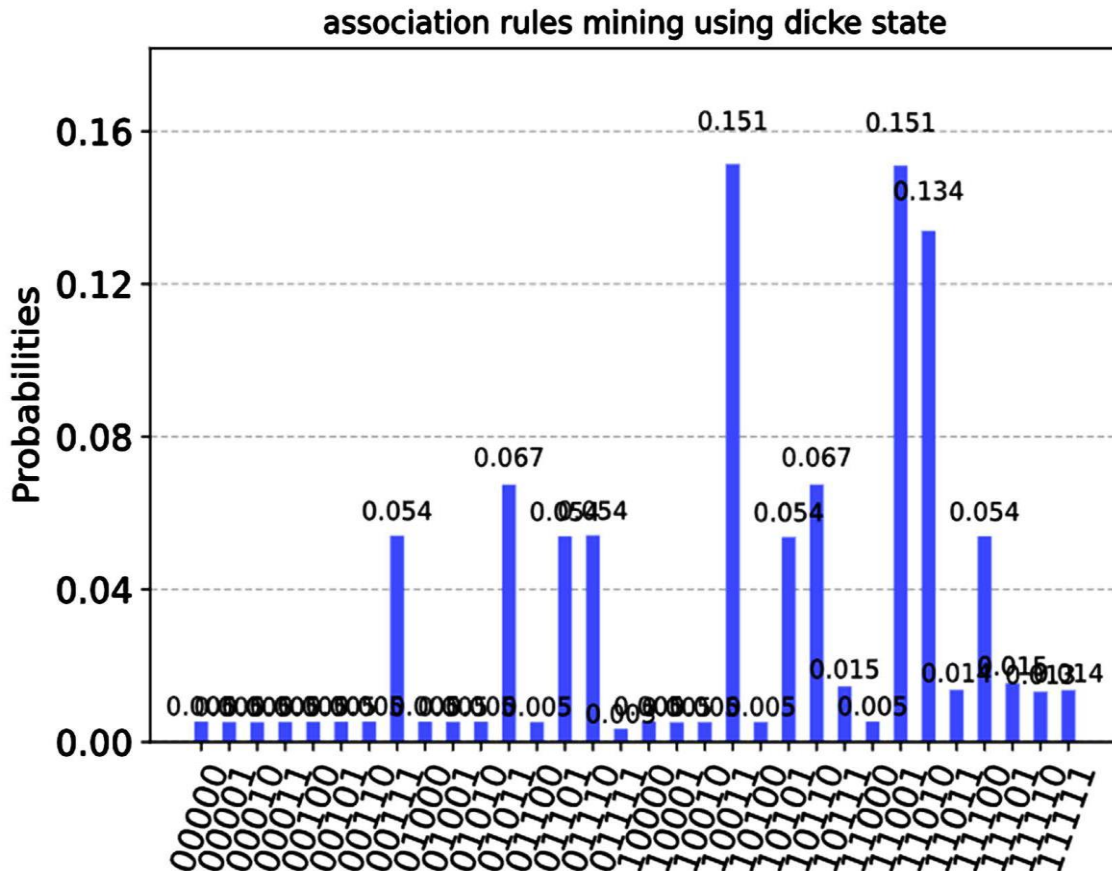


Figure 5.11 Histogram of measured value from the Figure 5.10

In Figure 5.12, we generalized our design to handle the associate rule mining to generate the maximum frequent k -itemset. As can be seen from Figure 5.12, the input of Dicke state is $|0\rangle^{\otimes n-k}|1\rangle^{\otimes k}$, where n is the number of items and k is the large itemset to check whether is the maximum frequent k -itemset or not. Each term from the SOP function is connected to the first qubit (Control Counter) of the quantum counter. The remaining number of qubits for the quantum counter is equal to $\lceil \log_2 T \rceil$, and the number of qubits of the quantum comparator is equal to $3\lceil \log_2 T \rceil + 1$, where T is the number of terms in the SOP function. Note that if the $\log_2 T$ is integer value, then add 1 to the $\log_2 T$ value. The out_0 is connected to the diffuser to amplify the solution. If out_0 is equal to one, then the oracle has recognized the solution. Finally, the Dicke state qubits for the items are measured. For checking purposes, the out_0 can be added to the measurement in order to check the solutions with a high probability that out_0 is equal to 1.

Scaling the required number of qubits in design is critical in the current generation of quantum computers because the number of qubits determines the degree of computational complexity. The more qubits a quantum processor possesses, the more complex and valuable the quantum circuits can be designed and analyzed [188]. Thus, to scale and optimize the number of qubits that can be accommodated in a quantum algorithm design directly reflects on the algorithm performance.

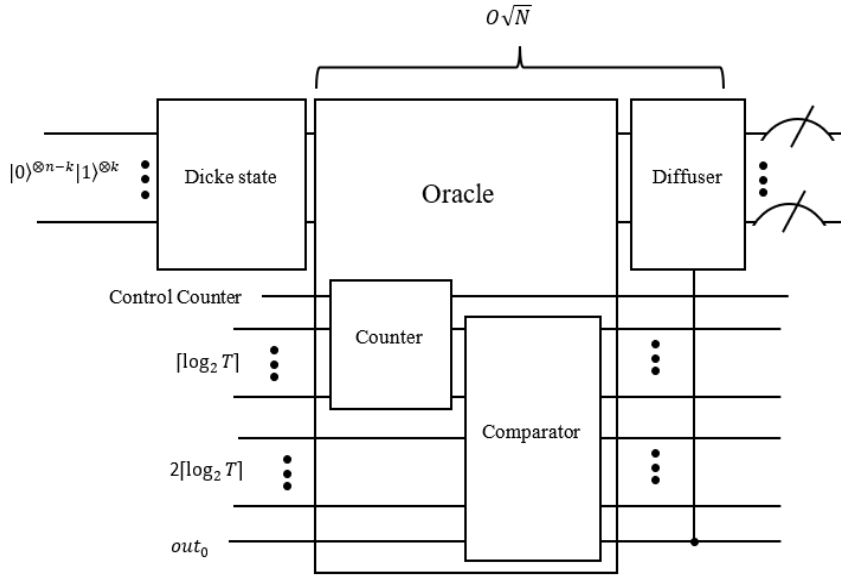


Figure 5.12 Proposed algorithm design for associate rule mining with diffuser in Figure 3.11.

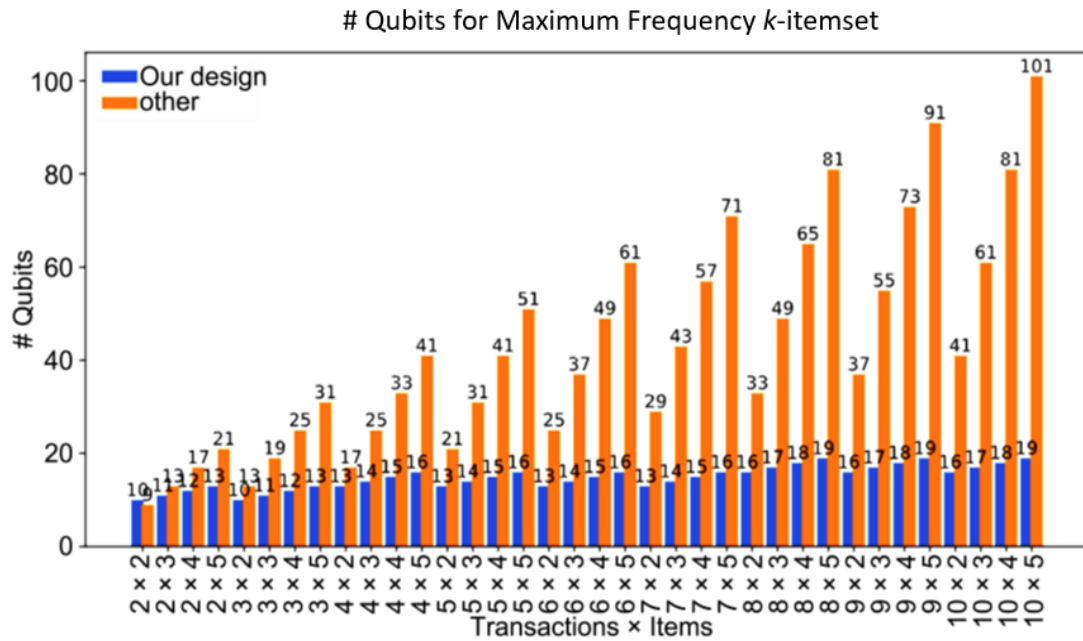


Figure 5.13 Histogram to compare number of qubits in our design (blue) and other designs from [182, 183] (orange).

We compare the number of qubits that are needed in our design and the designs proposed in [182, 183]. The number of required qubits in [182, 183] is equal to:

$$2TI + 1 \tag{5.2}$$

Our proposed design, the number of required qubits is:

$$\begin{cases} I + 3(\log_2 T + 1) + 2, & \text{If } \log_2 T \text{ is integer value} \\ I + 3\lceil \log_2 T \rceil + 2, & \text{If } \log_2 T \text{ not integer} \end{cases} \tag{5.3}$$

where T is the number of transactions and I is the number of items. For instance, 8×5 (8 transactions and 5 items), the number of required qubits in [182, 183] is equal to 81 qubits, while our design requires only 19 qubits.

As can be seen in the histogram from Figure 5.13, based on the equations 5.2 and 5.3, for 10 transactions and 5 items, 10×5 requires 101 qubits for the design in [182, 183], while our proposed design just requires just 19 qubits. According to the design in [182, 183], one qubit is needed for every item, as well as one qubit for every transaction and one qubit for the output qubit. A large transaction database typically contains massive transactions and large item sets. As a result, the number of qubits will be unreasonably high, even for large quantum computers. Our proposed quantum architecture employs the Dicke state, which reduces the search space into a sub-search space. In addition, we employ a quantum counter and a quantum comparator that can handle more transactions and items while still performing well with a small number of qubits.

6 QUANTUM RANDOM WALK

Quantum random walk algorithm is the quantum counterpart of the quantum analogues of classical random walk, aka Markov chains. Grover's search algorithm is a remarkable quantum search algorithm that exhibits a quadratic speedup over its counterpart classical algorithms. However, there are some problems with Grover's search algorithm, which cause in some cases that the Grover's algorithm is not more efficient than certain classical algorithms [191, 192, 194].

6.1 Limitation of Grover's search algorithm

Quantum walk is a quantum search algorithm that overcomes the limitations of Grover's search algorithm. Here, we present some problems that show the limitations of Grover's search algorithm.

6.1.1 Search on Grid

Consider a given N data points of 2-dimensional and arranged by $\sqrt{N} \times \sqrt{N}$ grid. Moving one point to another would take $O(\sqrt{N})$ steps for a local move, and Grover's algorithm performs $O(\sqrt{N})$ iterations. Thus, the search takes $O(\sqrt{N} \times \sqrt{N}) = O(N)$ time. In such a case, there is no quantum speedup because the classical algorithms take similar queries by traveling in the grid row by row for every cell. This limitation of Grover's algorithm was presented for the first time in [191] for a quantum robot. Such a limitation is also found in the d -dimensional $\sqrt[d]{N} \times \sqrt[d]{N} \dots \sqrt[d]{N}$ grid. Grover's search algorithm cannot find the solution in time $O(\sqrt[d]{N})$; instead, the solution would be found in time $O(\sqrt{N} \times \sqrt[d]{N})$. Where $\sqrt[d]{N}$ is for local move and \sqrt{N} for Grover iteration. Suppose N elements are

arranged in a 3-dimensional grid; then Grover's search algorithm would take

$O(\sqrt{N} \times \sqrt[3]{N}) = O\left(\sqrt[5]{6}\sqrt{N}\right)$ queries. This is better than $O(N)$ for the classical algorithm,

but still worse than $O(\sqrt{N})$ for the usual Grover's search algorithm. Quantum walk for

$d = 2$ takes $O(\sqrt{N} \log N)$ time, and $d \geq 3$ takes $O(\sqrt{N})$ time [193, 194].

6.1.2 Search on element distinctness

A bipartite graph is an undirected graph where the vertices can be divided into two disjoint sets. Every edge has one vertex in one set to a vertex in another set. For instance,

let $1 \leq M < N$ for element distinctness problem. Given set $S = \{x_1, \dots, x_N\}$, and there are $i, j \in [N]$. The bipartite graph can be created such that the vertices are subsets of S .

Each vertex consists of M elements or $M + 1$ elements. A vertex v_T is a subset of S of size M , and v_R is a subset of S of size $M + 1$, such that the two vertices v_T and v_R are connected if they differ by only one element and the subsets $|T| = M$ and $|R| = M + 1$.

The vertex v_T is marked if the set T contains $i \neq j$ and $x_i = x_j$. The goal of element distinctness is to find a marked vertex. The element distinctness based on Grover's search algorithm [192] takes $O(N)$ queries, similar to the classical exhaustive search algorithm.

However, the element distinctness takes $O(N^{2/3})$ queries using the quantum walk algorithm [195, 196].

6.1.3 Search on Hypercube

A hypercube is a graph that consists of $N = 2^n$ nodes, and each node is labeled by an n -bit string. The nodes are connected if they differ by only one single bit. The maximum distance between two nodes is $n = \log N$. The hypercube can be solved using Grover's

search algorithm [190] in $O(\sqrt{N} \log N)$ steps, but the quantum walk [190, 194] can be solved in $O(\sqrt{N})$ steps.

Most of the literature on quantum walk is theoretical, examples given by the authors have no practical applications. In contrast, in this dissertation, I present a circuit design for quantum walk implementation with some practical applications. First, we introduce the concept of the classical random walk and extend it to the quantum random walk.

6.2 Random walk on the line

In classical computing, random walk is a random walk without memory of the past states. For instance, given integer points on a line as in Figure 6.1, toss a fair coin to determine the direction of the next step either to the right or to the left. The transition probability p is only dependent on the current step. The probability of each step t at position n is calculated as:

$$p(t, n) = \frac{1}{2^t} \binom{t}{\frac{t+n}{2}}$$

Where $\binom{a}{b} = \frac{a!}{(a-b)!b!}$. This equation is valid only if $t + n$ is even and $n \leq t$. If $t + n$ is odd or $n > t$, the probability is zero. [189].

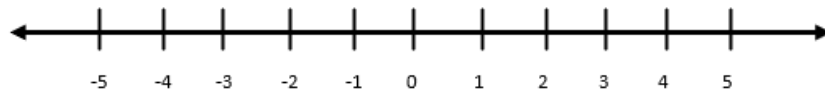


Figure 6.1 Integer points of a line.

Table 6.1 Probability of classical random walk on integer point of a line.

| $t \backslash n$ | -5 | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 |
|------------------|----------------|----------------|----------------|---------------|----------------|---------------|----------------|---------------|----------------|----------------|----------------|
| 0 | | | | | | 1 | | | | | |
| 1 | | | | | $\frac{1}{2}$ | | $\frac{1}{2}$ | | | | |
| 2 | | | | $\frac{1}{4}$ | | $\frac{1}{2}$ | | $\frac{1}{4}$ | | | |
| 3 | | | $\frac{1}{8}$ | | $\frac{3}{8}$ | | $\frac{3}{8}$ | | $\frac{1}{8}$ | | |
| 4 | | $\frac{1}{16}$ | | $\frac{1}{4}$ | | $\frac{3}{8}$ | | $\frac{1}{4}$ | | $\frac{1}{16}$ | |
| 5 | $\frac{1}{32}$ | | $\frac{5}{32}$ | | $\frac{5}{16}$ | | $\frac{5}{16}$ | | $\frac{5}{32}$ | | $\frac{1}{32}$ |

Let the initial position at 0 in Figure 6.1 then the probability at the origin point is 100% such that $p(t = 0, n = 0) = 1$. The next step of the walker will be at 1 or -1 for right and left respectively such that $p(t = 1, n = 1) = 1/2$ or that $p(t = 1, n = -1) = 1/2$. At each step, the walker chooses randomly left or right with a probability of 50%. As can be seen in Table 6.1 the probability distribution is symmetric around the origin.

6.3 Quantum Random walk on the line

Quantum random walks is a quantization of classical random walks aka Markov chains. For instance, quantum walk on an integer point of a line, toss a fair coin to determine which direction to go the next step left or right. In such case, we require two quantum registers $|n\rangle|c\rangle$, where $|n\rangle$ register stores an integer position state of n , and $|c\rangle$ register stores the direction (left, right) of travel which called a coin state. In the quantum walk, we need two operations: coin operator C , and shift operator S . The coin operator C creates a superposition that walker can walk all possible directions simultaneously. The coin operator C determines the direction of the next step either left (L) or right (R) which

is equivalent to the computational basis $|0\rangle$ and $|1\rangle$. The most used coin operators are the Hadamard coin which uses the Hadamard operator and the Grover coin which is the Grover diffusion operator from Grover's algorithm. In this dissertation, the Grover diffusion will be used as a coin operator. The shift operator acts based on the state of coin operator that the shift operator changes the walker state to left or right. For simplicity as an example, we use the Hadamard (H) operator as a coin operator:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$C|R\rangle = \frac{1}{\sqrt{2}}(|L\rangle + |R\rangle) \Rightarrow H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$C|L\rangle = \frac{1}{\sqrt{2}}(|L\rangle - |R\rangle) \Rightarrow H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

The shift operator acts on both registers:

$$S|R\rangle|n\rangle = |R\rangle|n+1\rangle \Rightarrow S|0\rangle|n\rangle = |0\rangle|n+1\rangle$$

$$S|L\rangle|n\rangle = |L\rangle|n-1\rangle \Rightarrow S|1\rangle|n\rangle = |1\rangle|n-1\rangle$$

One step of quantum walks is defined as a unitary operator:

$$U = SC$$

A quantum walk on integer points on a line for t steps consists of U^t to some initial state. Let consider quantum walk with the initial state $|R\rangle|0\rangle$ (position 0, facing to right), computing several steps:

$$t: 1 \mapsto |R\rangle|0\rangle \mapsto \frac{1}{\sqrt{2}}(|-1\rangle|L\rangle + |1\rangle|R\rangle)$$

$$t: 2 \mapsto \frac{1}{2}(|-1\rangle(|L\rangle - |R\rangle) + |1\rangle(|L\rangle + |R\rangle))$$

$$\mapsto \frac{1}{2}(|-2\rangle|L\rangle - |0\rangle|R\rangle + |0\rangle|L\rangle + |2\rangle|R\rangle)$$

$$t: 3 \mapsto \frac{1}{2\sqrt{2}}(|-2\rangle(|L\rangle - |R\rangle) - |0\rangle(|R\rangle + |L\rangle) + |0\rangle(|L\rangle - |R\rangle) + |2\rangle(|R\rangle + |L\rangle))$$

$$t: 3 \mapsto \frac{1}{2\sqrt{2}}(|-3\rangle|L\rangle + |-1\rangle|R\rangle - |1\rangle|R\rangle - |-1\rangle|L\rangle + |-1\rangle|L\rangle - |1\rangle|R\rangle + |3\rangle|R\rangle \\ + |1\rangle|L\rangle)$$

$$t: 3 \mapsto = \frac{1}{2\sqrt{2}}(|-3\rangle|L\rangle + |-1\rangle|R\rangle - 2|1\rangle|R\rangle + |3\rangle|R\rangle + |1\rangle|L\rangle)$$

Taking the squared values of the amplitude of each of the position state after $t = 3$ yields position $|1\rangle$ state with probability $\left(\frac{-2}{2\sqrt{2}}\right)^2 + \left(\frac{1}{2\sqrt{2}}\right)^2 = \frac{5}{8}$. While position $|-3\rangle, |3\rangle$ and $|-1\rangle$ states have probability $\frac{1}{8}$. As can be seen in Table 6.2, for $t = 3, 4, 5$ the probability distribution is not symmetric around the origin which is different from the classical random walk in Table 6.1. In the classical random walk $t = 3$, the position $|1\rangle$ and $|-1\rangle$ has probability $\frac{3}{8}$ each, and position $|3\rangle$ and $|-3\rangle$ has probability $\frac{1}{8}$ each.

Table 6.2 Probability of quantum random walk on integer points on a line.

| $n \backslash t$ | -5 | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 |
|------------------|----------------|----------------|----------------|---------------|---------------|---------------|---------------|---------------|-----------------|----------------|----------------|
| 0 | | | | | | 1 | | | | | |
| 1 | | | | | $\frac{1}{2}$ | | $\frac{1}{2}$ | | | | |
| 2 | | | | $\frac{1}{4}$ | | $\frac{1}{2}$ | | $\frac{1}{4}$ | | | |
| 3 | | | $\frac{1}{8}$ | | $\frac{1}{8}$ | | $\frac{5}{8}$ | | $\frac{1}{8}$ | | |
| 4 | | $\frac{1}{16}$ | | $\frac{1}{8}$ | | $\frac{1}{8}$ | | $\frac{5}{8}$ | | $\frac{1}{16}$ | |
| 5 | $\frac{1}{32}$ | | $\frac{5}{32}$ | | $\frac{1}{8}$ | | $\frac{1}{8}$ | | $\frac{17}{32}$ | | $\frac{1}{32}$ |

6.4 Random walk on the graph

A random walk on the graph is a process for traversing on a graph where at every step the walker moves from node to another connected node. When the graph is unweighted, the next node is chosen uniformly at random among the adjacent of the current node. When the graph is weighted, the next node is chosen according to the probability among the corresponding connected nodes. Let consider random walks on undirected graphs $G(V, E)$ where vertex (node) $V = \{v_1, v_2, v_3, \dots, v_n\}$ with n nodes and edge $E = \{e_1^j, \dots, e_k^m\} = \{(v_i, v_j), \dots, (v_k, v_m)\}$ with m edges. We consider the undirected graph such that moving from node A to B implies moving from node B to A . There are three different matrices for random walk: the adjacency matrix (A), the diagonal matrix D , and the transition matrix M . The adjacency matrix A of the graph G is defined as a $N \times N$ matrix with rows and columns indexed by vertices such that:

$$A_{i,j} = \begin{cases} 1, & (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

For undirected graph the matrix A is symmetric $A = A^T$, but for the directed graph the matrix A is not symmetric $A \neq A^T$. Also, every node has degree d such that each node had d connected edges. Let every vertex has label e_i^j an edge $e_i^j = (v_i, v_j)$ which on v_i 's end is labelled by j and $j \in 1 \dots d$. The degree d_i of a node v_i is the number of connected edges to v_i such that:

$$\text{deg}(d_i) = \sum_{j=1}^N A_{i,j}$$

The diagonal matrix D of the graph G is defined as a $N \times N$ matrix with rows and columns indexed by vertices such that:

$$D_{i,i} = \begin{cases} \frac{1}{\text{deg}(d_i)}, & v_i \in E \\ 0, & \textit{otherwise} \end{cases}$$

The transition matrix M of the graph G describes the transition probabilities from any node to any other node in the graph. $M_{i,j}$ is represented as a $N \times N$ matrix with rows and columns indexed by vertices such that:

$$M_{i,j} = \begin{cases} \frac{1}{\text{deg}(d_i)}, & \textit{if } i \textit{ is connected to } j \\ 0, & \textit{otherwise} \end{cases}$$

$M_{i,j}$ is the probability that going from i to j , given that we are at i .

For directed graph, there are two types of degree: in-degree which the number of edges end at node v_i and out-degree which the number of edges start at node v_i .

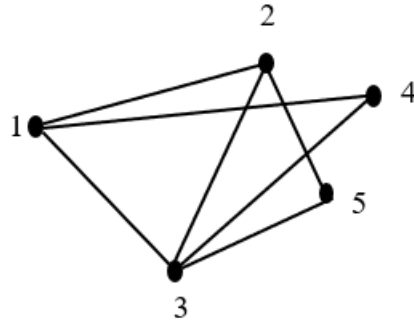


Figure 6.2 undirected graph of five nodes.

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}, \quad M = \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{4} & \frac{1}{2} & 0 \\ \frac{1}{3} & 0 & \frac{1}{4} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{3} & 0 & \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{4} & 0 & 0 \end{bmatrix}$$

For instance, in the Figure 6.2, the classical random walker at node 3 has transition probabilities $\frac{1}{3}, \frac{1}{3}, \frac{1}{2}, \frac{1}{2}$ each of jumping to vertices 1, 2, 4, 5 respectively.

Let v be a row vector which has 1 at i -th entry of v^0 if the node i is the starting point of the random walk.

$$v^0 = (1 \quad 0 \quad 0 \quad 0 \quad 0)$$

v^0 is the initial state that random walk starts at. We need to find the probability distribution at time t . We start:

$$t: 1 \mapsto P^1 = v^0 M = \left(0 \quad \frac{1}{3} \quad \frac{1}{4} \quad \frac{1}{2} \quad 0 \right)$$

This means at step $t = 1$, we move to node 2, 3, 4 with probabilities $1/3$, $1/4$, and $1/2$ respectively.

At $t = 2$, we use the previous step probability $t = 1$ such that:

$$t: 2 \mapsto P^2 = P^1 M = \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{4} & \frac{1}{2} & 0 \end{pmatrix} M = \begin{pmatrix} \frac{13}{36} & \frac{1}{12} & \frac{5}{24} & \frac{1}{8} & \frac{7}{24} \end{pmatrix}$$

This means at step $t = 2$, nodes 1, 2,3,4, and 5 have probabilities $\frac{13}{36}$, $\frac{1}{12}$, $\frac{5}{24}$, $\frac{1}{8}$, $\frac{7}{24}$ respectively.

To simplify, $P^2 = P^1 M = v^0 M.M = v^0 M^2$

At $t = 3$, we use previous step probability $t = 2$ such that:

$$t: 3 \mapsto P^3 = P^2 M = \begin{pmatrix} \frac{13}{36} & \frac{1}{12} & \frac{5}{24} & \frac{1}{8} & \frac{7}{24} \end{pmatrix} M$$

To simplify, $P^3 = P^2 M = v^0 M^2.M = v^0 M^3$.

The probability distribution P^t at time t given by initial probability v^0 at the starting point can be formulated as follows:

$$P^t = v^0 M^t$$

$$P^t = M P^{t-1}$$

6.5 Quantum Random walk on the graph

A quantum walk for graph takes steps in directions determined by coin and shift operators. In a similar way of the quantum walk on a line, we can expand the concept of

the quantum walk to the general graphs, such as hypercube, regular graphs, and irregular graphs. Graphs are used as a convenient way to represent systems in combinatorial problems, machine learning, and many other problems. Quantum walk on a graph is a process for walking on a graph where every step is a move from a node to another connected node. The quantum walk makes the walker moves all possible paths simultaneously in every node.

There are many versions of quantum walk design. Neil Shenvi, Julia Kempe, and K. Birgitta Whaley (SKW) [190] developed a quantum walk-based search method, known as the SKW algorithm. In this dissertation, I use quantum walk design based on SKW design to illustrate a practical application. The quantum walk consists of two steps: The first step is to create a superposition similar to Grover's algorithm. The second step consists of two oracles.

1. Verification oracle: This oracle is used to recognize the solution for a given problem similar to Grover's search algorithm.
2. Direction oracle: This oracle is the walker direction which consists of two blocks:
 - a. Coin operator: The coin operator determines the direction of the next step.
 - b. Shift operator: The shift operator shifts the position of the walker to all possible directions simultaneously. This is an advantage of the quantum walk that the walker can move all paths for the possible solution in one step.

The oracle design for shift operator depends on the type of direction of connected nodes for the given problem.

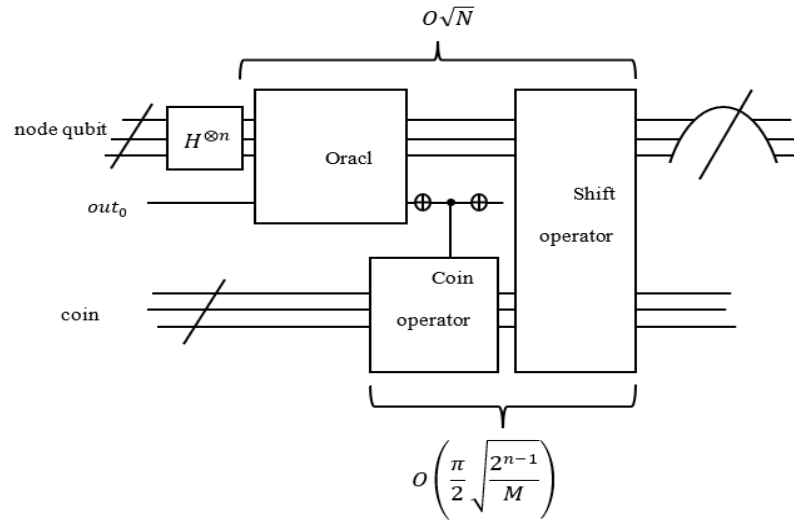


Figure 6.3 Quantum walk algorithm based on SKW design.

As can be seen in Figure 6.3, the quantum walk algorithm consists of four steps:

1. Creating superposition states at initial state
2. Oracle to recognize the marked elements of given problem.
3. Steps in direction which contains the coin operator and shift operator. Both operators repeat $O\left(\frac{\pi}{2}\sqrt{\frac{2^{n-1}}{M}}\right)$ times, where M , the number of marked elements.
4. Perform a measurement on node output.

Steps 1,2, and 3 are repeated $O\sqrt{N}$ times. Each oracle call is followed by a number of steps of coin and shift operators.

Standard Grover has no search direction, and the entire hypercube is searched symmetrically. This means that no direction of search is privileged. Quantum walk gives more opportunities. While Grover uses only an oracle for verification, quantum walk is potentially more powerful as it also uses an oracle for directing the direction of search. In this dissertation will be presented different types of problems, such as search on a hypercube, and search on regular graphs such symmetric graphs – all solved using the quantum walk algorithms.

While using Grover's algorithm we search the entire space which corresponds to the entire hypercube. While searching on the graph of n cube, called a hypercube with $N = 2^n$ nodes, each of the nodes is labelled by an n -bit binary string. The nodes in the hypercube are connected if the nodes differ by only one single bit such that the Hamming Distance between the nodes is one. In the symmetric graph, the nodes are connected if the nodes differ by Hamming weight. However, while using the quantum walk, the quantum algorithm design has the freedom to create special specific graphs. Therefore, at the cost of a more complicated design of the oracles, for special cases of problems an algorithm that is more efficient than Grover's algorithm can be designed.

In this dissertation, I present quantum walk algorithm for solving MAX-SAT problem and mining frequent pattern for association rule mining. In addition, I present an advanced design for shift operator oracle which can be used in practical problems that involve regular graphs.

6.6 Quantum Walk Algorithm for MAX-SAT

Boolean satisfiability expression can be represented as hypercube. For instance, the example presented in MAX-SAT section, suppose we have a Boolean function $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$. Let construct binary hypercube in 3-dimensional space since there is three variables (a, b, c) . Search on the graph of n cube is called hypercube which has $N = 2^n$ nodes, each of the nodes can be labeled by an n -bit binary string. The nodes in the hypercube are connected if the nodes differ by only one single bit such that the Hamming Distance is one. In Figure 6.4, there are 8 nodes, and each node is connected to all nodes that only differ in one single bit.

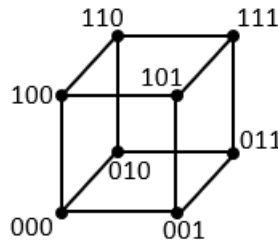


Figure 6.4 Three-dimensional hypercube.

In Figure 6.4 each node is connected to three nodes. In this case we need 2 qubits for the coin operator to create superposition to decide the next node to move. Based on the coin operator states, the shift operator moves the walker to the appropriate nodes by flipping one qubit. In this case each node is connected to three nodes that differ by one qubit. So, we apply a *NOT* gate to one of the node qubits. In Figure 6.5, the 2-qubit Grover coin operator creates four superposition states such as 00, 01, 10, and 11. We just need 3 states to move the walker to the next state. For instance, let start node 000 and

then flipping one bit at a time. The 00 is used move to 001 state, 01 to move to 010 state, and 10 to move to 100 state as can be seen in Figure 6.5.

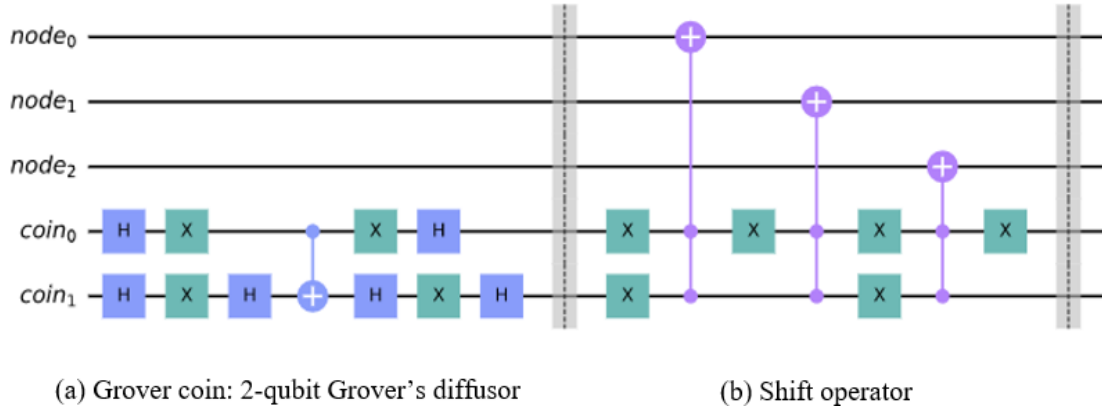


Figure 6.5 Direction oracle consists of: Grover coin for coin operator and shift operator.

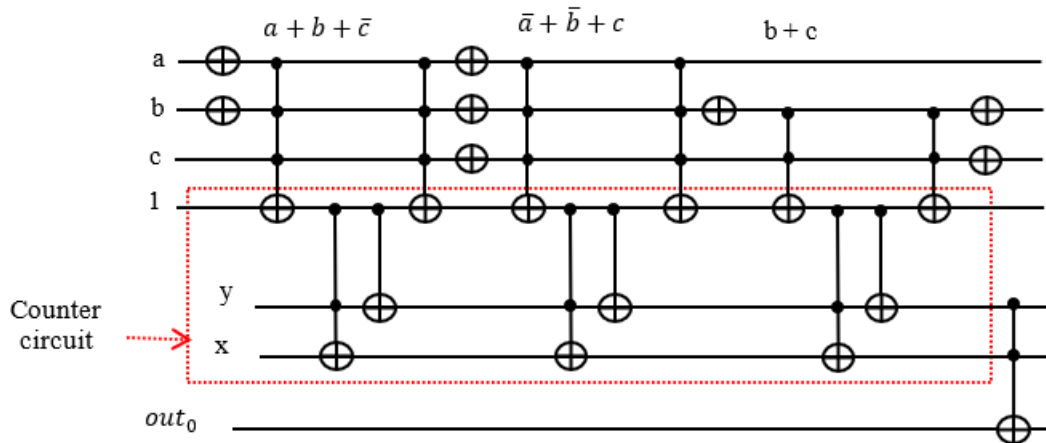


Figure 6.6 Oracle circuit for $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$

Figure 6.6 is the same oracle design that MAX-SAT is used in Grover's search algorithm to recognize the solution of MAX-SAT. Figure 6.7 is a complete quantum walk algorithm, where the oracle block is the oracle design from Figure 6.6, and the coin and shift operators are the circuit designs from Figure 6.5. The Boolean function $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$ has 4 solutions which is the number of the marked

elements as we discussed before. So, the steps for the coin and shift operator will be one

step based on $O\left(\frac{\pi}{2}\sqrt{\frac{2^{n-1}}{M}}\right) = O\left(\frac{\pi}{2}\sqrt{\frac{2^{3-1}}{4}}\right)$ for each oracle.

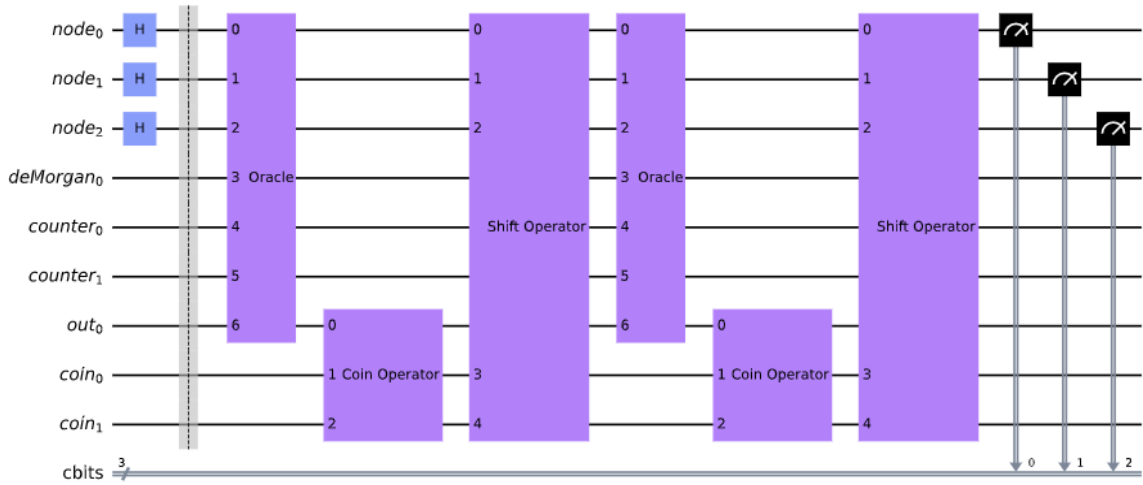


Figure 6.7 Quantum walk algorithm for solving $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$

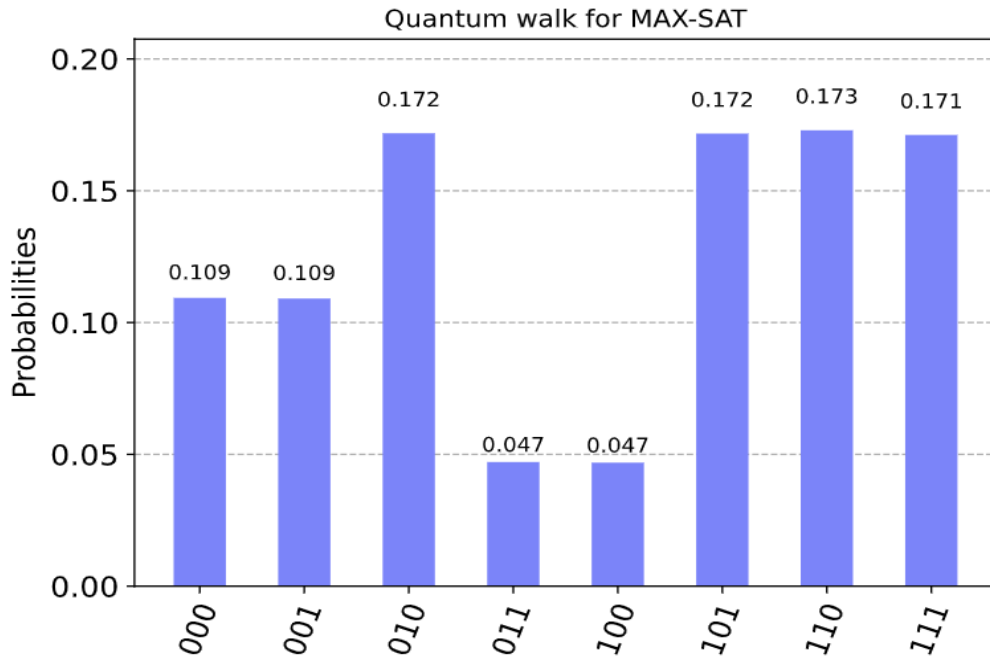


Figure 6.8 Measurement of the Boolean variables of $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$

After applying measurement on the qubits for $node_0node_1node_2$ in Figure 6.7, in Figure 6.8 the output from the simulation with high probability 010, 101, 110, and 111 for abc respectively matches the theoretical values for the solution $f(a, b, c) = (a + b + \bar{c})(\bar{a} + \bar{b} + c)(b + c)$.

In standard Grover there is no direction of search, and the entire hypercube is searched “symmetrically”. In general, the Grover’s search algorithm is a good option to solve for space shaped as a hypercube, while quantum walk is good for arbitrary graphs but especially for various grids. Grover's search algorithm is reputed to have a quadric speed up over the classical exhaustive search algorithm. However, Grover's algorithm is no more efficient than classical algorithms for some problems, including the searching a grid problem [191] and the searching element distinctness graph problem [195].

6.7 Quantum Walk Algorithm for Mining Frequent Patterns of Association Rule Mining

Mining frequent pattern can be represented as graph based on k -itemset. For instance, if we need to mine 3-itemset then all itemset equal or greater than 3-itemset are extracted. Recall the equation (3)

$$abd + abe + ade + bde + abd + abe + ade + abd + acd$$

The search space to mine k -itemset is only in $\binom{n}{k}$ itemset, where n is the number of items. For $n = abcde$, the search space of 3-itemset is:

$$abc, abd, abe, acd, ace, ade, bcd, bce, bde, cde$$

These 3-itemset can be represented as a regular graph in which each element is a node. Nodes are connected if they differ by just one symbol, as can be seen in Figure 6.9a. This graph in Figure 6.9a is known as the Johnson graph. The Johnson graph $J(n, k)$ can be represented as a binary string of n bits with k ones, as can be seen in Figure 6.9b. The number of vertices in Johnson graph is equal to $\binom{n}{k}$ and the degree d -regular is equal to $k(n - k)$. For instance, $J(5, 3)$, the number of vertices is equal to $\binom{5}{3} = 10$ and the degree d -regular is equal to 6.

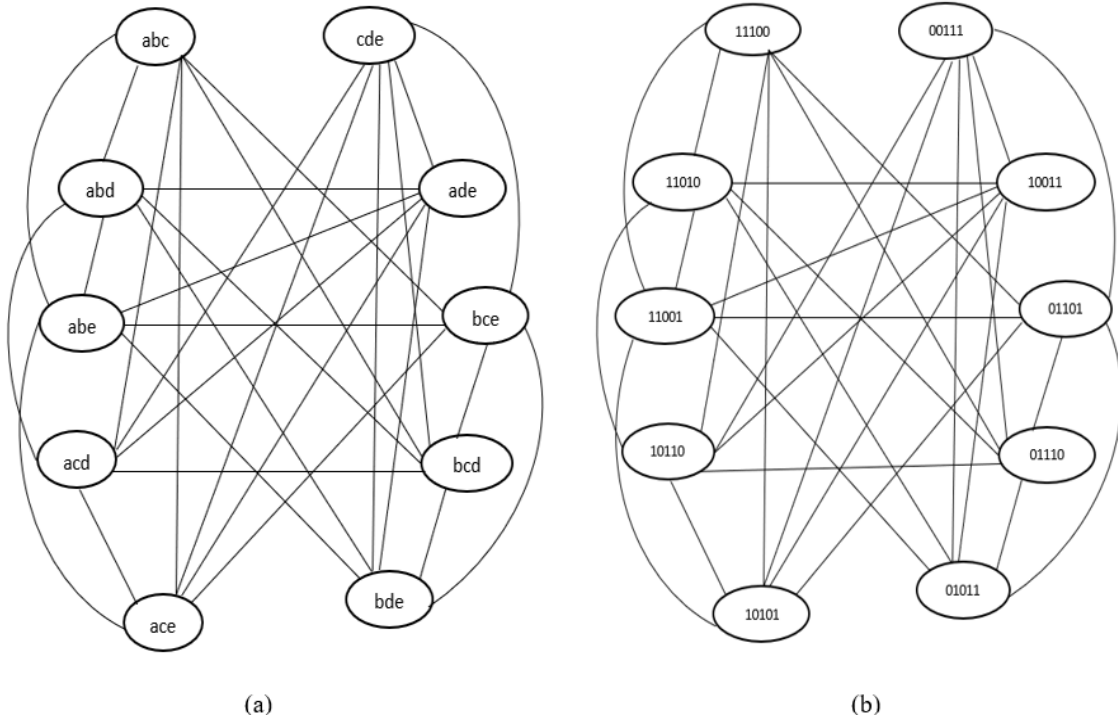


Figure 6.9 Johnson graph $\binom{n}{k}$, where $k = 3$ for 3-itemset of $n = 5$ for $abcde$

As we can see in Figure 6.9b, every node is connected to 6 other nodes with a Hamming distance of two. For instance, the 11001 node is connected to the 01101 node such that the Hamming distance is equal to 2. This Hamming distance can be represented

as a swap operation between the first (most significant) and second qubit of 11001 that transforms to 01101. Table 6.3 is an example 11100 node with swap qubits to reach all connected nodes.

Table 6.3 Swap operation for 11100 node to reach all other connected nodes.

| Node | Swap qubits | Connected nodes |
|-------|-------------|-----------------|
| 11100 | (0, 3) | 01110 |
| | (0, 4) | 01101 |
| | (1, 3) | 10110 |
| | (1, 4) | 10101 |
| | (2, 3) | 11010 |
| | (2, 4) | 11001 |

Based on the same approach as in Table 6.3, all other nodes can reach other connected nodes by SWAP operation, as can be seen in Table 6.4

Table 6.4 SWAP operation for all nodes

| Node | Swap qubits |
|-------|--|
| 11100 | (0, 3), (0, 4), (1, 3), (1, 4), (2, 3), (2, 4) |
| 11010 | (2, 3), (3, 4), (1, 2), (1, 4), (0, 2), (0, 4) |
| 11001 | (2, 4), (3, 4), (1, 2), (1, 3), (0, 2), (0, 3) |
| 10110 | (3, 4), (1, 2), (1, 3), (0, 4), (2, 4), (0, 1) |
| 10101 | (3, 4), (1, 2), (1, 4), (0, 3), (2, 3), (0, 1) |
| 01011 | (0, 3), (0, 4), (1, 2), (0, 1), (2, 3), (2, 4) |
| 01110 | (0, 2), (0, 3), (1, 4), (3, 4), (2, 4), (0, 1) |
| 01101 | (0, 4), (0, 2), (0, 1), (3, 4), (2, 3), (1, 3) |
| 10011 | (1, 4), (1, 3), (2, 4), (2, 3), (0, 2), (0, 1) |
| 00111 | (0, 4), (0, 3), (1, 2), (1, 4), (1, 3), (0, 2) |

Based on Table 6.4, we have all permutations (0, 1), (0, 2), (0, 3), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4) such that each of them is connected to node qubits. We use standard Grover diffusion as a coin operator, and the output of the coin operator is used

as a control for the shift operator, which is the SWAP gate. Thus, in this case, we design a multi-controlled SWAP gate as a shift operator, as can be seen in Figure 6.10.

Figure 6.10 is a complete design for the quantum walk algorithm for mining frequent patterns of association rule mining. We could not find a multi-controlled SWAP gate in QISKIT to simulate Figure 6.10. There are other approaches to decomposing the multi-controlled SWAP gate into multi-controlled Toffoli gates, or one multi-controlled Toffoli gate and two CNOT gates. However, such designs result in high quantum costs.

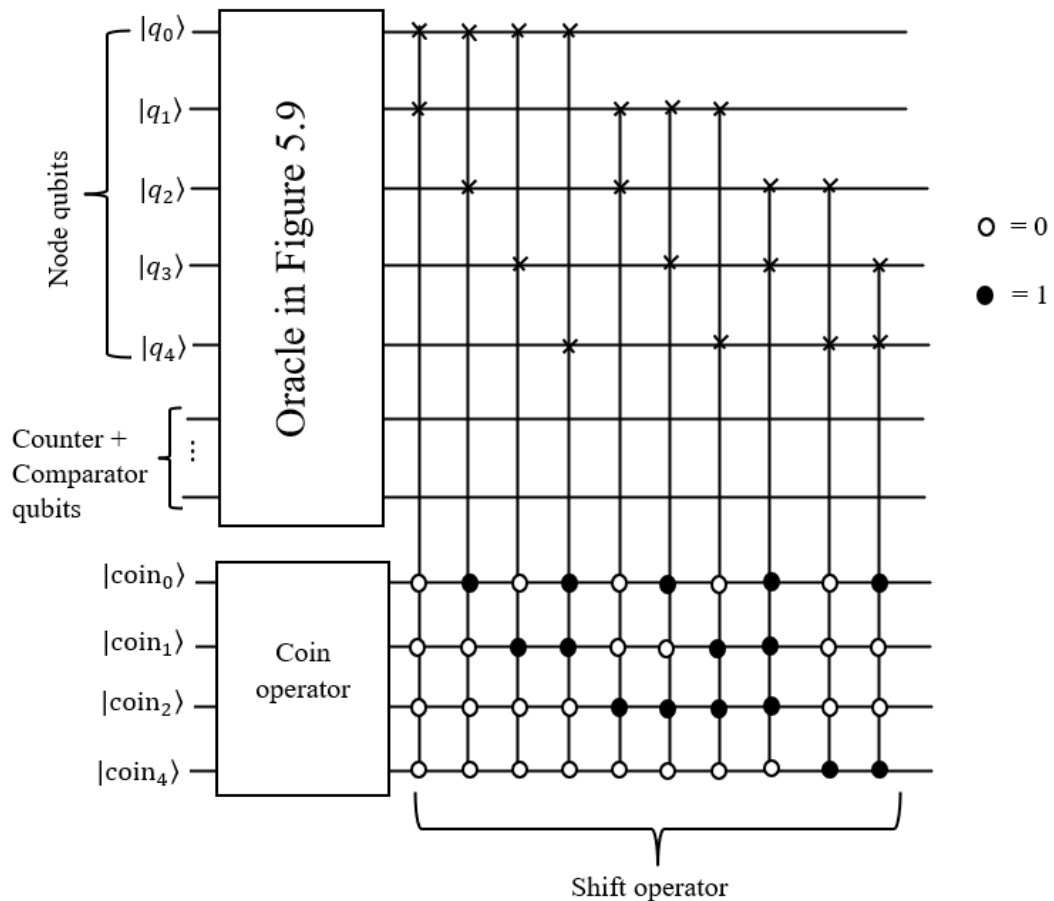


Figure 6.10 Oracle with coin and multi-controlled SWAP gates for the shift operators.

This chapter aims to represent and solve a practical problem in quantum walk since most of the literature on quantum walk used only mathematical models. I have shown a shift operator design for the hypercube and a Johnson graph for MAX-SAT and mining frequent patterns for association rule mining, respectively. Each problem has a different design for the shift operator; thus, the shift operator is the main component design that needs to be designed efficiently for each type of problem for quantum walk. The oracle and coin operator are very similar to Grover's search algorithms.

7 QUANTUM COMPLEXITY THEORY

Grover's search algorithm is a remarkable quantum search algorithm that exhibits a quadratic speedup over its counterpart classical algorithms. In order to comprehend the quantum complexity of Grover's search algorithms, it is necessary to first understand the complexity class of classical algorithms before moving on to quantum complexity. There are quantum algorithms that are faster than their counterparts of classical algorithms. How is it to measure the runtime of the computational tasks? To understand this question better, we first need to understand computational complexity theory for both classical and quantum computers.

7.1 Classical Complexity

An algorithm's runtime or time complexity depends on many parameters, such as the input size, software (programming language, compiler, OS), and hardware. In this dissertation, I use the input size of a problem to analyze the time complexity of the algorithm. The time complexity is measured by the number of operations (steps) executed for the given input size. In general, computational complexity deals with the study of the classification of computational problems according to their time and space complexity.

There are many computational problems, such as:

1. Search problems: involve finding a solution with certain properties if such a solution exists within a given dataset or structure.
2. Decision problems or constraint satisfaction problems: answer either 'yes' or 'no.'

3. Optimization problems: find the best possible solution among all possible solutions.
4. Function problems: compute the output based on functions or expressions for the given input.
5. Counting problems: count the number of solutions.

The goal is to find the best algorithm to solve the problem with the best time complexity. There are three ways to measure the time complexity of an algorithm. Each of these measurements has mathematical notations (asymptotic notation) to describe the time complexity, such as:

1. Worst-case time complexity: the longest amount of time for any input size of n to complete. This provides an upper bound and the asymptotic notation is O (big Oh)
2. Best-case time complexity: the shortest time for any input size of n . This provides a lower bound and the asymptotic notation is Ω (big Omega)
3. Average-case time complexity: the average time for all input size of n . This can be the same as worst-case or better. The asymptotic notation is Θ (Theta)

Some cases o (Little Oh) and ω (little Omega) are used as asymptotic notation for time complexity. The runtime for an algorithm depends on the input size, and the growth rate for an algorithm describes how the runtime will increase as the input size increases.

Let's describe the asymptotic notation as growth rate in mathematical terms that can be seen in Table 7.1. Also, some common equations for time complexity are in Table 7.2.

Table 7.1 Asymptotic notation

| Asymptotic notation | Mathematical meaning | Description |
|---------------------|----------------------|--|
| O | \leq | Growth rate is less than or equal (\leq) to given input value |
| Ω | \geq | Growth rate is greater than or equal (\geq) to given input value |
| Θ | $=$ | Growth rate is equal ($=$) to given input value |
| o | $<$ | Growth rate is less than ($<$) to given input value |
| ω | $>$ | Growth rate is greater than ($>$) to given input value |

Table 7.2 Common equation for time complexity

| O (Big Oh) | Class | Examples |
|-----------------------|------------------|---|
| $O(1)$ | Constant time | Adding an element to the front of a linked list |
| $O(\log n)$ | Logarithmic time | Finding an element in a sorted array |
| $O(n \log n)$ | Linearithmic | Sorting elements in array |
| $O(n)$ | Linear time | Finding an element in an unsorted array |
| $O(n^2)$ | Quadratic time | Shortest path between 2 nodes in a graph |
| $O(n^3)$ | Cubic time | Matrix Multiplication |
| $O(n^c)$ for some c | Polynomial time | Problems with exponent of c |
| $O(2^n)$ | Exponential time | Finding all subsets of an array |
| $O(n!)$ | Factorial time | Finding all permutations of a given set |

7.2 Classical Complexity Classes

Many computation problems can be formulated into equivalent decision problems. When studying complexity theory, we typically focus on decision problems where the answer is either **yes** or **no** to a specific question about the input. This is because the decision problems allow a simple definition of the problem and easily prove the complexity bounds. For instance, in primality testing, given an input x , we want to know if x is prime or not. Also, for a satisfiability problem, such as a given formula as input, we want to know whether it is satisfiable or not. For instance, the traveling salesman problem is a

decision problem that asks whether or not there is a route that passes through every city whose length is less than a fixed length for a given set of intercity distances. For many computation problems, we can re-state them as yes/no problem or a sequence of the yes/no problems. In complexity theory, the problem is called *language* which is used to formulate computational problems as an abstract and standardized formula. Language L is a set of binary strings $L = \{x: \text{answers is yes}\}$. For decision problem $f(x)$, the language $L = \{x: f(x) = 1\}$ which corresponds to $f(x) = 1$ for the set of all binary strings x . For instance, $L_{\text{primality}} = \{x: x \text{ is prime}\}$.

Let us define the main types of classical complexity classes:

P (polynomial time): the class of problems that can be solved in polynomial time by deterministic computing. For instance, checking if a given number is prime or not belongs to P. In general, P is the class of all problems that are efficiently solvable by a classical computer. There are various complexity classes that can be derived from the idea of polynomial time such as: non-deterministic polynomial (NP), NP-complete, and NP-hard. Table 7.3 and Figure 7.1 show the main classical complexity classes.

NP (non-deterministic polynomial): the class of problems where classical computation can verify the ‘yes-instance’ in polynomial time. However, NP problems cannot always be solved in polynomial time. For instance, what are all the prime factors of a given number? This is an NP problem. NP contains all P problems and other problems for which a polynomial time solution is not guaranteed to exist.

NP-hard: a set of problems as hard as the NP-complete but NP-hard contains some problems outside of NP-complete. NP-hard may or may not be verifiable in polynomial time.

NP-complete: a set of problems that can always be verified in polynomial time. NP-complete problems are both NP and NP-hard.

Table 7.3 P and NP complexity classes

| Complexity classes | Solvable in P time | Verifiable in P time | Examples |
|--------------------|--------------------|----------------------|----------------------------------|
| P | Yes | Yes | Find the greatest common divisor |
| NP | Yes/No | Yes | Traveling salesman problem |
| NP-complete | Unknown | Yes | SAT, graph coloring |
| NP-hard | Unknown | Yes/No | MAX-SAT, optimization problems |

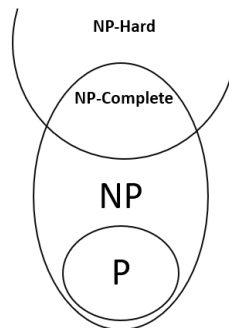


Figure 7.1 Main classical complexity classes.

The time required to solve or check a type of problem in P increases as the problem's size increases. NP problems can generally be solved using heuristics, approximations, probabilistic methods, and special cases but the solutions can be verified in polynomial time.

BPP (bound-error probabilistic polynomial): the class of problems that can be solved in polynomial time with bounded error such that the correct answer ‘yes’ at least with probability $\frac{2}{3}$ and ‘no’ at most with probability $\frac{1}{3}$.

PSPACE (polynomial space): the class problems that can be solved in polynomial space by classical deterministic computing.

EXP (exponential): the class problems solvable in exponential time on deterministic computing.

7.3 Quantum Complexity Classes

Some quantum algorithms, such as Shor’s algorithm, Grover’s search algorithm, and the quantum walk algorithm, are significantly faster than their counterparts in classical algorithms. The classical time complexity of classical algorithms is measured by the number of operations executed for the given input size. The complexity of quantum algorithms is known as query complexity. The quantum query complexity is measured by the number of queries that the oracle is using. For instance, the query complexity of Grover’s search algorithms is $O(\sqrt{N})$. The goal is to achieve by querying the oracle the least possible number of query repetition times. The most important quantum complexity class is BQP (bound-error quantum polynomial).

BQP (bound-error quantum polynomial) was first introduced in [197], this concept started the interest in quantum computational complexity field as a separate from classical complexity research. **BQP** is the class of problems that can be solved in polynomial time by a quantum computer with bounded probability error such that the

correct answer ‘yes’ is at least with probability $\frac{2}{3}$ and ‘no’ at most with probability $\frac{1}{3}$. The BQP corresponds to BPP in classical complexity. The BQP is the most important quantum complexity class.

In Figure 7.2, $L \in BQP$ such that a given input of $|X\rangle$ applied a sequence of quantum circuits Q_L and then measured to get the correct classical output. The quantum circuit Q_L provides a bound-error polynomial time. The quantum circuit is either an oracle circuit or a complete quantum algorithm.

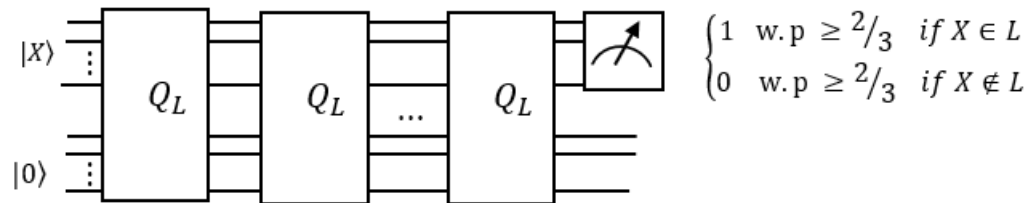


Figure 7.2 Quantum complexity circuit schematic.

One of the big questions in quantum complexity theory is: what is the power of BQP? All types of problems in BPP and P can be solved in BQP by a quantum computer such that BQP contains all the problems in BPP and class BQP is a subset of class PSPACE [48, 197, 199].

$$P \subseteq BPP \subseteq BQP \subseteq PSPACE \subseteq EXP$$

There is still ongoing research to determine if BQP contains problems outside of NP, but some problems have proven that BQP is outside of NP [201]. It was proven that BQP is not a subset of NP ($BQP \not\subseteq NP$) [198, 200]. For illustration, the relationship between P, NP and BQP can be seen in Figure 7.3.

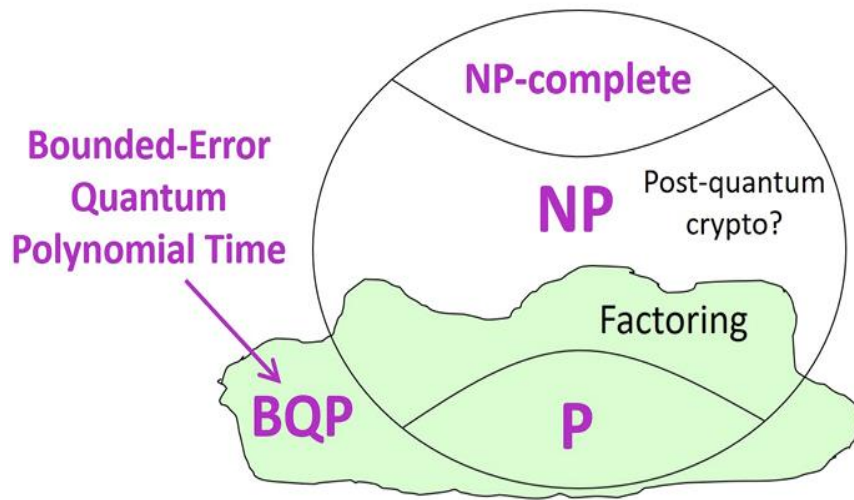


Figure 7.3 relationship P, NP and BQP complexities [200]

Other quantum complexity classes, such as EQP (exact quantum polynomial) and NQP (non-deterministic quantum polynomial), are quantum analogs of P and NP. These types of quantum complexities are not studied much because of the nature of quantum computing, which is associated with errors and noise. The results from quantum computing are probabilistic with a bound error, indicating that BQP is a fundamental quantum complexity class.

7.4 Practical Complexity Analysis

As quantum algorithms may exhibit different characteristics and behaviors compared to their classical counterparts, the primary performance metric in classical and quantum computing is query complexity (time complexity). However, the quantum query complexity hides the implementation-related cost of quantum resources. An oracle is the basic unit of quantum search algorithms, and the number of inquiries that the oracle calls determine the algorithm's time complexity. The quantum circuit cost and number of

qubits for the oracle are taken into account to calculate quantum complexity accurately. Taking only the time complexity disturbs accurate estimation of the speed of the quantum search algorithms. Different oracle circuit designs [207, 208, 209, 210, 211] are constructed to create a variety of configurations, offering choices like a decreased number of quantum gates at the expense of an increased number of qubits (or vice versa). The trade-off relation is then applied to the design candidates for comparison. For instance, A and B are two different oracle circuit designs for Grover's search algorithm, but they implement different architectural designs regarding the number of quantum gates and qubits. If we look purely at the time complexity, then the complexity of A and B oracles are the same, which takes $O(\sqrt{N})$. However, A and B oracles differ regarding quantum resources (number gates and qubits). Thus, the efficiency of different oracle designs is graded based on quantum resources. The quantum gates and qubits should be explicitly considered to calculate the algorithm's overall complexity in terms of performance efficiency and comparison analysis. Therefore, quantum complexity is divided into two categories: theoretical complexity and practical complexity. Theoretical complexity is related to the number of queries (time complexity) that quantum algorithm requires, and it is expressed as asymptotic notations. The theoretical complexity is proven when a quantum algorithm was developed. Practical complexity [202, 203, 206] is related to the practical implementation of the quantum algorithm design. The practical complexity, also known as a circuit complexity, is calculated using the following metrics:

1. Quantum circuit cost: determine the number of elementary quantum gates used in the quantum circuit.

2. Space complexity: number of qubits required.
3. The circuit depth: the number of steps for the quantum circuit requires. For instance, quantum gates that can be executed in parallel are counted as one step. The circuit depth depends on the physical properties of the quantum hardware.

The depth of a circuit is the longest path in the circuit, which is the number of elementary gates to execute on that path. Each gate has an execution time, and the longest path will be considered as the execution time. The quantum circuits are transpiled to run on the actual quantum hardware. The transpilation is a translation of logical qubits and gates onto a physical NISQ device [212]. The circuits are optimized based on the hardware technology during the transpilation process, and the actual depth may change.

In this dissertation, we focus on the two primary resources of quantum computing: quantum circuit cost and the number of qubits used to analyze the practical complexity of the quantum search algorithms. The quantum cost of a circuit is the total number of quantum gates in the given circuit. The elementary quantum gates are NOT, controlled-nth root of NOT, CNOT, Toffoli, and SWAP gates. The Toffoli gate is built using the elementary controlled-nth root of NOT gates to calculate the more detailed quantum cost.

The quantum cost of NOT, controlled-nth root of NOT, and CNOT is a unit cost that requires a single operation. The quantum cost of the Toffoli gate is $2^{m+1} - 3$, where m is the number of controlled qubits of the Toffoli gate. In Figure 5.9, we presented an example of quantum algorithm mining frequent patterns for association rule mining. To analyze the practical complexity, let's define:

$$\alpha = \lceil \log_2 T \rceil$$

$$\beta = \lceil \log_2 I \rceil$$

$$\gamma = k\text{-itemssets}$$

R = number of Grover iterations

For instance, referring to chapter 5, where T is the number of transactions, I is the number of items and k is the maximum number of frequent itemset. We can formulate the number of quantum gates as a general case (worst case) of our design as flows:

1. The number of SOP terms is equal to T , thus, the number of quantum counter blocks equals the number of SOP terms.
2. Each term of the SOP has γ of literals such that the number of controlled qubits of the Toffoli gate is equal to γ .
3. Quantum counter:
 - a. The maximum number of controlled qubits of Toffoli gates for the quantum counter is equal to α . The quantum counter consists of CNOT, Toffoli, 3-controlled Toffoli, up to α - controlled Toffoli gates.
4. Quantum comparator:
 - a. The quantum comparator requires 4α NOT gates, 2α Toffoli gates and one α -controlled Toffoli gate.

The total complexity would be the total number of gates in 1, 2, 3, and 4 steps times the number of iterations required by Grover's search algorithm:

$$R(T(\gamma\text{-controlled Toffoli} + \text{CNOT} + \text{Toffoli} + 3\text{-controlled Toffoli} + \dots + \alpha\text{-controlled Toffoli}) + 4\alpha \text{ NOT} + 2\alpha \text{ Toffoli gates} + \text{one } \alpha\text{-controlled Toffoli gate})$$

The number of total qubits for our design is:

$$\begin{cases} I + 3(\lceil \log_2 T \rceil + 1) + 2, & \text{if } \log_2 T \text{ is interger value} \\ I + 3\lceil \log_2 T \rceil + 2, & \text{if } \log_2 T \text{ is not interger value} \end{cases}$$

The number of qubits in [204] is equal to $2TI + 1$. The total number of quantum gates in [204] can be formulated as $2TI$ Toffoli gates and one I -controlled Toffoli gate. The number of iterations is equal to α . The total number of gates is equal:

$$\alpha (2TI \text{ Toffoli gates} + I\text{-controlled Toffoli gate})$$

The number of qubits in [205] is equal to $\alpha + \beta \gamma + \gamma + 1 + T$. Also, the total number of quantum gates in [205] can be formulated as:

$$\gamma T (\text{NOT gate} + \text{CNOT gate} + 3\text{-controlled Toffoli gate} + \beta (\text{Toffoli gate} + 3\text{-controlled Toffoli gate} + 4\text{-controlled Toffoli gate} + \dots + (\alpha + 1)\text{-controlled Toffoli gate}))$$

Table 7.4 General formula comparison for total number of gates and qubits

| Design | Total number of quantum gates | Total number of qubits |
|--------|--|--|
| [204] | $\alpha (2TI \text{ Toffoli gates} + I\text{-controlled Toffoli gate})$ | $2TI + 1$ |
| [205] | $\gamma T (\text{NOT gate} + \text{CNOT gate} + 3\text{-controlled Toffoli gate} + \beta (\text{Toffoli gate} + 3\text{-controlled Toffoli gate} + 4\text{-controlled Toffoli gate} + \dots + (\alpha + 1)\text{-controlled Toffoli gate}))$ | $\alpha + \beta \gamma + \gamma + 1 + T$ |

| | | |
|---|--|--|
| Proposed in this dissertation | $R(T(\gamma\text{-controlled Toffoli} + \text{CNOT} + \text{Toffoli} + 3\text{-controlled Toffoli} + \dots + \alpha\text{-controlled Toffoli}) + 4\alpha \text{ NOT} + 2\alpha \text{ Toffoli gates} + \text{one } \alpha\text{-controlled Toffoli gate})$ | $\begin{cases} I + 3(\lceil \log_2 T \rceil + 1) + 2, & \log_2 T \text{ is interger} \\ I + 3\lfloor \log_2 T \rfloor + 2, & \log_2 T \text{ is not interger} \end{cases}$ |
| NOTE: The quantum cost of NOT and CNOT is equal to 1. Toffoli gate cost is equal $2^{m+1} - 3$, where m is the number of controlled qubits of the Toffoli gate | | |

Table 7.4 presents a general formula for the total number of gates and total number of qubits that can be used to calculate the total complexity of the quantum cost. We use as an example in Tab 5.2 that we have 10 transactions with 6 items and generating 3 itemset as maximum frequent itemset (more details see in chapter 5). We minimized this example to equation 5.1. To construct the SOP function from equation 5.1 we need nine 3-controlled Toffoli gates plus nine 3-controlled Toffoli gates for the mirror.

$$\text{Cost: 3-controlled Toffoli} = 18(2^{3+1} - 3) = 18 * 13 = 234$$

For the quantum counter in Figure 5.9, we need nine quantum counter blocks, and each requires one CNOT, one Toffoli gate, one 3-qubit controlled Toffoli gate, and one 4-qubit controlled Toffoli gate.

Cost:

- CNOT = 9
- Toffoli (2-controlled Toffoli) = $9(2^{2+1} - 3) = 9 * 5 = 45$
- 3-controlled Toffoli = $9(2^{3+1} - 3) = 9 * 13 = 117$
- 4-controlled Toffoli = $9(2^{4+1} - 3) = 9 * 29 = 252$

For the quantum comparator in Figure 5.8, we need 17 NOT gates, 8 Toffoli gates, and one 4-qubit controlled Toffoli gate.

Cost:

- 17 NOT = 17
- Toffoli (2-controlled Toffoli) = $8(2^{2+1} - 3) = 8 * 5 = 40$
- 4-controlled Toffoli = $(1)(2^{4+1} - 3) = 1 * 29 = 29$

The proposed design in [204] for the same example 10 transactions and 6 items would require 4*120 Toffoli gates and four of 6-qubit controlled Toffoli gate.

Cost:

- Toffoli (2-controlled Toffoli) = $4*120(2^{2+1} - 3) = 480 * 5 = 2400$
- 6-controlled Toffoli = $4 * (1)(2^{6+1} - 3) = 4 * 125 = 500$

The other proposed design in [205] for the same example with 10 transactions and 6 items would require 30*(NOT, CNOT, and 3-qubit controlled Toffoli gates) and 36*(Toffoli gate, 3-qubit controlled Toffoli gate, 4-qubit controlled Toffoli gate, and 5-qubit controlled Toffoli gate).

Cost:

- 30 NOT = 30
- 30 CNOT = 30
- Toffoli (2-controlled Toffoli) = $90(2^{2+1} - 3) = 90 * 5 = 450$
- 3-controlled Toffoli = $(30 + 90)(2^{3+1} - 3) = 120 * 13 = 1560$
- 4-controlled Toffoli = $(90)(2^{4+1} - 3) = 90 * 29 = 2610$
- 5-controlled Toffoli = $(90)(2^{5+1} - 3) = 90 * 63 = 5670$

Table 7.5 Comparison of quantum circuit cost

| Gate | Cost [204] | Cost [205] | 2*Cost Proposed in this dissertation |
|-------------------------|-------------|--------------|--------------------------------------|
| NOT | 0 | 30 | 2*17 = 34 |
| CNOT | 0 | 30 | 2*9 = 18 |
| 2-controlled Toffoli | 2400 | 450 | 2*85 = 170 |
| 3-controlled Toffoli | 0 | 1560 | 2*351 = 702 |
| 4-controlled Toffoli | 0 | 2610 | 2*281 = 562 |
| 5-controlled Toffoli | 0 | 5670 | 0 |
| 6-controlled Toffoli | 500 | 0 | 0 |
| Total cost | 2900 | 10350 | 1486 |
| Number of qubits | 121 | 27 | 19 |

In Table 7.5, please note the number in the header of the last column is the optimal number of the Grover's iterations.

Toffoli gates are important in determining quantum costs, as seen in Table 7.5, where the cost of a quantum circuit increases with the number of controlled qubits of Toffoli gate. Table 7.5 shows that our method has better complexity in terms of the total number of qubits and quantum circuit cost of the oracle. In this example, we calculate only the oracle without adding the mirror and the diffusor for our design and Quantum Fourier Transform (QFT) in [204, 205]. We reduced the number of qubits with a low quantum cost. The overall quantum complexity is based on the quantum circuit, and the bottleneck is the trade-off between improved time performance and efficiency in qubits. We can calculate the overall quantum complexity as follows:

$$\text{overall quantum complexity} = \sqrt{N} \times \text{total quantum cost} \times \text{number of qubits}$$

The theoretical and practical quantum complexities are considered qualitative and quantitative complexities [190]. At the moment, there is a gap between them in that there

is not yet general guidance for the practical quantum complexities based on all technologies because the practical quantum complexities evolve in the quantum hardware system [190]. For the future, we need to have a methodology independent of the technology to calculate quantum total complexity, which consists of the query complexity (qualitative) of the quantum algorithm and the quantum computation circuit cost (quantitative) to solve.

8 METHODOLOGY FOR ADVANCED QUANTUM ORACLE DESIGN

Chapter 8

Methodology for Advanced Quantum Oracle Design

Alasow, Abdirahman, and Marek Perkowski. "Quantum Algorithms for Unate and Binate Covering Problems with Application to Finite State Machine Minimization." *Journal of Applied Logics* (2023).

Authors: Abdirahman Alasow, Marek Perkowski

Abdirahman Alasow:

Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Writing—original draft

Marek Perkowski:

Conceptualization, Methodology, Supervision, Visualization, Writing—review & editing.

https://pdxscholar.library.pdx.edu/ece_fac/768/

In this Chapter, I proposed a uniform methodology quantum oracle design for quantum search algorithms where many ideas from several algorithms are combined together and applied to the design of oracles for Grover's search algorithm and quantum walk algorithm. The methodology of the oracle design can be applied to many problems in combinatorial optimization problems, such as Unate Covering Problem (UCP) and Binate Covering Problem (BCP), logic design such as all forms of SAT and SAT related problems, machine learning, and logic puzzles.

8.1 Covering Problems

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

Many optimization problems can be formulated as the selection of a subset from a larger set. One familiar form is the covering problems [221, 222, 223]. Covering problems are minimization problems for combinatorial optimization problems in which a certain combinatorial structure covers another. For instance, in logic design, the covering

problems can be formulated as a set of individual minterms each to be covered by a subset of the set minterms (these subsets can be for instance prime implicants for a specific example of the covering problem). Minterms for a Boolean function of n variables are products of literals for all variables of this function. A literal is a variable or the negation of a variable. True minterms are those for which the value of the function is 1. The prime implicant is the product of literals that cannot be extended by removing some of the literals and which covers some subset of true minterms. Covering problems are generally given as a table with rows corresponding to the set elements and columns corresponding to the subsets.

Covering problems include two main types: unate covering problem (UCP) and binate covering problem (BCP) [213, 214, 231]. The unate covering problem is the problem of finding a minimum cost assignment to variables for which a given Boolean function f is equal to 1. The literals are all in positive form (uncomplemented). The Binate covering problem has the extra constraint that negative literals may be present. The covering problem can be generalized by assuming that the choice of a subset implies the choice of another subset. This additional constraint can be represented by an implication clause. For example, if the selection of group a implies the choice of b ($a \Rightarrow b$), then the clause $(\bar{a} + b)$ is added. Note that the implication clause makes the product of sums form binate in variable a , because a (uncomplemented) is also part of some covering clauses. Therefore, this class of problems is called a “binate covering” or a “covering with closure”. There are two main ways to express the covering problems: constraints in the form of a matrix or a product of sum (POS) form of a Boolean equation

$f = 1$. This formulation of POS is known as the Petrick's method [220]. If the covering problems are expressed in the form of a matrix, then the corresponding matrix (covering matrix) of the UCP is filled with elements from the set $\{1, 0\}$ while the BCP is filled with elements from the set $\{1, -1, 0\}$. A -1 entry corresponds to a complemented variable, and a 1 entry to an uncomplemented one. In the covering matrix, the rows correspond to each term of the expression, and the columns correspond to each variable. Covering problems can also be expressed in the form of a POS formula, such that $f(x_1, x_2, x_3, x_4) = (x_1 + x_2)(x_2 + x_3)(x_1 + x_4)x_3x_4 = 1$ then this is called an unate covering, which always has a solution. If some of the variables in the function appear both positive and negative (complement) such that $f(x_1, x_2, x_3, x_4) = (x_1 + x_3 + x_4)(x_1 + \bar{x}_2 + x_4)(\bar{x}_2 + x_3 + x_4)(x_2 + \bar{x}_3 + x_4)$ then it is called the binate covering, which may or may not have a solution. In this Chapter, we will express the covering problems in POS form.

8.1.1 Related work

The fundamental covering problem known from computer science has many practical applications in electronic design automation (EDA) and digital systems. There are many combinatorial optimization problems for covering problems in various areas, such as logic minimization [215], scheduling [229], parallel computation on a GPU [228], and allocation, encoding, and routing [216] for unate covering problem. Binate covering problem is used in finite state machine minimization [214, 217], technology mapping [213], Boolean relations [218], and Directed Acyclic Graphs (DAG) covering problems [215, 219, 226]. Logic minimization is the process of finding the optimal implementation

of a logic function. The minimum-cost implementation is achieved when the cover of a given function consists of the minimum number of prime implicants to represent the function. In another variant, the solution should have the total minimum cost of selected prime implicants. The Unate covering can be used to achieve exact logic minimization [215]. Scheduling problems can be formulated as UCP, such as vehicle and crew scheduling problems in [229], to minimize the combined vehicle and crew cost. Parallel computation on a GPU [228, 234, 235, 236] was explored using the Unate covering problem. The classical task of the GPU is matrix multiplication, which can be mapped to the UCP implementation. Also, UCP has many other applications in logistic problems such as allocation, encoding, and routing [216]. Finite state machine minimization [214, 217] can be formulated as a binate covering problem such that the number of internal states in the finite state machine can be reduced. A standard-cell library must first comply with the available library primitives in VLSI, a process known as technology mapping. Finding the optimal mapping of logic gates to VLSI library cells can be accomplished via binate covering [213]. Boolean relations such as two-level logic minimization under the Sum-of-Product (SOP) representation can be solved based on the binate covering problem formulation [218]. Finding the minimum set of nodes or paths covering every node in a directed acyclic graph is called DAG covering. This can be formulated as a binate covering problem [215, 219, 226] by constructing the closure condition of the graph.

8.1.2 Classical Algorithms for Covering Problems

The covering problem is considered NP-hard [224, 233], and much effort has been spent on it because of its wide applications. Several classical algorithms are proposed for covering problems based on exact and heuristic algorithms. Several exact algorithms are proposed for covering problems, such as the most widely known approach, the branch and bound algorithm [225, 218, 227, 232], with many techniques suggested for lower bound and upper bound improvement using pruning techniques. In the branch and bound technique, the covering table is expressed as the POS of the constraints, and the problem solution explores, in the worst case, all possible solution instances. Branch and bound employs the upper bound and lower bound methods. For each solution to the constraints, upper bounds on the value of the cost function are identified, and lower bounds are estimated using the current set of variable assignments. The upper bound value is updated each time a new lower-cost solution is discovered. When the lower bound estimation is greater than or equal to the most recently computed upper bound, the search can be pruned. As a result, a better solution will not be found using the current variable assignments, allowing us to prune the search. Branch and bound used reduction and bounding techniques of search space for solutions to avoid the generation of some of the suboptimal solutions. The upper bound is the cost of the cheapest solution seen so far. Eventually, it will be the cost of the optimal solution. While the lower bound is an estimate of the minimum cost of a solution for the problem. The lower bound is the most important factor for runtime.

A Binary Decision Diagrams (BDDs) based algorithm [225] was proposed to solve the covering problem such that finding the solution only requires computing the shortest path in the BDD. The number of variables in the BDD is equal to the number of columns in the binate table. However, the BDD tree is too large to be built when there are many variables. A mixed technique of both branch-bound and BDD-based algorithms was proposed in [218], such that the constraints are represented as a conjunction of BDDs. This method leads to an effective method to compute a lower bound on the cost of the solution. Exact algorithms are computationally expensive for large problems because of exhaustive searches. Although there are several improvements using pruning and reduction techniques for exact algorithms, in general, the computational time can be exponential. Thus, a heuristic approach [233, 237] has been proposed for covering problems that provide suboptimal solutions. The heuristic approach in [237] has time complexity $O(n^2m)$ where n is the number of variables and m the number of terms in the covering problem. While the literature proposes both exact and heuristic approaches for classical algorithms, there is still a need for efficient algorithms that may take advantage of quantum algorithms to solve covering problems efficiently.

8.1.3 Quantum Algorithm for Covering Problem

Since classical optimization techniques are inefficient for solving NP-hard problems in terms of computational complexity, we present a quantum algorithm for solving the covering problems. To the best of our knowledge, this is the first quantum algorithm for solving covering problems. Algorithms with quantum oracles are better than the corresponding classical search algorithms because they operate using quantum

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

A hybrid algorithm (a combination of classical and quantum) can be used to solve the covering problems, which assumes an arbitrary number of terms and variables. This algorithm would be a direct generalization of the algorithm presented in this dissertation. A classical computer can use any type of heuristic or algorithmic search method to expand a search tree. The upper part of the tree is created on a classical computer using all kinds of general search strategies, heuristic functions, cost functions, parameters, and constraints such as those discussed in [238, 231, 232]. This way, the sizes of the macro-leaves of the tree are reduced step by step such that the number of terms in them is less than m and the number of variables is less than n . For each macro-leaf, a quantum computer is called and executes a full search based on Grover's algorithm. Suppose the problem in the macro-leaves with less than m terms and less than n variables is recognized as a special type of problem. In that case, a special algorithm on a classical computer is executed for this reduced tree. This is a standard method used recently by

several authors because Grover's algorithm gives a quadratic speedup only for problems for which a more efficient algorithm than a complete search does not exist. Also, this method allows for excellent scalability by sharing subtasks between the classical and quantum processors based on the availability of the quantum computer size (parameters of m terms and n variables).

The covering problem contains n variables from the given Boolean function that are used to represent the search space of $N = 2^n$ elements. To solve the covering problem in Grover's algorithm, these N elements are applied in a superposition state, which is the input to the oracle. Our proposed concept of using a quantum counter can be used to design a quantum oracle for both decision and optimization problems, such as SAT-like, MAX-SAT problems, and many other problems in machine learning, such as mining frequent pattern generation [242]. In traditional quantum oracle design for SAT-like, MAX-SAT, and covering problems, every clause is built as a multi-input Toffoli gate. The number of qubits in the multi-input Toffoli gate is equal to the number of variables in the clause plus one extra ancilla qubit to save the result of the clause. In our design, there is no need for extra ancilla qubits for each clause, but several clauses have a quantum counter which shares ancilla qubits. For instance, if there are 30 clauses, our design requires only 5 ancilla qubits for all 30 clauses, rather than the 30 ancilla qubits required in the traditional quantum oracle. Thus, our design reduces the number of qubits logarithmically. In this section will be presented the following problems.

1. Unate Covering Problems:

- a. Finding all prime implicants for minimal covering of a SOP circuit.

2. Binate Covering Problem:

- a. Finding the minimum covering for an implication graph.
- b. Finding minimum cost constraint.
- c. Minimization of finite state machines

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

Circuit

Minterms for a Boolean function of n variables are products of literals for all variables of the function. A literal is a variable or the negation of a variable. True minterms are those for which the value of the function is 1. The prime implicant in a sum-of-product (SOP) structure is a product of literals that cannot be extended by removing some of the literals and which covers some subset of true minterms. For instance, given a function from Figure 8.1a, all its prime implicants are marked as ovals (loops). Using the minterm compatibility graph G , all prime implicants are found as maximum cliques. Prime implicants can also be found as maximum independent sets of graphs (G complement). Based on the truth table (or a Karnaugh Map) and the prime implicants, the covering table from Figure 8.1b is created. In this table, every row represents a prime implicant as a product of Boolean literals, and each column represents a true minterm. The covering table is filled with symbol X for every minterm included in a prime implicant.

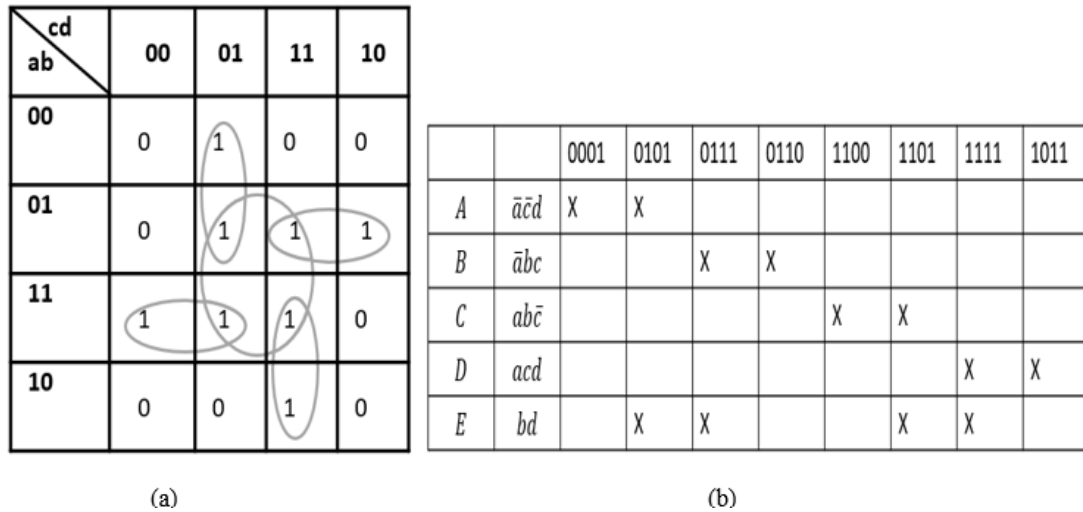


Figure 8.1 (a) Truth table in form of a Karnaugh Map for all prime implicants of a SOP circuit. (b)

Covering table for SOP function from (a).

From the table in Figure 8.1b, denoting rows A, B, C, D, E we compile the Petrick function in a standard way such that each column is created as one term by adding the variables in the row corresponding with symbol X. For instance, column 0101 has two cells filled with X. Adding the two rows A + E as one term that corresponds to the symbols X in column 0101. In such a way, equation 8.1 is created:

$$A(A + E)(B + E)BD(D + E)(C + E)C = 1 \tag{8.1}$$

Equation 8.1 can be simplified using the Boolean law: $A(A + E) = A \cdot A + A \cdot E = A(1 + E) = A$ to the following equation: $1 = A \cdot B \cdot C \cdot D$. Therefore, $f = A + B + C + D = \bar{a}\bar{c}d + \bar{a}bc + acd + ab\bar{c}$ is the minimum sum of products of expression of function f . In the case of many variables and clauses, solving this problem exactly is very difficult.

The main goal of this example is to show how SOP minimization can be solved using the UCP. Then the quantum oracle is built from UCP to apply Grover's algorithm. The quantum oracle is built from the 8.1 equation. First, we need to convert each OR term into a product using De Morgan's Law $A + E = \overline{\overline{A + E}} = \overline{\overline{A} \cdot \overline{E}} = 1 \oplus \overline{A} \overline{E}$; $B + E = \overline{\overline{B + E}} = \overline{\overline{B} \cdot \overline{E}} = 1 \oplus \overline{B} \overline{E}$; $C + E = \overline{\overline{C + E}} = \overline{\overline{C} \cdot \overline{E}} = 1 \oplus \overline{C} \overline{E}$; $D + E = \overline{\overline{D + E}} = \overline{\overline{D} \cdot \overline{E}} = 1 \oplus \overline{D} \overline{E}$. From 8.1 equation we can rearrange $A \cdot B \cdot C \cdot D(A + E)(B + E)(D + E)(D + E)(C + E)$. To simplify the oracle design, we consider $A \cdot B \cdot C \cdot D$ as one term, which needs one quantum circuit. Then we connect one block of the iterative quantum counter after each Toffoli gate representing the OR term of the function POS formula. We put the ancilla qubit back to its original state by mirroring each Toffoli gate after each counter. In this case, we need five quantum counter circuits, as can be seen in Figure 8.2. Also, we need to add the NOT gate in the output circuit block, which makes the last qubit out_0 to produce 1 if the variables xyz in the counter circuit are 101 such that the Boolean function in equation 8.1 is equal to 1.

In Figure 8.3, we applied the oracle circuit in Figure 8.2 in the Grover's search algorithm for iterations $R = 4$ from this formula: $R \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil$ where $N = 2^5 = 32$ is the number of all search space elements since there are 5 variables for A, B, C, D, E . $M = 2$ is the number of solutions. $M = 2$ because in equation 8.1 is equal to 1 either $ABCDE = 11110$ or $ABCDE = 11111$. We run the circuit on the 'qasm_simulator' from QISKIT for 1024 shots (independent runs to get high precision probability) which the circuit produces the correct answers. We measured a_0, a_1, a_2, a_3 and a_4 in Figure 8.3 where $a_0,$

a_1, a_2, a_3, a_4 correspond to the Boolean variables A, B, C, D, E respectively. As can be seen in Figure 8.4, the diagram illustrates the QISKIT [125] output graphics for the simulated circuit. The measured values $a_4 a_3 a_2 a_1 a_0$ with high probability are 11111, 01111. The values 11111 and 01111 correspond to E, D, C, B, A respectively which are two solutions to equation 8.1: A, B, C, D and A, B, C, D, E .

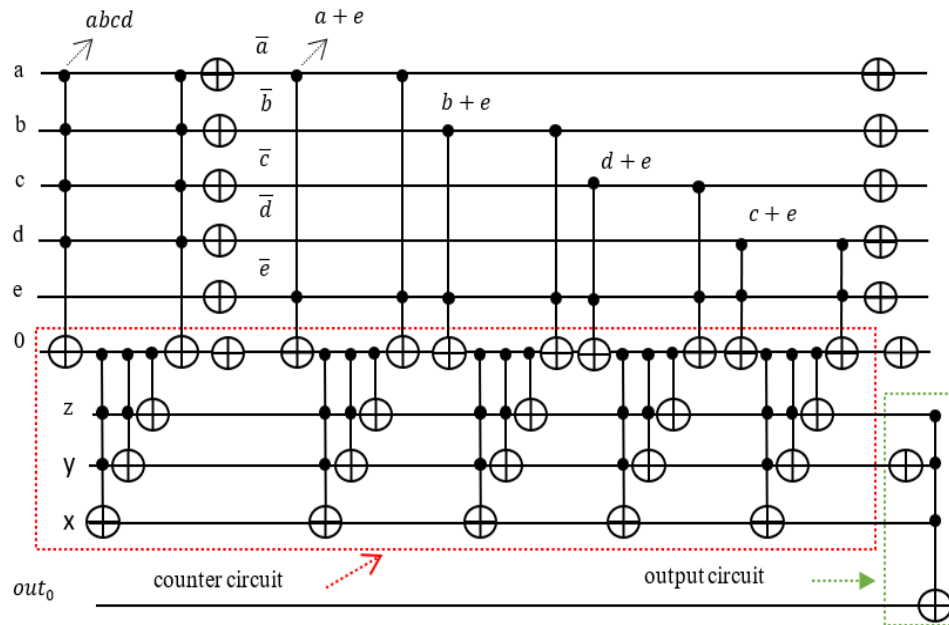


Figure 8.2 Quantum oracle for $f = A \cdot B \cdot C \cdot D(A + E)(B + E)(D + E)(C + E)$

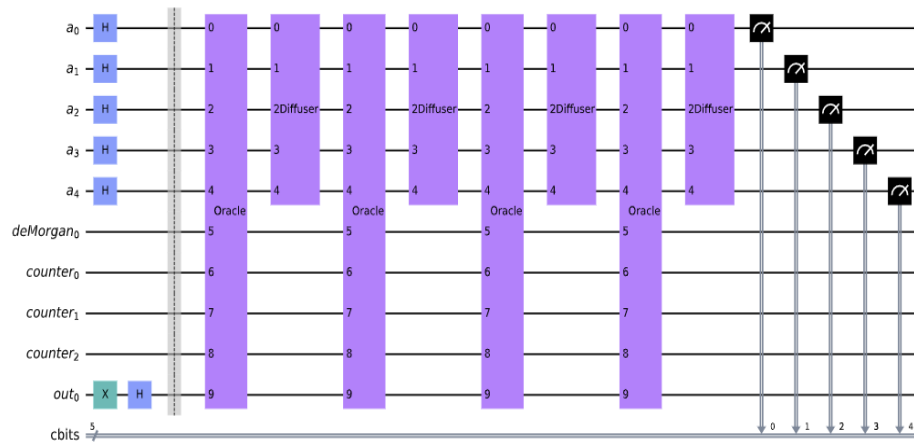


Figure 8.3 Grover's algorithm with 4 iterations using the oracle circuit from Figure 8.2

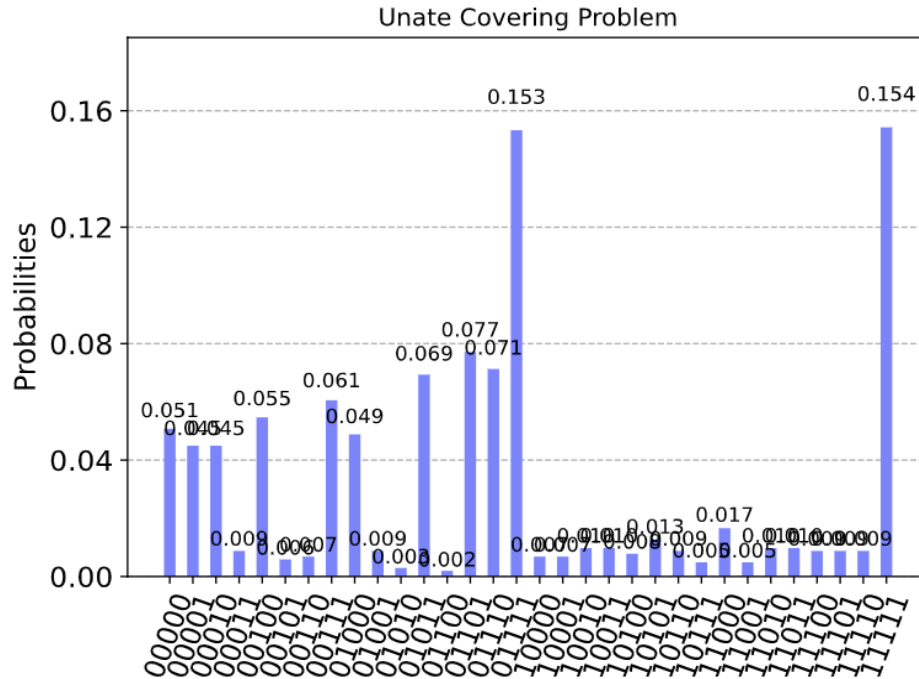


Figure 8.4 Measurement of the Boolean variables from Figure 8.3

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

8.1.5 Finding the Minimum Covering for an Implication Graph

An implication graph is a directed acyclic graph (DAG) where each node represents a variable assignment. An implication graph represents the implication relations between pairs of variable assignments. Given a set S and a family of subsets $F = \{s_1, s_2, \dots, s_n\}$, a closure conditions are represented as an implication graph. For instance, assuming given a family of subsets of the set $\{1,2,3,4,5,6\}$ and the implication graph from in Figure 8.5,

the optimization task is to select a subset of nodes from the set A, B, C, D, E that satisfies all three conditions:

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

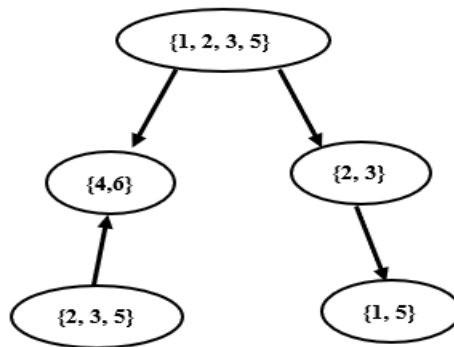


Figure 8.5 Implication graph for the set $\{1,2,3,4,5,6\}$

The general optimization can be solved by constructing a covering and closure table. For a particular problem, as specified above, the table is shown in Figure 8.6. Here, the rows correspond to the nodes (subsets) of the implication graph in Figure 8.5, while the columns correspond to the individual elements of the set $\{1,2,3,4,5,6\}$ for covering table and the columns for the closure table correspond to the nodes in the implication graph which are the same subsets as the rows of the table.

| | 1 | 2 | 3 | 4 | 5 | 6 | A{1,2,3,5} | B{4,6} | C{2,3} | D{1,5} | E{2,3,5} |
|------------|---|---|---|---|---|---|------------|--------|--------|--------|----------|
| A{1,2,3,5} | X | X | X | | X | | | ● | ● | | |
| B{4,6} | | | | X | | X | | | | | |
| C{2,3} | | X | X | | | | | | | ● | |
| D{1,5} | X | | | | X | | | | | | |
| E{2,3,5} | | X | X | | X | | | ● | | | |

Covering table
Closure table

Figure 8.6 Covering-closure table based on the implication graph from Figure 8.5

The closure conditions are illustrated in Figure 8.5. Assuming that we select set $A = \{1,2,3,5\}$ on top of the implication graph, it is implied that the set $B = \{4,6\}$ and the set $C = \{2,3\}$ must be also selected. Set $\{2,3\}$ implies set $\{1,5\}$ and set $\{2,3,5\}$ implies set $\{4,6\}$ (see Figure 8.5 above). This way, the table from Figure 8.6 is created. For instance, the black dots in the row A mean that set $A = \{1,2,3,5\}$ implies sets $B = \{4,6\}$ and $C = \{2,3\}$. Set $C = \{2,3\}$ implies sets $D = \{1,5\}$. Set $E = \{2,3,5\}$ implies sets $B = \{4,6\}$. Based on the covering and closure table from Figure 8.6, Petrick's method creates a Boolean formula in equation 8.2. Next, this formula is transformed into the general POS formula from equation 8.3.

$$(A + D)(A + C + E)B(A + D + E)(A \Rightarrow BC)(C \Rightarrow D)(E \Rightarrow B) = 1 \quad (8.2)$$

In equation 8.2, the first four terms describe the covering conditions, and the last three terms correspond to closure conditions (closure constraints). Equation 8.2 can be converted to equation 8.3 by using the logic transformation rule $(A \Rightarrow B) \Leftrightarrow (\bar{A} + B)$:

$$f = (A + D)(A + C + E)B(A + D + E)(\bar{A} + BC)(\bar{C} + D)(\bar{E} + B) \quad (8.3)$$

Next , we build a quantum oracle for the general POS formula from equation 8.3 by converting each OR term into Products using De Morgan's law $A + D = \overline{\overline{A + D}} = \overline{\overline{A} \overline{D}} = 1 \oplus \overline{A} \overline{D}$; $A + C + E = \overline{\overline{A + C + E}} = \overline{\overline{A} \overline{C} \overline{E}} = 1 \oplus \overline{A} \overline{C} \overline{E}$; $A + D + E = \overline{\overline{A + D + E}} = \overline{\overline{A} \overline{D} \overline{E}} = 1 \oplus \overline{A} \overline{D} \overline{E}$; $\overline{A} + BC = \overline{\overline{\overline{A} + BC}} = \overline{\overline{A} \overline{BC}} = 1 \oplus \overline{A} \overline{BC} = 1 \oplus \overline{A} (1 \oplus BC)$; $\overline{C} + D = \overline{\overline{\overline{C} + D}} = \overline{\overline{\overline{C}} \overline{D}} = 1 \oplus \overline{C} \overline{D}$; $\overline{E} + B = \overline{\overline{\overline{E} + B}} = \overline{\overline{\overline{E}} \overline{B}} = 1 \oplus \overline{E} \overline{B}$. Then we connect one block of the iterative quantum counter after each Toffoli gate, representing the OR term of the function POS formula. We put the ancilla qubit back to its original state by mirroring each Toffoli gate after each counter. In this case, we need seven quantum counter circuits (one for B term and six for each POS term), as can be seen in Figure 8.7. The oracle circuit in Figure 8.7 is inserted into Grover's Oracle. This is similar to the previous examples in that it demonstrates one more time how inefficient is using the global AND gate in quantum, even for very small practical binate covering problems. Rather than using AND, we use a quantum counter.

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

In Figure 8.8, we applied the oracle circuit from Figure 8.7 in Grover's search algorithm for iterations $R = 2$ from this formula: $R \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil$ where $N = 2^5 = 32$ and $M = 5$ (this can be verified by a truth table). We run the circuit on the 'qasm_simulator' from QISKIT for 1024 shots, and the circuit produces the correct answers. We measured

a_0, a_1, a_2, a_3 and a_4 in Figure 8.8 where a_0, a_1, a_2, a_3, a_4 corresponds to the Boolean variables A, B, C, D, E respectively. As can be seen in the histogram in Figure 8.9, this illustrates the QISKIT [125] output graphics for the simulated circuit. The measured values $a_4 a_3 a_2 a_1 a_0$ with high probability are 01110, 01111, 11010, 11110, and 11111 corresponding to variables E, D, C, B, A respectively. For instance, the vector 01110 corresponds to the solution of DCB . In the measurement, the solutions have much higher probabilities than the non-solutions. These solutions are verified outside of Grover's algorithm, just by using the oracle with function $(A + D)(A + C + E)B(A + D + E)(\bar{A} + BC)(\bar{C} + D)(\bar{E} + B)$.

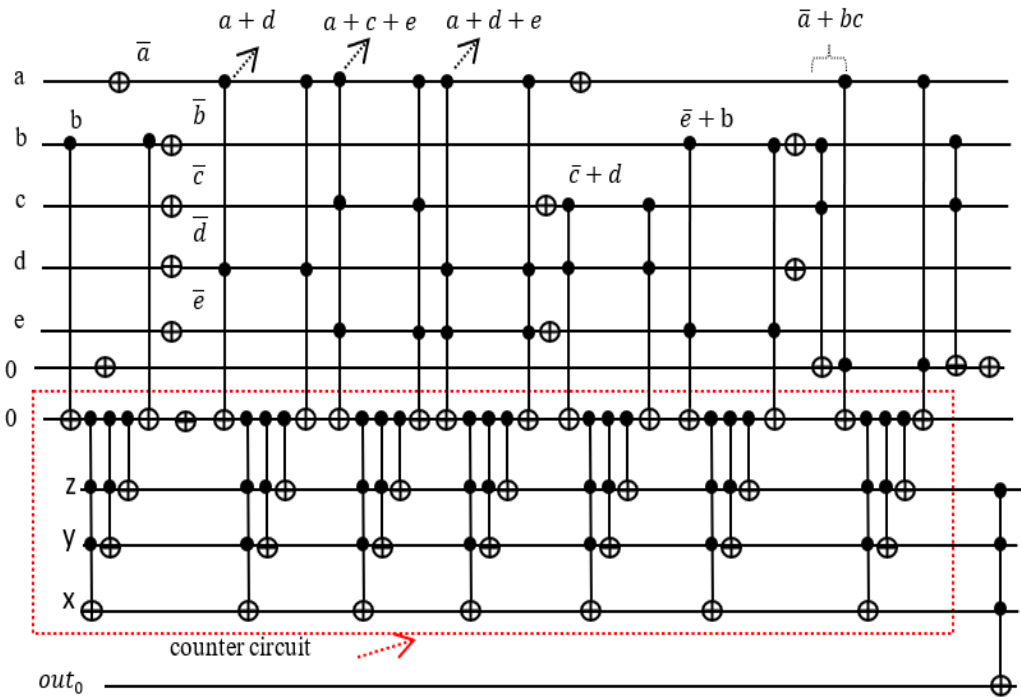


Figure 8.7 Quantum oracle for solving the Implication Graph Problem for $(A + D)(A + C + E)B(A + D + E)(\bar{A} + BC)(\bar{C} + D)(\bar{E} + B)$

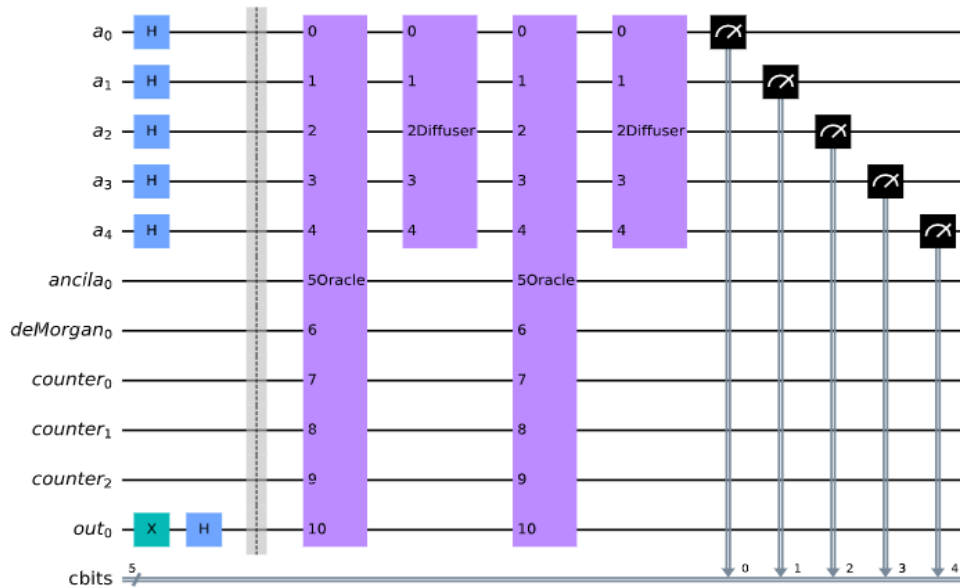


Figure 8.8 Grover's algorithm with 2 iterations using the oracle circuit.

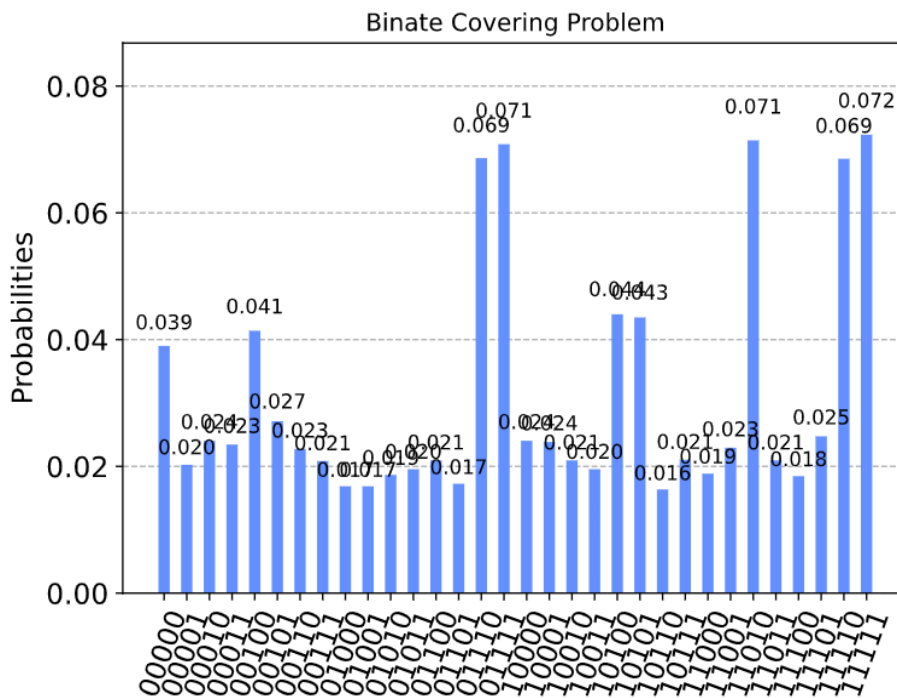


Figure 8.9 Measurement of the Boolean variables from $(A + D)(A + C + E)B(A + D + E)(\bar{A} + BC)(\bar{C} + D)(\bar{E} + B)$

8.1.6 Finding Minimum Cost Constraint

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

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

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

where $w_i \geq 0$ is the weight of variable x_i and y_m is a clause which means a sum of x_i . This is called a weighted binate covering. The practical application of finding the minimum cost constraint can be found in [230], which can be formulated as a binate covering problem. However, we present in this Chapter a general example of a Boolean function that can be solved using Grover's algorithm. For instance, given is the following set of clauses Y :

$$y_0 = x_0 + x_3$$

$$y_1 = x_2 + \bar{x}_3$$

$$y_2 = x_1 + \bar{x}_2$$

$$y_3 = x_1 + x_2 + x_3$$

and let the weight w_i for each value of x_i as follow: $w_0 = 4$, $w_1 = 2$, $w_2 = 1$, and $w_3 = 1$. The optimization task is to find the minimum cost assignment based on the given weight. First, we construct the POS function, and then we design a quantum oracle for the

Grover's algorithm to find the exact minimum solution. Based on the solution, we apply the weight for each clause to find the minimum cost constraint. Here the Y clauses are represented by a POS Boolean formula:

$$(x_0 + x_3)(x_2 + \bar{x}_3)(x_1 + \bar{x}_2)(x_1 + x_2 + x_3) \quad (8.4)$$

The solution found in Grover's algorithm is subject to the equation 8.5 for the minimum cost function:

$$4x_0 + 2x_1 + x_2 + x_3 \quad (8.5)$$

Equation 8.5 is an arithmetic function that is computed in a classical processor of a hybrid computer, while equation 8.4 is computed in Grover's algorithm. First, we build a quantum oracle from equation 8.4, similar to the previous examples, after applying De Morgan's law: $x_0 + x_3 = \bar{x}_0 \cdot \bar{x}_3 \oplus 1$; $x_2 + \bar{x}_3 = \bar{x}_2 \cdot x_3 \oplus 1$; $x_1 + \bar{x}_2 = \bar{x}_1 \cdot x_2 \oplus 1$; $x_1 + x_2 + x_3 = \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \oplus 1$.

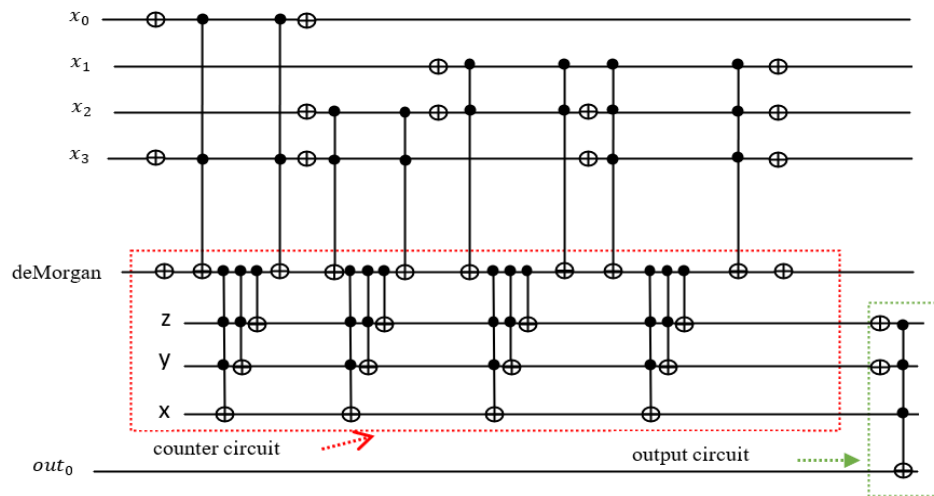


Figure 8.10 Quantum oracle for the Binate Covering Problem with $(x_0 + x_3)(x_2 + \bar{x}_3)(x_1 + \bar{x}_2)(x_1 +$

We need to add the two NOT gates in the output circuit block, which makes the last qubit out_0 to produce one if the variable xyz in counter circuit is 100, such that the Boolean function in equation 8.4 is equal to 1.

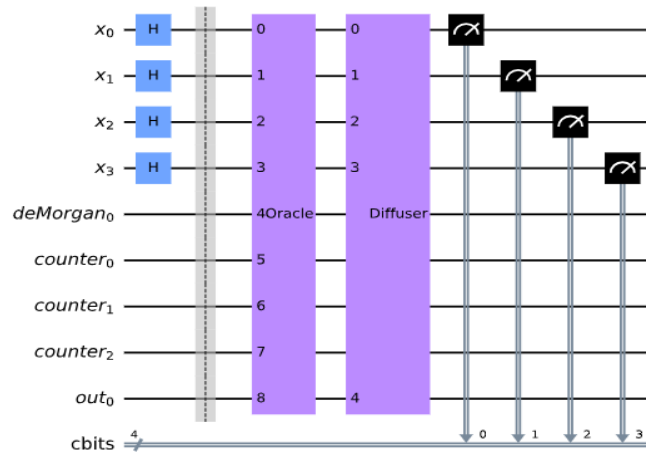


Figure 8.11 Grover's algorithm with 2 iterations using the oracle circuit.

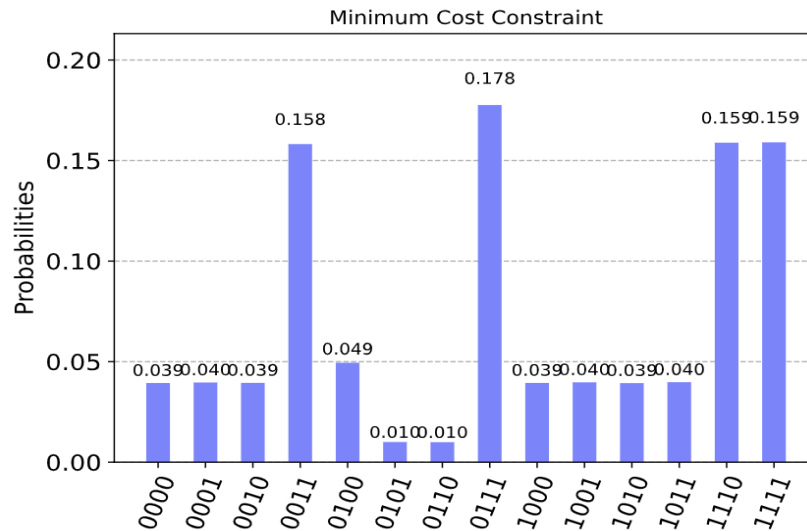


Figure 8.12 Measurement of the Boolean variables from Binate Covering Problem $(x_0 + x_3)(x_2 + \bar{x}_3)(x_1 + \bar{x}_2)(x_1 + x_2 + x_3)$

In Figure 8.11, we applied the oracle circuit from Figure 8.10 with the diffuser operator discussed in Figure 3.11. Also, it can be used with the other diffuser from Figure

3.12. We need $R = 2$ for Grover's iterations from this formula: $R \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil$ where $N = 2^4 = 16$ is the number of all search space elements. $M = 4$ is the number of solutions that can be verified by creating a truth table. We run the circuit on the 'qasm_simulator' from QISKIT for 1024 shots, and the circuit produces the correct answers. We measured $x_3x_2x_1x_0$ in Figure 8.11. As can be seen in Figure 8.12, the measured values $x_3x_2x_1x_0$ with high probability are {1111,1110,0111,0011}. Applying these values to equation 8.4, we found 1110, which gives the minimum cost of 4.

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

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

8.1.7 Minimization of Incompletely Specified Finite State Machines

A Finite State Machine (FSM) is an abstract model used in design to model problems in various fields of science and engineering. A finite state machine consists of input states, output states, and internal states that can change from one state to another state based on

input. The change of the internal state is described by a transition function. For various reasons, finite state machines can have incompletely specified output functions and transition functions. The minimization of incompletely specified finite state machines is considered an NP-hard problem [217]. We present the complete oracle design for Grover's algorithm for the well-known classical problem of minimization of the number of states of incompletely specified finite state machines. For a given incompletely specified finite state machine, the solution is achieved by the following steps:

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

Below we will illustrate the above general hybrid algorithm on a particular example. Given is an incompletely specified Mealy finite state machine described as a transition/output table from Figure 8.13a. Dashes represent don't care in internal states or output states. This table has internal states A, B, C, D, E, F , two input signals for columns, and one binary output under a slash symbol in cells of the map. A triangular table in Figure 8.13b was generated based on the table from Figure 8.13a. The table from Figure 8.13b covers all possible cases to minimize the number of states in the finite state machine. The 'X' symbol in the table indicates no possibility for grouping the corresponding states. Symbol 'V' in the table indicates that the states can be combined without any problem. A pair of state variables in a cell of the triangular map V indicates that states can be grouped only if the states mentioned in the block can be combined without any problem. For instance, states B and F can be combined under the condition that states C, F are compatible (can be combined). The method of creating the triangular table is well-known from [222].

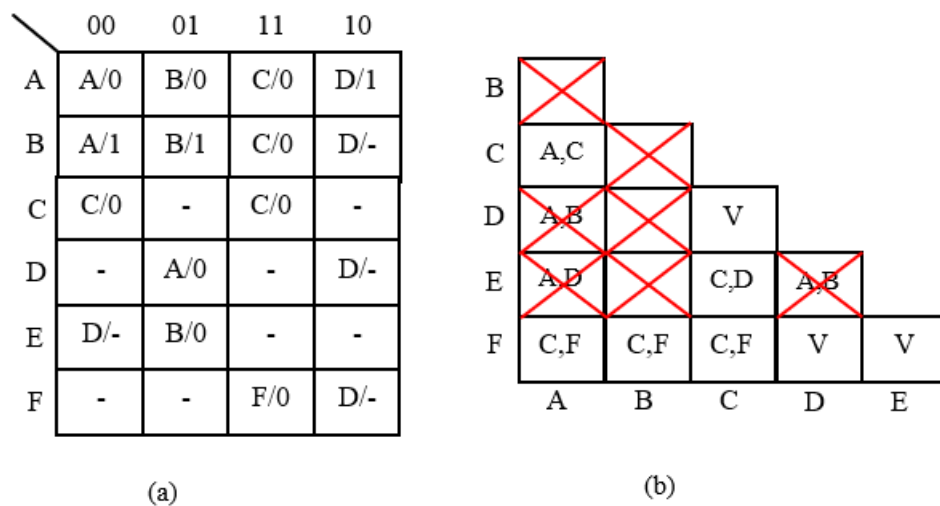


Figure 8.13 (a) Truth table of FSM (b) FSM triangular table generated based on the table from (a).

Based on the triangular table, a compatibility graph for the state machine is generated, as shown in Figure 8.14a. Every cell in the triangular map that has no symbol X corresponds to a compatible pair of states. For instance, the cell at the intersection of row C and column A is compatible. The cell at the intersection of row B and column A is not compatible as it has a symbol X in it. Similarly, states A and D are not compatible. In Figure 8.13b, states A and D are compatible under the condition that states A and B are compatible. But we found earlier that states A and B are not compatible; thus, states A and D are not compatible. The cell at the intersection of row D and column A is crossed-out as symbol X . In the same way, the cell at the intersection of column A and row E is replaced with X because states A and D are not compatible. A simple recursive classical algorithm creates symbols X for every incompatible pair of internal states.

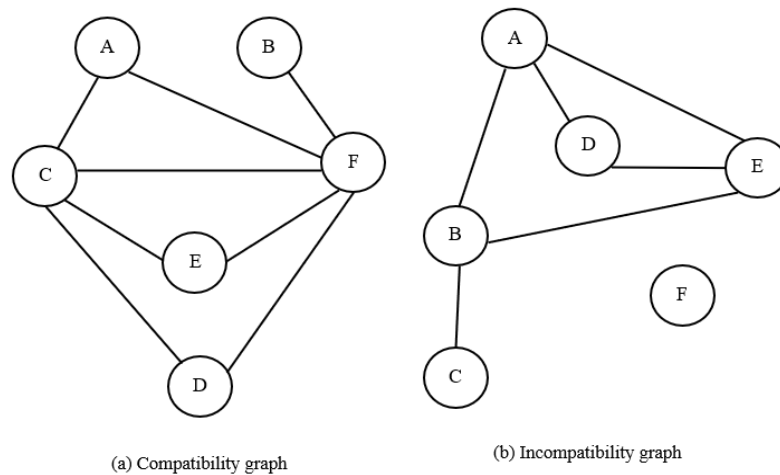
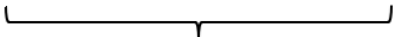


Figure 8.14 (a) Compatibility graph (b) Incompatibility graph.

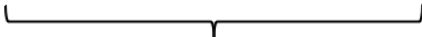
From Figure 8.14a, the maximum cliques of the graph are identified as $\{A, C, F\}$, $\{B, F\}$, $\{C, D, F\}$, $\{C, E, F\}$. Finding of all cliques in a graph is done by a SAT-based algorithm similar to those discussed earlier in this paper. The compatibility graph from

Figure 8.14a is a complement of the incompatibility graph from Figure 8.14b. As we see, the maximum cliques in the compatibility graph are the same as the maximum independent sets in the incompatibility graph from Figure 8.14b. These maximum independent sets can be found from the graph coloring [243] of the incompatibility graph. The graph coloring can be solved by a special Grover's oracle. It can be solved by finding the Maximum Independent Sets and then solving the unate covering problem with them. These problems are reducible to SAT-like oracles for Grover's algorithm. This example explains the relation between the graph coloring of the incompatibility graph, finding the maximum cliques of the compatibility graph, and covering problems. These partial quantum algorithms are also useful in a quantum algorithm for solving the Ashenurst-Curtis Decomposition [131].

| | A | B | C | D | E | F | V{A,C} | Q{C,F} | R{C,D} |
|----------|---|---|---|---|---|---|--------|--------|--------|
| X{A,C,F} | X | | X | | | X | ● | ● | |
| Y{B,F} | | X | | | | X | | ● | |
| Z{C,D,F} | | | X | X | | X | | ● | |
| U{C,E,F} | | | X | | X | X | | ● | ● |
| V{A,C} | X | | X | | | | ● | | |
| P{A,F} | X | | | | | X | | ● | |
| Q{C,F} | | | X | | | X | | ● | |
| R{C,D} | | | X | X | | | | | |
| S{D,F} | | | | X | | X | | | |
| T{C,E} | | | X | | X | | | | ● |
| W{E,F} | | | | | X | X | | | |



Covering table



Closure table

Figure 8.15 Covering-Closure table for the FSM.

To minimize the finite state machine, a covering-closure table, shown in Figure 8.15, is created by considering the maximum cliques and all their subsets as rows of the table. All of the states $\{A, B, C, D, E, F\}$ in the machine correspond to columns of the covering table, and the implications $\{A, C\}$, $\{C, F\}$, $\{C, D\}$ in the compatibility graph from the cell of the triangular table correspond to columns of the closure table. From the table in Figure 8.15, a binate covering problem can be specified using the equation:

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

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

$$(X + V + P)Y(X + Z + U + V + Q + R + T)(Z + R + S)(U + T + W)(X + Y + Z + U + P + Q + S + W)(\bar{X} + VQ)(\bar{Y} + Q)(\bar{Z} + Q)(\bar{U} + QR)(\bar{V} + V)(\bar{P} + Q)(\bar{Q} + Q)(\bar{T} + R) = 1 \quad (8.6)$$

The number of search space $N = 2^{11} = 2048$ where 11 is the number of variables in the rows of covering-closure table. There are 155 solutions that the Boolean equation 8.6 is equal to 1. The presented method is not yet practical as contemporary quantum computers have not enough qubits. However, with a sufficient number of qubits, the presented algorithm will allow to minimize large machines with quadratic speedup. To visualize all these solutions in a histogram is difficult such that we use a more general case in Figure 8.16, we repeat the Grover's algorithm for iterations $R = 3$ with tuning values of thresholds until equal to counter value. The comparator $G = H$ compares the

output from the counter with the threshold value given as constant values n_1, n_2, n_3 and n_4 . (Using the threshold with a comparator has many other applications such as finding the minimum set of support [126]).

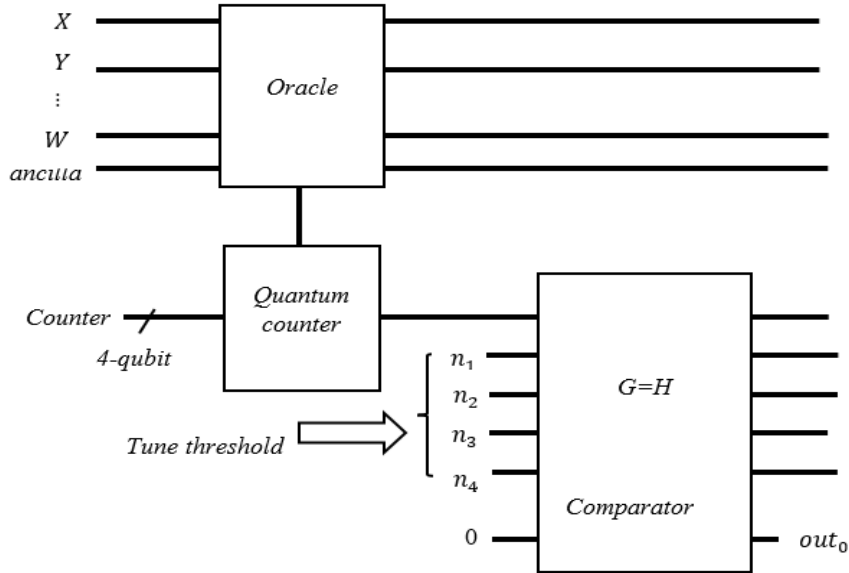


Figure 8.16 FSM Oracle Design with counter circuit and threshold with comparator.

The solution with the minimum number of positive literals is $\{V, Q, R, Y, U\}$ which simplifies to $\{V, R, Y, U\}$ because group $Q = \{C, F\}$ is included in $U = \{C, E, F\}$, thus $Q \Rightarrow U$. Symbol V requires combining states A and C , symbol Y requires combining states B and F , symbol R combines states C and D and symbol U combines states C, E, F together. As the result, we obtain the minimized state machine from Figure 8.17a. We combine states from the respective states of Figure 8.13. Thus, combining in column 00 for row V we obtain symbols A and C and output 0. This way, combining states from groups V, Y, R and U the entire table from Figure 8.17a is created. Now for every subset of initial states A, B, C, D, E, F corresponding to each symbol from set of sets $\{V, Y, R, U\}$ we check to

which set this subset belongs. For instance, state C is included in sets V, R and U . Therefore symbol C in the table from Figure 8.17a is replaced with symbols V, R and U in the transition cells. This way, the non-deterministic machine from Figure 8.17b is created. Now select any state among V, R and U to create one of the deterministic machines described by the non-deterministic machine. Choose every row U in column 11 in order to improve the logic realization of the machine. Similarly, for the purpose of good encoding (encoding not explained here), select state R in column 00 to have two transitions to V and two transitions to R in this column. One final finite state machine minimized in this way is shown in Figure 8.17c.

a)

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

b)

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

c)

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

Figure 8.17 Steps to create an exactly minimized deterministic FSM using binate covering problem. (a) the table created directly from the solution to the covering-closure problem; (b) a non-deterministic automaton created from the table in (a); (c) one deterministic automaton in (b)

There are not yet benchmarks for quantum algorithms. However, there exist benchmarks for classical algorithms, such as those in [231, 232, 239, 240]. The current

quantum computers are too small to run the classical benchmarks on them. One can, however, speculate on the speedup of future quantum and hybrid computers based on these classical benchmarks. Suppose a benchmark takes m terms and n variables, using our method, this benchmark would require n qubits for variables and $\lceil \log_2 m \rceil + 1$ ancilla qubits for terms to represent the problem in a quantum algorithm design. In contrast, the traditional quantum oracle design would require n qubits for variables and $m + 1$ ancilla qubits for terms [185]. Thus, when compared to the traditional Grover, our proposed design requires fewer qubits with a quadratic speedup of Grover's algorithm. Assuming a complete search, the complexity of the classical algorithm would be $N = 2^n$. The complexity of our quantum algorithm would be $O(\sqrt{N})$. When quantum computers have enough qubits, comparing practical benchmarks will be possible. Because IBM aims to build a quantum computer with 100,000 qubits in 10 years [241], we hope that in this time frame, our quantum algorithm for EDA problems will become practical.

8.2 Quantum oracles based on arithmetic for constraint satisfaction and optimization problems

The class of problems we presented are logic problems, some associated with arithmetic operations and solved using classical methods. Arithmetic problems are very important in both classical and quantum computing. For instance, given an arithmetic equation 8.7, we are supposed to find the values of the variables a and b :

$$\begin{cases} a + b = 5 \\ a * b = 6 \end{cases} \quad (8.7)$$

Can we design a quantum oracle to solve the given arithmetic equation and then find a and b ? To answer this question, let's study some arithmetic operations such as addition and multiplication to build a quantum oracle that will apply to Grover's search algorithm to find a and b . Let's first address the quantum oracle design for a full adder for addition and then extend to multiplication.

8.2.1 Classical Half and Full Adders

In digital circuit design, a half adder and a full adder are fundamental building blocks that are used in arithmetic circuits. A half adder performs the addition of two binary digits: two inputs (A and B) and produces a sum (S) and carry output (C_{out}). A full adder performs the addition of three binary digits: two inputs (A and B) and carry input (C_{in}). It produces a sum (S) and carry output (C_{out}). Figure 8.18 and Figure 8.19 are half adder and full adder logic circuit with truth table.

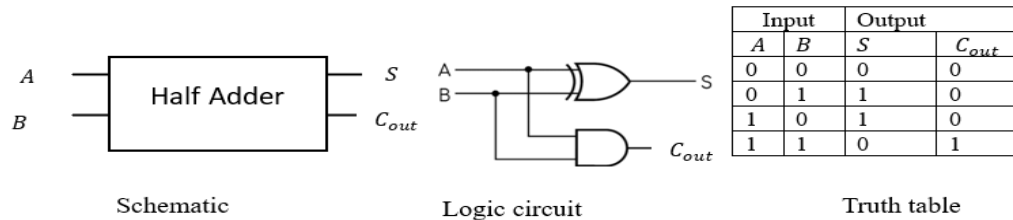


Figure 8.18 Half adder logic circuit with truth table.

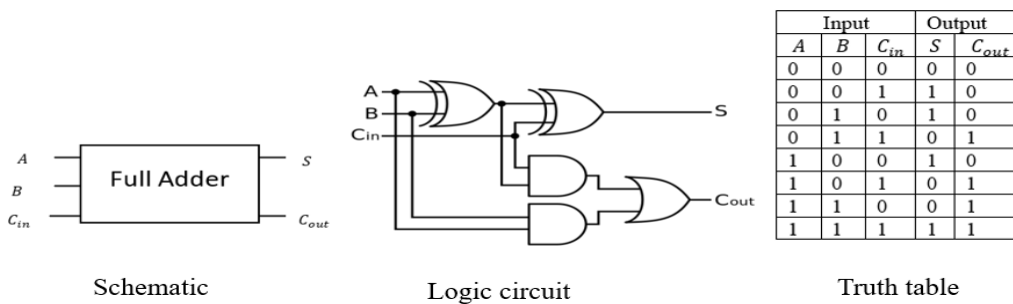


Figure 8.19 Full adder logic circuit with truth table.

A full adder is extended to a ripple carry adder, which consists of stages of full adders, and the carry-out becomes the carry-in of the next full adder through each stage.

8.2.2 Quantum Half and Full Adders

Classical full adder in Figure 8.19 can be easily converted to corresponding quantum gates, which are NOT, CNOT, and Toffoli gates. However, many versions of the quantum full adder proposed [244, 245, 246] tried to minimize the quantum resources, such as the number of quantum gates and qubits. The proposed technique in [244] performs the quantum addition such that the rotations are controlled by one input, and a second input is used as a target, a technique that is very similar to the quantum Fourier transform. Ancilla qubits are not used in this technique. However, for the output at the end of the computation, one of the inputs would be able to reverse, and the second input would be used as the sum of the two numbers. Assume $a + b = 5$, and the task is to find a and b . The issue with this method is that the sum of the two values and one of a or b will be found at the end of the computation. A quantum ripple adder was proposed in [245, 246], in which the circuits are built from NOT, CNOT, and Toffoli gates. These approaches minimized the number of qubits at the cost of a high number of Toffoli gates, and still, one of the inputs is used as the sum of the output, a similar issue in [244]. Another ripple carry adder was proposed in [247] that uses a single Toffoli gate and 6 CNOT gates with one ancilla qubit in the full adder, as can be seen in Figure 8.20. As we presented in Table 7.5, the Toffoli gate is a high-cost gate in determining the total quantum circuit cost; even a single Toffoli gate in a full adder became high-cost because the ripple carry adder consists of full adders.

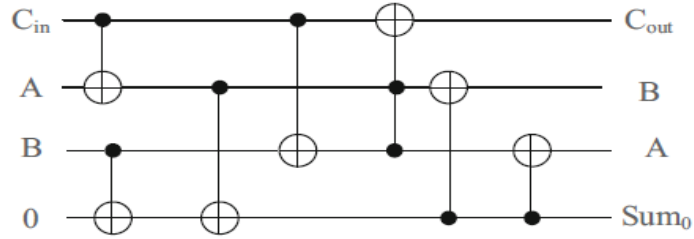


Figure 8.20 Ripple carry adder proposed in [247]

An efficient quantum circuit design for full adder design was proposed in [250, 251] that does not need any Toffoli gates; thus, the quantum circuit is small in terms of quantum cost, especially the number of gates. The circuit design of the full adder in [250, 251] is constructed using only the controlled-nth root of NOT gate (V gate) and CNOT gates, as can be seen in Figure 8.21

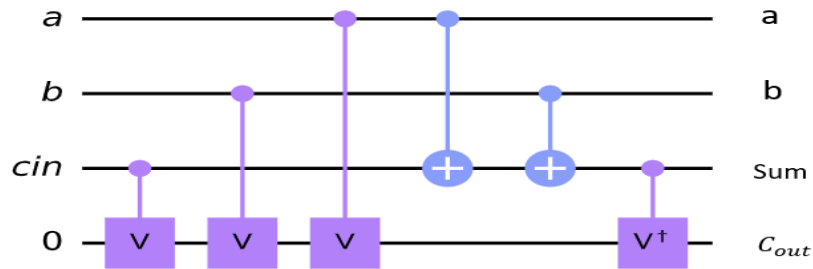


Figure 8.21 Quantum circuit design for full adder proposed in [250]

Table 8.1 Detailed operations for Figure 8.21

| Input | | | Output | |
|-------|-----|----------|--------|---|
| A | B | C_{in} | Sum | C_{out} |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | $VV^\dagger \Rightarrow 0$ |
| 0 | 1 | 0 | 1 | $VV^\dagger \Rightarrow 0$ |
| 0 | 1 | 1 | 0 | $VV = NOT \Rightarrow 1$ |
| 1 | 0 | 0 | 1 | $VV^\dagger \Rightarrow 0$ |
| 1 | 0 | 1 | 0 | $VV = NOT \Rightarrow 1$ |
| 1 | 1 | 0 | 0 | $VV = NOT \Rightarrow 1$ |
| 1 | 1 | 1 | 1 | $VVVV^\dagger = NOT VV^\dagger = NOT \Rightarrow 1$ |

NOTE:

- V gate = \sqrt{NOT} gate.
- V^\dagger gate is inverse of the V gate which called conjugate of V .
- $VV^\dagger = I$ (identity, means no operation).
- $VV \Rightarrow NOT$ gate.

Table 8.1 shows detailed operations for Figure 8.21. The V or V^\dagger gate is activated only when the control of the V or V^\dagger gate is one. For instance, when a, b or C_{in} is one, then the V gate is activated. When both V and V^\dagger are active, then they cancel each other, and no operation is applied on the qubit (0). When two of V 's are active, then they operate as NOT gate. The full adder can be extended as a ripple carry adder, and the carry-out becomes the carry-in of the next full adder through each stage. For instance, Figure 8.22 shows a quantum circuit for a 3-bit ripple carry adder where the sum bits are $S_0S_1S_2$ and the carry out is C_{out} .

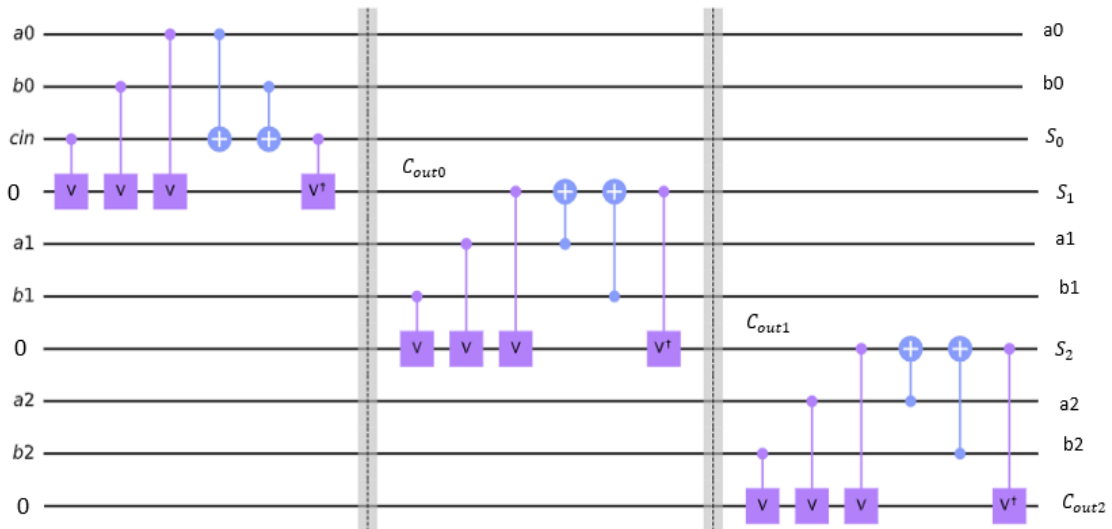


Figure 8.22 3-bith ripple carry adder.

The quantum half adder circuit in [250, 251] is built using only the controlled- n th root of NOT gate (V gate) and CNOT gates, as can be seen in Figure 8.23.

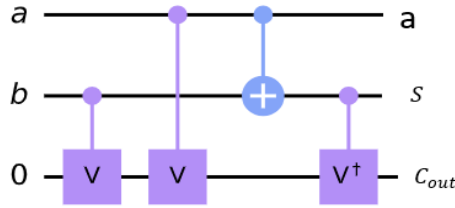


Figure 8.23 Quantum circuit for half adder in [251]

8.2.3 Comparison analysis

We need to analyze the quantum circuit designs regarding the number of gates and qubits required to trade off different designs.

Table 8.2 Comparison analysis for quantum circuit

| Circuit | # qubits | Gates | | | | Total cost |
|------------------------|----------|----------|-----------|----------|----------------|------------|
| | | # CNOT | # Toffoli | # NOT | # \sqrt{NOT} | |
| Cuccaro et al. [245] | $2n + 2$ | $5n - 3$ | $2n - 1$ | $2n - 2$ | 0 | $17n - 10$ |
| Takahashi et al. [246] | $2n + 1$ | $5n$ | $2n - 1$ | 0 | 0 | $15n - 5$ |
| Gayathri et al. [247] | $3n + 1$ | $6n$ | n | 0 | 0 | $11n$ |
| [250, 251] | $3n + 1$ | $2n$ | 0 | 0 | $4n$ | $6n$ |

Where n is the number of bits to be added, as can be seen in Table 8.2, the trade-off is between the number of qubits and the total quantum circuit cost. The approach [250, 251] has the lowest quantum cost and can be used to solve arithmetic addition problems ($a + b = 5$) because, at the measurement stage, we keep the inputs, sums, and final carry-out at the end of the computation, as can be seen in Figure 8.22. Separation of inputs, sums, and final carry-out is useful when solving problems related to Boolean constraint satisfaction (weighted sums: weighted MAX-SAT and partial weighted MAX-SAT) and

objective functions associated with cost. This method in [250, 251] is also useful in machine learning, such as when we need to tune the weights of a neural network. Also, the proposed ripple carry adder circuit design can be used for the Wallace tree multiplier [248], a high-speed multiplier that uses a full adder and half adder to add partial products.

In digital computers, the Wallace multiplier is considered the fastest multiplication circuit other than traditional multipliers [249]. In digital design, the Wallace tree multiplier multiplies two integers that employ carry save addition using a full adder and a half adder. There are several stages to go through to get the final product of the multiplication. The first stage of the Wallace tree multiplier is partial multiplication. The second stage is partial product reduction using half adders and full adders. The third stage is an addition to get a complete product. For instance, we need to multiply 4-bit number $x_3x_2x_1x_0$ and $y_3y_2y_1y_0$ that generates partial products $P_7P_6P_5P_4P_3P_2P_1P_0$, as can be seen in Figure 8.24.

| | | | | | | | | |
|-------|----------|----------|----------|----------|----------|----------|----------|--|
| | | | | x_3 | x_2 | x_1 | x_0 | |
| | | | \times | y_3 | y_2 | y_1 | y_0 | |
| | | | | | | | | |
| | | | | x_3y_0 | x_2y_0 | x_1y_0 | x_0y_0 | |
| | | | x_3y_1 | x_2y_1 | x_1y_1 | x_0y_1 | | |
| | | x_3y_2 | x_2y_2 | x_1y_2 | x_0y_2 | | | |
| | x_3y_3 | x_2y_3 | x_1y_3 | x_0y_3 | | | | |
| | | | | | | | | |
| P_7 | P_6 | P_5 | P_4 | P_3 | P_2 | P_1 | P_0 | |

Figure 8.24 Multiplication 4-bit number $x_3x_2x_1x_0$ and $y_3y_2y_1y_0$

We can analyze each stage separately. In the first stage, we add three rows of partial products, as can be seen in Figure 8.25. We need two Half adders and two full adders that generate $x_0y_0, s_0, c_0, s_1, c_1, s_2, c_2, s_3, c_3$, and x_3y_2 . For instance, adding x_1y_0 and x_0y_1 produces the sum s_0 and the carry-out c_0 . Also x_2y_0, x_1y_1 , and x_0y_2 produce the sum s_1 and the carry-out c_1 . The rest goes the same way.

| | | | | | | |
|--|----------|----------|----------|----------|----------|----------|
| | | | x_3y_0 | x_2y_0 | x_1y_0 | x_0y_0 |
| | | x_3y_1 | x_2y_1 | x_1y_1 | x_0y_1 | |
| | x_3y_2 | x_2y_2 | x_1y_2 | x_0y_2 | | |
| | | | | | | |
| | x_3y_2 | s_3 | s_2 | s_1 | s_0 | x_0y_0 |
| | c_3 | c_2 | c_1 | c_0 | | |

Figure 8.25 First stage of partial products.

In the second stage, we add the fourth row and outputs of the first stage, as can be seen in Figure 8.26. We need this stage one half adder and three full adders that generates $x_0y_0, s_0, s_4, s_5, s_6, s_7, c_4, c_5, c_6, c_7$, and x_3y_3 . For instance, adding s_1 and c_0 generates the s_4 and c_4 .

| | | | | | | | |
|--|----------|----------|----------|----------|-------|-------|----------|
| | | x_3y_2 | s_3 | s_2 | s_1 | s_0 | x_0y_0 |
| | | c_3 | c_2 | c_1 | c_0 | | |
| | x_3y_3 | x_2y_3 | x_1y_3 | x_0y_3 | | | |
| | | | | | | | |
| | x_3y_3 | s_7 | s_6 | s_5 | s_4 | s_0 | x_0y_0 |
| | c_7 | c_6 | c_5 | c_4 | | | |

Figure 8.26 Second stage of partial products.

In the final stage, we add the outputs of the second such that we need one half adder, and three full adders that generates $x_0y_0, s_0, s_4, s_8, s_9, s_{10}, s_{11}$, and c_{11} corresponding $P_0, P_1, P_2, P_3, P_4, P_5, P_6$, and P_7 respectively, as shown in Figure 8.27.

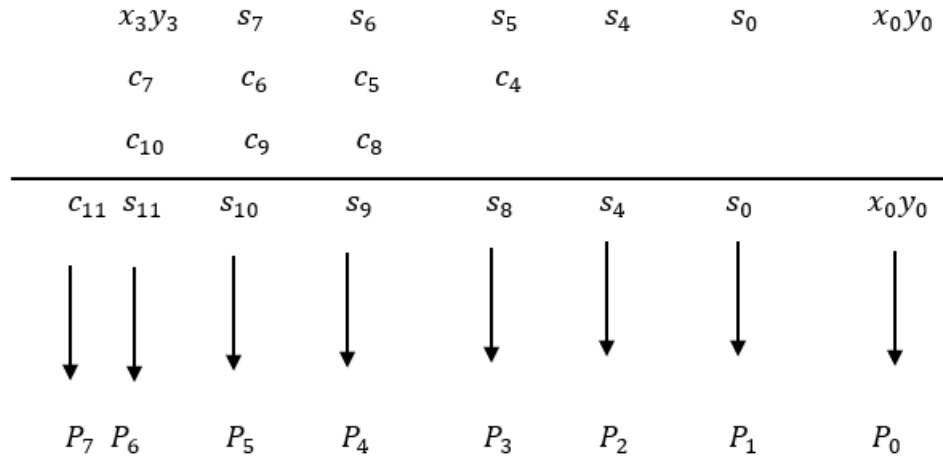


Figure 8.27 Final stage of partial products.

8.2.4 Quantum Wallace Multiplier

A quantum circuit design for the Wallace multiplier was proposed in [247, 250, 251], in which the full adder is the main building block to perform the multiplication. It is wise to consider an efficient design that reduces the quantum cost of circuit design. We chose the half adder and the full adder in Figures 8.23 and 8.21, respectively, to construct a quantum circuit for the Wallace tree multiplier. We use 4-bit multiplication to illustrate the concept of the Wallace tree multiplier. To generate the partial product in the first stage, we use Toffoli gates, as can be seen in Figure 8.28.

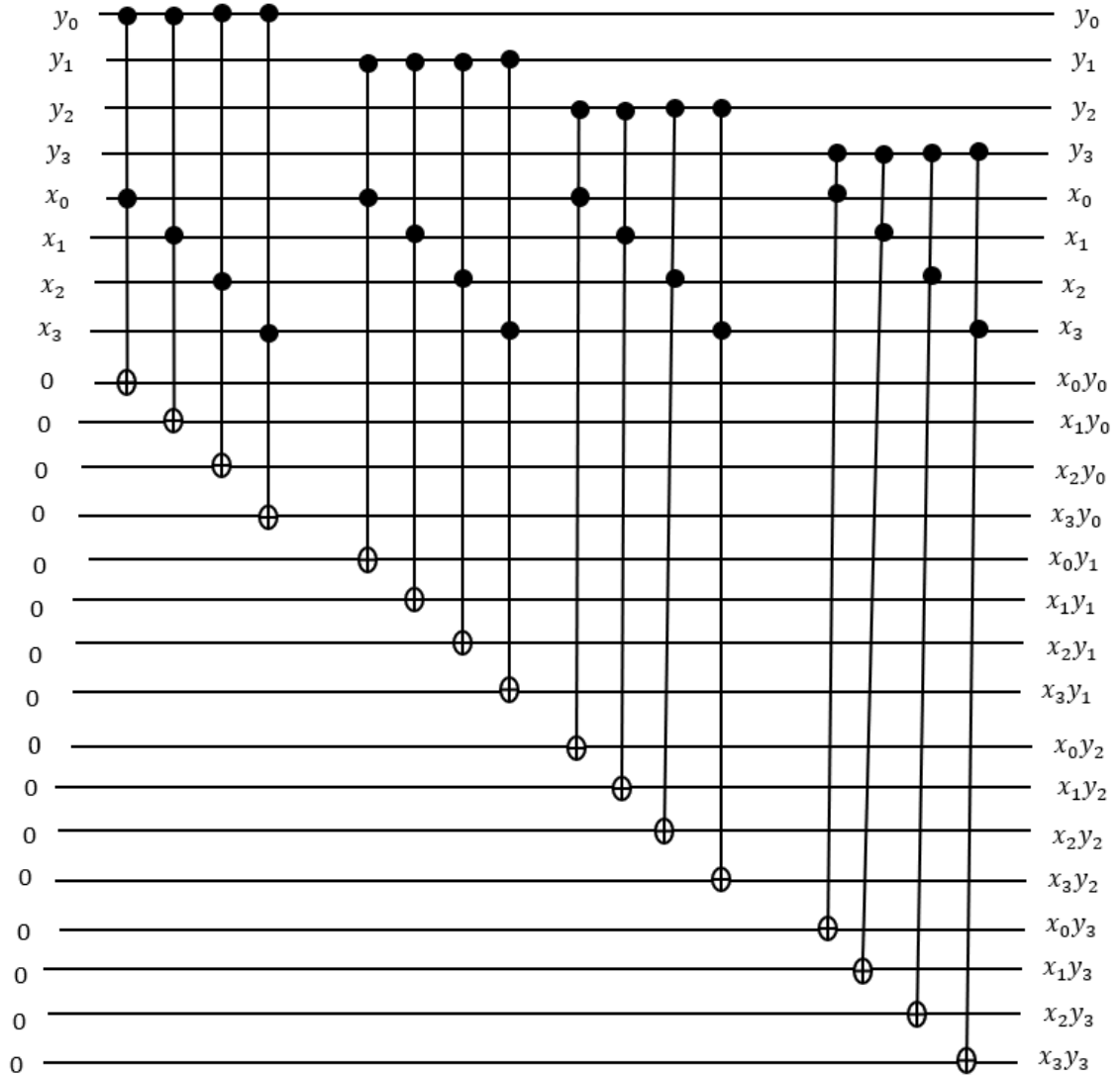


Figure 8.28 Quantum circuit to generate the partial products in the first stage.

The partial products $x_0y_0 \dots x_3y_3$ in the first stage in Figure 8.28 are used as input to the schematic circuit in Figure 8.29. We design the quantum circuit of the Wallace multiplier in stages in the same way as the classical method, as shown in Figure 8.29.

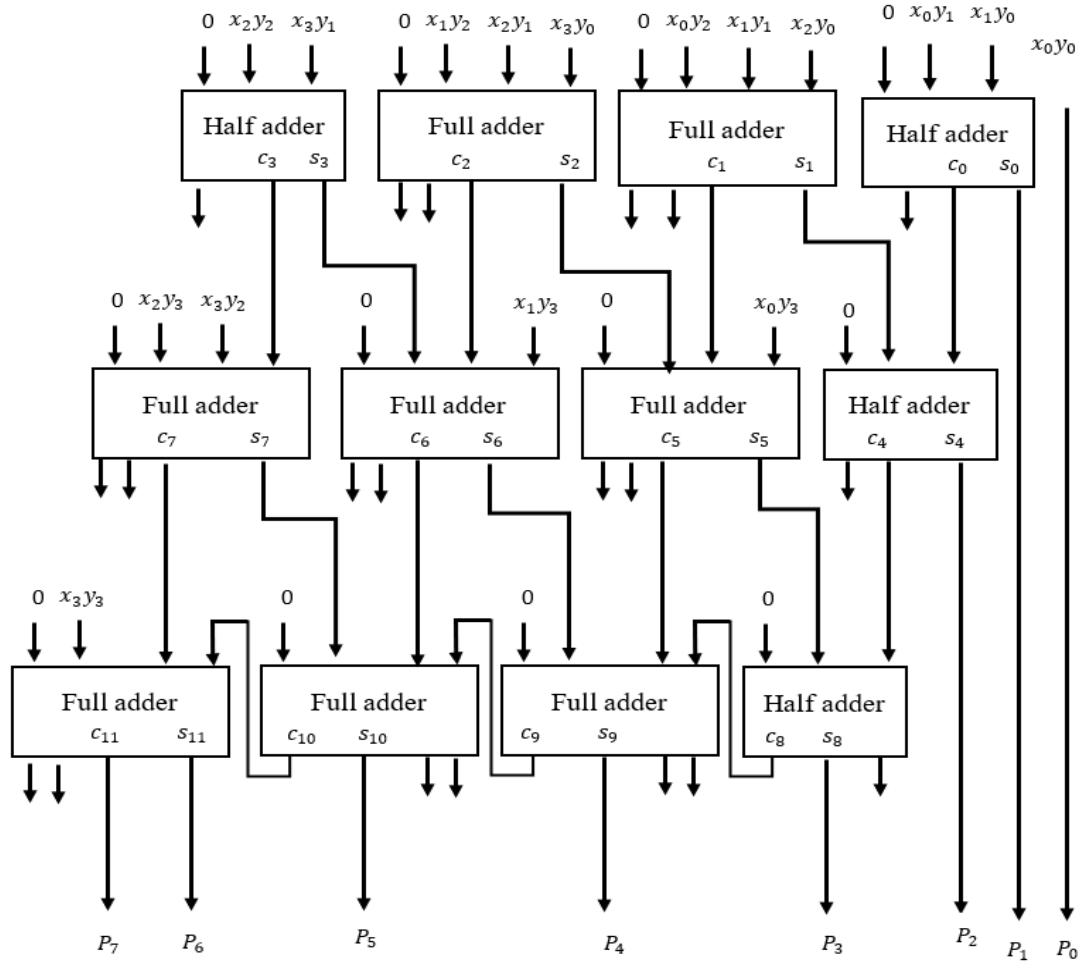


Figure 8.29 Schematic for quantum circuit design of Wallace tree multiplier.

We constructed the quantum circuit for addition and multiplication separately so that we can design a quantum oracle circuit to solve a and b for the equation 8.7:

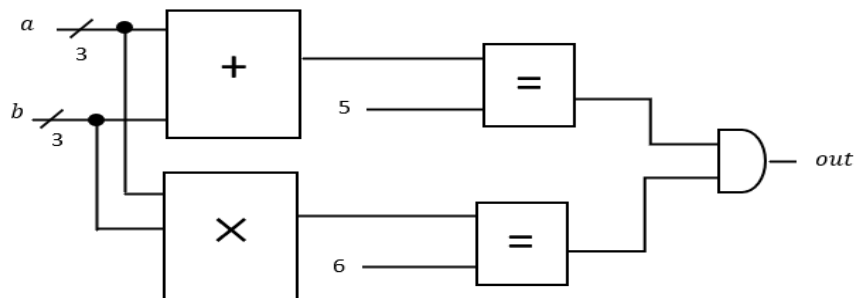


Figure 8.30 Quantum oracle schematic for solving a and b

In Figure 8.30, assume a and b are three bits. For the addition equation, we use the quantum circuit design in Figure 8.22, and we need 9 qubits. For the multiplication equation, we use the design in Figure 8.29. We need 9 qubits for Figure 8.28 for the first stage and 6 qubits for the Wallace multiplier for three half adders and three full adders. For the equality comparator, we need 9 and 13 qubits for addition and multiplication, respectively. Finally, 1 qubit for the *out*. Thus, the total number of qubits needed to solve the equation is 47 qubits for only a 3-bit value. As we can see, solving arithmetic equations is expensive in terms of the number of qubits and quantum gates.

The goal of the arithmetic equation we presented is to show that we can design arithmetic quantum oracles and apply Grover's search algorithm to solve the arithmetic equations. The class of arithmetic problems is more general than the class of logic problems that we have been solving so far. One class of problems that can be done is $TWO + TWO = FOUR$ and similar problems, but solving this problem is time-consuming. Another class of problems are arbitrary equations and optimization problems involving equations, inequalities, and complicated cost functions. They give a lot of freedom for circuit and algorithm design inside oracle optimization. Just to show the concept in a simple example, I selected a problem with addition and multiplication. This simple example points out the wide category of problems that can be solved using Grover's algorithm. For instance, we presented the SAT and MAX-SAT problems and used a quantum counter to count the satisfied number of terms. When the terms are associated with a weight of an integer value other than one, then we need to design the quantum counter using an arithmetic adder. We presented a class of logic problems that

can be combined with quantum arithmetic to solve a large class of problems in both logic and arithmetic.

8.3 Logic Puzzles

One of the main challenges of classical computing, both theoretical and practical, is solving logic puzzles. Logic puzzles are studied in mathematics and computer science to prove computational complexity. Logic puzzles are used as entertainment to develop concentration and strengthen cognitive reasoning. In a logic puzzle, given a set of rules and constraints, the task is to satisfy the given conditions, which may have one solution or many solutions. There are many different types of logical puzzles, such as sudoku, masyu, nonogram, and calcudoku. Logic puzzles are characterized by a grid that needs to be filled to find the solution based on the rules and constraints. Sudoku is one of the main types of problems belonging to combinatorial and constraint satisfaction problems. Thus, we use it as a case study in this dissertation. For instance, the classical sudoku puzzle is a logic puzzle game played on a grid filled with digit numbers from 1 to 9. There is also binary sudoku, in which the grid is filled with only 0 and 1. The classical complexity of $n \times n$ sudoku is considered NP-hard [252].

There are many classical algorithms and techniques for solving sudoku puzzles, such as the genetic algorithm [253], backtracking [254], and brute force search [255]. The primary purpose of these techniques is to speed up the computation time for sudoku solvers. One of the main advantages of quantum computing is that it allows for faster computation of complex problems. Sudoku can be solved using quantum search

algorithms to leverage the advantage of quantum computing. A binary sudoku [256, 257, 258] is an $n \times n$ grid that satisfies the following conditions:

1. Each entry is filled with either zero or one.
2. There are no three consecutive ones or three consecutive zeros in each row and column.
3. The number of ones and zeros are balanced so that in every row and column, the number of ones and zeros are equal.
4. Every row and every column must be distinct.

To satisfy these conditions, the binary sudoku can be encoded into SAT format in the form of CNF, which is product-of-sum (POS). However, this encoding creates a large number of clauses, which can lead to an exponential number of clauses. The performance of SAT solvers depends on the number of CNF clauses.

8.3.1 Quantum binary sudoku

I proposed an efficient quantum oracle design that can solve the binary sudoku such that all possible solutions can be found in one run of Grover's search algorithm. The binary sudoku constraints are encoded into satisfiability terms based on the given rule. The proposed SAT-like terms are the product of sum-of-XOR (PSOX), a format different from other formats discussed in Chapter 4. First, we converted each rule for each row and column into PSOX format and then we constructed the oracle using a nested quantum counter. Second, we applied the oracle to Grover's search algorithm. We present a complete quantum oracle design for 4×4 binary sudoku to illustrate our architecture.

Table 8.3 Binary sudoku 4×4

| | | | |
|----------|----------|----------|----------|
| x_0 | x_1 | x_2 | x_3 |
| x_4 | x_5 | x_6 | x_7 |
| x_8 | x_9 | x_{10} | x_{11} |
| x_{12} | x_{13} | x_{14} | x_{15} |

In Table 8.3, given 4×4 binary sudoku where x_i is a binary value of 0 or 1. To satisfy all constrains, we design the quantum oracle based on each condition as follows:

Condition 1: x_i is binary value of 0 or 1.

Condition 2: No three consecutive 1 or 0. We design the oracle as a product of sum-of-XOR (PSOX) terms for each row and column as flow:

a) Row terms:

$$(x_0 \oplus x_1 + x_1 \oplus x_2)(x_1 \oplus x_2 + x_2 \oplus x_3)$$

$$(x_4 \oplus x_5 + x_5 \oplus x_6)(x_5 \oplus x_6 + x_6 \oplus x_7)$$

$$(x_8 \oplus x_9 + x_9 \oplus x_{10})(x_9 \oplus x_{10} + x_{10} \oplus x_{11})$$

$$(x_{12} \oplus x_{13} + x_{13} \oplus x_{14})(x_{13} \oplus x_{14} + x_{14} \oplus x_{15})$$

b) Column terms:

$$(x_0 \oplus x_4 + x_4 \oplus x_8)(x_4 \oplus x_8 + x_8 \oplus x_{12})$$

$$(x_1 \oplus x_5 + x_5 \oplus x_9)(x_5 \oplus x_9 + x_9 \oplus x_{13})$$

$$(x_2 \oplus x_6 + x_6 \oplus x_{10})(x_6 \oplus x_{10} + x_{10} \oplus x_{14})$$

$$(x_3 \oplus x_7 + x_7 \oplus x_{11})(x_7 \oplus x_{11} + x_{11} \oplus x_{15})$$

To optimize the number of qubits, we use one ancilla qubit to save the computation. For instance, $x_0 \oplus x_1 + x_1 \oplus x_2$, the x_1 is used two times, so the XOR operation result in $x_0 \oplus x_1$ is saved in one extra ancilla qubit (r) and the XOR operation in $x_1 \oplus x_2$ is computed without ancilla qubit. The output of $x_0 \oplus x_1 + x_1 \oplus x_2$ is used as the first qubit (d) for the output counter which also is used as De Morgan's Law to convert the OR into a product ($a + b = \overline{\overline{a} \cdot \overline{b}} = \overline{a} \cdot \overline{b} \oplus 1$), as can be seen in Figure 8.31. The rest of the terms for condition two are constructed using the same method.

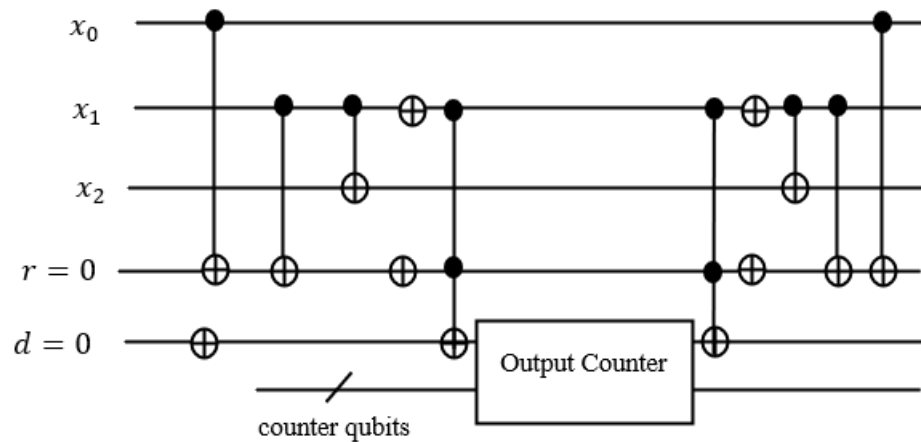


Figure 8.31 Oracle circuit for $x_0 \oplus x_1 + x_1 \oplus x_2$

Condition 3: Balance the number of ones and zeros in every row and column. This condition counts the number of ones in every row and column. We connect every cell in the table as the first qubit to a counter (we call this counter a balance counter), and then the output of the balance counter is connected to another counter (output counter) that counts all conditions. In this case of the 4×4 binary sudoku, the number of ones in each row and each column are equals two. If all rows and columns are balanced, then the output of the balance counter is 16. We check separately for every row and column, then

cascade the output of the balance counter from all rows and columns such that the output of the balance counter result is 2, 4, 6, and 8 for rows 1, 2, 3, and 4 respectively, and 10, 12, 14, 16 for columns 1, 2, 3 and 4 respectively. For instance, if the row $x_0x_1x_2x_3$ is balanced then $balance_counter_0$, $balance_counter_1$, $balance_counter_2$, $balance_counter_3$, and $balance_counter_4$ is equal to 00010 respectively. To count this row as a satisfied row, we need to change 00010 to 11111 by applying NOT gates and then connect to the output counter that counts all satisfied conditions, as can be seen in Figure 8.32. Figure 8.33 is a complete oracle to check the balance of each row and each column.

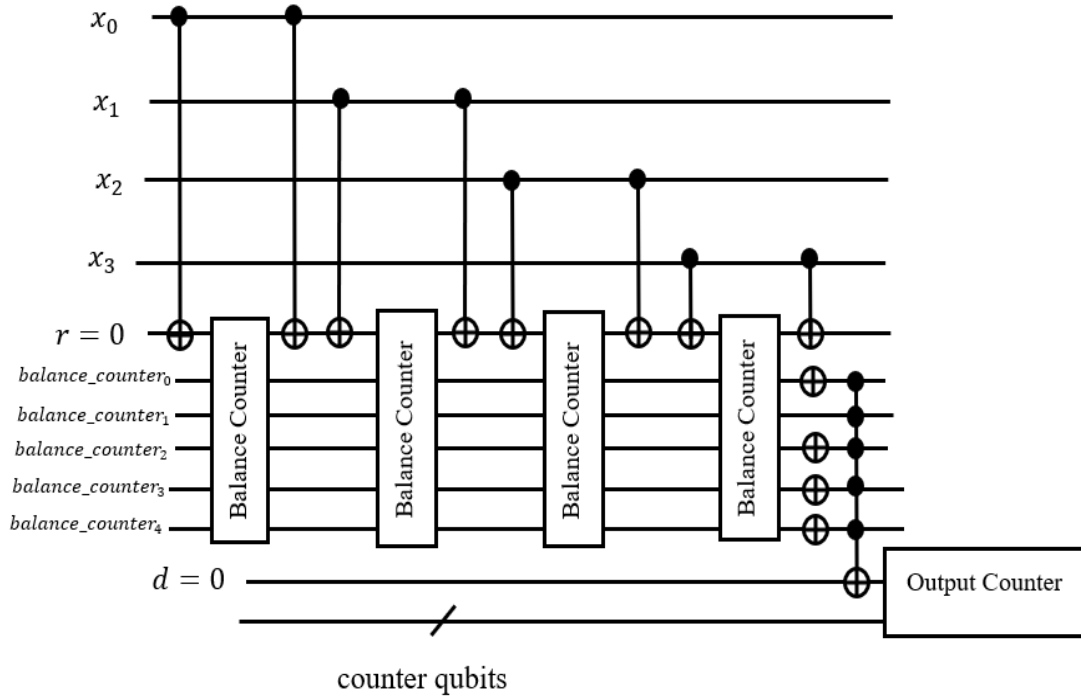


Figure 8.32 Oracle balance circuit for row $x_0x_1x_2x_3$

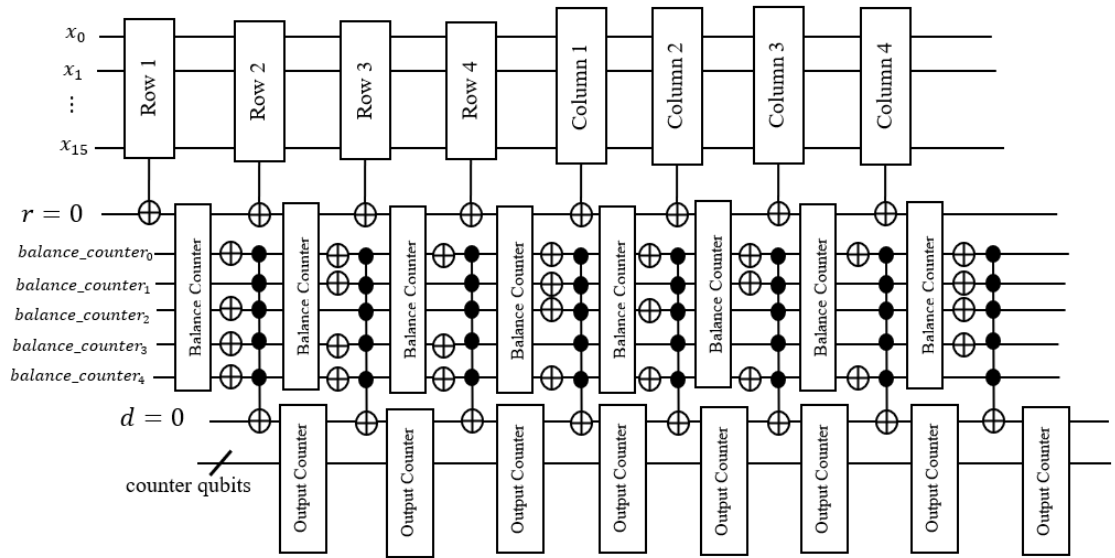


Figure 8.33 Oracle balance circuit for all rows and columns.

Condition 4: Every row and every column must be distinct. Similar to condition 2, we design the oracle using PSOX format as follows:

a) Row terms:

$$(x_0 \oplus x_4 + x_1 \oplus x_5 + x_2 \oplus x_6 + x_3 \oplus x_7)$$

$$(x_0 \oplus x_8 + x_1 \oplus x_9 + x_2 \oplus x_{10} + x_3 \oplus x_{11})$$

$$(x_0 \oplus x_{12} + x_1 \oplus x_{13} + x_2 \oplus x_{14} + x_3 \oplus x_{15})$$

$$(x_4 \oplus x_8 + x_5 \oplus x_9 + x_6 \oplus x_{10} + x_7 \oplus x_{11})$$

$$(x_4 \oplus x_{12} + x_5 \oplus x_{13} + x_6 \oplus x_{14} + x_7 \oplus x_{15})$$

$$(x_8 \oplus x_{12} + x_9 \oplus x_{13} + x_{10} \oplus x_{14} + x_{11} \oplus x_{15})$$

b) Column terms:

$$(x_0 \oplus x_1 + x_4 \oplus x_5 + x_8 \oplus x_9 + x_{12} \oplus x_{13})$$

$$(x_0 \oplus x_2 + x_4 \oplus x_6 + x_8 \oplus x_{10} + x_{12} \oplus x_{14})$$

$$(x_0 \oplus x_3 + x_4 \oplus x_7 + x_8 \oplus x_{11} + x_{12} \oplus x_{15})$$

$$(x_1 \oplus x_2 + x_5 \oplus x_6 + x_9 \oplus x_{10} + x_{13} \oplus x_{14})$$

$$(x_1 \oplus x_3 + x_5 \oplus x_7 + x_9 \oplus x_{11} + x_{13} \oplus x_{15})$$

$$(x_2 \oplus x_3 + x_6 \oplus x_7 + x_{10} \oplus x_{11} + x_{14} \oplus x_{15})$$

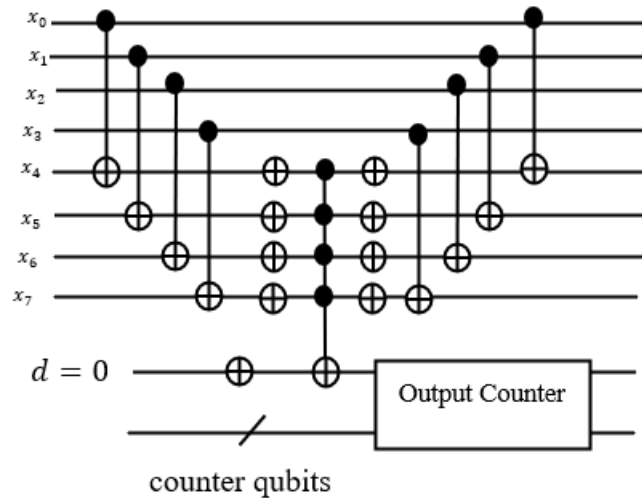


Figure 8.34 Oracle circuit for term $x_0 \oplus x_4 + x_1 \oplus x_5 + x_2 \oplus x_6 + x_3 \oplus x_7$

To illustrate the oracle for condition 4, we use the term $x_0 \oplus x_4 + x_1 \oplus x_5 + x_2 \oplus x_6 + x_3 \oplus x_7$ as an example to construct the quantum oracle, as can be seen in Figure 8.34, and the rest of the terms in condition four can be constructed using the same method. The total number of terms that need to be satisfied for all conditions is 36, such as 16 terms for condition 2, 8 for condition 3, and 12 for condition 4. As can be seen in Figure 8.35, we easily check either by checking the oracle output qubit (*out*) that is equal to one when

all terms for conditions 2, 3, and 4 are satisfied or $ouput_counter_0$, $ouput_counter_1$, $ouput_counter_2$, $ouput_counter_3$, $ouput_counter_4$, and $ouput_counter_5$ are equal to 100100.

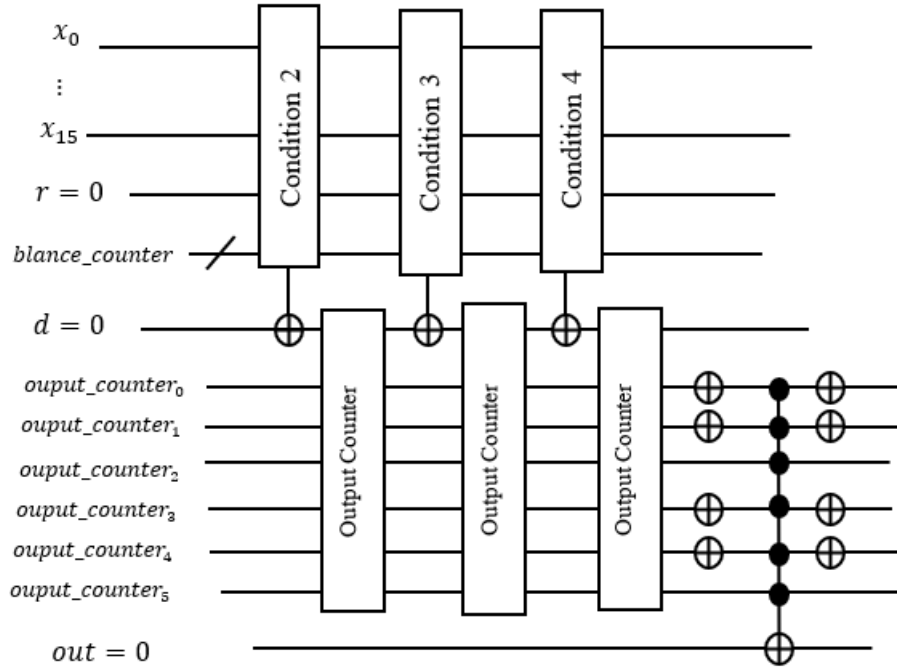


Figure 8.35 Oracle circuit for 4×4 binary sudoku.

To solve 4×4 binary sudoku in this design, we need a total of 30 qubits for quantum oracle. To find a complete solution for all 16 cells in 4×4 binary sudoku, we need 2^{16} superposition states that the current simulators for IBM qiskit simulators cannot handle it because the required memory exceeds the maximum memory allowed. However, we were able to test different test cases applied directly to oracle without applying the Hadamard operator to create superposition states, and we got the expected result. The proposed quantum oracle design can easily find all possible solutions in one iteration of Grover's algorithm flowed classical check by checking only the output qubit such that if

the *out* qubit is equal to one, then the corresponding input for x_0, \dots, x_{15} are the solutions for binary sudoku.

Based on these and on the previous examples, as well as several examples calculated by our research team, I can state that very many problems can be solved with Grover algorithm with quadratic speed up using in oracle the following blocks:

1. Logic gates: NOT, AND, OR, XOR, EQUIVALENCE, IMPLY, and INHIBITION.
2. Arithmetic operators such as ADDER, MULTIPLIER, DIVIDER, SQUARE ROOT, POWER, and etc.
3. Predicates for comparison such as $A > B$, $A < B$, $A = B$, $A \geq B$, $A \leq B$, and etc.
4. Arbitrary code converters (encoders, decoders) such as binary, Gray, One Hot, two natural order, etc. that are specified by various lookup tables.
5. Arbitrary functions such as cosine, sine, etc. specified by lookup tables.

9 SUMMARY AND CONCLUSIONS

9.1 Conclusion

In this dissertation, I have designed and analyzed a novel quantum oracle circuit that requires a logarithmically reduced number of qubits for solving SAT-like and MAX-SAT-like problems. The oracle circuit uses the iterative quantum counter circuit, which replaces the ancilla qubits of a global large AND gate for traditional oracle design. My design showed a significant reduction overall in the number of qubits in Grover's search algorithm for MAX-SAT. Also, my design calculates the quantum measurable number of the maximum satisfiable OR terms for unsatisfiable Boolean functions. In addition, I compared using Peres and Toffoli gates in terms of quantum cost, where the Peres gates built from truly quantum primitives provide lower quantum costs. In Chapter 4, I tested and showed several examples using the IBM QISKIT simulator [93] that provided the expected results. Other variants of SAT oracles were also presented that can be designed for the oracle circuit using the quantum counter. Also, many other potential problems in the area of EDA were also discussed that can be reduced to SAT and MAX-SAT such that the oracle can construct the quantum counter idea. There is still active research to overcome the challenges in the classical SAT solver to handle CNF-XOR clauses. I presented a quantum oracle design for XOR clauses that does not need to convert to CNF clauses. I presented a uniform quantum oracle design that can handle all the forms of clauses, such as CNF clauses, CNF-XOR clauses, and XOR clauses. In the optimistic advent of quantum SAT solvers, my uniform quantum oracle design with Grover's search

algorithm will be an asset to have such a few qubits with one uniform design for all forms of CNF clauses, CNF-XOR clauses, and XOR clauses.

In Chapter 5, I presented a novel quantum design for associate rule mining to discover the maximum frequent k -itemset. The transaction database was converted into a new type of SOP Boolean function and then the SOP function was reduced into large k -items with only Hamming weight that satisfies the minimum support threshold. In addition, I proposed using the Dicke state to prepare the superposition $\binom{n}{k}$ states for Grover's search algorithm instead of the conventional Hadamard operator, which would require 2^n states. The SOP function was reduced to large k -items where the likelihood of the maximum frequent k -itemset can be found. Then, I designed an advanced quantum oracle with a quantum counter and a quantum comparator. This way I achieved a reduced search space and the required number of qubits in the design using these three different quantum blocks: Dicke state, quantum counter, and quantum comparator. Then I showed a full implementation of this design on the IBM Qiskit simulator and observed the correct results. I compared my proposed design with [182, 183] design, that my design requires fewer qubits. My research demonstrated that now the user of my methodology for this type of problem can solve many transactions and items by scaling and optimizing the search space and required qubits.

In Chapter 6, I presented the efficient quantum circuit implantation for shift operators for mining frequent patterns for association rule mining. These examples are considered regular graphs; however, irregular graphs need a more complex design with a high cost regarding the number of quantum gates and qubits. The quantum walk algorithm has

more potential applications on complex graphs, both regular and irregular graphs. The quantum walk algorithm is more suited to graphs than the Grover's algorithm.

In Chapter 8, I proposed a uniform methodology for Boolean quantum oracle design for quantum search algorithms where many ideas from several algorithms are combined together and applied to the design of oracles for quantum search algorithms. All these algorithms either use Grover, repeated Grover, or hybrid algorithms that use Grover's algorithm. Optimization problems are reduced to a repetition of constraint satisfaction problems solved by Grover's algorithm. My approach is based on creating various oracles, which are, however, based on the same basic principle of designing oracles from typical digital blocks bottom-up, rather than decomposing a unitary matrix top-down as proposed by some previous authors. The presented quantum oracle designs are for various well-known EDA applications of the Unate and Binate covering problems. Innovative quantum algorithms for exact and incompletely specified FSM minimization have been presented here for the first time. This is the first quantum algorithm ever for the classical problem known from many logic design textbooks used in undergraduate and graduate education. In addition, the methodology presented here can be used with very small modifications to solve several other logic design problems known from the areas of "logic synthesis" or "digital design". These problems are all very similar to one another and basically can be reduced to SAT-like approaches with more complicated oracles than only product of sums. These are all fundamental problems known from textbooks and research papers over last 50 years [55, 126, 128, 130, 131, 132, 134, 136,

137, 139, 140, 142, 143, 213, 214, 215, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 231, 232, 233, 234, 235, 236, 237, 238].

Also, I presented a quantum oracle design for binary sudoku, a logic puzzle problem that can be solved using Grover's search algorithm. Each of these problems is converted to a general quantum oracle. Our quantum oracle design uses an iterative quantum counter block used inside the oracles for the Grover-like algorithms. The concept of introducing this quantum counter can be applied to all these algorithms, allowing them to solve in a uniform way both SAT-like and MAX-SAT-like problems. Most importantly, this approach reduces the number of qubits logarithmically [127]. The introduction of the iterative quantum counter circuit replaces the ancilla qubits of the global large AND gate for traditional quantum oracle design for SAT-like problems. This is important because it reduces not only the number of required qubits but also avoids designing a quantum AND gate with many inputs, which is known to be a very complicated gate. I presented experimental results from the QISKIT simulator that showed the circuit worked correctly. I showed a quantum oracle design for arithmetic problems such as addition and multiplication that can be solved using Grover's search algorithm.

9.2 Achievements

Being familiar with the importance of various search problems and corresponding algorithms, I ask myself questions about how to create a unified methodology to solve all these problems using quantum computers. When I started my work in 2018, this area was less developed than it is in recent years. I found then that there are basically two main approaches: Grover's algorithm and quantum walk algorithm. Most of the problems in

the literature were purely mathematical and theoretical. Based on my electrical engineering and programming experience, I decided to solve several practical problems using the above-mentioned quantum algorithms. I asked myself the following questions:

1. Can problems such as minimization of automata or creating association rules base systems for which quantum algorithms have not been done before be solved practically using Grover's algorithm and the quantum walk algorithm?
2. Most of the published papers described the oracles as only unitary matrices. Can I create detailed quantum circuits from realistic based gates that are available in QISKIT?
3. What are all possible practical quantum circuit design problems that can be solved by methods? The same for cryptography problems
4. How to define the complexity of oracles for quantum search problems so that this complexity will be minimized in my designs?
5. I found the following generally useful techniques, such as
 - a. Quantum counter
 - b. Dicke state
 - c. Quantum comparator
 - d. Improved diffusion circuits
 - e. Approaches to solve repeated constraint satisfaction.
 - f. Boolean oracles vs phase oracles

I asked myself questions, what are possible uses of these blocks and what other similar blocks can be invented?

Based on my solutions to several individual problems and applied them to specific quantum circuits, I found that I created a framework for a general methodology that can be applied to a very wide category of problems that I was only able to illustrate partially in the chapters of my dissertation. Specifically, I believe that the main achievements of the dissertation that can be used both by me and several other authors to solve problems in various areas are the following:

- 1) I designed several advanced oracle circuits for various practical problems, in which I exponentially reduced the number of qubits. Moreover, my designs generalized these problems from SAT to MAX-SAT. These make them more general and more practically useful.
- 2) I developed a new version of Grover's search algorithm for mining frequent patterns, symmetric Boolean functions, and other machine learning-related problems. These new versions scale the search space by using only subspace states as superposition states instead of the whole space.
- 3) I designed the first quantum MAX-SAT, which can handle any size of constraints and variables. In literature, 2-SAT, and 3-SAT were mostly proposed for new technologies.
- 4) I created and presented a general methodology to create complex quantum search algorithms. Some of them are shown in detail, while others are only mentioned, but a careful reader can use my methodology and examples of these problems. These problems include many other variants of the SAT format, such as covering

problems, logic puzzles, and SAT-like and MAX-SAT-like problems that can be solved by my design methodology. See also above for all design automation problems in classical binary design.

- 5) My methodology allows any constraint satisfaction problem that constraints are formulated in binary Boolean algebra and digital arithmetic. I use logic operator blocks and arithmetic blocks such as counters, comparators, adders, and multipliers.
- 6) The authors of published oracles often used unitary matrices that are not directly realizable in quantum hardware. It is also well-known that decomposing unitary matrices into elementary gates creates very complicated circuits. Therefore, in my methodology, instead of decomposing unitary matrices, I compose these matrices from well-known elementary quantum gates. This method allowed me to realistically simulate quantum circuits, taking into account the complexity of circuits and the analysis of errors. Therefore, I was able to simulate most of the quantum oracles in QISKIT at the level of primitive gates built by IBM.
- 7) I found a class of practical problems that can be solved better than Grover's algorithm. For instance, I proved a quantum walk design for mining frequent patterns for association rule mining.

9.3 Publications

- [1] Alasow, Abdirahman, and Marek Perkowski. "Quantum Algorithm for Maximum Satisfiability." In 2022 IEEE 52nd International Symposium on Multiple-Valued Logic (ISMVL), pp. 27-34. IEEE, 2022.
- [2] Alasow, Abdirahman, Peter Jin, and Marek Perkowski. "Quantum Algorithm for Variant Maximum Satisfiability." *Entropy* 24, no. 11 (2022): 1615.
- [3] Alasow, Abdirahman, and Marek Perkowski. "Quantum Algorithm for Mining Frequent Patterns for Association Rule Mining." *Journal of Quantum Information Science* 13, no. 1 (2023): 1-23.
- [4] Alasow, Abdirahman, and Marek Perkowski. "Quantum Algorithms for Unate and Binate Covering Problems with Application to Finite State Machine Minimization." *Journal of Applied Logics* (2023).

9.4 Future Work

- 1) My paper XOR SAT was not accepted by a journal. However, the critical remarks of the reviewers, further review of literature and my current deeper understanding of my developed methodology allow me to believe that I will be able to improve and resubmit this paper "Quantum Algorithm for XOR SAT".
- 2) Further improvement is to extend the quantum logic oracle design to add an arithmetic circuit for problems that involve both logic and arithmetic, such as weighted MAX-SAT, weighted covering problems, and objective functions for constraint satisfaction problems.

- 3) A quantum design to generate association rules from a maximum frequent k -itemset for associate rule mining can be developed using our proposed design methodology.
- 4) Standard quantum circuit designs for shift operators for regular graphs for quantum walk algorithm since the shift operator can be formulated based on the number of vertices of each type of regular graph.

References

- [1] Wack, A., Paik, H., Javadi-Abhari, A., Jurcevic, P., Faro, I., Gambetta, J.M. and Johnson, B.R., 2021. Quality, speed, and scale: three key attributes to measure the performance of near-term quantum computers. arXiv preprint arXiv:2110.14108.
- [2] Donkers, H., Mesman, K., Al-Ars, Z. and Möller, M., 2022. QPack Scores: Quantitative performance metrics for application-oriented quantum computer benchmarking. arXiv preprint arXiv:2205.12142.
- [3] Collins, H. and Easterly, K. (2021) IBM Unveils Breakthrough 127-Qubit Quantum Processor. <https://newsroom.ibm.com/2021-11-16-IBM-Unveils-Breakthrough-127-Qubit-Quantum-Processor>
- [4] Moore, Gordon E. "Cramming more components onto integrated circuits." Proceedings of the IEEE 86, no. 1 (1998): 82-85.
- [5] Lu, Chien-Ping. "AI, native supercomputing and the revival of Moore's Law." APSIPA Transactions on Signal and Information Processing 6 (2017): e9.
- [6] Shalf, John. "The future of computing beyond Moore's Law." Philosophical Transactions of the Royal Society A 378, no. 2166 (2020): 20190061.
- [7] Theis, Thomas N., and H-S. Philip Wong. "The end of moore's law: A new beginning for information technology." Computing in science & engineering 19, no. 2 (2017): 41-50.
- [8] Kasirajan, Venkateswaran. Fundamentals of quantum computing. Springer International Publishing, 2021.

- [9] Ladd, Thaddeus D., Fedor Jelezko, Raymond Laflamme, Yasunobu Nakamura, Christopher Monroe, and Jeremy Lloyd O'Brien. "Quantum computers." *Nature* 464, no. 7285 (2010): 45-53.
- [10] Popkin, Gabriel. "Quest for qubits." (2016): 1090-1093. [Online]. Available: <https://www.science.org/doi/full/10.1126/science.354.6316.1090>
- [11] Devoret, Michel H., and Robert J. Schoelkopf. "Superconducting circuits for quantum information: an outlook." *Science* 339, no. 6124 (2013): 1169-1174.
- [12] Huang, He-Liang, Dachao Wu, Daojin Fan, and Xiaobo Zhu. "Superconducting quantum computing: a review." *Science China Information Sciences* 63 (2020): 1-32.
- [13] Gambetta, Jay M., Jerry M. Chow, and Matthias Steffen. "Building logical qubits in a superconducting quantum computing system." *NPJ Quantum Information* 3, no. 1 (2017): 2.
- [14] Arute, Frank, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas et al. "Quantum supremacy using a programmable superconducting processor." *Nature* 574, no. 7779 (2019): 505-510.
- [15] Khammassi, Nader, Imran Ashraf, J. V. Someren, Razvan Nane, A. M. Krol, M. Adriaan Rol, Lingling Lao, Koen Bertels, and Carmen G. Almudever. "Openql: A portable quantum programming framework for quantum accelerators." *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 18, no. 1 (2021): 1-24.

- [16] Monroe, Chris, David M. Meekhof, Barry E. King, Wayne M. Itano, and David J. Wineland. "Demonstration of a fundamental quantum logic gate." *Physical review letters* 75, no. 25 (1995): 4714.
- [17] Ballance, C. J., T. P. Harty, N. M. Linke, M. A. Sepiol, and D. M. Lucas. "High-fidelity quantum logic gates using trapped-ion hyperfine qubits." *Physical review letters* 117, no. 6 (2016): 060504.
- [18] Wright, Kenneth, Kristin M. Beck, Sea Debnath, J. M. Amini, Y. Nam, N. Grzesiak, J-S. Chen et al. "Benchmarking an 11-qubit quantum computer." *Nature communications* 10, no. 1 (2019): 5464.
- [19] Pino, Juan M., Jennifer M. Dreiling, Caroline Figgatt, John P. Gaebler, Steven A. Moses, M. S. Allman, C. H. Baldwin et al. "Demonstration of the trapped-ion quantum CCD computer architecture." *Nature* 592, no. 7853 (2021): 209-213.
- [20] Wang, Jianwei, Fabio Sciarrino, Anthony Laing, and Mark G. Thompson. "Integrated photonic quantum technologies." *Nature Photonics* 14, no. 5 (2020): 273-284.
- [21] Madsen, Lars S., Fabian Laudenbach, Mohsen Falamarzi Askarani, Fabien Rortais, Trevor Vincent, Jacob FF Bulmer, Filippo M. Miatto et al. "Quantum computational advantage with a programmable photonic processor." *Nature* 606, no. 7912 (2022): 75-81.
- [22] Bombin, Hector, Isaac H. Kim, Daniel Litinski, Naomi Nickerson, Mihir Pant, Fernando Pastawski, Sam Roberts, and Terry Rudolph. "Interleaving: Modular

- architectures for fault-tolerant photonic quantum computing." arXiv preprint arXiv:2103.08612 (2021).
- [23] Maurand, R., X. Jehl, D. Kotekar-Patil, A. Corna, H. Bohuslavskyi, R. Laviéville, L. Hutin et al. "A CMOS silicon spin qubit." *Nature Communications* 7, no. 1 (2016): 13575.
- [24] Intel's New Chip to Advance Silicon Spin Qubit Research for Quantum Computing.
<https://www.intel.com/content/www/us/en/newsroom/news/quantum-computing-chip-to-advance-research.html#gs.6idy36>
- [25] Cai, Xinxin, Elliot J. Connors, Lisa F. Edge, and John M. Nichol. "Coherent spin–valley oscillations in silicon." *Nature Physics* 19, no. 3 (2023): 386-393.
- [26] Dejpasand, Mohamad Taghi, and Morteza Sasani Ghamsari. "Research Trends in Quantum Computers by Focusing on Qubits as Their Building Blocks." *Quantum Reports* 5, no. 3 (2023): 597-608.
- [27] Deutsch, D., 1985. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818), pp.97-117.
- [28] Deutsch, D. and Jozsa, R., 1992. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907), pp.553-558.
- [29] Simon, D.R., 1997. On the power of quantum computation. *SIAM Journal on Computing*, 26(5), pp.1474-1483.

- [30] Shor, P.W., 1994, November. Algorithms for quantum computation: discrete logarithms and factoring. In Proceedings 35th Annual Symposium on Foundations of Computer Science (pp. 124-134). IEEE.
- [31] Grover, L.K., 1996, July. A fast quantum mechanical algorithm for database search. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (pp. 212-219).
- [32] Shenvi, N., Kempe, J. and Whaley, K.B., 2003. Quantum random-walk search algorithm. *Physical Review A*, 67(5), p.052307.
- [33] Harrow, A.W., Hassidim, A. and Lloyd, S., 2009. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15), p.150502.
- [34] Peruzzo, A., McClean, J., Shadbolt, P., Yung, M.H., Zhou, X.Q., Love, P.J., Aspuru-Guzik, A. and O'brien, J.L., 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), p.4213.
- [35] Farhi, E., Goldstone, J. and Gutmann, S., 2014. A quantum approximate optimization algorithm. arXiv preprint arXiv:1411.4028.
- [36] Zhang, S. and Li, L., 2022. A brief introduction to quantum algorithms. *CCF Transactions on High Performance Computing*, 4(1), pp.53-62.
- [37] Benioff, P., 1980. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of statistical physics*, 22, pp.563-591.
- [38] Manin, Yuri. "Computable and uncomputable." Sovetskoye Radio, Moscow 128 (1980).

- [39] Bertels, K. O. E. N., Aritra Sarkar, Thomas Hubregtsen, M. Serrao, Abid A. Mouedenne, Amitabh Yadav, A. Krol, Imran Ashraf, and C. Garcia Almudever. "Quantum computer architecture toward full-stack quantum accelerators." *IEEE Transactions on Quantum Engineering* 1 (2020): 1-17.
- [40] Kadowaki, Tadashi, and Hidetoshi Nishimori. "Quantum annealing in the transverse Ising model." *Physical Review E* 58, no. 5 (1998): 5355.
- [41] Fankhauser, Tobias, Marc E. Solèr, Rudolf M. Füchslin, and Kurt Stockinger. "Multiple Query Optimization using a Gate-Based Quantum Computer." *IEEE Access* (2023).
- [42] Johnson, Mark W., Mohammad HS Amin, Suzanne Gildert, Trevor Lanting, Firas Hamze, Neil Dickson, Richard Harris et al. "Quantum annealing with manufactured spins." *Nature* 473, no. 7346 (2011): 194-198.
- [43] Codognet, Philippe, Daniel Diaz, and Salvador Abreu. "Quantum and Digital Annealing for the Quadratic Assignment Problem." In *2022 IEEE International Conference on Quantum Software (QSW)*, pp. 1-8. IEEE, 2022.
- [44] D-WaveSystems, Inc. *D-Wave Ocean Software Documentation*, [Online]. Available: <https://docs.ocean.dwavesys.com>
- [45] Bunyk, Paul I., Emile M. Hoskinson, Mark W. Johnson, Elena Tolkacheva, Fabio Altomare, Andrew J. Berkley, Richard Harris et al. "Architectural considerations in the design of a superconducting quantum annealing processor." *IEEE Transactions on Applied Superconductivity* 24, no. 4 (2014): 1-10.

- [46] Yanofsky, Noson S., and Mirco A. Mannucci. Quantum computing for computer scientists. Cambridge University Press, 2008.
- [47] Woody III, Leonard Spencer. "Essential mathematics for quantum computing: a beginner's guide to just the math you need without needless complexities." Packt Publishing Limited, Birmingham, 2022.
- [48] Nielsen, Michael A., and Isaac L. Chuang. Quantum computation and quantum information. Cambridge University Press, 2010.
- [49] Kaye, Phillip, Raymond Laflamme, and Michele Mosca. An introduction to quantum computing. Oxford University Press, Oxford, 2007.
- [50] Mermin, N. David. Quantum computer science: an introduction. Cambridge University Press, 2007.
- [51] Figgatt, Caroline, Dmitri Maslov, Kevin A. Landsman, Norbert M. Linke, Shantanu Debnath, and Christofer Monroe. "Complete 3-qubit Grover search on a programmable quantum computer." Nature Communications 8, no. 1 (2017): 1918.
- [52] Li, ZhenQiang, BinBin Cai, HongWei Sun, HaiLing Liu, LinChun Wan, SuJuan Qin, QiaoYan Wen, and Fei Gao. "Novel quantum circuit implementation of Advanced Encryption Standard with low costs." Science China Physics, Mechanics & Astronomy 65, no. 9 (2022): 290311.
- [53] Jang, Kyungbae, Gyeongju Song, Hyunjun Kim, Hyeokdong Kwon, Hyunji Kim, and Hwajeong Seo. "Efficient implementation of PRESENT and GIFT on quantum computers." Applied Sciences 11, no. 11 (2021): 4776.

- [54] Langenberg, Brandon, Hai Pham, and Rainer Steinwandt. "Reducing the cost of implementing the advanced encryption standard as a quantum circuit." *IEEE Transactions on Quantum Engineering* 1 (2020): 1-12.
- [55] Perkowski, Marek. "Inverse Problems, Constraint Satisfaction, Reversible Logic, Invertible Logic and Grover Quantum Oracles for Practical Problems." In *Reversible Computation: 12th International Conference, RC 2020, Oslo, Norway, July 9-10, 2020, Proceedings* 12, pp. 3-32. Springer International Publishing, 2020.
- [56] Jaques, Samuel, Michael Naehrig, Martin Roetteler, and Fernando Virdia. "Implementing Grover oracles for quantum key search on AES and LowMC." In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II* 30, pp. 280-310. Springer International Publishing, 2020.
- [57] Aïmeur, Esma, Gilles Brassard, and Sébastien Gambs. "Quantum speed-up for unsupervised learning." *Machine Learning* 90 (2013): 261-287.
- [58] Wittek, Peter. *Quantum machine learning: what quantum computing means to data mining*. Academic Press, 2014.
- [59] Lloyd, Seth, Masoud Mohseni, and Patrick Rebentrost. "Quantum algorithms for supervised and unsupervised machine learning." *arXiv preprint arXiv:1307.0411* (2013).

- [60] Zeguendry, Amine, Zahi Jarir, and Mohamed Quafafou. "Quantum machine learning: A review and case studies." *Entropy* 25, no. 2 (2023): 287.
- [61] Djordjevic, Ivan B. *Quantum information processing, quantum computing, and quantum error correction: an engineering approach*. Academic Press, 2021.
- [62] Wiedemann, Simon, Daniel Hein, Steffen Udluft, and Christian Mendl. "Quantum Policy Iteration via Amplitude Estimation and Grover Search--Towards Quantum Advantage for Reinforcement Learning." arXiv preprint arXiv:2206.04741 (2022).
- [63] Ganger, Michael, and Wei Hu. "Quantum multiple q-learning." *International Journal of Intelligence Science* 9, no. 01 (2019): 1.
- [64] Rebentrost, Patrick, Masoud Mohseni, and Seth Lloyd. "Quantum support vector machine for big data classification." *Physical Review Letters* 113, no. 13 (2014): 130503.
- [65] Pronin, C. B., O. I. Maksimychev, A. V. Ostroukh, A. V. Volosova, and E. N. Matukhina. "Creating Quantum Circuits for Training Perceptron Neural Networks on the Principles of Grover's Algorithm." In *2022 Systems of Signals Generating and Processing in the Field of on Board Communications*, pp. 1-5. IEEE, 2022.
- [66] Dang, Yijie, Nan Jiang, Hao Hu, Zhuoxiao Ji, and Wenyin Zhang. "Image classification based on quantum K-Nearest-Neighbor algorithm." *Quantum Information Processing* 17 (2018): 1-18.

- [67] Wang, Yushi, and Marek Perkowski. "Improved complexity of quantum oracles for ternary grover algorithm for graph coloring." In 2011 41st IEEE International Symposium on Multiple-Valued Logic, pp. 294-301. IEEE, 2011.
- [68] Haverly, A., and S. López. "Implementation of grover's algorithm to solve the maximum clique problem." In 2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 441-446. IEEE, 2021.
- [69] Chang, Weng-Long, Renata Wong, Wen-Yu Chung, Yu-Hao Chen, Ju-Chin Chen, and Athanasios V. Vasilakos. "Quantum Speedup for the Maximum Cut Problem." arXiv preprint arXiv:2305.16644 (2023).
- [70] Dörn, S., 2005. Quantum complexity bounds for independent set problems. arXiv preprint quant-ph/0510084.
- [71] Marques-Silva, J.; Glass, T. Combinational equivalence checking using satisfiability and recursive learning. In Proceedings of the Conference on Design, Automation and Test in Europe, Munich, Germany, 1 January 1999; pp. 33-es.
- [72] Konuk, H.; Larrabee, T. Explorations of sequential ATPG using Boolean satisfiability. In Proceedings of the Digest of Papers Eleventh Annual 1993 IEEE VLSI Test Symposium, Atlantic City, NJ, USA, 6–8 April 1993; IEEE: Manhattan, NY, USA; pp. 85–90.
- [73] Biere, A.; Cimatti, A.; Clarke, E.M.; Fujita, M.; Zhu, Y. Symbolic model checking using SAT procedures instead of BDDs. In Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, New Orleans, LA, USA, 21–25 June 1999; pp. 317–320.

- [74] Hong, T.; Li, Y.; Park, S.B.; Mui, D.; Lin, D.; Kaleq, Z.A.; Hakim, N.; Naeimi, H.; Gardner, D.S.; Mitra, S. QED: Quick error detection tests for effective post-silicon validation. In 2010 IEEE International Test Conference, Austin, TX, USA, 2–4 November 2010; IEEE: Manhattan, NY, USA; pp. 1–10.
- [75] Wang, P.W.; Donti, P.; Wilder, B.; Kolter, Z. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In Proceedings of the International Conference on Machine Learning; Long Beach, CA, USA, 10–15 June 2019; PMLR: New York, NY, USA; pp. 6545–6554.
- [76] Cook, S.A. The complexity of theorem-proving procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing, Shaker Heights, OH, USA, 3–5 May 1971; pp. 151–158.
- [77] Kohli, R.; Krishnamurti, R.; Mirchandani, P. The minimum satisfiability problem. *SIAM J. Discret. Math.* 1994, 7, 275–283.
- [78] Biere, A.; Heule, M.; van Maaren, H. (Eds.) *Handbook of Satisfiability*; IOS press: Washington, DC, USA, 2009; Volume 185.
- [79] Fu, Z.; Malik, S. On solving the partial MAX-SAT problem. In Proceedings of the International Conference on Theory and Applications of Satisfiability Testing, Seattle, WA, USA, 12–15 August 2006; Springer: Berlin, Heidelberg; pp. 252–265.
- [80] Berg, O.J.; Hyttinen, A.J.; Jarvisalo, M.J. Applications of MaxSAT in data analysis. In Proceedings of the Pragmatics of SAT 2015 and 2018, Oxford, UK, July 7, 2018.

- [81] Berg, J.; Jarvisalo, M. Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. *Artif. Intell.* 2017, 244, 110–142.
- [82] Berg, J.; Jarvisalo, M.; Malone, B. Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In *Proceedings of the Artificial Intelligence and Statistics, Reykjavik, Iceland, 22–25 April 2014*; PMLR: New York, NY, USA; pp. 86–95.
- [83] Hyttinen, A.; Saikko, P.; Jarvisalo, M. A core-guided approach to learning optimal causal graphs. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017), Melbourne, Australia, 19–25 August 2017*.
- [84] Malioutov, D.; Meel, K.S. MLIC: A MaxSAT-based framework for learning interpretable classification rules. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming, Lille, France, 27–31 August 2018*; Springer: Cham, Switzerland, 2018; pp. 312–327.
- [85] Dimitrova, R.; Ghasemi, M.; Topcu, U. Maximum realizability for linear temporal logic specifications. In *Proceedings of the International Symposium on Automated Technology for Verification and Analysis, Los Angeles, CA, USA, 7–10 October 2018*; Springer: Cham, Switzerland, 2018; pp. 458–475.
- [86] Zhang, L.; Bacchus, F. MAXSAT heuristics for cost optimal planning. In *Proceedings of the AAAI Conference on Artificial Intelligence, Toronto, ON, Canada, 22–26 July 2012*; Volume 26, pp. 1846–1852.

- [87] Muise, C.; Beck, J.C.; McIlraith, S.A. Optimal partial-order plan relaxation via MaxSAT. *J. Artif. Intell. Res.* 2016, 57, 113–149.
- [88] Demirović, E.; Musliu, N.; Winter, F. Modeling and solving staff scheduling with partial weighted maxSAT. *Ann. Oper. Res.* 2019, 275, 79–99.
- [89] Safarpour, S.; Mangassarian, H.; Veneris, A.; Liffiton, M.H.; Sakallah, K.A. November. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer Aided Design (FMCAD'07)*; IEEE: Cham, Switzerland, 2007; pp. 13–19.
- [90] Chen, Y.; Safarpour, S.; Veneris, A.; Marques-Silva, J. Spatial and temporal design debug using partial MaxSAT. In *Proceedings of the 19th ACM Great Lakes symposium on VLSI, Boston Area, MA, USA, 10–12 May 2009*; pp. 345–350.
- [91] Chen, Y.; Safarpour, S.; Marques-Silva, J.; Veneris, A. Automated design debugging with maximum satisfiability. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 2010, 29, 1804–1817.
- [92] Jose, M.; Majumdar, R. Cause clue clauses: Error localization using maximum satisfiability. *ACM SIGPLAN Not.* 2011, 46, 437–446.
- [93] Zhu, C.S.; Weissenbacher, G.; Malik, S. Post-silicon fault localisation using maximum satisfiability and backbones. In *Proceedings of the 2011 Formal Methods in Computer-Aided Design (FMCAD), Austin, TX, USA, 30 October–2 November 2011*; IEEE: Cham, Switzerland, 2011; pp. 63–66.

- [94] Wickramaarachchi, G.T.; Qardaji, W.H.; Li, N. An efficient framework for user authorization queries in RBAC systems. In Proceedings of the 14th ACM symposium on Access control models and technologies, Stresa, Italy, 3–5 June 2009; pp. 23–32.
- [95] Liao, X.; Zhang, H.; Koshimura, M. Reconstructing AES key schedule images with SAT and MaxSAT. *IEICE TRANSACTIONS Inf. Syst.* 2016, 99, 141–150.
- [96] Shabani, A.; Alizadeh, B. PMTP: A MAX-SAT-based approach to detect hardware trojan using propagation of maximum transition probability. *IEEE Trans. Comput. -Aided Des. Integr. Circuits Syst.* 2018, 39, 25–33.
- [97] Feng, Y.; Bastani, O.; Martins, R.; Dillig, I.; Anand, S. Automated synthesis of semantic malware signatures using maximum satisfiability. *arXiv* 2016, arXiv:1608.06254.
- [98] Lin, P.C.K.; Khatri, S.P. Application of Max-SAT-based ATPG to optimal cancer therapy design. *BMC Genom.* 2012, 13, 1–10.
- [99] Guerra, J.; Lynce, I. Reasoning over biological networks using maximum satisfiability. In Proceedings of the International conference on principles and practice of constraint programming, Québec City, QC, Canada, 8–12 October 2012; Springer: Berlin, Heidelberg; pp. 941–956.
- [100] Martins, R. Solving RNA alignment with MaxSAT. In *MaxSAT Evaluation 2017 Solver and Benchmark Descriptions*, pages 29–30. University of Helsinki, 2017.
- [101] Graça, A.; Marques-Silva, J.; Lynce, I.; Oliveira, A.L. Haplotype inference with pseudo-Boolean optimization. *Ann. Oper. Res.* 2011, 184, 137–162.

- [102] Li, C.M.; Quan, Z. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence, Atlanta, GA, USA, 11–15 July 2010.
- [103] Li, C.M.; Jiang, H.; Xu, R.C. Incremental MaxSAT reasoning to reduce branches in a branch-and-bound algorithm for MaxClique. In Proceedings of the International Conference on Learning and Intelligent Optimization, Lille, France, 12–15 January 2015; Springer: Cham, Switzerland; pp. 268–274.
- [104] Fang, Z.; Li, C.M.; Qiao, K.; Feng, X.; Xu, K. Solving Maximum Weight Clique Using Maximum Satisfiability Reasoning. In Proceedings of the ECAI, Prague, Czech, 18–22 August 2014; pp. 303–308.
- [105] Berg, J.; Jarvisalo, M. November. SAT-based approaches to treewidth computation: An evaluation. In Proceedings of the 2014 IEEE 26th international conference on tools with artificial intelligence, Limassol, Cyprus, 10–12 November 2014; IEEE: Cham, Switzerland; pp. 328–335.
- [106] Morgado, A.; Marques-Silva, J. Combinatorial optimization solutions for the maximum quartet consistency problem. *Fundam. Inform.* 2010, 102, 363–389.
- [107] Smyth, K.; Hoos, H.H.; Stützle, T.; 2003, June. Iterated robust tabu search for MAX-SAT. In Proceedings of the Conference of the Canadian Society for Computational Studies of Intelligence, Halifax, Canada, 11–13 June 2003; Springer: Berlin, Heidelberg; pp. 129–144.
- [108] Mastrolilli, M.; Gambardella, L.M. Maximum satisfiability: How good are tabu search and plateau moves in the worst-case?. *Eur. J. Oper. Res.* 2005, 166, 63–76.

- [109] Cai, S.; Lei, Z. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artif. Intell.* 2020, 287, 103354.
- [110] Marchiori, E.; Rossi, C. A flipping genetic algorithm for hard 3-SAT problems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, Orlando, FL, USA, 13–17 July 1999; pp. 393–400.
- [111] Layeb, A.; Deneche, A.H.; Meshoul, S. A new artificial immune system for solving the maximum satisfiability problem. In *Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, Cordoba, Spain, 1–4 June 2010; Springer: Berlin, Heidelberg; pp. 136–142.
- [112] Munawar, A.; Wahib, M.; Munetomo, M.; Akama, K. Hybrid of genetic algorithm and local search to solve max-sat problem using NVIDIA CUDA framework. *Genet. Program. Evolvable Mach.* 2009, 10, 391–415.
- [113] Lardeux, F.; Saubion, F.; Hao, J.K. GASAT: A genetic local search algorithm for the satisfiability problem. *Evol. Comput.* 2006, 14, 223–253.
- [114] Davis, M.; Logemann, G.; Loveland, D. A machine program for theorem-proving. *Commun. ACM* 1962, 5, 394–397.
- [115] Li, C.M.; Manyá, F.; Planes, J. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*; Sitges, Spain, 1–5 October 2005; Springer, Berlin, Heidelberg; pp. 403–414.

- [116] Li, C.M.; Xu, Z.; Coll, J.; Manyà, F.; Habet, D.; He, K. Combining clause learning and branch and bound for MaxSAT. In Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming (CP 2021), Montpellier, France, 25–29 October 2021; Schloss Dagstuhl-Leibniz-Zentrum für Informatik: Wadern, Germany, 2021.
- [117] Gu, J. Efficient local search for very large-scale satisfiability problems. *ACM SIGART Bull.* 1992, 3, 8–12.
- [118] Bian, Z.; Chudak, F.; Macready, W.; Roy, A.; Sebastiani, R.; Varotti, S. Solving sat and maxsat with a quantum annealer: Foundations and a preliminary report. In Proceedings of the International Symposium on Frontiers of Combining Systems, Brasília, Brazil, 27–29 September 2017; Springer: Cham, Switzerland; pp. 153–171.
- [119] Cheng, S.T.; Tao, M.H. Quantum cooperative search algorithm for 3-SAT. *J. Comput. Syst. Sci.* 2007, 73, 123–136.
- [120] Lee, S.; Lee, S.-J.; Kim, T.; Biamonte, J.; Perkowski, M. The cost of Quantum Gate Primitives. *J. Mult.-Valued Log. Soft Comput.* 2006, 12, 561–573.
- [121] Barenco, A.; Bennett, C.H.; Cleve, R.; DiVincenzo, D.P.; Margolus, N.; Shor, P.; Sleator, T.; Smolin, J.A.; Weinfurter, H. Elementary gates for quantum computation. *Phys. Rev. A* 1999, 52, 3457.
- [122] Maslov, D.; Dueck, G.W. Improved quantum cost for n-bit Toffoli gates. *Electron. Lett.* 2003, 39, 1790–1791.
- [123] Peres, A. Reversible logic and quantum computers. *Phys. Rev. A* 1985, 32, 3266.

- [124] Szyprowski, M.; Kerntopf, P. August. Low quantum cost realization of generalized peres and toffoli gates with multiple-control signals. In Proceedings of the 2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013), Beijing, China, 5–8 August 2013; IEEE: Manhattan, NY, USA; pp. 802–807.
- [125] Aleksandrowicz, G.; Alexander, T.; Barkoutsos, P.; Bello, L.; Ben-Haim, Y.; Bucher, D.; Cabrera-Hernández, F.J.; Carballo-Franquis, J.; Chen, A.; Chen, C.F.; et al. Qiskit: An Open-Source Framework for Quantum Computing; Zenodo: Honolulu, HI, USA, 2019.
- [126] Perkowski, M., 2022. Inverse problems, constraint satisfaction, reversible logic, invertible logic and Grover quantum oracles for practical problems. *Science of Computer Programming*, 218, p.102775.
- [127] Alasow, A. and Perkowski, M., 2022, May. Quantum Algorithm for Maximum Satisfiability. In 2022 IEEE 52nd International Symposium on Multiple-Valued Logic (ISMVL) (pp. 27-34). IEEE.
- [128] Csanky, L. On the Generalized Reed-Muller Canonical Form of Boolean Functions: Research Project. PhD Thesis, University of California: Berkeley, CA, USA, 1972.
- [129] Garey, M.R.; Johnson, D.S. A Guide to the Theory of NP-Completeness. *Computers and Intractability*; W. H. Freeman & Co: New York, NY, USA, 1979.
- [130] Lin, H.P.; Jiang, J.H.R.; Lee, R.R. To SAT or not to SAT: Ashenurst decomposition in a large scale. In Proceedings of the 2008 IEEE/ACM

International Conference on Computer-Aided Design, San Jose, CA, USA, 10–13 November 2008; IEEE: Manhattan, NY, USA, 2008; pp. 32–37.

- [131] Li, Y.; Tsai, E.; Perkowski, M.; Song, X. Grover-based Ashenurst-Curtis decomposition using quantum language quipper. *Quantum Inf. Comput.* 2019, 19, 35–66.
- [132] Breuer, M.A.; Preiss, R.J. *Design Automation of Digital Systems*; Prentice Hall: Hoboken, NJ, USA, 1972; Volume 1.
- [133] Slagle, J.R. *Artificial Intelligence: The Heuristic Programming Approach*; McGraw-Hill: New York, NY, USA, 1971.
- [134] Kohavi, Z.; Jha, N.K. *Switching and Finite Automata Theory*; Cambridge University Press: Cambridge, UK, 2009.
- [135] Lieberherr, K.; Specker, E. Complexity of partial satisfaction. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, Washington, DC, USA, 29–31 October 1979; IEEE: Manhattan, NY, USA, 1979; pp. 132–139.
- [136] Perkowski, M. *State-Space Approach to the Design of a Multipurpose Problem-Solver for Logic Design*; KU Leuven: North Holland, Amsterdam, 1978.
- [137] Perkowski, M. "Synthesis of multioutput three level NAND networks." In *Proceedings of the Seminar on Computer Aided Design*. Budapest, Hungary, pp. 238-265. 1976.
- [138] Perkowski, M. "Minimization of two-level networks from negative gates." In *Proceedings Midwest 86 Conference on Circuits and Systems*. 1986.

- [139] Perkowski, M.; Liu, J.; Brown, J. A System for Fast Prototyping of Logic Design Programs. In Proceedings of the 1987 Midwest Symposium on Circuits and Systems, Syracuse, NY, USA, 17–18 August 1987.
- [140] Sasao, T. Input variable assignment and output phase optimization of PLA's. IEEE Trans. Comput. 1984, 33, 879–894.
- [141] Nilsson, J. Problem-Solving Methods in Artificial Intelligence; McGraw-Hill: New York, NY, USA, 1971.
- [142] Nguyen, L.B.; Perkowski, M.A.; Goldstein, N.B. Palmi—Fast Boolean minimizer for personal computers. In Proceedings of the 24th ACM/IEEE Design Automation Conference, Miami Beach, FL, USA, 28 June–1 July 1987; pp. 615–621.
- [143] Perkowski, M.; Mishchenko, A. Logic synthesis for regular layout using satisfiability. In Proceedings of the International Symposium on Boolean Problems, Freiberg, Germany, 19–20 September 2002.
- [144] Soos, Mate, Karsten Nohl, and Claude Castelluccia. "Extending SAT solvers to cryptographic problems." In International Conference on Theory and Applications of Satisfiability Testing, pp. 244-257. Springer, Berlin, Heidelberg, 2009.
- [145] Soos, Mate. "CryptoMiniSat 2.5. 0." http://baldur.iti.uka.de/sat-race-2010/descriptions/solver_13.pdf (2007).
- [146] Haanpää, Harri, Matti Jarvisalo, Petteri Kaski, and Ilkka Niemelä. "Hard satisfiable clause sets for benchmarking equivalence reasoning

- techniques." *Journal on Satisfiability, Boolean Modeling and Computation* 2, no. 1-4 (2006): 27-46.
- [147] Gomes, Carla P., Joerg Hoffmann, Ashish Sabharwal, and Bart Selman. "Short XORs for model counting: from theory to practice." In *International Conference on Theory and Applications of Satisfiability Testing*, pp. 100-106. Springer, Berlin, Heidelberg, 2007.
- [148] Soos, Mate, and Kuldeep S. Meel. "BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting." In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 1592-1599. 2019.
- [149] Dudek, Jeffrey M., Kuldeep S. Meel, and Moshe Y. Vardi. "The hard problems are almost everywhere for random CNF-XOR formulas." *arXiv preprint arXiv:1710.06378* (2017).
- [150] Schaefer, Thomas J. "The complexity of satisfiability problems." In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pp. 216-226. 1978.
- [151] Gomes, Carla P., Ashish Sabharwal, and Bart Selman. "Model counting: A new strategy for obtaining good bounds." In *AAAI*, vol. 10, pp. 1597538-1597548. 2006.
- [152] Massacci, Fabio, and Laura Marraro. "Logical cryptanalysis as a SAT problem." *Journal of Automated Reasoning* 24, no. 1 (2000): 165-203.
- [153] Creignou, Nadia, and Hervé Daude. "Satisfiability threshold for random XOR-CNF formulas." *Discrete Applied Mathematics* 96 (1999): 41-53.

- [154] Dzyuba, Vladimir, Matthijs van Leeuwen, and Luc De Raedt. "Flexible constrained sampling with guarantees for pattern mining." *Data Mining and Knowledge Discovery* 31, no. 5 (2017): 1266-1293.
- [155] Soos, Mate. "Enhanced Gaussian Elimination in DPLL-based SAT Solvers." In *POS@ SAT*, pp. 2-14. 2010.
- [156] Soos, Mate, Stephan Gocht, and Kuldeep S. Meel. "Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling." In *International Conference on Computer Aided Verification*, pp. 463-484. Springer, Cham, 2020.
- [157] Trimoska, Monika, Sorina Ionica, and Gilles Dequen. "Parity (XOR) reasoning for the index calculus attack." In *International Conference on Principles and Practice of Constraint Programming*, pp. 774-790. Springer, Cham, 2020.
- [158] Davis, Martin, and Hilary Putnam. "A computing procedure for quantification theory." *Journal of the ACM (JACM)* 7, no. 3 (1960): 201-215.
- [159] Aslan, Murat, Mesut Gunduz, and Mustafa Servet Kiran. "JayaX: Jaya algorithm with xor operator for binary optimization." *Applied Soft Computing* 82 (2019): 105576.
- [160] Sinha, Prokash. "A memory-efficient doubly linked list." *Linux Journal* 2005, no. 129 (2005): 10.
- [161] Bittman, Daniel, Darrell DE Long, Peter Alvaro, and Ethan L. Miller. "Optimizing Systems for {Byte-Addressable} {NVM} by Reducing Bit Flipping."

In 17th USENIX Conference on File and Storage Technologies (FAST 19), pp. 17-30. 2019.

- [162] Lewin, Michael. "All About XOR," June 2012. [Online]. Available: https://accu.org/journals/overload/20/109/lewin_1915/
- [163] Wittek, P., 2014. Quantum machine learning: what quantum computing means to data mining. Academic Press.
- [164] Zhao, Q. and Bhowmick, S.S., 2003. Association rule mining: A survey. Nanyang Technological University, Singapore, 135.
- [165] Kaur, M. and Kang, S., 2016. Market Basket Analysis: Identify the changing trends of market data using association rule mining. Procedia Computer Science, 85, pp.78-85.
- [166] Ünvan, Y.A., 2021. Market basket analysis with association rules. Communications in Statistics-Theory and Methods, 50(7), pp.1615-1628.
- [167] Yi, K., Chen, T. and Cong, G., 2018. Library personalized recommendation service method based on improved association rules. Library Hi Tech, 36(3), pp.443-457.
- [168] Lin, W., Alvarez, S.A. and Ruiz, C., 2002. Efficient adaptive-support association rule mining for recommender systems. Data Mining and Knowledge Discovery, 6, pp.83-105.
- [169] Jooa, J., Bangb, S. and Parka, G., 2016. Implementation of a recommendation system using association rules and collaborative filtering. Procedia Computer Science, 91, pp.944-952.

- [170] Langhnoja, S.G., Barot, M.P. and Mehta, D.B., 2013. Web usage mining using association rule mining on clustered data for pattern discovery. *International Journal of Data Mining Techniques and Applications*, 2(1), pp.141-150.
- [171] Singh, A.K., Kumar, A. and Maurya, A.K., 2014, August. Association rule mining for web usage data to improve websites. In *2014 International Conference on Advances in Engineering & Technology Research (ICAETR-2014)* (pp. 1-6). IEEE.
- [172] Naulaerts, S., Meysman, P., Bittremieux, W., Vu, T.N., Vanden Berghe, W., Goethals, B. and Laukens, K., 2015. A primer to frequent itemset mining for bioinformatics. *Briefings in bioinformatics*, 16(2), pp.216-231.
- [173] Rajak, A. and Gupta, M.K., 2008, February. Association rule mining: applications in various areas. In *Proceedings of international conference on data management, Ghaziabad, India* (pp. 3-7).
- [174] Agrawal, R., Imieliński, T. and Swami, A., 1993, June. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data* (pp. 207-216).
- [175] Han, J., Pei, J. and Kamber, M., 2011. *Data mining: concepts and techniques*. Elsevier.
- [176] Agrawal, R. and Srikant, R., 1994, September. Fast algorithms for mining association rules. In *Proc. 20th int. conf. Very Large Data Bases, VLDB (Vol. 1215, pp. 487-499)*.

- [177] Xie, H., 2021. Research and case analysis of apriori algorithm based on mining frequent item-sets. *Open Journal of Social Sciences*, 9(04), p.458.
- [178] Liu, H. and Wang, B., 2007. An association rule mining algorithm based on a Boolean matrix. *Data Science Journal*, 6, pp.S559-S565.
- [179] Park, J.S., Chen, M.S. and Yu, P.S., 1995. An effective hash-based algorithm for mining association rules. *Acm Sigmod Record*, 24(2), pp.175-186.
- [180] Savasere, A., Omiecinski, E.R. and Navathe, S.B., 1995. An efficient algorithm for mining association rules in large databases. Georgia Institute of Technology.
- [181] Akash, M.B., Mandal, I. and Al Mamun, M.S., 2023. Backward Support Computation Method for Positive and Negative Frequent Itemset Mining. *Journal of Data Analysis and Information Processing*, 11(1), pp.37-48.
- [182] Yu, C.H., Gao, F., Wang, Q.L. and Wen, Q.Y., 2016. Quantum algorithm for association rules mining. *Physical Review A*, 94(4), p.042311.
- [183] Yu, C.H., 2022. Experimental implementation of quantum algorithm for association rules mining. *arXiv preprint arXiv:2204.13634*.
- [184] Bärttschi, A. and Eidenbenz, S., 2019, August. Deterministic preparation of Dicke states. In *International Symposium on Fundamentals of Computation Theory* (pp. 126-139). Springer, Cham.
- [185] Alasow, A., Jin, P. and Perkowski, M., 2022. Quantum Algorithm for Variant Maximum Satisfiability. *Entropy*, 24(11), p.1615.
- [186] Canteaut, A. and Videau, M., 2005. Symmetric boolean functions. *IEEE Transactions on information theory*, 51(8), pp.2791-2811.

- [187] Wong, T.G., 2016. Quantum walk search on Johnson graphs. *Journal of Physics A: Mathematical and Theoretical*, 49(19), p.195303.
- [188] Collins, H. and Easterly, K., 2021. IBM unveils breakthrough 127-qubit quantum processor. *Atualizado em*, 16(11).
- [189] Portugal, Renato. *Quantum walks and search algorithms*. Vol. 19. New York: Springer, 2013.
- [190] Brown, Adam R., and Leonard Susskind. "Second law of quantum complexity." *Physical Review D* 97, no. 8 (2018): 086015.
- [191] Benioff, Paul. "Space searches with a quantum robot." *arXiv preprint quant-ph/0003006* (2000).
- [192] Ambainis, Andris. "Quantum search algorithms." *ACM SIGACT News* 35, no. 2 (2004): 22-35.
- [193] Ambainis, Andris, Julia Kempe, and Alexander Rivosh. "Coins make quantum walks faster." *arXiv preprint quant-ph/0402107* (2004).
- [194] Ambainis, Andris. "Quantum walks and their algorithmic applications." *International Journal of Quantum Information* 1, no. 04 (2003): 507-518.
- [195] Ambainis, Andris. "Quantum walk algorithm for element distinctness." *SIAM Journal on Computing* 37, no. 1 (2007): 210-239.
- [196] Shi, Yaoyun. "Quantum lower bounds for the collision and the element distinctness problems." In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pp. 513-519. IEEE, 2002.

- [197] Bernstein, Ethan, and Umesh Vazirani. "Quantum complexity theory." In Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, pp. 11-20. 1993.
- [198] Librande, Jonah. "BQP is not in NP." arXiv preprint arXiv:2209.10398 (2022).
- [199] Arora, Sanjeev, and Boaz Barak. Computational complexity: a modern approach. Cambridge University Press, 2009.
- [200] Aaronson, Scott. "How much structure is needed for huge quantum speedups?." arXiv preprint arXiv:2209.06930 (2022).
- [201] Raz, Ran, and Avishay Tal. "Oracle separation of BQP and PH." ACM Journal of the ACM (JACM) 69, no. 4 (2022): 1-21.
- [202] Kim, Panjin, Daewan Han, and Kyung Chul Jeong. "Time–space complexity of quantum search algorithms in symmetric cryptanalysis: applying to AES and SHA-2." Quantum Information Processing 17 (2018): 1-39.
- [203] Campbell, Earl, Ankur Khurana, and Ashley Montanaro. "Applying quantum algorithms to constraint satisfaction problems." Quantum 3 (2019): 167.
- [204] Yu, Chao-Hua, Fei Gao, Qing-Le Wang, and Qiao-Yan Wen. "Quantum algorithm for association rules mining." Physical review A 94, no. 4 (2016): 042311.
- [205] Yu, Chao-Hua. "Experimental implementation of quantum algorithm for association rules mining." IEEE Journal on Emerging and Selected Topics in Circuits and Systems 12, no. 3 (2022): 676-684.

- [206] Wang, Ze-Guo, Shi-Jie Wei, and Gui-Lu Long. "A quantum circuit design of AES requiring fewer quantum qubits and gate operations." *Frontiers of Physics* 17, no. 4 (2022): 41501.
- [207] Langenberg, Brandon, Hai Pham, and Rainer Steinwandt. "Reducing the cost of implementing the advanced encryption standard as a quantum circuit." *IEEE Transactions on Quantum Engineering* 1 (2020): 1-12.
- [208] Grassl, Markus, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. "Applying Grover's algorithm to AES: quantum resource estimates." In *International Workshop on Post-Quantum Cryptography*, pp. 29-43. Cham: Springer International Publishing, 2016.
- [209] Li, Zhenqiang, Fei Gao, Sujuan Qin, and Qiaoyan Wen. "New record in the number of qubits for a quantum implementation of AES." *Frontiers in Physics* 11 (2023): 1171753.
- [210] Dasu, Vishnu Asutosh, Anubhab Baksi, Sumanta Sarkar, and Anupam Chattopadhyay. "LIGHTER-R: optimized reversible circuit implementation for sboxes." In *2019 32nd IEEE International System-on-Chip Conference (SOCC)*, pp. 260-265. IEEE, 2019.
- [211] Jaques, Samuel, Michael Naehrig, Martin Roetteler, and Fernando Virdia. "Implementing Grover oracles for quantum key search on AES and LowMC." In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*,

Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30, pp. 280-310. Springer International Publishing, 2020.

- [212] Wilson, Ellis, Sudhakar Singh, and Frank Mueller. "Just-in-time quantum circuit transpilation reduces noise." In 2020 IEEE International Conference on Quantum Computing and Engineering (QCE), pp. 345-355. IEEE, 2020.
- [213] De Micheli, Giovanni. Synthesis and optimization of digital circuits. McGraw Hill, 1994.
- [214] Hachtel, Gary D., and Fabio Somenzi. Logic synthesis and verification algorithms. Springer Science & Business Media, 2007.
- [215] Rudell, Richard L. "Logic synthesis for VLSI design." PhD diss., University of California, Berkeley, 1989.
- [216] Bramel, Julien, and David Simchi-Levi. "On the effectiveness of set covering formulations for the vehicle routing problem with time windows." Operations Research 45, no. 2 (1997): 295-301.
- [217] Kam, Timothy, Tiziano Villa, Robert Brayton, and Alberto Sangiovanni-Vincentelli. "A fully implicit algorithm for exact state minimization." In 31st Design Automation Conference, pp. 684-690. IEEE, 1994.
- [218] Jeong, Seh-Woong, and Fabio Somenzi. "A new algorithm for the binate covering problem and its application to the minimization of boolean relations." In ICCAD, vol. 92, pp. 417-420. 1992.
- [219] Liao, Stan, Srinivas Devadas, Kurt Keutzer, and Steve Tjiang. "Instruction selection using binate covering for code size optimization." In Proceedings of

- IEEE International Conference on Computer Aided Design (ICCAD), pp. 393-399. IEEE, 1995.
- [220] Petrick, Stanley R. "A direct determination of the irredundant forms of a Boolean function from the set of prime implicants." Air Force Cambridge Res. Center Tech. Report (1956): 56-110.
- [221] Breuer, Melvin A., and Ralph J. Preiss. Design automation of digital systems. Vol. 1. Prentice Hall, 1972.
- [222] Kohavi, Zvi, and Niraj K. Jha. Switching and finite automata theory. Cambridge University Press, 2009.
- [223] McCluskey, Edward J. "Minimization of Boolean functions." The Bell System Technical Journal 35, no. 6 (1956): 1417-1444.
- [224] Rudell, Richard L. Multiple-valued logic minimization for PLA synthesis. Tech. Rep. UCB/ERL M86/65 EECS Department University of California Berkeley, 1986.
- [225] Villa, Tiziano, Timothy Kam, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. "Explicit and implicit algorithms for binate covering problems." IEEE Transactions on computer-Aided Design of integrated Circuits and Systems 16, no. 7 (1997): 677-691.
- [226] Liao, Stan, Kurt Keutzer, S. T. E. V. E. N. Tjiang, and Srinivas Devadas. "A new viewpoint on code generation for directed acyclic graphs." ACM Transactions on Design Automation of Electronic Systems (TODAES) 3, no. 1 (1998): 51-75.

- [227] Sasao, Tsutomu, ed. Logic synthesis and optimization. Vol. 2. MA: Kluwer Academic Publishers, 1993.
- [228] Paul, Eric, Bernd Steinbach, and Marek Perkowski. "Application of CUDA in the Boolean domain for the unate covering problem." (2010).
- [229] Mesquita, Marta, and Ana Paia. "Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem." *Computers & Operations Research* 35, no. 5 (2008): 1562-1575.
- [230] Li, Xiao Yu. Optimization algorithms for the minimum-cost satisfiability problem. North Carolina State University, 2004.
- [231] Kam, T., Villa, T., Brayton, R.K. and Sangiovanni-Vincentelli, A.L., 2013. Synthesis of finite state machines: functional optimization. Springer Science & Business Media.
- [232] Li, Xiao Yu, Matthias F. Stallmann, and Franc Brglez. "Effective bounding techniques for solving unate and binate covering problems." In *Proceedings of the 42nd annual Design Automation Conference*, pp. 385-390. 2005.
- [233] Kagliwal, Ankit, and Shankar Balachandran. "Set-cover heuristics for two-level logic minimization." In *2012 25th International Conference on VLSI Design*, pp. 197-202. IEEE, 2012.
- [234] Steinbach, Bernd, and Matthias Werner. "Alternative Approaches for Fast Boolean Calculations Using the GPU." In *Computational Intelligence and Efficiency in Engineering Systems*, pp. 17-31. Cham: Springer International Publishing, 2015.

- [235] Steinbach, Bernd, and Christian Posthoff. "Fast calculation of exact minimal unate coverings on both the CPU and the GPU." In Computer Aided Systems Theory-EUROCAST 2013: 14th International Conference, Las Palmas de Gran Canaria, Spain, February 10-15, 2013. Revised Selected Papers, Part II 14, pp. 234-241. Springer Berlin Heidelberg, 2013.
- [236] Steinbach, Bernd, and Christian Posthoff. "Sources and obstacles for parallelization-a comprehensive exploration of the unate covering problem using both CPU and GPU." GPU Computing with Applications in Digital Logic 63 (2012).
- [237] Servit, M. and Zamazal, J., 1992, January. Heuristic approach to binate covering problem. In Proceedings The European Conference on Design Automation (pp. 123-124). IEEE Computer Society.
- [238] Perkowski, Marek A., Jiuling Liu, and James E. Brown. "Rapid software prototyping: CAD design of digital CAD algorithms." In Progress in computer-aided VLSI design, pp. 353-401. 1990.
- [239] The MCNC Benchmark Problems for VLSI Floorplanning. [Online]. Available: <http://www.mcnc.org>
- [240] Brglez, Franc, and Hideo Fujiwara. "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran." In Proc. Intl. Symp. Circuits and Systems, 1985. 1985.

- [241] Michael Brooks, IBM wants to build a 100,000-qubit quantum computer. URL: <https://www.technologyreview.com/2023/05/25/1073606/ibm-wants-to-build-a-100000-qubit-quantum-computer/> (visited on 6/15/2023)
- [242] Alasow, Abdirahman, and Marek Perkowski. "Quantum Algorithm for Mining Frequent Patterns for Association Rule Mining." *Journal of Quantum Information Science* 13, no. 1 (2023): 1-23.
- [243] Tommy R Jensen and Bjarne Toft. *Graph coloring problems*, volume 39. John Wiley & Sons, 2011.
- [244] Draper, Thomas G. "Addition on a quantum computer." arXiv preprint [quant-ph/0008033](https://arxiv.org/abs/quant-ph/0008033) (2000).
- [245] Cuccaro, Steven A., Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton. "A new quantum ripple-carry addition circuit." arXiv preprint [quant-ph/0410184](https://arxiv.org/abs/quant-ph/0410184) (2004).
- [246] Takahashi, Yasuhiro, Seiichiro Tani, and Noboru Kunihiro. "Quantum addition circuits and unbounded fan-out." arXiv preprint [arXiv:0910.2530](https://arxiv.org/abs/0910.2530) (2009).
- [247] Gayathri, S. S., R. Kumar, Samiappan Dhanalakshmi, Brajesh Kumar Kaushik, and Majid Haghparast. "T-count optimized wallace tree integer multiplier for quantum computing." *International Journal of Theoretical Physics* 60, no. 8 (2021): 2823-2835.
- [248] Wallace, Christopher S. "A suggestion for a fast multiplier." *IEEE Transactions on electronic Computers* 1 (1964): 14-17.

- [249] Waters, Ron S., and Earl E. Swartzlander. "A reduced complexity wallace multiplier reduction." *IEEE transactions on Computers* 59, no. 8 (2010): 1134-1137.
- [250] Haghparast, Majid, Majid Mohammadi, Keivan Navi, and Mohammad Eshghi. "Optimized reversible multiplier circuit." *Journal of Circuits, Systems, and Computers* 18, no. 02 (2009): 311-323.
- [251] PourAliAkbar, Ehsan, and Mohammad Mosleh. "An efficient design for reversible wallace unsigned multiplier." *Theoretical Computer Science* 773 (2019): 43-52.
- [252] Yato, Takayuki, and Takahiro Seta. "Complexity and completeness of finding another solution and its application to puzzles." *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 86, no. 5 (2003): 1052-1060.
- [253] Mantere, Timo, and Janne Koljonen. "Solving and rating sudoku puzzles with genetic algorithms." In *New Developments in Artificial Intelligence and the Semantic Web, Proceedings of the 12th Finnish Artificial Intelligence Conference STeP*, pp. 86-92. 2006.
- [254] Maji, Arnab Kumar, and Rajat Kumar Pal. "Sudoku solver using minigrid based backtracking." In *2014 IEEE International Advance Computing Conference (IACC)*, pp. 36-44. IEEE, 2014.

- [255] McGuire, Gary, Bastian Tugemann, and Gilles Civario. "There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem via hitting set enumeration." *Experimental Mathematics* 23, no. 2 (2014): 190-217.
- [256] De Biasi, Marzio. "Binary puzzle is NP-complete." 2012. URL: <http://www.nearly42.org/vdisk/cstheory/binary.pdf>.
- [257] Utomo, Putranto Hadi, and G. R. Pellikaan. "Binary puzzles as an erasure decoding problem." In *Proceedings of the 36th WIC Symposium on Information Theory in the Benelux (Brussels, Belgium, May 6-7, 2015)*, pp. 129-134. Werkgemeenschap voor Informatie-en Communicatietheorie (WIC), 2015.
- [258] Nori, Uday Kiran, and Mihir Vishindas. "TORUS BINARY PUZZLE." (2023). [online]. Available: https://www.researchgate.net/profile/Uday-Kiran-16/publication/375060856_Torus_Binary_Puzzle/links/653e329bf7d021785f1e4671/Torus-Binary-Puzzle.pdf
- [259] Grassl, M., Langenberg, B., Roetteler, M. and Steinwandt, R., 2016, February. Applying Grover's algorithm to AES: quantum resource estimates. In *International Workshop on Post-Quantum Cryptography* (pp. 29-43). Cham: Springer International Publishing.
- [260] Ju, Y.L., Tsai, I.M. and Kuo, S.Y., 2007. Quantum circuit design and analysis for database search applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(11), pp.2552-2563.

- [261] Yang, W.L., Wei, H., Zhou, F., Chang, W.L. and Feng, M., 2009. Solution to the satisfiability problem using a complete Grover search with trapped ions. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 42(14), p.145503.
- [262] Gueddana, A., Chatta, R. and Attia, M., 2014. CNOT-based design and query management in quantum relational databases. *International Journal of Quantum Information*, 12(04), p.1450023.
- [263] Shi-man, X. and Xin-zhi, S., 2012. The building and optimization of quantum database. *Physics Procedia*, 25, pp.1602-1609.
- [264] Wong, R. and Chang, W.L., 2021. Quantum speedup for protein structure prediction. *IEEE Transactions on NanoBioscience*, 20(3), pp.323-330.
- [265] Ngoc, V.P., Tuckey, D. and Wiklicky, H., 2022. Tunable Quantum Neural Networks in the QPAC-Learning Framework. arXiv preprint arXiv:2205.01514.
- [266] A. A-Bayaty and M. Perkowski, "A concept of controlling Grover diffusion operator: A new approach to solve arbitrary Boolean-based problems," submitted to *Scientific Reports*, 2023. Nature.
- [267] Martonosi, Margaret, and Martin Roetteler. "Next steps in quantum computing: Computer science's role." arXiv preprint arXiv:1903.10541 (2019).
- [268] Shor, Peter W. "Why haven't more quantum algorithms been found?." *Journal of the ACM (JACM)* 50, no. 1 (2003): 87-90.
- [269] Cao, Hao, and Wenping Ma. "Multiparty quantum key agreement based on quantum search algorithm." *Scientific reports* 7, no. 1 (2017): 45046.

- [270] Grassl, Markus, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. "Applying Grover's algorithm to AES: quantum resource estimates." In International Workshop on Post-Quantum Cryptography, pp. 29-43. Cham: Springer International Publishing, 2016.
- [271] Bonnetain, Xavier, Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki, and André Schrottenloher. "Quantum attacks without superposition queries: the offline Simon's algorithm." In International Conference on the Theory and Application of Cryptology and Information Security, pp. 552-583. Cham: Springer International Publishing, 2019.
- [272] Bhatia, Vaishali, and K. R. Ramkumar. "An efficient quantum computing technique for cracking RSA using Shor's algorithm." In 2020 IEEE 5th international conference on computing communication and automation (ICCCA), pp. 89-94. IEEE, 2020.
- [273] Nagata, Koji, Tadao Nakamura, and Ahmed Farouk. "Quantum cryptography based on the Deutsch-Jozsa algorithm." International Journal of Theoretical Physics 56 (2017): 2887-2897.
- [274] Wang, G., 2017. Quantum algorithm for linear regression. Physical review A, 96(1), p.012335.
- [275] Zhao, Zhikuan, Jack K. Fitzsimons, and Joseph F. Fitzsimons. "Quantum-assisted Gaussian process regression." Physical Review A 99, no. 5 (2019): 052331.
- [276] Chakraborty, Shantanav, Aditya Morolia, and Anurudh Peduri. "Quantum regularized least squares." Quantum 7 (2023): 988.

- [277] Li, YaoChong, Ri-Gui Zhou, RuQing Xu, Jia Luo, and WenWen Hu. "A quantum deep convolutional neural network for image recognition." *Quantum Science and Technology* 5, no. 4 (2020): 044003.
- [278] Lloyd, Seth, and Christian Weedbrook. "Quantum generative adversarial learning." *Physical review letters* 121, no. 4 (2018): 040502.
- [279] de Souza, Luciano S., Jonathan HA de Carvalho, and Tiago AE Ferreira. "Classical artificial neural network training using quantum walks as a search procedure." *IEEE Transactions on Computers* 71, no. 2 (2021): 378-389.
- [280] Bauer, Bela, Sergey Bravyi, Mario Motta, and Garnet Kin-Lic Chan. "Quantum algorithms for quantum chemistry and quantum materials science." *Chemical Reviews* 120, no. 22 (2020): 12685-12717.
- [281] Cao, Yudong, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan et al. "Quantum chemistry in the age of quantum computing." *Chemical reviews* 119, no. 19 (2019): 10856-10915.
- [282] Botsinis, Panagiotis, Dimitrios Alanis, Zunaira Babar, Hung Viet Nguyen, Daryus Chandra, Soon Xin Ng, and Lajos Hanzo. "Quantum search algorithms for wireless communications." *IEEE Communications Surveys & Tutorials* 21, no. 2 (2018): 1209-1242.
- [283] Nawaz, Syed Junaid, Shree Krishna Sharma, Shurjeel Wyne, Mohammad N. Patwary, and Md Asaduzzaman. "Quantum machine learning for 6G communication networks: State-of-the-art and vision for the future." *IEEE access* 7 (2019): 46317-46350.

- [284] Qu, Zhiguo, and Hanrong Sun. "A secure information transmission protocol for healthcare cyber based on quantum image expansion and Grover search algorithm." *IEEE Transactions on Network Science and Engineering* (2022).
- [285] Egger, Daniel J., Claudio Gambella, Jakub Marecek, Scott McFaddin, Martin Mevissen, Rudy Raymond, Andrea Simonetto, Stefan Woerner, and Elena Yndurain. "Quantum computing for finance: State-of-the-art and future prospects." *IEEE Transactions on Quantum Engineering* 1 (2020): 1-24.
- [286] Herman, Dylan, Cody Googin, Xiaoyuan Liu, Yue Sun, Alexey Galda, Ilya Safro, Marco Pistoia, and Yuri Alexeev. "Quantum computing for finance." *Nature Reviews Physics* 5, no. 8 (2023): 450-465.