Methods for Efficient Synthesis of Large Reversible Binary and Ternary Quantum Circuits and Applications of Linear Nearest Neighbor Model

Maher Mofeid Hawash

Portland State University

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

Part of the Other Electrical and Computer Engineering Commons

Let us know how access to this document benefits you.

Recommended Citation
https://doi.org/10.15760/etd.1090

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.
Methods for Efficient Synthesis of Large Reversible Binary and Ternary Quantum Circuits and Applications of Linear Nearest Neighbor Model

by

Maher Mofeid Hawash

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
Douglas Hall
Malgorzata Chrzanowska-Jeske
Christof Teuscher
John Caughman

Portland State University
2013
Abstract

This dissertation describes the development of automated synthesis algorithms that construct reversible quantum circuits for reversible functions with large number of variables. Specifically, the research area is focused on reversible, permutative and fully specified binary and ternary specifications and the applicability of the resulting circuit to the physical limitations of existing quantum technologies.

Automated synthesis of arbitrary reversible specifications is an NP hard, multi-objective optimization problem, where 1) the amount of time and computational resources required to synthesize the specification, 2) the number of primitive quantum gates in the resulting circuit (quantum cost), and 3) the number of ancillary qubits (variables added to hold intermediate calculations) are all minimized while 4) the number of variables is maximized. Some of the existing algorithms in the literature ignored objective 2 by focusing on the synthesis of a single solution without the addition of any ancillary qubits while others attempted to explore every possible solution in the search space in an effort to discover the optimal solution (i.e., sacrificed objective 1 and 4). Other algorithms resorted to adding a huge number of ancillary qubits (counter to objective 3) in an effort minimize the number of primitive gates (objective 2).

In this dissertation, I first introduce the MMDSN algorithm that is capable of synthesizing binary specifications up to 30 variables, does not add any ancillary variables, produces better quantum cost (8-50% improvement) than algorithms which limit their search to a single solution and within a minimal amount of time compared to
algorithms which perform exhaustive search (seconds vs. hours). The MMDSN algorithm introduces an innovative method of using the Hasse diagram to construct candidate solutions that are guaranteed to be valid and then selects the solution with the minimal quantum cost out of this subset.

I then introduce the Covered Set Partitions (CSP) algorithm that expands the search space of valid candidate solutions and allows for exploring solutions outside the range of MMDSN. I show a method of subdividing the expansive search landscape into smaller partitions and demonstrate the benefit of focusing on partition sizes that are around half of the number of variables (15% to 25% improvements, over MMDSN, for functions less than 12 variables, and more than 1000% improvement for functions with 12 and 13 variables). For a function of $n$ variables, the CSP algorithm, theoretically, requires $n$ times more to synthesize; however, by focusing on the middle $k$ ($k<n$) partitions, the CSP algorithms only requires $k$ times the amount of time required by MMDSN which typically yields lower quantum cost. I also show that using a Tabu search for selecting the next set of candidate from the CSP subset results in discovering solutions with even lower quantum costs (up to 10% improvement over CSP with random selection).

In Chapters 9 and 10 I question the predominant methods of measuring quantum cost and its applicability to physical implementation of quantum gates and circuits. I counter the prevailing literature by introducing a new standard for measuring the performance of quantum synthesis algorithms by enforcing the Linear Nearest Neighbor Model (LNNM).
constraint, which is imposed by the today’s leading implementations of quantum technology. In addition to enforcing physical constraints, the new LNNM quantum cost (LNNQC) allows for a level comparison amongst all methods of synthesis; specifically, methods which add a large number of ancillary variables to ones that add no additional variables. I show that, when LNNM is enforced, the quantum cost for methods that add a large number of ancillary qubits increases significantly (up to 1200%).

I also extend the Hasse based method to the ternary and I demonstrate synthesis of specifications of up to 9 ternary variables (compared to 3 ternary variables that existed in the literature). I introduce the concept of ternary precedence order and its implication on the construction of the Hasse diagram and the construction of valid candidate solutions. I also provide a case study comparing the performance of ternary logic synthesis of large functions using both a CUDA graphic processor with 1024 cores and an Intel i7 processor with 8 cores. In the process of exploring large ternary functions I introduce, to the literature, eight families of ternary benchmark functions along with a Multiple Valued file specification (the Extended Quantum Specification XQS). I also introduce a new composite quantum gate, the multiple valued Swivel gate, which swaps the information of qubits around a centrally located pivot point.

In summary, my research objectives are as follows:

- Explore and create automated synthesis algorithms for reversible circuits both in binary and ternary logic for large number of variables.
• Study the impact of enforcing *Linear Nearest Neighbor Model (LNNM)* constraint for every interaction between qubits for reversible binary specifications.

• Advocate for a revised metric for measuring the *cost* of a quantum circuit in concordance with LNNM, where, on one hand, such a metric would provide a way for balanced comparison between the various flavors of algorithms, and on the other hand, represents a realistic cost of a quantum circuit with respect to an ion trap implementation.

• Establish an open source repository for sharing the results, software code and publications with the scientific community.

With the dwindling expectations for a new lifeline on silicon-based technologies, quantum computations have the potential of becoming the future workhorse of computations. Similar to the automated CAD tools of classical logic, my work lays the foundation for creating automated tools for constructing quantum circuits from reversible specifications.
Dedication

*I dedicate this dissertation to*

My loving parents Mofeid and Raefah Hawash;

My wife Lisa for her unwavering support;

My wonderful children Jared, Sarra and Amani;

My brother Shaher;

My Sisters Shaheera, Maheera, Mohaya & Majdoleen;

And my family and friends around the globe.
Acknowledgments

I offer my sincere gratitude to many individuals for their support, advice and motivation to help fulfill my aspiration to attain a PhD in Electrical and Computer Engineering. I am deeply indebted to Dr. Marek A. Perkowski for his transcendent patronage and mentorship throughout the years of work on my PhD and for his interminable motivation and inspirations to help me blaze forward. I am greatly impressed by the tremendous effort he expends for the sake of his students specially the time and energy he spends in nurturing middle and high school students in quantum computing and robotics. I have personally benefited throughout the years from the lectures and discussions in his weekly Friday meeting and from working directly with him in developing ideas, reviewing results and drafting papers for publishing. I am committed to, in addition to absorbing his well of knowledge, by following in Dr. Perkowski’s footsteps in mentoring young students to attain their own scientific achievement.

I would also like to extend my deepest appreciations to the committee members who have given me valuable feedback for improving the quality of results from my simulation of quantum logic synthesis and for introducing me to new concepts. I am grateful for Dr. Douglas Hall for the multiple roles that he took as my graduate advisor and a supportive member of my committee. Dr. Malgorzata Chrzanowska-Jeske has been extremely helpful in reviewing my dissertation and giving me valuable feedback in regards to organization and content. I am greatly appreciative of her insightful questions and
constructive advice during my comprehensive exam and dissertation proposal. I cannot thank Dr. Christof Teuscher enough for expanding my field of vision by introducing me to the stimulating and promising nanotechnology, which I hope to dedicate more time to learn and explore in the future. Since my first encounter of Dr. John Caughman, I have been greatly impressed, and slightly jealous, by his fluency in one of my favorite childhood topics, Mathematics. I am grateful for the spark that he and Dr. Steven Bleiler ignited which allowed me to develop the CSP algorithm.

My mother, brother and sisters have been a beacon of guidance, a fount of love and a whisper of encouragement. Undoubtedly, I would be ungrateful to overlook the arduous burden that I have levied upon my wife and children as I thieved the time meant for them for the sake of my PhD and who have tolerated my rants and unpleasantness as I struggled with some code or papers to finish. Their love and support account for nine tenth of my PhD. I close by thanking my friends Naveed and Rania Ali, Steven McGeady, Stephen Houze, Leora Gregory, Rohan and Kathryn Coelho and my fellow co-workers at Renewable Funding for their unrelenting support and encouragement.
# TABLE OF CONTENTS

Abstract i
Dedication v
Acknowledgments vi
List of Tables xv
List of Figures xix
Glossary of Terms xxxii
Introduction 1
Summary of Research Objectives 4
My Contributions 10
Chapter 1 Introduction 18
  1.1 Included in this research 26
  1.2 Excluded from this research 26
  1.3 Approach 27
  1.4 Organization of Chapters 28
Chapter 2 Background 30
  2.1 Feasibility of Quantum Computation 32
Chapter 3 Binary and Ternary Quantum Gates

3.1 Introduction

3.2 Binary Quantum Gates

3.2.1 Inverter NOT gate
3.2.2 Feynman or Controlled NOT gate
3.2.3 Toffoli and MCT gates
3.2.4 Fredkin or Swap gate
3.2.5 Square Root of Not (V) gate
3.2.6 Hadamard Gate

3.3 Ternary (and Multiple Valued) Quantum Logic gates

3.3.1 Galois Field 3 Logic – GF(3)

3.4 Mathematical representation of qubit state

3.4.1 Bloch Sphere and Quantum State
3.4.2 Evolution of Quantum State

3.5 History of Ion Trap

3.5.1 Operation of the trap
3.5.2 One Qubit Operation
3.5.3 Entanglement
3.5.4 CNOT Gate
3.5.5 Quantum Cost Calculation
3.5.6 Ternary Quantum gates in an ion-trap system

3.6 Methods of Quantum Logic Synthesis
3.6.1 Heuristic methods
3.6.2 Cycle Decomposition
3.6.3 Hierarchical Diagrams
3.6.4 Linear Nearest Neighbor Model (LNNM)
3.6.5 Measurement Quality of Reversible Gates

Chapter 4 MMDSN and MP algorithms

4.1 Introduction
4.2 Explanation of MMDs main idea
4.3 MMDS and MMDSN Orderings

1) Proof of Convergence:

4.4 Multi-Pass Algorithm
4.5 Results of the MMDSN/MP for more than four Variables
4.6 Analysis and Conclusion

Chapter 5 Covered Set Partitions

5.1 Introduction
5.2 MMD Style Algorithms
5.3 Partial Covering Set Partitions
5.4 CSP Algorithm
5.5 Experimental Results
5.6 Analysis and Conclusion

Chapter 6 Covered Set Partition with Evolutionary Algorithms

6.1 Introduction
6.2 MMD Style Algorithms
6.3 Anatomy of Covered Set Partition Algorithm

6.3.1 Structure
6.3.2 Steps for creating valid sequences

6.4 Algorithmic Contest

6.4.1 Objective function using Quantum Cost
6.4.2 Method 1: A Random Skip and Hop
6.4.3 Method 2: Genetic Algorithm
6.4.4 Method 3: Tabu Search

6.5 Experimental Results
6.6 Conclusion and Analysis

Chapter 7 Other attempts at discovering better sequences

7.1 Hamming Distance Predictor
7.2 Absolute Distance Predictor
7.3 Vector Length Predictor

Chapter 8 Linear Nearest Neighbor Model for Binary

8.1 Introduction
8.2 Toffoli Gate Cost
8.3 Count of CNOT and CV Gates for MCT Gate
8.4 LNN Quantum Cost of MCT\textsubscript{n} Gates
8.5 Circuit Depth
8.6 Experimental Results
8.7 Conclusion and Analysis
Chapter 9 Multi-Dimensional LNNM Architecture

9.1 Introduction 157
9.2 MCT_{4} in 2-D 158
9.3 MCT_{5} in 2-D 160
9.4 Conclusion and Analysis 162

Chapter 10 Synthesis of Ternary Quantum Circuits

10.1 Prologue 165
10.2 Ternary Logic System 168

10.2.1 Measurement of a qubit 168
10.2.2 Trits and Ternary States 169
10.2.3 Reversible Operations 170

10.3 Ternary Reversible Operators 173
10.4 Synthesis by Example 176
10.5 Ternary Logic Synthesis Algorithm 179

10.5.1 Control Line Blocking 180
10.5.2 Ternary Hasse Input Sequence 181
10.5.3 Construction of Input Sequence 183
10.5.4 Hasse Precedence Quandary 186

10.6 Convergence of Algorithm 188
10.7 Convergence of Triangular Hasse Precedence Orders 195
10.8 Selection through Genetic algorithm 197

10.8.1 Objective function using Quantum Gate Count 199
10.8.2 Genetic Algorithm 200
12.6 Quantum Gate Count of Ternary Swivel Gate 240
12.7 MV Swap Gate 241
12.8 MV Swivel Gate 241
12.9 Quantum Gate Count of Ternary Swivel Gate 242
12.10 Conclusion 243

Chapter 13 Final Conclusion 244

13.1 Accomplishments of my research 247

13.1.1 Automated Logic Synthesis of Large Binary Specifications 247
13.1.2 MMDSN Algorithm: 248
13.1.3 Covered Set Partition Algorithm 249
13.1.4 Redefinition of Benchmarks Measurement 250
13.1.5 Ternary Quantum Synthesis 251
13.1.6 Extended Quantum Specification and MV function generator 253
13.1.7 Generalized Multiple Valued Swivel Gate 254

13.2 Future Work 255

List of Publications 256

Bibliography 258
List of Tables

Table 3-1 Galois Field 3 operations (modulo 3) for a single qutrit with addition operator (a) and product operator in (b). The addition operator uses the same symbol as the binary XOR symbol (⊕) since, mathematically, the binary XOR operation is a Galois Field 2 (modulo 2) operation. ................................................................. 45

Table 3-2 Ternary operators based on GF(3) addition operator where each gate (operator) is a mathematical bijection with one-to-one and onto mapping from the input to the output. ................................................................. 45

Table 4-1 MMD method illustrated with truth tables of intermediate functions. Notation a → c means c = c ⊕ a which “inverts c if a=1”.

Control lines are underlined and shaded minterms are completed and should not be modified. The goal of the algorithm is to insert quantum gates in order to transform the input vector (column ABC) into the input vector (column abc = column 6). Starting from the top, the algorithm skips the first row since abc = ABC and processes the second row where the output minterm ‘100’ is transformed into the input minterm ‘001’ by placing two conditional inverters on qubits ‘A’ and ‘C’. The process continues until column abc = column 6. ................................................................. 78

Table 4-2 Comparison of MMDSN to both MMD and MMDS with respect to quantum cost and duration of synthesis ................................................................. 92

Table 5-1 Results of synthesizing 16 functions of different number of variable for every possible partitions size for the specific function. Each table includes the average quantum cost over 5 runs, along with the standard deviation for the runs and the ratio of the standard deviation to the mean (a measure of variance in results). The majority of results show consistency between runs with a variance below 3% while few exhibited higher error around 9%. The shaded number represents the partition size with the lowest average quantum cost. For functions below 12 bits,
all functions witnessed the best quantum cost where the partition size is around half the number of variables.  

Table 6-1 Quantum Cost of elementary gates.  
Table 6-2 Comparison for five large functions between four different selection methods (random, GA with single point crossover, GA with double point crossover, and Tabu search). By far, the Tabu search has outperformed all other methods when compared for the same partitions size as shown by the lightly shaded areas. For each function, the partition size also had an impact the outcome where similar to the random selection; partition sizes around the midpoint of the number of variables yield the best results – shaded in dark gray.  

Table 7-1 Hamming distance between input (ab) and output (AB) vectors. In (a), the HD=4 when the qubits are arranged in this manner, however, if the outputs are swapped (BA), then the hamming distance=0. I assumed that it is feasible that by rearranging the qubits and synthesizing the modified function might yield better results. However, preliminary experimentation with different functions did not indicate that this would be a successful strategy.  

Table 7-2 Absolute Distance |d| of minterms between input and output vectors. In (a) the input minterm (01) is one position away from its location in the output vector. Similarly, the (10) input minterm is one position away from its location in the output vector, which combined, results in |d|=2. Similarly for (b) where |d|=4.  

Table 8-1 LNNQC for MCT_3 to MCT_9 gates.  
Table 8-2 Current quantum cost used in literature compared to LNNQC.  
Table 8-3 RevLib Benchmarks with LNNQC [30].  

Table 10-1 Shows the band number where each minterm appears where the value of x is shown in the upper header. For row (2x), the minterms (20, 21, 22) appear in bands (2,3,4) consecutively which is guaranteed by the structure of the Hasse diagram. I rely on this ordering for the proof of convergence.
Table 10-2 All possible permutations vs. total permutations for Hasse based sequences where all possible solutions are shown in column 3, while the number of solutions constructed using the Hasse structure are shown in column 4................................. 199

Table 10-3 Comparison between using the natural order of the input vector vs. using the Hasse structure to construct valid input vector arrangements. As the search space increases, the probability of discovering better solutions decreases while the time required to discover such solutions increases drastically. ........................................ 203

Table 10-4 Performance times for the HWT5 function which shows that, for CUDA, at full thread capacity, it is able to execute at 55 microseconds per sample compared to 304 microseconds for the CPU. .......................................................... 206

Table 10-5 Structure of GA algorithm on both CPU and CUDA implementations........ 207

Table 10-6 Performance times for the HWT6 function where, at full capacity, CUDA takes 376 microseconds per sample compared to the CPU at 645 710 microseconds per sample. The CUDA speedup is lower than for the HWT5 function since the dataset for HWT6 is larger forcing some of the data to exist in CUDA memory buffers shared amongst multiple threads. For the HWT5 function, all the data was able to fit in local memory buffers each exclusively dedicated for each processor core. .......................................................... 208

Table 11-1 Example of multiple-valued (ternary) reversible function where AB are the inputs and PQ are the outputs. ................................................................. 216

Table 11-2 Matrix representation of ternary (a-c) and binary (d) quantum gates. ........ 217

Table 11-3 Generalized Ternary Gates based on Galoi Field 3 (GF3) with the gate name given in the first column and the mathematical equation for calculating the outputs based on the current input. The third column shows the output for the three possible inputs shown in the header, and finally the gate symbol is shown in the last column. ................................................................. 219

Table 11-4 Superposition gates for self-inverting ternary(a-c) and binary (d) logic..... 219

Table 11-5 Ternary Square Root of [+1] and [+2] gates. ........................................ 221
Table 11-6 Basic Data Types supported by YAML language which can be nested to as many levels as needed. The first column shows the YAML syntax of the type shown in the second column. The last column represents the parsed YAML in most modern programming languages................................................................. 224

Table 11-7 Sample benchmark functions available on MV Benchmark Repository: http://quantumlib.cecs.pdx.edu................................................................. 229

Table 12-1 Quantum Ternary Operators based on GF(3) logic........................................... 238

Table 12-2 Number of gates of swivel gates for binary, ternary and multiple valued radix-r basis of computation................................................................. 242

Table 13-1 Comparison of my binary synthesis algorithms, MMDSN and CSP, to the work of others in the field. My algorithm offers the best compromise for this multiple optimization problem, where the number of variables is maximized while the time of synthesis, quantum cost are minimized without the addition of any ancillary variables. ........................................................................................................... 248
List of Figures

Figure 3-1 The classical inverter in (a) is unidirectional where it inverts the value of the input ‘a’ while the quantum inverter in (b) is bidirectional (reversible) which transforms ‘a’ into ‘a’ and vice versa. ............................................................... 38

Figure 3-2 Controlled NOT gate where the ‘b’ qubit is inverted when the ‘a’ qubit = 1. Otherwise, the ‘b’ qubit remains unmodified. The value of the ‘b’ qubit represents the XOR relation which is equivalent to the XOR classical gate. The classical XOR gate is not reversible while the CNOT gate is reversible.............. 39

Figure 3-3 Demonstration of quantum reversibility of the CNOT gate. An application of the first CNOT gate to the ‘ab’ inputs produces a⊕b (middle table) on the ‘b’ qubit. Applying the second CNOT gate reverses the operation and restores the original value of ‘b’. The value of qubit ‘a’ remains unchanged throughout the entire process. ............................................................................................................. 39

Figure 3-4 Equivalent implementation of the classical AND and NAND logic gates (a) using the Toffoli quantum gate (b). The NAND gate is implemented by strapping the ‘c’ qubit to ‘1’ which results in (1 ⊕ ab) = ab............................................... 40

Figure 3-5 Universal Toffoli gate (C²NOT) which is classically implemented by the AND/XOR gates (a), however, the quantum Toffoli gate (b) is reversible where, similar to the CNOT gate above, application of two Toffoli gates restores the state of qubit ‘c’ to its original value................................................................. 41

Figure 3-6 (a) Fredkin or Swap gate is a composite structure which is implemented using three CNOT gates arranged according to the figure in (b). In classical logic, a swap of two bits is accomplished through the exchange of physical wires but in quantum technology a set of gates are used to exchange the information of two qubits....................................................................................................................... 41

Figure 3-7 Symbol of the Hadamard gate and its matrix representation. The Hadamard gate, which has no classical equivalence, places the qubit in a state of 

xix
superposition between the pure states of 0 and 1. The gate is also reversible where an application of two Hadamard gates back to back restores the original value of the qubit. ............................................................ 43

Figure 3-8 Bloch sphere illustrating a qubit state \( \psi(t) \) where the North and South poles are designated as the binary basis states of 0 and 1. Unlike classical logic, the qubit could assume any state between the basis states of 0 and 1 where such states are symbolized by all the points on the surface of the unit sphere. .......... 46

Figure 3-9 Ion trap apparatus showing a vacuum chamber in the middle where ions are linearly aligned at the center of the narrow cavity................................................. 51

Figure 3-11 Pictorial representation of the Paul ion-trap with ions trapped along the center z-axis. Each ion acts as a quantum register (qubit) where finely detuned laser beams (shown as two arrows pointing upwards) represent the quantum gate. The laser beams are used to affect the state of each individual ion (a single qubit gate), or the state of two neighboring ions in an entangled state......................... 52

Figure 3-14 Binary quantum system showing two energy levels ground \( g \) and excited \( e \) states where an electron is transitioned between the two states with a finely detuned laser pulse with the exact energy necessary to cause such transition...... 57

Figure 3-15 Ion's electronic excitation state coupled to its vibrational motion. Each electronic transition at frequency \( \omega_0 \) has sideband frequencies as a result of the vibrational motion \( \omega_1 \)....................................................................................................... 59

Figure 3-16 Decomposition of the composite Toffoli gate two the five primitive 2-qubit gates (CNOT and CV gates). The Toffoli gate is assumed to have a quantum cost of 5 which is the number of one and two qubit primitive gates necessary to implement it................................................................. 61

Figure 3-17 Implementation of ternary energy states of an ion-trap system (a) transitions between any two states is possible with a single pulse which requires three distinctly detuned laser systems for each qubit, while in (b) only transitions
between neighboring energy levels are possible using two distinctly detuned laser systems but requiring two pulses for transitions between the states 0 and 2. 

Figure 3-19 Cycle decomposition of 3-bit binary function shown in (a) as a table, and (b) in a Karnaugh map where the function is represented by the 3-tuple cycle (0,1,2) which is equivalent to the cycles (1,2,0) and (2,0,1). The synthesis process decomposes each of the 3-tuple cycles into their equivalent 2-tuple cycles (P1, P2, P3) as shown in (b). A cascade of 2-tuple cycles is then substituted with the corresponding gate cascade as shown in (c), (d) and (e) and the solution with the least quantum cost is selected as the best solution.

Figure 3-20 Reed Muller Reversible Logic Synthesis algorithm [60] where the algorithm starts from a set of equations rather than a truth table. A solution is found by recursively factorizing the set of equations in every possible way (at each node), until all inputs and outputs are exactly the same. The number of nodes for this method grow exponentially and require huge amount of resources (memory, computation, and time) and, as a result, is limited to small number of variables.

Figure 3-21 The universal Toffoli gate is known to have a quantum cost of 5 which is the number of primitive two-qubit gates needed to implement (a); however, the first CV gate performs remote interaction between qubits ‘a’ and ‘c’ which violates the LNNM architecture. An LNNM compliant composition of the Toffoli gate is shown in (b) at a cost of 11 two-qubit primitive gates which is further reduced to 9 two-qubit gates as shown in section 8.2 below.

Figure 4-1 The solution circuit found from MMD in Table 4-1 drawn and created from outputs to inputs. The arrow shows the flow of signal from inputs to outputs. This method is possible because each reversible gate used in this figure is its own self-inverse.

Figure 4-2 Algorithm and examples of control line blocking detection.

Figure 4-3 (a) MMDSN orders are created by first constructing a Hasse diagram where each level corresponds to the sum of digits at that level. (b) shows all possible
transitions which have a single bit difference between the two minterms in the transition. (c) Finally, an MMDSN order is constructed from the Hasse diagram where all the minterms of lower levels have to be processed before any terms in the succeeding level. In this example, the order of levels is \( \{0\}, \{4,1,2\}, \{5,3,6\}, \{7\} \) which yields the input vector of the following order of minterms \( \{0,4,1,2,5,3,6,7\} \).

Figure 4-4 A valid MMDS order \( \{0,2,1,3,4,6,5,7\} \) for MMD-like binary synthesis which is shown to be algorithmically convergent according to MMDS convergence rule. This algorithm is outside the subset that MMDSN algorithm creates since the minterm ‘3’ (on the third level) is taken before the minterm ‘4’ on the second level.

Figure 4-5 Quantum cost comparison of MMDSN to both MMD and MMDS for 50 AHP functions which are randomly generated reversible specifications. Each function has three points corresponding to one of the algorithms. The dotted line represents a linear trendline which demonstrates that on average, the MMDS algorithm always gives the lowest quantum cost while the MMD yields the highest quantum cost. The MMDSN results in quantum costs that are close to the MMDS and much better than MMD.

Figure 4-6 Comparison of the performance of MMD’s algorithm to the MMDSN algorithm described in this chapter. For MMD the single sample with natural binary order was processed. For MMDSN the average quantum cost of 5 runs over 100,000 samples for each run is reported along with the standard deviation of the samples and the percentage of deviation compared to the mean. The first improvement column reports the percentage improvement/degradation of MMDSN over MMD and the final column reports the final result of MMDSN which always considers the MMD sequence in the process.

Figure 5-2 Row 2 illustrates control line blocking where the output minterm ‘10’ is to be mapped to the input minterm ‘01’. Following the MMD method of syntheses, I

xxii
can invert the lower bit of ‘10’ while using the upper bit for control. Had I done
so, however, would alter the completed minterm in the first row ‘11’ to ‘10’
(effectively restoring the original state of the first row. Altering any completed
minterm is a violation of the MMD algorithm which indicates that the algorithm
is blocked and will never converge................................................................. 98

Figure 5-3 Construction of Binary Hasse Diagram for any 3-bit function starts with the
minterm with all zeros first in the first level (level 0). Every subsequent level
includes all minterms which have the same sum of their digits. Level 1, for
example, has the minterms \{001, 010, 100\} since the digits of each minterm add
up to ‘1’. The last minterm has all digits set to ‘1’........................................... 100

Figure 5-4 (a) MMDSN orders created by first constructing a Hasse diagram where valid
transitions from one minterm to another is shown in (b). A valid MMDSN order
is constructed from the Hasse diagram where all the minterms of lower levels are
processed before any terms in the succeeding level. ........................................ 102

Figure 5-5 The input sequence of a four variable \(B_3B_2B_1B_0\) can be partitioned in
multiple ways. Using the upper variable \(B_3\) only would create two partitions
(visualy represented as the two cubes) and the remaining bits \(B_2B_1B_0\) are
arranged according to a Hasse diagram (the levels of the Hasse diagram are
shown as diagonal lines – described in earlier sections). Using the upper two bits
\(B_3B_2\) for partitioning would create 4 partitions (shown as the four planes) where
the remaining two bits of each partition would also be arranged according to the
Hasse structure. A CSP input sequence is then constructed by arranging the
minterms according to the natural order of the bits forming the partition then for
each partition the remaining bits are arranged according to the MMDSN sequence
described in the previous chapter................................................................. 104

Figure 5-6 Normalized quantum cost for 16 benchmark functions ranging from 5 to 13
variables. In order to visualize the outcome for functions of different number of
variables, I scaled the quantum cost for all functions to the range between 0 and
100% (y-axis) and scaled the partitions size between 0 and 1 (x-axis). The normalized quantum cost for functions of the same number of variables were averaged and plotted here. Notice that the lowest quantum cost (normalized) for all functions is found for normalized partitions size between 0.4 and 0.6; hence, the best quantum costs are for partition sizes which are close to the half-way point relative to the number of variables. Notice the trendlines clearly show the minimums for each dataset are close to the middle partition.

Figure 5-7 The graph plots the normalized partition size where the best normalized quantum cost was discovered from Table 5-1, in which, the partition size and best average quantum cost are shaded in gray. Notice that for functions of 11 variables or less, the best quantum cost occurred around the midpoint of the normalized partition size (between 0.4 and 0.6) which indicates that for functions of this number of variables, a partition size in the vicinity of \( \frac{1}{2} \) the number of variables typically yields the best results.

Figure 5-8 Results for functions of 12 and 13 variables. Similar to the other functions, the midpoints of the partition size appears to have the best normalized quantum cost. The left edge represents the MMDSN algorithm which, in this case, suffers greatly in discovering the quantum cost.

Figure 6-1 Covering Set Partitions with partition size=1 using bit 3 to create two partitions of 3 bits each (upper and lower cubes), or partition size=2 using bits 3-2 to create 4 partitions of 2 bits each (four planes). The dark line separating the partitions is referred to as the partition boundary where minterms are not allowed to cross such boundary in the process of rearranging minterms to create different input sequences.

Figure 6-2 Genotype of a valid CSP input sequence showing valid and invalid mutations and cross over operations. A valid crossover can only occur at the partition boundary for each specific partition and at the band boundary as stipulated by the
Hasse diagram. Valid mutations, in this case swap, can only swap elements within the band to assure that the Hasse order is not violated.

Figure 6-3 Quantum Cost for the URF9 (9 bit) function for the four selection methods at each partition point. Notice that, in general, the Tabu search performs the best compared to the other selection methods and that the Random selection method is always the worst. Also notice that the best quantum cost for all methods happens around the midpoint of the number of variables (here at partition size = 4).

Figure 6-4 Plots of four functions with quantum cost vs. partition size for the four selection methods. Similar to the urf5 function above, the Tabu search performs the best and all selection methods have lower quantum cost around the midpoint of the size of function.

Figure 7-1 For the same sequence, swapping bit positions in the output and calculating the bit to bit hamming distance vs. the quantum cost of the resultant circuit. From the plot, it is clear that using a preprocessor to determine the arrangement of variables with the lowest hamming distance does not give a good predictor of the result.

Figure 7-2 Cumulative Positional Distance (CPD) of minterms vs. quantum cost. I hypothesized that a preprocessor which, for a specific minterm, calculates the Absolute Distance $|d|$ between its position in the input vector to its position in the output vector, and then accumulating all these distances to determine the CPD. The CPD would, supposingedly, then could be used as a predictor of which input/output sequence could yield better quantum costs. However, as seen from the plot, no clear sign emerges to fortell such a result.

Figure 7-3 Difference in Euclidean vector length between input and output vectors as a predictor of best quantum cost does not appear to show a clear pattern of which candidate solutions could yield better quantum costs. The vector length is determined by considering that the value of each minterm as a decimal distance away from zero and calculating the Euclidean distance for a the multi-
dimensional input and output vectors. The absolute difference between such values is plotted vs the quantum cost................................................................. 137

Figure 8-1 Toffoli gate decomposed to set of 5 primitive gates. Notice the remote interaction of the first CV gate which violates the LNNM architecture............ 142

Figure 8-2 Toffoli gate with qubit interaction constrained to LNNM where the qubits of the first CV gates are brought next to one another through a set of swap gates. The first 3 CNOT gates act as a swap gate which brings the a qubit next to c, and the next set of CNOT gates restore the location of the a qubit back to its original location. The quantum cost here is 11 but since the two highlighted CNOT gates would cancel each other, the quantum cost of the LNNM Toffoli gate is 9....... 142

Figure 8-3 Decomposition of the 4-bit MCT gate without consideration of LNNM. Most algorithms assume this gate to cost 13 primitive gates, which ignores the fact that there are five gates which interact remotely in violation of the LNNM architecture........................................................................................................ 144

Figure 8-4 Decomposition of the 5-bit MCT gate into primitive 2 qubit gates without consideration of the LNNM architecture................................................................. 145

Figure 8-5 LNNM equivalent of CNOT gate acting on qubits ‘a’ and ‘d’ which are two qubits apart. Rather than a quantum cost of 1, the LNNMQC = 13 (a) which is further reduced to 9 (b) once the shaded gates cancel one another. ................. 149

Figure 8-6 4-qubit swivel gate which pivots the qubits around the center point between them (abcd \(\rightarrow\)dcba). The binary swivel gate of ‘n’ variables requires \(= n^2 - 1\) CNOT gates to implement. ................................................................. 150

Figure 8-7 Method for minimal swapping of distant two-qubit gates within a decomposed MCT\(_n\) gate. In stage 0, rather than adding 4 swap gates to bring qubit ‘a’ next to ‘f’ and then 4 swap gates to bring it back, I opted to bring ‘f’ next to ‘a’, and slowly bring ‘f’ back through each stage. Qubit ‘f’ is the target line and it interacts with all qubits in each of the stages. This method reduces the number of swap gates necessary to implement this MCT\(_6\) gate. Also notice that
recursive patterns emerge in every stage which build on the structure of the previous stage................................................................. 151

Figure 9-1 MCT$_4$ in two dimensional grid layout where, now, qubits ‘a’ and ‘d’ are considered neighbors and the first CV gate is compliant with the LNNM model (a cost of 1) compared to the linear arrangement which requires 8 additional CNOT gates to bring it into compliance. Notice that the pairs ‘ac’ and ‘bd’ are still one qubit apart. ...................................................................................................................... 158

Figure 9-2 MCT$_4$ in a planer layout with qubit ‘a’ duplicated into two additional ancilla qubits in order to bring pairs ‘ac’ and ‘bd’ next one another. Notice that with this arrangement, ‘b’ and ‘c’ are now the only pair which are one qubit apart. Using this arrangement reduced the quantum cost further to 21 (rom 29) at a cost of 2 ancialla qubits. ...................................................................................................................... 159

Figure 9-3 MCT$_5$ gate with a star-shaped 2-D layout centered around the target qubit ‘e’. The LNNMQC for this arrangement (85) is actually worse than the linear arrangement of qubits (77) since now qubits ‘a’, ‘b’, ‘c’, and ‘d’ are each remote to one another which is not the case for the linear arrangement................. 160

Figure 9-4 A modified planer arrangement of the MCT$_5$ gate with the duplication of qubit ‘a’ into additional ancilla qubits. Since qubits ‘a’ and ‘e’ interact with all other qubits, our strategy is to bring these two qubits as close to the others as possible. Since qubit ‘a’ never acts as a target qubit, it is easy to just mirror its value upfront to other ancilla qubits which will interact in a ‘neighborly’ manner with the other qubits. This pattern reduced the quantum cost to 53 compared to the linear arrangement of 77................................................................. 161

Figure 9-5 Another improvement of the planer arrangement of the MCT$_5$ gate where qubit ‘b’ was transported to another location which makes it next to both qubits ‘c’ and ‘d’. This shuttling of qubit ‘b’ can only occur after the first stage once qubit ‘b’ is no longer a target. The LNNM quantum cost is further reduced to 43 from 53................................................................. 162

xxvii
Figure 10-1 Measurement of qubit in an optical quantum system can be performed by placing horizontal and vertical filters along with photon detectors on two orthogonal axis for detecting the number of photons on each point. One of the detectors is assigned the state 0 and the other state 1.

Figure 10-2 (a) non-reversible functions f and g since separately nor combined (fg) since the value 01 is repeated twice on the output and there is no way to reconstruct the input; (b) ax is reversible function since it is possible to determine the input minterm for each output minterm.

Figure 10-3 Symbols for quantum binary Identity (wire) and NOT gates.

Figure 10-4 Generalized Ternary Gates with the gate name given in the first column and the mathematical equation for calculating the outputs based on the current input. The third column shows the output for the three possible inputs shown in the header, and finally the gate symbol is shown in the last column.

Figure 10-5 Ternary energy states of an ion-trap system. In (a) transitions between any two states is possible with a single laser pulse which would require three finely detuned laser rays targeted at each ion. In (b) only transitions between neighboring energy levels are possible but only two laser rays are required.

Figure 10-6 (a) Ternary Inverter vs. (b) binary Inverter; (c) ternary controlled op (C-OP) vs. (d) Feynman CNOT; and (e) ternary (C^2-OP) vs. (f) Toffoli gate (C^2-NOT).

Figure 10-7 Ternary Synthesis Example transforming output vector AB (column 2) into input vector ab (columns 1 and 8). The [[+1]] in the header represents the ternary gate inserted at that point which is also shown pictorially in the circuit below the column. The shaded values indicate the values impacted by the gate; the [[+1 1]] in the header of column 5 indicate that the lower qutrit is used as a control of value ‘1’ while the upper bit has the [[+1]] gate as shown in the pictorial below the column.
Figure 10-8 When the algorithm reaches the shaded minterm which needs to be transformed from 20 to 02. If the upper trit is used as control to avoid changing the second minterm, any gate applied to change the lower trit from 0 to 2 will also change the first minterm. The same thing will happen if I use the lower trit (0) as control where in this case, the second minterm will be altered. Either way, this is a violation of the algorithm and the algorithm is blocked at this point.

Figure 10-9 Structure of the ternary Hasse diagram for a 2-variable function. Each level (band) contains the set of minterms which has the same mathematical sum of digits. The sum of digits represents the band number shown on the right hand side.

Figure 10-10 Construction of the Ternary Hasse Diagram for a 2-variable ternary function. (a) Starting from the bottom with 00, I add two new minterms by adding 1 to each digit resulting in (01, 10). (b) for each new minterm (e.g. 01), I repeat the process where I add 1 to each digit resulting in (02, 11). (c) For each digit, the process is repeated until the upper value ‘2’ is reached.

Figure 10-11 Three possible Ternary Value Precedence diagrams where the ternary values can be arranged with different precedence orders. (a) indicates a natural mathematical order of precedence. (b) indicates that the values 0 and 1 are equivalent in precedence and are lower than 2. (c) indicates that 1 and 2 are at the same level and are both above 0. The precedence order plays a role in the structure of the Hasse diagram and the consequent construction of ternary input sequences.

Figure 10-12 Precedence order of low and high bits in a ternary Hasse diagram. This Hasse structure follows the natural precedence order (0→1→2) which is also evident for each specific digit as it moves from one level to a higher level. The lower digit exhibits the same pattern in the NW direction, while the upper bit exhibits the pattern in the NE direction.
Figure 10-13 Example for case 4c. In an attempt to change 111 to 120, the two lower trits have to be modified one at a time. Were I to attempt changing the lower trit from 1 to 0 first, I would had to use the ‘11’ control pattern which was encountered before, and would have modified the first minterm (110). But by always doing ‘low to high’ transitions first (pass 1), in this case the middle trit (from 1 to 2), the output minterm will change properly to allow for new control values to emerge which would not affect previously completed minterms. In pass 2, I can safely modify the lower trit from 1 to 0 since the ‘new controlling value (12)’ will be guaranteed not to modify any previously completed minterms.....

Figure 10-14 (a) One of the triangular precedence orders (\{0\} \rightarrow \{1,2\}) can be represented by the binary equivalent class (\{0\} \rightarrow \{1\}) since the \{1,2\} are of the same precedence level. (b) The ternary Hasse diagram based on this triangular precedence order can be reduced to the binary equivalent Hasse diagram shown to the right...

Figure 10-15 Number of permutations for all possible input vectors (all possible solutions) vs. Hasse based sequences (valid solutions constructed by this algorithm).

Figure 10-16 Genotype of a valid input sequence showing valid and invalid mutations and cross over operations. A valid crossover can only occur at the band boundary as stipulated by the Hasse diagram. Valid mutations, in this case swap, can only swap elements within the band to assure that the Hasse order is not violated....

Figure 10-17 Time required to synthesize the family of Hidden Weighted Trit functions.

Figure 11-1 Example of multiple-valued reversible function represented as a circuit. In this illustration, the A=P qubit is used as a controlling value for the gate [[02]]. When A is set, the [[02]] gate is active, otherwise, it is a passthrough.

Figure 11-2 Operation of Ternary Inverters (a,b,c) shows three self inverters, while (d) show the two complementary inverters (+1 and +2). (e) shows substitution of +1
to (12, 02) and +2 to (02,12) where the middle 02 gates cancel one another, and then the 12 gates cancel one another yielding identity; hence, +1 and +2 are complementary invertors.

Figure 11-3 (a) Example of binary reversible quantum circuit and (b) its specification in the PLA format.

Figure 11-4 Pseudo-code of an XQS specification describing a Full Adder with 2 inputs and 1 output. The implementation section shows a solution for this adder.

Figure 11-5 XQS specification of a binary/ternary hybrid function where the radix is defined for each qubit (322 is ternary for the upper qubit, and binary for the other two). “Don’t care” is shown as ‘-‘ in the output variable definition.

Figure 11-6 XQS specification of Moore State Machine.

Figure 11-8 Index page of quantum function generator.

Figure 11-9 details of function along with downloadable specification.

Figure 12-1 (a) LNNM rendition of distant CNOT gate (on far left) which is brought into LNNM compliance with the aid of swap gates (block of 3 CNOTs), (b) minimized version of (a) where the shaded CNOT gates cancel one another.

Figure 12-2 Pictorial of swivel gate with (a) odd number of qubits, (b) even number of qubits.

Figure 12-3 Four bit binary swivel gate composed of a set of CNOT gates. The upper right triangle shows the well-known swap gate composed of three CNOT gates.

Figure 12-4: 4-qubit ternary swivel gate which has similar structure as its binary counterpart with the addition of the 2x gate and the double return gates at the far right.

Figure 12-7 Four variable Multiple Valued (radix = r) Swivel Gate which has the same structure as the ternary and binary counterparts with the use of bias gates (r-1)x and the (r-1) cascade of controlled gates at the far right.
Glossary of Terms

Ancilla Qubits  Qubits which are introduced for the sake of holding intermediate results of calculations and are not part of the original specification. They are also referred to as garbage bits.

Ancillary Ratio  The ratio of synthesized circuit width to the number of input variables which reflects the magnitude of increase of circuit width due to the addition of ancilla qubits. An ancillary ratio of one(1) indicates no ancilla bits.

Application Specific Quantum Circuit (ASQC)  Also known as a quantum oracle refers to the automated synthesis of quantum circuit, using the set of primitive quantum gates, for any given specification.

Band  Used in the construction of input vector sequences from a Hasse diagram where band \( n \) holds all minterms whose individual digits add up to the decimal number \( n \). For example, the minterms (011, 110, 101) belong to band 2.

Bijective  A bijection, or one-to-one correspondence, is a function giving an exact pairing of the elements of two sets. Every element of one set is paired with exactly one element of the other set, and every element of the other set is paired with exactly one element of the first set. (There are no unpaired elements.)

Bloch Sphere  In quantum mechanics, the Bloch sphere is a geometrical representation of the pure state space of a 2-level quantum system represented as points on the surface of the unit sphere in three dimensions. Alternatively, it is the pure state space of a 1 qubit
quantum register.

**C\(^n\)NOT**  
Controlled NOT, or Multi Control Toffoli (MCT), gate with \(n\) control qubits and single target qubit.

**Completely Mapped Pair**  
For an MMD style algorithm, an input/output pair which have already been mapped and should not be altered by any subsequent synthesis step.

**Control line**  
A qubit whose value represents a conditional switch to enable or disable the operation of a quantum gate of another qubit. In binary circuits, typically the value of \(\text{one}(1)\) allows the controlled (target) gate to operate while a value of \(\text{zero}(0)\) disables the target gate.

**Control Line Blocking**  
A condition that occurs during the synthesis process which prevents MMD style algorithms from converging because later synthesis steps cannot proceed without altering completely mapped pairs which is a violation of the convergence criteria.

**Convergence**  
The ability of an algorithm to always compute a solution without suffering from the control line blocking syndrome.

**Crossover**  
In genetic algorithms, crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next. In other words, crossover is a process of taking more than one parent solutions and producing a child solution from them.

**CSP**  
Acronym for Covered Set Partition which, for an \(n\)-variable function, is a method of subdividing the input vector into partitions of size \(k\)-bits, \(k < n\) where partitions are processed in sequential order according to the natural order of the \(k\)-bits.

**CV**  
Controlled V gate, which enables the V gate when the control qubit is \(\text{one}(1)\) and disables it when the control qubit is \(\text{zero}(0)\).

**Cycle**  
A process where a permutative specification \(S \rightarrow S\) is redefined
Decomposition as the product of subsets each consisting of a disjoint cycle.

FPRM Fixed Polarity Reed Muller decomposition

Fredkin Gate Invented by Edward Fredkin, is the controlled swap gate of three qubits which swaps the last two bits when the control bit is one(1). It is a universal gate which means that it can be used to construct any quantum binary logic circuit.

Genetic Algorithm Is heuristic search algorithm used whenever an exhaustive search is impractical or impossible. A genetic algorithm mimics the biological process of natural evolution employing procedures of inheritance, mutation and random selection.

Genotype In a genetic algorithm, the genotype represents the structure of a chromosome (solution).

Grover’s database search algorithm Invented by Lov Grover in 1996, is a quantum algorithm for searching an unsorted database with N entries in \( O(N^{1/2}) \) time and using \( O(\log N) \) storage space.

Hasse Diagram By Helmut Hasse (1898-1979), is a type of mathematical diagram used to represent a finite partially ordered set in a form of directed acyclic graph representing the relation \((S, \leq)\) where vertical position represents the relation \( \leq \) between elements of \( S \) such that \( a \leq b \) when \( a \) is below \( b \).

Heuristic Refers to experience based techniques for problem solving, learning and discovery where an exhaustive search is impractical.

Hidden Weight Bit/Trit functions Reversible binary/Ternary functions where the output minterm is generated by circularly shifting the input minterm by the number of its non-zero bits/Trits.

Ion Trap Special apparatus which confines charged particles into a definite region in space, typically in a linear array of particles uniformly lined like a tightly held string of beads. An ion-trap
provides the environment for quantum computation where, on one hand, qubits must be isolated from the surrounding environment in order to minimize decoherence (loss of internal state), while on the other hand, qubits must be individually addressable by external laser pulses to transform their internal state according to the quantum gate under operation.

**Kronecker Product**

Used to assemble the composite operator spanning multiple qubits when control lines are not present between the qubits.

\[
\begin{bmatrix}
a_{11} & a_{12} \\ a_{21} & a_{22}
\end{bmatrix} \otimes \begin{bmatrix}
b_{11} & b_{12} \\ b_{21} & b_{22}
\end{bmatrix} = \begin{bmatrix}
a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12}
\end{bmatrix}
\]

For example:

\[
I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad I \otimes X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}
\]

**LNNM**

Linear Nearest Neighbor Model which constrains interaction amongst qubit to their nearest neighbors which would require swapping qubits to comply with such constraint.

**MCT\(_n\)**

Multi-Control Toffoli Gate of \(n\)-variables.

**MMD algorithm**

Refers to Miller, Maslov and Dueck’s algorithm for synthesis of binary quantum circuits

**MMDS**

Stedman’s improvement over MMD where he discovered the formula for detecting convergent sequences.

**MMDSN**

Nouraddin’s (and Hawash) limited the generation of input sequences to those constructed using the Hasse diagram.

**Mutation**

Genetic algorithm operator used to imbue diversity into the population by altering phenotype of a solution.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMR</td>
<td>Nuclear Magnetic Resonance is used to manipulate the state of qubits in order to implement quantum primitive gates.</td>
</tr>
<tr>
<td>Qubit</td>
<td>Binary Quantum digit</td>
</tr>
<tr>
<td>Qudit</td>
<td>$d$-valued Quantum digit</td>
</tr>
<tr>
<td>Qutrit</td>
<td>Ternary Quantum digit</td>
</tr>
<tr>
<td>Recombination</td>
<td>Genetic algorithm operator where two genotypes (solutions) are used to <em>breed</em> an offspring through <em>n-point</em> crossover.</td>
</tr>
<tr>
<td>Reversible Specification</td>
<td>Represents a function in the form of a truth table or equation where every input pattern maps to a unique output pattern.</td>
</tr>
<tr>
<td>Tabu Search</td>
<td>Is a met heuristic local search algorithm which iteratively moves from one potential solution $x$ to an improved solution $x'$ in the neighborhood of $x$ until some stopping criterion is satisfied.</td>
</tr>
<tr>
<td>Target line (qudit)</td>
<td>The qubit on which a quantum gate applies its operation or produces its results.</td>
</tr>
<tr>
<td>Ternary Precedence Order</td>
<td>Precedence hierarchy amongst the three members of the ternary domain ${0, 1, 2}$ are a member of an ordered graph where $\forall a, b \in {0, 1, 2} \mid a \leq b$.</td>
</tr>
<tr>
<td>Toffoli Gate</td>
<td>Invented by Tommaso Toffoli, a universal reversible logic gate of two control qubits $(a, b)$ and a single target qubit $(c)$ where $c = ab \oplus c$.</td>
</tr>
</tbody>
</table>
Introduction

This dissertation is devoted to quantum computing and information processing from an engineering perspective with specific focus on automated and efficient synthesis of quantum logic circuits of relatively large number of variables. My research topic assumes that there exists a set of universal reversible quantum gates capable of operating as binary and ternary logic gates, which have been experimentally demonstrated to operate in accordance to the principles of quantum mechanics [15, 18]. The physical design of such quantum gates and the fabrication of quantum circuits, however, are outside the scope of this dissertation.

The entire body of my research centered on developing and analyzing algorithms to synthesize quantum logic specifications of large number of variables. My work has progressed through three successive and complimentary stages:

Stage 1: I first started by focusing on the problem of synthesizing binary quantum specifications and attempted to automatically construct quantum circuits while minimizing the number of primitive quantum gates used. In this stage, I adhered to the prevalent theoretical assumption that any two qubits can interact remotely regardless of the number of other qubits between them. The artifacts of this effort are a set of algorithms capable of synthesizing any arbitrary reversible binary specification which attempt to minimize the objective function measuring quantum cost by employing classical techniques in data structures, random, genetic algorithm
and Tabu search.

**Stage 2:** I later realized, however, that certain technological constraints exist in a physical implementation of quantum circuit. For example, Ion Trap and Nuclear Magnetic Resonance (NMR) technologies, limit interaction amongst qubits to their nearest neighbor, and as a result, distant the information held within two qubits has to be relocated to two neighboring qubits in order to facilitate the interaction (i.e., computation) and the result is then relocated back to the source qubit. I realized that, clearly, such restriction would greatly impacts the yardstick employed in measuring circuit performance, and therefore, I have shifted our focus to the Linear Nearest Neighbor Model (LNNM). Since then, I continue to calculate and promote a set of standard benchmarks based on the LNNM architecture. As part of my work, I have recalculated the cost of the most commonly used gates, the Multi-Control Toffoli gate (MCT) and suggested an efficient LNNM Quantum Cost for the MCT gate family. I then recalculated the performance of my latest binary synthesis algorithms to the LNNM metric.

**Stage 3:** Given the higher information carrying capacity of a quantum bit, I also examined synthesis of quantum ternary specifications and applied methods similar to our binary work from the first stage to the ternary domain. As I was trying to synthesize ternary functions larger than what is available in existing literature, I defined and introduced a set of new benchmark functions into the literature. I initially introduced the Hidden Weighted Trit (HWT) [19] functions which extend
the well-known Hidden Weighted Bit (hwb) binary benchmark functions [20]. I later introduced seven additional Multiple-Valued (MV) functions and built an online multiple-valued function generator capable of creating one of the defined functions for any radix and any number of variables. In the process of introducing these functions, I also realized that the existing file formats used to define quantum specifications, the PLA and REAL formats, are inadequate for defining multiple-valued and hybrid quantum functions. As a result, I invented and introduced a new file specification, the *Extensible Quantum Specification (XQS)* which is specifically designed for multiple-valued and hybrid quantum specifications.

**Stage 4 (current work):** As the number of variables increases, by definition of the specification or as a result of the introduction of ancillary qubits, the number of control lines required to implement the circuit grow. For a Multi-Control Toffoli (MCT) gate, every additional control qubit effectively doubles the number of primitive gates necessary to simulate the MCT gate, i.e., doubles the quantum cost. In this stage of my research, I am exploring post-processing methods in order to group gates of similar control pattern and partition their control functionality into subgroups through the introduction of a minimal number of ancillary qubits.
Summary of Research Objectives

I. Binary Logic Synthesis

Problem Statement: Automated synthesis of binary specifications into quantum circuits using the Not, C-Not and Toffoli (NCT) library.

Description: A combinatorial problem by nature, quantum logic synthesis algorithms have been bound to low number of variables (3-4) due to the enormous computing resources required to explore all \(2^n\) possible solutions for the set of Not Controlled Not and Toffoli gates (NCT library). The solution space is much larger, in fact, if we chose to include other sets of gates for the synthesis process.

Objective: Devise methods for exploring the enormous search space of large number of binary variables (9+) and discover valid solutions with low quantum cost.

In order to compare the results of different synthesis algorithm, I use the de facto benchmark metrics found in the literature where the quantum cost of the circuit is measured by the number of one and two-qubit gates needed to implement the circuit.

Completed Work: I have completed work in this area with publication of 4 papers:

1. MMDSN algorithm [21, 22].
2. CSP-R algorithm with Random selection [23].
3. CSP-EV with selection using evolutionary algorithms (Genetic algorithm and Tabu search) [24].
II. Binary Logic Synthesis with LNNM

**Problem Statement:** Automated synthesis of binary quantum circuits with consideration to technological constraint of *nearest neighbor* interaction between qubits.

**Description:** The majority of logic synthesis algorithms calculate the complexity of resultant circuit using an *idealistic* cost of a quantum gate without regard to the limitations of such gates from a physics point of view.

For example, Ion trap and NMR (both leading technologies for quantum gates) constrains interaction between qubits to their nearest neighbor; yet, the majority of existing algorithms disregard this fact when calculating the cost.

It is not *realistic* to compare results amongst different methods of synthesis where some algorithms consider controlled gates with large distance between the control and target qubits to be equivalent in cost to the primitive gates where the target and control qubits are next to one another.

**Objective:** (1) Analyze and (2) compare the impact of enforcing nearest neighbor model upon the quantum cost of a circuit and (3) advocate the LNNM architecture as the foundation for establishing new metrics for calculating the cost of binary quantum circuits.

**Completed Work:** I calculated the LNNM quantum cost of the well-known Multiple Control Toffoli (MCT) gates and derived a set of equations that determine a
minimized quantum cost based on the LNNM architecture. I also proposed a two-element tuple, \((\text{lnnqc}, \text{ancillary ratio})\), definition of quantum cost which considers both the width and depth of a circuit. I believe that this definition will allow for a level comparison amongst different methods of quantum logic synthesis. [25]

Finally, I re-implemented my latest binary algorithm (CSP) to calculate the LNNQC measure performance of circuits. I am also currently exploring post processing algorithms for sharing control lines among consecutive gates in an effort to drastically reduce the LNNQC
III. Multiple Valued (Ternary) Logic Synthesis

**Problem Statement:** Automated and efficient synthesis of *ternary* specifications into quantum circuits using the Permutative Ternary Gate (PTG) library based on Galois Field 3 logic.

**Description:** Quantum mechanical systems are inherently suitable for multi-valued logic of much higher information density per qubit. However, the majority of research today has mainly focused on binary systems and the limited research in multi-valued logic synthesis has been only demonstrated for functions of two variables.

**Objective:** Explore logic synthesis algorithms for multiple-valued systems with large number of variables relative the radix of our logic considered. For ternary systems, the search space expands at an exponential rate of \(3^n\)!; hence, a 5 ternary variable can hold the same information capacity as a 9 binary variable.

**Completed Work:** HP\(_3\) algorithm [26, 27, 28, 29]:

1. Synthesis algorithm of up to 9 ternary variables.
2. Introduced the 8 new multiple-valued functions into the literature.
3. Published and online multiple-valued function generator and a repository of MV functions at http://quantumlib.cecs.pdx.edu.
4. Introduced the *Extended Quantum Specification (XQS)* file format, specifically designed for multiple-valued and hybrid quantum functions.
IV. Collaborate

**Motivation:** The following is a personal objective that I do not intend to be a requirement for my PhD. Unlike my work in the industry which dictates secrecy, I intend that all my academic work will be available to anyone interested in using it for the advancement of this field. I firmly believe in the spirit of collaboration and the principles of open source.

**Objective:** I realized that there is a unique opportunity for establishing an online forum for discussion and hosting an open source repository for source code, publications and references. I also believe that it would be beneficial for me personally, and for others, to build a compendium of existing synthesis methods in the field of quantum logic synthesis.

Completed Work:

1. Already established open source repository containing all source code for all variants of the binary and ternary algorithms: [http://github.com/quhub](http://github.com/quhub)
2. I relocated my quantumlib.org site to the Portland State University’s subdomain [http://quantumlib.cecs.pdx.edu](http://quantumlib.cecs.pdx.edu).
3. **QuGenerator:** I implemented a new function generator which is capable of generating specifications for quantum functions with any radix and number of variables: [http://quantumlib.cecs.pdx.edu/specifications](http://quantumlib.cecs.pdx.edu/specifications).
4. **QuCircuit:** I created an online tool for converting a quantum circuit definition to a circuit diagram: [http://quantumlib.cecs.pdx.edu/circuits](http://quantumlib.cecs.pdx.edu/circuits).

5. The majority of my work has been in collaboration with professors and students at Portland State University [21, 22, 25, 23, 26], in Palestine [24, 23] and in Japan [28, 27].
My Contributions

<table>
<thead>
<tr>
<th>MMDSN/MP Algorithms</th>
<th>Collaborators: N. Alhagi</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reference:</strong> Chapter 4 <strong>MMDSN and MP algorithms</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Motivation:</strong> Alhagi has designed this algorithm based on Hasse diagram to discover and synthesize quantum binary circuits. He attempted to implement the algorithm in C but lacked the proficiency in programming so he resorted to demonstrate its operation manually including a manual random number generator and hand calculation. Although he hypothesized correctly that the algorithm will converge, he could not provide experimental results to demonstrate. The algorithm intended to extend the MMD algorithm to address other possible sequences, which MMDS did, but still do it in a reasonable amount of time, and perform the synthesis for larger specifications. MMDS was computationally limited to 5 bits since it attempted to synthesize every possible permutation of sequences. The MMDSN/MP algorithm proved successful in providing better results than MMD, tremendously faster results than MMDS and for functions up to 32 bits.</td>
<td></td>
</tr>
<tr>
<td><strong>My Contribution:</strong> Dr. Perkowski recommended that I collaborate with Alhagi to help him with this algorithm and to learn about quantum logic synthesis. I learned the details of the algorithm and implemented the entire algorithm in C++ and demonstrated that the algorithm will converge for any arbitrary reversible binary function. I also calculated benchmarks demonstrating that this algorithm provides better quantum cost than MMD and takes a fraction of the time that MMDS does. I then edited (with Dr. Perkowski) finalized and submitted the paper for the ISMVL 2010 conference and for publication in the <em>FACTA UNIVERSITATIS</em> journal.</td>
<td></td>
</tr>
</tbody>
</table>
**CSP Algorithm**  **Collaborators: Amjad Hawash**

**Reference:** Chapter 5 *Covered Set Partitions*

**Motivation:** Dr. Perkowski and I met with Drs. Caughman and Bleiler to explore other possible ways of exploring the search space. I realized that our MMDSN algorithm is skipping certain search spaces and wanted to attempt additional methods of constructing valid sequences that explore other segments of the search landscape. I used one of the methods from that meeting to design and implement the CSP algorithm.

My implementation of the CSP algorithm with the ability to adjust the partition size has proved useful in discovering solutions with superior quantum cost. Depending on the number of partitions attempted, this algorithm takes a bit longer, and performs better than MMDSN and in much faster time than MMDS can. I also was able to synthesize functions up to 32 bits but it takes a long time to complete.

I also saw an opportunity to involve some of the professors from the University of Najah, in my hometown of Nablus, in my research for the sole purpose of providing them with an opportunity to learn, research and contribute. My cousin professor Amjad Hawash showed interest and I setup a weekly meeting on Skype where I introduced him to quantum logic synthesis and reviewed my CSP code in details. He worked with me in collecting, analyzing the data and reviewing the paper.

**My Contribution:** I designed the CSP algorithm and transformed the MMDSN program to support the CSP method of generating sequences. I added support for a variable partition size in order to test the impact of the size of partition on the result. I also added support for multi-threading on the CPU, and purchased a new PC with Intel i7 with 8 cores, in order to process a larger number of sequences. I also modified the MMDSN algorithm for multi-threading and compared results from both algorithms.

I recently updated the logic for calculating the quantum cost and automated the process of running the algorithm for all possible partitions of a specifications based on the number of variables. I ran this modified algorithm on a larger number of functions from the RevLib site.
I wrote a paper about CSP and presented it at ITNG 2011 with the initial results and will be revising the paper with the new results for submission later this year.

**CSP-EV Algorithm** Collaborators: Amjad Hawash, Baker Abdalhaq

**Reference:** Chapter 6 Covered Set Partition with Evolutionary Algorithms

**Motivation:** Up to this point, I had been using random generators in selecting CSP sequences, so I decided to explore whether the selection methods of CSP sequences has an impact on discovering sequences with better quantum cost. I decided to implement a genetic algorithm (GA) using Roulette Wheel to decide on the next set of CSP sequences to synthesize.

I also invited another professor, B. Abdalhaq, from the University of Najah in my research who suggested exploring Tabu Search as well. In my weekly meeting with them, I discussed various ideas and challenges in their effort and I gave them advice and feedback on how to proceed. Abdalhaq added the Tabu search to my source code and Amjad collected the data from my implementation of the GA. Amjad also experimented with various derivations of the GA and both of them helped in reviewing the paper.

**My Contribution:** I updated my CSP code and implemented the genetic algorithm to select CSP sequences. I designed the mutation and cross-over operators to preserve the internal structure of the CSP. I chose to use the Roulette Wheel method of selection in order to give the offspring of better fit solutions a higher priority for selection. I also implemented both single point and two point crossover and experimented with uniform crossover.

I then wrote the paper for the ISIICT conference in Amman, Jordan which was presented by B. Abdalhaq.
**Reference:** Chapter 9 *Linear Nearest Neighbor Model for Binary*

**Motivation:** Learning about the nearest neighbor constraints that the underlying technologies impose upon the design of a circuit, I realized that the *de facto* quantum cost metric used in the literature is not realistic. I decided to explore the Linear Nearest Neighbor Model as a realistic alternative. I also realized that attempts by other researchers to address the nearest neighbor constraint was incomplete since they addressed the gaps between gates but did not address the gaps within a gate (specifically MCT gates). None provided a comprehensive set of equations to calculate the LNNM quantum cost.

**My Contribution:** I first derived a set of equations which calculate the number of swap gates necessary to bring any binary quantum into LNNM compliance. Inserting swap gates was the common method of enforcing LNNM in the literature and everyone ignored the internal structure of the MCT gates. I then derived equations to calculate the number of swap gates necessary to bring any of the MCT gates into compliance with LNNM. Finally, I implemented a program to calculate the LNNM quantum cost for all existing benchmark functions on RevLib [30] and Maslov’s [20] websites.

I wrote and submitted a paper with the results of comparing the idealistic vs. the LNNM quantum costs and proposed that the LNNQC should be used for comparison because it is compliant with technology and that the LNNQC brings evenness to comparison amongst different methods of synthesis.

I then derived a set of equations to recalculate the LNNM quantum cost for the family of MCT gates by reducing the number of CNOT gates needed for enforcing LNNM architecture. I modified my program to calculate the new LNNM quantum cost and applied them to the functions from RevLib and Maslov’s websites.

I also investigated arranging the qubits on a 2-D grid and calculate the quantum cost of some of the MCT gates when arranged in a 2 dimensional plan (rather than a straight line).
Finally, I am in the process of updating my previous paper with the new results and plan on submitting it to future conferences and journals.

**Ternary Synthesis Algorithm**

**Reference:** Chapter 10 *Multi-Dimensional LNNM Architecture*

**Motivation:** The majority of literature in quantum synthesis focused on the binary domain, and few research papers addressed the ternary and multiple-valued domain. Even then, such papers demonstrated manual calculations on functions of 2 to 3 variables.

I saw an opportunity to address ternary functions with large number of variables and decided to apply the knowledge of constructing Hasse diagrams in the ternary domain.

**My Contribution:** I designed three models for constructing the Hasse diagrams in the ternary domain and implemented a synthesis application using C++ to synthesize any arbitrary reversible ternary circuit. Since the literature lacked a set of ternary benchmarks for large number of variables, I created and introduced a new set of functions *HWT* for any number of variables. I was able to run my implementation on functions of up to 9 ternary variables.

I wrote a paper with the results and effectively set the yardstick for future algorithms. In the paper I highlighted the importance of precedence order amongst the ternary literals (0,1,2) and provided a proof of convergence for the three prominent orders. I also described the process of constructing Hasse structures for each of the orders.

I wrote and published a paper in ISMVL 12.

I then purchased a CUDA graphics card with 1024 cores in an effort to speed up this time intensive algorithm and compared the speed up to running the same algorithm on the host CPU. I wrote and published all my results as a chapter in the book: *GPU Computing with Applications in Digital Logic*. 
**MV Functions and XQS file format**  
**Collaborator:** Martin Lukac

**Reference:** Chapter 11 *MV Benchmarks and Extensible Quantum Specification* (XQS)

**Motivation:** During my work in the ternary domain I introduced the HWT family of functions in order to verify my results because ternary functions with large number of variables were not available. I worked with Dr. Martin Lukac from Japan to create a set of general-purpose multiple-valued functions and introduce them into the literature. In the process of defining these functions, I realized that the current file format used to define quantum binary functions becomes limiting when I move to higher levels of computation, and as a result, I also introduced the XQS file format.

**My Contribution:** Martin and I explored some of the existing binary benchmark functions from RevLib and decided on seven of them. I wrote a set of generators in Ruby language which is capable of generating any of the seven functions based on the radix of computation and number of variables. I then integrated these generators into the *Portland Quantum Logic Group* website and provided it online for anyone to use.

As for the XQS file format, I suggested the architecture of YAML format as the structure for the XQS format, and suggested the specific format which will accommodate multiple-valued and hybrid quantum specifications.

I jointly wrote a paper of these results which I will be presenting in ISMVL 13.
### Multiple Valued Swivel Function

**Reference:** Chapter 12 *Generalized Multiple Valued Swivel Gate*

**Motivation:** In my work on LNNM architecture, I recognized a pattern of gates which reverse the order of qubits around a central access. I then derived a generalized pattern which is applicable to any multiple-valued domain and decided to share these results.

**My Contribution:** I designed the swivel gate pattern for multiple valued specifications (binary, ternary … etc.) and provided a set of equations to calculate the number of primitive gates necessary to implement. The swivel gate can be used as a functional block in a circuit. I have a paper accepted for the Reed-Muller workshop 2013.
PART I

INTRODUCTION

TO

QUANTUM COMPUTING
Chapter 1 Introduction

Although it appears that the probability of making quantum computing devices available for the masses is miniscule, the eminent end of Moore’s law continue to kindle the fire for exploring and laying a foundation for such a futuristic computing device. Today where the number of quantum bits, qubits, employed for computing is extremely limited, automated logic synthesis tools (CAD) which aim at reducing the number of such resources is of extreme importance. When I started my research into quantum logic synthesis, I found that the vast body of research in quantum logic synthesis used an abstract, mathematical model of quantum gates, and, by large, ignored inherent physical constraints imposed by the underlying technology used to construct quantum circuits. I also implemented my first set of algorithms using the abstract model. However, as I realized that there are physical constraints imposed by the technology used to construct quantum gates and circuits and, I shifted the focus of my research to take into account the physical technological constraints and redefined the quantum cost metrics to take into account the physical constraint imposed by the underlying technologies. The abstract model of a quantum gate is useful at the conceptual level of quantum CAD, however, from a practical engineering perspective, considering the constraint of Linear Nearest Neighbor Model (LNNM) dictated by technology yields better estimates to the width and depth of a quantum circuit. Since a quantum gate requires a finite amount of time to
*execute*, the width of the circuit determines the length of time to execute the specification implemented by said circuit. The depth of a quantum circuit dictates the number of additional gates necessary to bring an arbitrary quantum circuit into compliance with the LNNM architecture. Depending on the specification at hand, every additional qubit can result in doubling the number of gates needed to implement the function.

A drastic shift of thinking and comprehension had to manifest as I started working in this field. Quantum mechanics seemed as the strangest field I have ever encountered judged by the existing foundations of engineering acquired in my previous academic pursuit and professional career. Basic concepts of quantum computing made little sense as they violated well-known, and solid engineering principles. So the question continues to be asked, *is quantum computing feasible and do the postulates of quantum mechanics hold water?* I can now answer both questions with a resounding affirmation and can point to the body of experimental results demonstrating quantum computation in the lab setting. For instance, Ion-trap is one of the leading quantum computing technologies used today to build quantum circuits. Quantum registers with up to 14 qubits, trapped in a straight line, have demonstrated the ability to perform computations with high repeatability and fidelity [15]. In these experiments, the set of single and double primitive gates have been demonstrated. In classical logic, a NAND gate is considered universal such that any logic circuit can be constructed purely from a set of NAND gates. In quantum computing, the Toffoli gate is one of many equivalent *universal quantum*
gates which have been demonstrated in the lab and can also be used to construct any quantum logic specification [31, 32, 33].

My first exposure to quantum logic synthesis started by aiding N. Alhagi in his implementation of the Multi-Pass (MP) and MMDSN algorithms, described shortly, which allowed for synthesis of any arbitrary quantum reversible circuit [21, 22]. As part of his PhD endeavors, Alhagi was exploring quantum logic synthesis and had developed a heuristic algorithm to find minimal solutions for arbitrary reversible quantum specifications. He had developed the MMDSN algorithm as an extension to one of the well-known quantum logic synthesis algorithms, MMD [34], and on another algorithmic extension introduced by Stedman, MMDS [11]. MMD operates on all possible minterms of a quantum specification by organizing them in a truth table format in ascending natural binary order of the input vector. The algorithm then implements the input/output mapping of each minterm pair by inserting the necessary quantum gates to the circuit, and using control lines to avoid modifying previously synthesized minterm pairs.

Stedman correctly hypothesized that attempting to reorder the order of treating the input/output minterms would surely produce different, and ultimately better, results. For a specification of $n$ variables, Stedman calculated that the solution space consist of $2^n!$ unique ways of ordering the input vector. He was able to implement his MMDS algorithm to visit each of the input vector ordering, solution, and calculate its quantum cost. Although Stedman was able to demonstrate that, truly, better solutions exist, his algorithm could only handle a small number of variables, as the number of solutions grew
exponentially with each additional variable, and hence, his algorithm blocked at 5 variables.

Building on the work of Stedman, Alhagi realized that the majority of solutions that MMDS explores do not algorithmically converge, and thus must be discarded. He developed an algorithm to construct different orderings of the input vector which are guaranteed to converge and demonstrated the ability of the algorithm to discover better solutions than MMD but not as optimal as MMDS. Yet, his algorithm would easily work for larger number of variables and discover solutions better than MMD. Lacking proficiency in computer programming, Alhagi had trouble implementing the algorithm in order to provide simulation results of the algorithm and compare with existing algorithms. He had done the majority of his calculations on small number of variables and had done the simulation by hand. I quickly learned the details of the algorithm, implemented it and collected results for functions of up to 16 variables. I then implemented the MultiPass (MP) algorithm that builds on the MMDSN work by operating on binary equations rather than truth table. Using binary equations is superior to the truth table as the number of variables increase, as the memory requirements to store truth tables doubles with each additional qubit. With the MP algorithm, I was able to synthesize functions with up to 32 variables – both unfeasible for MMD and MMDSN.

I built on my work with Alhagi and developed the Covered Set Partition (CSP) algorithm, which in addition to exploring different orders, splits the function into subgroups or partitions and then reorders the terms within a partition. Through simulation, I
was able to demonstrate that creating partitions of the input vector always results in discovering better solutions than MMDSN and that the partition width also affects the outcome. For this algorithm, I also experimented with different methods of selecting input vector orderings in an attempt to study the impact of selection on the result. I implemented random selection, selection through a genetic algorithms and Tabu Search. Although both evolutionary methods produced slightly better results than random selection, the choice and overhead of the evolutionary algorithms did not prove highly superior to random selection.

But, how does it really work? I initially had trouble internalizing how a quantum gate or circuit works and could not conceptualize what a quantum gate looks like nor how it operates. I took an introductory course in quantum mechanics in the physics department in hopes of understanding the concepts of superposition, entanglement and the technicalities of quantum computation. From that course, I was able to understand the framework of quantum mechanics that helped me understand the internal state of a qubit and how a qubit could exist in a state of superposition (anywhere between the values 0 and 1). I was also able to reason out the ability of a qubit to assume multiple basis states for the implementation of multiple values systems and I was able to better comprehend the concept of entanglement. It was clear to me at the time, that, in quantum computation, a qubit represents the storage unit to hold the information necessary for computation. It was still unclear, however, what a quantum gate is and how it operates until I started learning about NMR and ion-trap technologies. Ion-trap, specifically,
made it clear that a the state of qubits, suspended in a vacuum chamber which isolates them from external environments, could be manipulated with a highly focused laser beam with a specific wavelength for a specific period of time. The paper “Ion-Trap Quantum Computation” by Holzscheiter [15] gave a good description of the Paul’s ion-trap and its operation and detailed the implementation of a NOT and CNOT quantum gates. It became clear that linearly trapped ions limit their interaction to their immediate neighbors [6]. Nonetheless, quantum logic synthesis algorithms in the literature have largely ignored such physical constrains and have made the following invalid assumptions:

1. The possibility of any two qubits to interact in an operation regardless of the physical distance and barriers between them,
2. The possibility for a quantum gate to operate on a large number of qubits,
3. The feasibility of using thousands of qubits in a quantum circuit despite the demonstrated hurdles of attaining 10 qubits to date.

Only recently have researchers turned their gaze to the concept of Linear Nearest Neighbor Model (LNNM) as it applies to quantum logic synthesis. Still, there is naivety in the approach where an application of LNNM is limited to the pathways between quantum gates while the gates themselves violate the same LNNM principle. I shifted my research focus to the LNNM model and apply this model from a holistic point of view by considering all aspects of the problem of quantum logic synthesis. I directed my
effort to redefine the *yardstick* used to quantify the performance of automated quantum synthesis algorithm for the reasons that follows:

1. Application of the LNNM constraint to any synthesized quantum circuit is a superior predictor of the complexity of the circuit as it considers the natural constraints imposed by the underlying technology,

2. Uniform measurement of a quantum circuit using the LNNM architecture allows for a *normalized benchmark* for comparing of different solutions utilizing different methods to synthesize the circuit. Notably, benchmarks of algorithms which disregard the cost of interaction between distant or add additional qubits for holding intermediate results cannot be compared to solutions which adhere to the physical realization of quantum gates.

I am confident that my contribution in this field, along with the work of other research institutions around the world, will lay the foundation for the next generation of automated logic synthesis CAD tools which aim to construct application specific quantum circuits (ASQC). Kindled by the spirit of collaboration, I continue to work with other researchers in the field to explore alternative methods of quantum logic synthesis and share our findings with the rest of the world. Guided by my personal belief that fruitful collaborations are the direct outcome of open and transparent disposition amongst researchers in the community, I established an online portal to share the results of our research, and provide an online open repository for sharing algorithms and source code.
Existing internet portals for quantum benchmarks, namely RevLib [30] and Maslov’s Reversible Logic Synthesis Benchmark [20] pages, continue to promote the theoretical, unrealistic, embodiment of a quantum gate as it favors their method of logic synthesis; i.e., lower quantum cost. As I will soon demonstrate, application of the LNNM architecture to existing solutions expose that the methods that some of the researchers use result in a very expensive cost when LNNM is considered.

I also extended my binary algorithms to the ternary domain and discovered new set of challenges as the level of computation increases. I explored the benefit of exploring different ternary orderings to discover solutions with lower quantum gate count. In the binary case, I only had two permutative gates to work with, a wire and an inverter. In the ternary domain, I have five permutative gates to choose from which adds additional dimensions to the solution space. For each additional variable, the increase of the number of solutions is even larger for the ternary space where, for a function with $n$ variables, there exists $3^n!$ possible orderings of the input vector (compared to $2^n!$ for the binary domain). Convergence of the ternary algorithm takes the center stage where I provide multiple proofs of convergence of my ternary algorithm where each proof depends on the relative precedence of the values 0, 1 and 2. This topic is discussed further in Chapter 10.
1.1 **Included in this research**

My research is mainly concerned with the problem of logic synthesis of permutative quantum circuit specifications\(^1\) which are represented as a set of equations or a truth table. Quantum circuit specifications represent any arbitrary \(n\)-input vector mapped to a bijective \(n\)-output vector such that elements of the output vector are a permutation of the input vector; i.e., one-to-one and onto mathematical relation. The field of research will explore quantum logic synthesis of binary and ternary specifications and will progressively focus on the applicability of synthesized solutions to the technology used to construct the quantum circuit: namely, application of Linear Nearest Neighbor Model (LNNM) to the synthesized circuit. A brief overview of the LNNM, the shortcoming of existing synthesis algorithms, and the benefit of applying LNNM architecture to logic synthesis can be found in sections 3.6.4 and 0 below.

The research field is further limited to the set of *reversible* functions which are completely specified; i.e., there are no terms which are designated as a *do not care*.

1.2 **Excluded from this research**

The research will not address the technical implementation of a quantum gate or the fabrication of synthesized quantum circuits. Discussion of quantum mechanical principles and allegories are also excluded along with some of the advance mathematical

\(^1\) A permutative binary specification for an \(n\)-variable circuit represents the set \(S\) consisting of \(2^n\) input minterms where every input minterm maps to an output minterm of the same set \(S\) and each mapping is unique. In group theory, a permutation of a set \(S\) is defined as a bijection from \(S\) to itself \(S \rightarrow S\). Such specification is clearly reversible due to the one-to-one and unique mapping of input and output minterms.
derivations and manipulation of quantum state, gates and propagation of a quantum bit from one state to another. I follow the path of classical CAD tools which relies on the supposition that basic universal primitive gates exist at the abstract level of Boolean logic (NOT, AND, OR, XOR), and the goal of such tools is to arrange and interconnect such gates to implement the desired function. I make a similar assumption where a basic set of quantum primitives exist (in this case NOT, Controlled NOT, Toffoli, and Fredkin gates) which when placed appropriately would implement the desired specification. With the exception of the LNNM approach, I assume that the synthesis algorithm is oblivious to the technology implementing a quantum circuit.

1.3 Approach

Motivated by the potential of quantum computing and other reversible technologies, synthesis of reversible specifications (circuits) has been intensely pursued both in academic and professional circles. Some researchers approached their solutions solely on a pure mathematical model of reversible circuits while others constrained the mathematical model to the limits of quantum mechanics, viz. nature. Methods for finding solutions with minimal quantum gate cost typically require an exploration of the entire search space which typically limits such algorithms to a small number of variables. Exhaustive depth and breadth first traversal algorithms [35], group theory [36] and Boolean satisfiability (SAT) [37] methods suffer a similar fate where the exploration
space grows exponentially with each additional variable, and as a result, are limited to a maximum of six binary variables.

1.4 **Organization of Chapters**

Part I of this dissertation provides background information about quantum computing in general and quantum logic synthesis in particular. Chapter 2 starts with information about and feasibility of quantum computation. Chapter 3 presents some of the quantum gates typically used for constructing quantum circuits and are the main building blocks used in this research. The chapter also contains a description of Ion trap technology which has been used to demonstrate quantum gates and quantum computation. Chapter 4 describes some of the quantum logic synthesis method used in the literature and demonstrates their operations and discusses their strengths and weaknesses.

The remainder of this dissertation is organized in three parts. Part II covers my work in the binary logic synthesis domain and includes the MMDSN/MP (Chapter 5), CSP (Chapter 6) and CSP-EV (Chapter 7) algorithms which are based on the idealistic cost of quantum gates. Chapter 8 covers some of my other experiments in the binary synthesis domain which yielded little success.

Part III shifts focus to the Linear Nearest Neighbor Model in the binary domain and demonstrates methods for calculating the quantum cost for the MCT gates and for any quantum circuit (Chapter 9). Chapter 10 discusses the application of the LNNM to a 2-D
arrangement of qubits and the potential for reduced quantum cost using such arrangement.

Part IV is focused on the ternary and multiple valued domain. Starting with Chapter 11, I cover my work in synthesizing ternary quantum specifications with relatively large number of variables and introduce a set of ternary specifications as benchmark functions. In Chapter 12 I introduce seven more multiple valued benchmark functions and introduce a new file format (XQS) specifically designed for multiple valued functions. In Chapter 13, I introduced a new functional quantum block, the generalized swivel gate, and demonstrate how to calculate the cost for any multiple valued computation basis.
Chapter 2 **Background**

Practically, computation has been the single driving force behind modern technology over the last few decades. The theoretical foundation of modern information processing was established by Church and Turing in 1930s and has become one of the foundational principles of modern computer science. The Church-Turing thesis outlined a classical deterministic model of computation and set the stage for computing feasibility and complexity of computations. I soon came to realize that, for a certain class of problems (NP problems), a vast number of computations become necessary, and impossible to perform, as the solution space for such problems grows exponentially. All the more surprising was the recent claim that there exists a more efficient way of computing, which can, for a certain class of problems, result in exponentially faster algorithms. The newly discovered method depends on the quantum mechanical description of information which gained the appropriate name of *quantum computing*.

While a classical *bit* of information can assume one of two states, the quantum mechanical description of an equivalent two level system, a qubit, is represented by a unit vector in a 2-dimensional complex plane which is represented graphically by the Bloch Sphere – described in 3.4.1. Qubits combine via a tensor product where $n$ qubits are represented by a unit vector in a $2^n$-dimensional complex plane\(^2\). A transistor based memory cell or register can hold a single value at any moment of time, (0 or 1 for binary

---

\(^2\) The geometrical representation of the Bloch sphere is naturally three dimensional, however, by constraining the vector length to 1, the state of a qubit resides only on the surface of the unit sphere; hence, only two dimensions are required to pinpoint the state of the qubit on the surface of the Bloch sphere (analogous to latitude and longitude).
registers). A quantum register, a string of qubits, on the other hand, can be a 0, 1, or anything in between. Similarly, a classical gate expects a single discrete value on each of its inputs, and produces single discrete value on each of its outputs. Assuming binary logic, a quantum gate accepts any value between 0 and 1, inclusive, and produces a value in the same range. When the input values are in a superposition state, halfway between 0 and 1, a quantum gate calculates the results of applying all possible combinations of inputs at the same time.

Despite of the fact that our knowledge of quantum mechanics and its potential applicability to computation has spanned more than a century, advances in quantum computing technology have been moderately small. Such technologies and experiments, however, have been instrumental in validating the concepts of quantum mechanics and have further demonstrated through the construction of quantum gates that the concepts of quantum computing are valid. Quantum computation requires the setup of a very special physical environment where numerous operations must be performed on the register of qubits while those qubits are in a total isolation from external environmental stimuli, events or disturbances; yet, the same set of qubits can interact with, and affect the state of, one another as manipulated by an accurate source of laser or magnetic force. Additionally, once the qubits are initialized to a state, computation must be performed in quick succession as to prevent the loss of information through decoherence - a loss of energy which results in weakening, and eventual ruin of, the interlocking relationship amongst qubits in the register.
In 1958, the German researcher Wolfgang Paul [38] proposed a design principle for isolating single ions in a linear trap and, in 1995, Ignacio Cirac and Peter Zoller [16] proposed the architecture of quantum computing machine based on the Paul ion-trap. The Linear Paul ion-trap uses a linear radio-frequency quadrupole placed within a vacuum chamber at very low temperature where the positively charged ions are suspended, and confined, at the center of the trap through opposing and alternating current applied to the four poles of the trap. The positively charged ions maintain an even distance between one another through under the influence of the Coulomb repulsive force. A set of laser pulses detuned to a very narrow bandwidth are then used to alter the state of a specific ion, or cause it to enter a state of entanglement with another ion. In this arrangement, the set of suspended ions represent the information vessel, the qubit, and the bandwidth, sequence and length of the laser pulse represent quantum gate. The operation of the Paul trap is detailed further in section 3.4.1.

2.1 Feasibility of Quantum Computation

While the fundamental operations required for quantum computation have been demonstrated successfully in the lab, the efforts to scale these experiments to a large scale quantum computer are still in their infancy. The first generation of quantum computers might not be suited for general purpose computing nor malleable to be software driven. Rather, quantum computing devices would be designed, initially, to implement custom algorithms for solving a specific problem of a higher class of
complexity. Such algorithms, however, would still require thousands of qubits to implement, with possibly larger numbers of qubits for error correcting in order to achieve an acceptable level of fidelity.

There are a number of physical systems which have been proposed for the realization of a quantum computer. For example, enormous progress has been made in implementing qubits using Josephson junctions, semiconductor quantum dots, nuclear magnetic spins, trapped ions and many others. DiVincenzo [39, 40, 41], of IBM, outlined the necessary requirements for implementing a quantum computing apparatus as:

1. An extensible set of registers (qubits),
2. Ability to initialize registers to a known initial state,
3. Universal set of quantum gates,
4. Ability to transfer quantum information between spatially separated registers,
5. Ability to extract (measure) the state of registers,
6. Decoherence time of quits longer than duration of quantum gate.

The system which has met most of the requirements deemed necessary for a functional quantum computer uses trapped atomic ions. In particular, state readout, high fidelity and universality of one and two qubit gates, teleportation and error correction have all been demonstrated [41]. In 2009, NIST demonstrated a 2-qubit quantum computer using trapped ions where laser pulses were used to manipulate the state of the qubits [42]. In 2000, Chuang’s team at IBM demonstrated a five (5) qubit system for solving the
problem of “order finding” which finds the periods of a particular function, [43] and in the same year, the scientists at Los Alamos National Laboratories announced that they have achieved quantum coherence in a seven-qubit molecule [43].

As for our technology of choice, ion trap, Monz et al have recently demonstrated the largest number of qubit entanglement in a confined linear Paul ion trap consisting of 14 $^{40}\text{Ca}^+$ ions where each ion represents a qubit [32]. The original linear Paul trap proposed by Wolfgang Paul [38] confines a set of charged particles (ions) to a definite region of space with magnetic or electric field. Raizen et al. [44] and Walther proposed the linear radio-frequency quadrupole (RFQ) trap, known as the linear Paul trap which uses a time-varying electric fields to suspend a set of positively charged ions in free space in a straight line. In their trap, Raizen demonstrated the ability to trap 15 and 33 ions suspended in space forming a line, mimicking a string of beads.

I now point our attention to a list of concrete examples where quantum computation has been developed and implemented:

**Quantum Cryptography** was one of the earliest, and so far the most developed application of quantum information. Bennett and Brassard have proposed a communication method which relied on the impossibility of measuring a quantum system without disturbing it [45]. Quantum cryptography is different from traditional cryptographic systems in that it relies more on physics, rather than mathematics, as a key aspect of its security model. Unlike classical CMOS, *sampling* or measuring a quantum system in the midst of computation, or teleportation, *cannot be done undetected*. Any
act of measurement will always leave a mark on the system which makes it impossible for an eavesdropper to listen-in, i.e. sample the information, without leaving a trace. Amazingly, the protocol can be proven to be secure, regardless of resources available to a potential eavesdropper. Today, there exist commercial implementations of this protocol where it has been used to build secure networks to transmit election data, truly random number generators and quantum RADAR [46].

**Quantum Simulations**, proposed by Feynman [47] and Deutsch [48], provided the initial impetus for studying quantum computation. A full description of a quantum system consists of \( n \) 2-level systems which require \( O(2^n) \) complex numbers, and as a result, simulations of quantum systems can be performed only for small systems, due to the memory requirements involved in tracking the variables. However, if the information is encoded in qubits, only storage of the order \( O(n) \) is required, reducing the complexity exponentially. The ability to simulate quantum systems could help resolve a number of important problems in a condensed matter and particle physics.

**Quantum Algorithms** such as the Shor’s factoring algorithm [49] and Grover’s database search algorithm [50], pushed quantum computation into the mainstream of physics and computer science. The best known classical algorithms for factoring a large number \( N \) need \( O(e^{3\sqrt{\log N}}) \) operations, while the quantum equivalent uses only \( O((\log N)^3) \), an exponential improvement in speed. Factoring turns out to be very important for cryptography, and a realization of a polynomial time factoring code would render most of current encryption schemes useless.
These advantages do not come without a cost. The increased size of the description required for quantum bits is a result of the importance of relative qubit phases. In an experimental realization of quantum information processing, these phases must be carefully preserved throughout all performed operations. However, any interaction with the environment may alter the energy states of the qubits, resulting in uncontrolled phase shifts, or decoherence. Such interactions are inevitable, though, and for a while it seemed that complex computations would not be possible to practically realize. The discovery of quantum error correction has dramatically altered the landscape, demonstrating that as long as the errors are small enough, they can be reduced further by constructing “logical qubits” out of a number of physical ones [51, 52]: storage qubits plus error correcting qubits. The current theoretical estimates place the necessary threshold of maximum failure probability at $O(10^{-3})$, which is likely within experimental reach for an ion trap implementation. Ion trap features high trap depth which allows for long storage times, and hence lower decoherence (>10 min have been observed) [40]. The fact that the charges are confined in free space, away from any objects, the ion trap remains undisturbed, for the most part, allowing for precise control of individual qubits.
Chapter 3 **Binary and Ternary Quantum Gates**

3.1 **Introduction**

This chapter provides background information about quantum gates, circuits, technologies and the performance of a quantum algorithm is measured. I will start by describing the basic quantum primitive gates used to construct more complex gates which are further used to construct logic blocks (oracles) or circuits implementing a functional specification. I will also give details of ion-trap technology as it is relevant to our work in the space of Linear Nearest Neighbor Model architecture.

3.2 **Binary Quantum Gates**

Similar to classical logic, binary quantum gates are given representative symbols to designate their functionality. The literature describes many quantum gates, or operators, where some operate on a single qubit and others operate on two qubits or more. There is, however, a limited set of primitive quantum gates which are used to construct composite gates of greater complexity. The NOT, Controlled-NOT and Controlled-V gate represent the NCV library of primitive quantum gates which are commonly used in quantum logic synthesis and are the building blocks of composite gates. Even though the Fredkin (swap) gate is composed of a cascade of CNOT gates, it is also typically considered as a primitive gate and included in the NCVS library of gates. Similarly, the Toffoli 3-qubit gate is considered as a functional primitive and is included in the NCT library which is used in this dissertation.
One of the commonly used gates is the 3-variable Toffoli gate and the Multiple-Control Toffoli (MCT) gate. Although these gates are ubiquitously used in the literature, they are composite gates built from the NCV library gates as shown in section 8.2.

3.2.1 **Inverter NOT gate**

Figure 3-1 shows the classical inverter (NOT) gate, on the left hand side, and its equivalent quantum inverter symbolized with the XOR symbol used in binary logic equations, on the right hand side. Unlike its classical counterpart, the quantum NOT gate is reversible by definition allowing information to flow in either direction.

![Figure 3-1](image)

*Figure 3-1 The classical inverter in (a) is unidirectional where it inverts the value of the input ‘a’ while the quantum inverter in (b) is bidirectional (reversible) which transforms ‘a’ into ‘\(\bar{a}\)’ and vice versa.*

3.2.2 **Feynman or Controlled NOT gate**

Figure 3-2 shows the two qubit Controlled NOT gate (CNOT) which is also known as the Feynman gate after the well-known physicist Richard Feynman. The CNOT gate allows for conditional modification of the lower qubit based on the value of the upper qubit. A value of zero on qubit \(a\) renders the lower part of the gate as pass-through, while
a value of *one* on qubit *a* enables the inverter on lower part of the gate. From the truth table, the CNOT gate performs the same functionality as the classical XOR gate. Again, unlike the classical counterpart, the CNOT gate is reversible. In Figure 3-3 for example, placing two CNOT gates back to back returns the state of both qubits to the original state.

![Figure 3-2 Controlled NOT gate where the ‘b’ qubit is inverted when the ‘a’ qubit = 1. Otherwise, the ‘b’ qubit remains unmodified. The value of the ‘b’ qubit represents the XOR relation which is equivalent to the XOR classical gate. The classical XOR gate is not reversible while the CNOT gate is reversible.](image)

Figure 3-3 Demonstration of quantum reversibility of the CNOT gate. An application of the first CNOT gate to the ‘ab’ inputs produces \(a \oplus b\) (middle table) on the ‘b’ qubit. Applying the second CNOT gate reverses the operation and restores the original value of ‘b’. The value of qubit ‘a’ remains unchanged throughout the entire process.

### 3.2.3 Toffoli and MCT gates

Figure 3-4 shows one of the universal quantum gates, the Toffoli gate, which, similar to the NAND gate, can be used to implement any quantum circuit. The Toffoli gate enables the inverter on qubit *c* only when both qubits *a* and *b* hold the value *one*. We can easily implement the quantum equivalents of both the AND and NAND gates by initializing qubit *c* to either a *zero* or *one* as shown in Figure 3-4.
Figure 3-4 Equivalent implementation of the classical AND and NAND logic gates (a) using the Toffoli quantum gate (b). The NAND gate is implemented by strapping the 'c' qubit to '1' which results in \((1 \oplus ab) = \overline{ab}\).

The Toffoli gate is not considered a primitive element of quantum gates as it decomposes to a set of CNOT and Controlled-V gates, discussed in section 8.2. The symbolism is extended further where I define the Multiple Control Toffoli gates (MCT\(n\)) as follows:

**Definition 3-1:** A multiple-control Toffoli (MCT) gate of \(n\) inputs with target line \((x_t)\) and control lines \((x_{c1}, x_{c2}, x_{c3}, \ldots x_{c(n-1)})\) maps all control to their original value and the target \((x_t)\) to \((x_{c1} \cdot x_{c2} \cdot x_{c3} \ldots x_{c(n-1)}) \oplus x_t\). In essence, all control lines must be active for the target \((x_t)\) to be inverted.
Figure 3-5 Universal Toffoli gate ($C^2$NOT) which is classically implemented by the AND/XOR gates (a), however, the quantum Toffoli gate (b) is reversible where, similar to the CNOT gate above, application of two Toffoli gates restores the state of qubit ‘c’ to its original value.

3.2.4 Fredkin or Swap gate

The Fredkin gate swaps the values of two qubits as shown in Figure 3-6. Unlike classical registers, where swapping bits of information involves transporting values over a physical connection, in a quantum register, no such wires exist, and swapping here refers to exchanging the information amongst the two qubits in a one for one manner. The no-cloning theorem driven by the preservation of quantum momentum (quantum mechanics) prohibits a one way delivery of data, but some level of an exchange must occur. Figure 3-6a shows the symbolic representation of the Fredkin gate, while Figure 3-6b shows the composition of the gate using the CNOT primitive gate.

Figure 3-6 (a) Fredkin or Swap gate is a composite structure which is implemented using three CNOT gates arranged according to the figure in (b). In classical logic, a swap of two bits is accomplished through the exchange of physical wires but in quantum technology a set of gates are used to exchange the information of two qubits.
A Controlled Swap gate also exists where the operation of the swap gate is active only when all control qubits are active.

3.2.5 *Square Root of Not (V) gate*

This is one of the special gates of quantum computing and has no analogy in binary logic. The V gate transitions a qubit from one of its pure states (0 or 1) to a state of superposition halfway between 0 and 1. The state of superposition in quantum computing is represented on the Bloch sphere, described shortly, by the circle around the equator of the Bloch sphere. The V gate places a qubit in a state of superposition which is represented on the Bloch sphere as a single point on the equator of the sphere.

The V gate is also referred to as the Square Root of NOT because multiplying its matrix representation by itself, i.e. squaring it, yields the matrix representing the NOT gate. The effect of the V Gate can be visualized on the Bloch sphere as rotating the unit vector pointing to the zenith (state $|0\rangle$) or nadir (state $|1\rangle$) points by 90° in the clockwise direction. The $V^\dagger$ gate is an inverse gate which rotates the unit vector around the Bloch sphere by -90°. In effect $V\cdot V^\dagger = I$. The V gate is represented mathematically by the matrix:

$$V = \sqrt{\text{NOT}} = \begin{pmatrix} \frac{1+i}{2} & \frac{1-i}{2} \\ \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix}$$
Similar to the NOT and CNOT gates, the V and V\(^\dagger\) gates also exist with a controlling qubit where such gates are referred to as the Controlled-V and Controlled-V\(^\dagger\) gates (CV and CV\(^\dagger\)).

3.2.6 Hadamard Gate

The Hadamard Gate (H) acts on single qubits and maps the basis state \(|0\rangle\) to the superposition state \(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\) and state \(|1\rangle\) to the superposition state \(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\) which represent a \(\pi\) rotation around the \(x\)- and \(z\)-axes. The gate is represented by the following symbol and Hadamard matrix shown in Figure 3-7.

\[
\begin{pmatrix}
1 \\
1 \\
\end{pmatrix}
\]

Figure 3-7 Symbol of the Hadamard gate and its matrix representation. The Hadamard gate, which has no classical equivalence, places the qubit in a state of superposition between the pure states of \(|0\rangle\) and \(|1\rangle\). The gate is also reversible where an application of two Hadamard gates back to back restores the original value of the qubit.

3.3 Ternary (and Multiple Valued) Quantum Logic gates

Ternary logic is the closed logic system with certain ternary operators that operate on three logic values \{0, 1, 2\}. In quantum mechanics, the three ternary values could correspond to the different polarization of a photon, alignment of nuclear spin in a uniform magnetic field, or different energy levels in an ion-trap system. To date, Nuclear Magnetic Resonance (NMR) and Ion Trap are the most promising technologies which were used to demonstrate “quantum circuit model” of quantum computation.
In addition to the basis states mentioned above, a qudit can exist in a state of superposition anywhere in between two or three states (including a phase component). The Dirac notation allows for describing the state of a qudit as follows:

\[ \Psi = \alpha |0\rangle + \beta |1\rangle + \gamma |2\rangle \]

where, \( |\alpha|^2 + |\beta|^2 + |\gamma|^2 = 1 \). The \( \alpha \), \( \beta \), and \( \gamma \) components are referred to as the amplitude and their square \( |\alpha|^2 \), \( |\beta|^2 \), \( |\gamma|^2 \) represent the probability amplitude of each state; meaning \( |\alpha|^2 \) is the probability amplitude that the qudit will be measured in the \( |0\rangle \) state.

An \( n \)-qudit quantum register has \( 3^n \) computational basis states donated as \( |00...,0\rangle, |00...,1\rangle, \ldots, |11...,1\rangle \). At any moment, a register of \( n \)-qudits will be in a composite superposition state composed the superposition states of each qudit.

3.3.1 Galois Field 3 Logic – GF(3)

A Galois Field is a field that contains a finite set of elements which is of the order \( p \) or \( p^k \) where \( p \) is a prime number and \( k \) is a positive integer. In the ternary case, the GF(3) is a prime Galois Field of three elements \( \{0,1,2\} \) which are closed under the addition and multiplication \textit{modulo 3} operators. Table 3-1 demonstrates the closure of GF(3) under the addition and multiplication operators [53].
Table 3-1 Galois Field 3 operations (modulo 3) for a single qutrit with addition operator (a) and product operator in (b). The addition operator uses the same symbol as the binary XOR symbol ($\oplus$) since, mathematically, the binary XOR operation is a Galois Field 2 (modulo 2) operation.

\[
\begin{array}{c|ccc}
\oplus & 0 & 1 & 2 \\
\hline
0 & 0 & 1 & 2 \\
1 & 1 & 2 & 0 \\
2 & 2 & 0 & 1 \\
\end{array}
\quad
\begin{array}{c|ccc}
\circ & 0 & 1 & 2 \\
\hline
0 & 0 & 0 & 0 \\
1 & 0 & 1 & 2 \\
2 & 2 & 0 & 1 \\
\end{array}
\]

(a)  (b)

The modulo 3 addition operator can be used as the foundation for a set of the ternary quantum operators shown in Table 3-2 [54]. Notice that the set of six gates are the total number of permutations of the basis states \{0,1,2\}. A thorough discussion of these gates is found in Chapter 10 and Chapter 11.

Table 3-2 Ternary operators based on GF(3) addition operator where each gate (operator) is a mathematical bijection with one-to-one and onto mapping from the input to the output.

<table>
<thead>
<tr>
<th>Gate</th>
<th>$F(a)$</th>
<th>Input</th>
<th>Gate symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0</td>
<td>$(a+0) \ mod \ 3$</td>
<td>0 1 2</td>
<td>a --- a</td>
</tr>
<tr>
<td>+1</td>
<td>$(a+1) \ mod \ 3$</td>
<td>1 2 0</td>
<td>a ---+a_1</td>
</tr>
<tr>
<td>+2</td>
<td>$(a+2) \ mod \ 3$</td>
<td>2 0 1</td>
<td>a ---+a_2</td>
</tr>
<tr>
<td>12</td>
<td>$2(a+0) \ mod \ 3$</td>
<td>0 2 1</td>
<td>a ---+a_{12}</td>
</tr>
<tr>
<td>02</td>
<td>$2(a+1) \ mod \ 3$</td>
<td>2 1 0</td>
<td>a ---+a_{02}</td>
</tr>
<tr>
<td>01</td>
<td>$2(a+2) \ mod \ 3$</td>
<td>0 1 2</td>
<td>a ---+a_{01}</td>
</tr>
</tbody>
</table>
3.4 Mathematical representation of qubit state

Classical multiple valued logic has been trivially simple to represent as a finite set of literals representing the state of a classical bit. For example, the set \{0,1\} represents all possible states of a binary bit, while the set \{0,1,2\} represent all possible states of a ternary bit. In quantum logic, the most common representation of a qubit is the Bloch Sphere that is used to visualize the current state of a qubit. The Bloch sphere represents a continuous landscape for the qubit’s state to occupy rather than the discreet set of a classical multiple valued system. The Bloch sphere represents both a magnitude and phase element of the internal state of the qubit, where, for an optical quantum system, for example, describes the magnitude and phase of the light photons employed as the qubit.

![Bloch sphere illustrating a qubit state \(\psi(t)\) where the North and South poles are designated as the binary basis states of \(|0\rangle\) and \(|1\rangle\). Unlike classical logic, the qubit could assume any state between the basis states of \(|0\rangle\) and \(|1\rangle\) where such states are symbolized by all the points on the surface of the unit sphere.](image)

In quantum computing technologies, measuring the state of a qubit is a probabilistic event, where, rather than measuring and exact zero, one or two, of a single computation, an ensemble of computing registers (possibly thousands or millions) are all constructed in...
the same manner, prepared in the same manner, and excited with the same set of inputs. The answer to the computation is the answer with the highest probability.

3.4.1 Bloch Sphere and Quantum State

A qubit’s state can be visualized as a unit vector on the well-known Bloch sphere shown in Figure 3-8 which can be represented mathematically as:

$$|ψ\rangle = e^{iγ} \left( \cos \frac{θ}{2} |ψ\rangle + e^{iφ} \sin \frac{θ}{2} |1\rangle \right)$$

Where $θ$, $φ$ and $γ$ are real numbers, and $0 \leq θ \leq π$ and $0 \leq φ \leq 2π$ are the only two angles necessary to define any point on the sphere (similar to the longitude and latitude lines on a map). The $e^{iγ}$ term is a global phase which has no observable effects and can be ignored resulting in the equation:

$$|ψ\rangle = \left( \cos \frac{θ}{2} |ψ\rangle + e^{iφ} \sin \frac{θ}{2} |1\rangle \right)$$

In fact, the Bloch sphere merely extends the two dimensional complex $(x + iy)$ plane into the third dimension $z$ and limiting the radius (vector length) to 1. By doing so, only two variables ($θ$ and $φ$) are necessary to locate any point on the surface of the unit sphere. For a two dimensional complex plan, the length of the vector $\vec{v} = (x + iy)$ can be found by:

$$|\vec{v}| = v^*v = (x - iy) \cdot (x + iy) = x^2 + y^2 = 1$$

which is the equation for the unit circle. In polar form, the vector can be represented as:

$$\vec{v} = r(\cos θ + i \sin θ) = re^{iθ}$$
which for the unit circle would become:

\[ \vec{v} = \cos \theta + i \sin \theta = e^{i\theta} \]

Similarly, the general qubit state can be written in the Dirac notation as:

\[ |\psi\rangle = \alpha |0\rangle + \beta |1\rangle \]

where \( \alpha \) and \( \beta \) are complex numbers which, for a pure quantum state, has the unit length constraint:

\[ \langle \psi | \psi^* \rangle = 1 \rightarrow |\alpha|^2 + |\beta|^2 = 1 \]

In quantum mechanics, the coefficients \( \alpha \) and \( \beta \) represent the *probability amplitude* of the corresponding states; meaning, \( |\alpha|^2 \) is the probability of measuring the qubit in state \( |0\rangle \) and \( |\beta|^2 \) is the probability of measuring state \( |1\rangle \).

### 3.4.2 Evolution of Quantum State

In the following section about Ion trap, I will show that an application of a finely tuned laser pulse can transition a qubit’s state from ground to excited states. The length of time that the laser pulse is applied has an impact on the final state of the qubit. Using the Bloch sphere as a visualization aid, the qubit’s state can be thought of as a vector touching a single point on the Bloch sphere and the application of the laser pulse transitions that point around the surface of the sphere. For a binary system, the north and the south poles of the sphere represent the basis states \( |0\rangle \) and \( |1\rangle \) respectively. Any point in between is said to be in a *superposition state* between the two basis states. I describe such a state by the application of a \( \frac{\pi}{2} \) pulse representing which is symbolically
represented by the Hadamard gate \( (H) \) which evolves either of the basis states of a qubit to a state of superposition. In essence, we can visualize the application of a quantum gate as a rotation \( (\sigma_x, \sigma_y, \sigma_z) \) of the qubit’s state vector around the three orthogonal axes. The set of rotation operators, known as the Pauli matrices are shown below:

\[
\begin{align*}
\sigma_x &= X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\
\sigma_y &= Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \\
\sigma_z &= Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}
\end{align*}
\]

The Hadamard gate is a composition of two Pauli matrix rotations as follows:

\[
H = \frac{1}{\sqrt{2}} (X + Z) = \frac{1}{\sqrt{2}} \left( \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \right) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}
\]

Notice that the above matrices are all unitary matrices which preserve the unit length of the vector within the Bloch sphere. By definition, a matrix \( U \) is unitary if

\[
U \cdot U^\dagger = U^\dagger \cdot U = I
\]

The NOT gate is the same as the X gate and can be simulated with HZH as follows:

\[
X = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}
\]

Similarly a CNOT gate can be simulated with the following primitive gates:
Multiplying the Hadamard gate matrix by itself results in the identity matrix which indicates that, similar to the CNOT gate, the Hadamard gate is a self-inverting gate.

3.5 History of Ion Trap

Ignacio Cirac and Peter Zoller proposed the original utility of Ion-Trap for quantum computation in 1995 [16]. An ion-trap is a special apparatus which, as the name implies confines positively charged particles into a definite region in space, typically in a linear array of particles uniformly lined like a tightly held string of beads. An ion-trap provides the environment for quantum computation where, on one hand, qubits must be isolated from the surrounding environment in order to minimize decoherence (loss of internal state), while on the other hand, qubits must be individually addressable by external laser pulses to transform their internal state according to the quantum gate under operation.

Figure 3-9 shows the Radio-Frequency Quadra-pole (RFQ) trap, aka linear Paul trap [44] which utilizes a radio-frequency time-varying electric field along four long poles to suspend the ions in a vacuum chamber and effectively isolates them from the
environment. Each ion which serves as a single quantum register, a qubit, is individually addressable with a sharply defined laser beam targeted at the exact location in space.

Figure 3-9 Ion trap apparatus showing a vacuum chamber in the middle where ions are linearly aligned at the center of the narrow cavity.

Figure 3-10 and Figure 3-11 illustrate the construction of an RFQ ion trap with four rods (poles) where each pair of opposing rods are connected to the same source (polarity) and neighboring rods are of opposite polarity. In a theoretically symmetrical trap, the opposing positive pairs of rods exact a repellant force upon the positively charged ion of the same magnitude but in the opposite direction. At the same time, the negatively charged rods attract the positively charged ion with forces of the same magnitude but in opposite direction. However, minuscule differences in timing or the construction of the trap distorts the idealistic symmetry and the ions attempt to leak out of the trap. By switching the polarity of the poles, the ion will reverse its attempt to escape and move in z-axis direction (back to the center of symmetry). Applying an alternating source with the appropriate switching frequency, the charged particles will remain within the trap along the axis of symmetry.
Despite their confinement along the axis of symmetry, ions will attempt to escape in either direction along that axis. The Coulomb repulsion force of similarly charged particles typically maintains a distance between each neighboring ion; however, due to the constant vibration of each ion, such force will eventually push the ions at the edge out of the trap. To avoid such an incident, a static DC voltage (end cap electrodes) is applied on both ends of the axis as shown in Figure 3-11.

Despite their confinement along the axis of symmetry, ions will attempt to escape in either direction along that axis. The Coulomb repulsion force of similarly charged particles typically maintains a distance between each neighboring ion; however, due to the constant vibration of each ion, such force will eventually push the ions at the edge out of the trap. To avoid such an incident, a static DC voltage (end cap electrodes) is applied on both ends of the axis as shown in Figure 3-11.

**Figure 3-10** Paul ion-trap with four electrical rods having exact characteristics (material, radius, length, ...etc) placed in parallel at equal distances. The two opposite electrical rods are connected together to a time varying alternating electrical field of the same charge. The other set of poles carry the exact electrical field but with opposite charge (180° phase shift). Positively charged ions are trapped in the center of the four rods along the central z-axis by the Coulomb forces of the alternating magnetic field which alternates at a high enough frequency to keep pushing and pulling on the ions along the two orthogonal axes (x and y). The ions are kept from escaping from the open ends (along the z-axis) by placing positive static charges at both ends of the z-axis.

**Figure 3-11** Pictorial representation of the Paul ion-trap with ions trapped along the center z-axis. Each ion acts as a quantum register (qubit) where finely detuned laser beams (shown as two arrows pointing upwards) represent the quantum gate. The laser beams are used to affect the state of each individual ion (a single qubit gate), or the state of two neighboring ions in an entangled state.
It is also essential to reduce the vibrational motion of individual ions in order to ensure their exact spatial position within the trap, such that, when a laser beam is applied, it can be applied to a specific ion. Therefore, in order to individually address each ion, the motion along the trap must below the wavelength of the laser beam while the distance between neighboring ions along the axis of symmetry must be larger than the wavelength of the beam. The vibrational motion of the ions can be reduced by cooling the ions to an extremely low temperature (below 10 K) which is effectively accomplished by multi-step process of laser pumping, Doppler cooling and broadband cooling [15].

3.5.1 Operation of the trap

A string of ions confined within a Paul trap, even cooled ions, will vibrate within a sphere of motion centered around a single point in space, and therefore, a string of N ions have 3N vibrational modes. Along the axis of symmetry, ions will vibrate back and forth pushing one another along that axis. Similar to a set of strongly coupled pendulum connected with a spring, the motion of a single pendulum will cause all others to move in a similar pattern. Unlike a spring, however, vibrations exhibited by a string of ions are quantized, where the amplitude of motion depends on the number of quanta (phonons) in the vibrational mode. At the lowest energy, or absence of individual energy phonons, all ions will vibrate back and forth in unison in the same direction. This is known as the common mode of vibration and represents the common-mode quantum state (stationary = |0⟩ and vibrating = |1⟩) – analogous to a modulated wave in a microwave signal.
In addition to the common vibrational mode, each ion exhibits its own vibrational mode which can be addressed individually by a laser beam of an appropriate wavelength. As a result, the vibrational state of each ion can represent another vessel of quantum information (a qubit) – *logical qubit state*. As a result, the state of a string of ions is given as:

\[
|q_1, q_2, \ldots, q_j \rangle |n\rangle
\]

The first ket, \( |q_1, q_2, \ldots, q_j \rangle \), represents the logical qubit states of the \( j \) ions in the trap, and the second ket, \( |n\rangle \), represents the common-mode vibrational state \( (n=0, 1, 2\ldots etc) \) where \( n \) represents the number of energy phonons in common-mode. For the lowest energy common-mode \( (n=0) \), ions are not vibrating along the axis of symmetry which represents a *Common-Mode Quantum State* (CMQS) of \( |0\rangle \), and when all ions are vibrating in unison along the same axis, the CMQS \( = |1\rangle \), see Figure 3-12. The common mode vibrational state is typically used for establishing a state of entanglement between two ions.

*Figure 3-12 Common Mode Quantum State (a) all ions are stationary \( \rightarrow \) State \( |0\rangle \), (b) A single energy phonon excites all ions to vibrate in unison in the same direction \( \rightarrow \) State \( |1\rangle \).*
3.5.2 One Qubit Operation

A single qubit operation can be implemented by affecting the internal state of each individual ion or affecting the vibration state of the entire trap. The latter, however, will only allow for the realization of a single qubit regardless of the number of ions in the trap (a single quantum of information) as they all share the common mode vibrational state. Consequently, the internal state of an ion is used as the carrier of information, while the vibrational state is used to entangle multiple qubits which is used for implementing multiple qubit gates – entanglement will be discussed shortly.

3.5.2.1 The electron model – a preview

The electron model proposed by Rutherford and Bohr with the electrons spinning around the nucleus at discrete distances, the planter model, was superseded by the cloud model proposed by Erwin Schrödinger. By proposing the probability function for the Hydrogen atom, he surmised that the electron does not follow a specific orbit, but rather, it can be found within a region. Heisenberg proposed his landmark uncertainty principle declaring that simultaneous measurement of both the exact location of an electron and its precise speed and direction is not possible; and hence, it is not possible to measure the path a specific electron takes as it orbits the nucleus which invalidates the Bohr model of the atom. Scientists can, however, determine the area an electron will probably occupy, and the probability of finding the electron at some place inside this area. A map of this area and its probabilities forms a cloudlike pattern known as an orbital. Each orbital can
contain two electrons, but these electrons cannot have identical properties, so they must spin in opposite directions. Orbitals are grouped into the set of shells around the nucleus. Each shell can contain a limited number of orbitals, which means that each shell can contain a limited number of electrons. Each shell corresponds to a certain level of energy, and all the electrons in the shell have this same level of energy Figure 3-13.

Within a shell, the \textit{s-orbital} has the lowest energy level followed by the \textit{p, d and f-orbitals} consecutively. Electrons fill orbitals with lower energy levels first followed by higher energy levels of orbitals. Exceptions occur where, for example, the \textit{4s-orbital} (4\textsuperscript{th} shell) has a lower energy level than the \textit{3d-orbital}, which results in electrons occupying the \textit{4s-orbital} before the \textit{3d-orbital}. An injection of a photon at the appropriate frequency can cause an electron to move from a lower energy orbital to a higher energy orbital or vice versa.

\textit{Figure 3-13} Electron orbitals around the atom showing a cloudlike structure where the electron is most likely to occupy depending on the energy level that it carries. An electron can be excited to move to a higher energy level through the injection of a photon of a specific wavelength (finely detuned laser beam). Since multiple energy levels are available for the electron to occupy at each orbital shell level, Multiple Valued logic is implemented by moving the electron between multiple distinct energy levels.
The ion’s electronic ground state ($s$-orbital) can be used to represent the $|0\rangle$ while one of the higher energy states ($p$- and $d$-orbitals) are used to represent the $|1\rangle$ state [Figure 3-14]. In an ion trap, a sharply focused laser beam of specific amplitude and duration detuned to the transition frequency ($\omega_0$) moves the electron from its ground to the long lived excited state (1 second). The duration of such a transition is known as a $\pi$-pulse transition. An application of the same laser beam for half the time a $\frac{\pi}{2}$ pulse, places the electron in an equally weighted superposition state between $|0\rangle$ and $|1\rangle$ states $[\frac{1}{\sqrt{2}} (|0\rangle + |1\rangle)]$ – (A Hadamard gate).

3.5.3 Entanglement

Entanglement was initially a mysterious and controversial aspect of quantum mechanics which led Einstein, Podolsky and Rosen (EPR) [55] to reject the notion that the universe may have a fundamentally non-local interpretation and expressed as much in the so-called EPR paradox. Entanglement is the property whereby a measurement on one
part of a quantum system affects the measurement outcome of another. Later, however, Bell, Clauser, Horne, Shimony and Holt [55] where able to confirm, experimentally, that the concept of entanglement is a part of Nature and an integral part of quantum mechanics.

Entanglement is the cornerstone of implementing two qubit gates and is at the heart of describing quantum many-body physics harnessing the many-body phenomena, such as quantum phase transitions, to entangle many-body particles together, at absolute zero temperature. The implementation of the CNOT gate in an ion-trap system relies on entangling the two ions in order to achieve the conditional control of one qubit over the other.

3.5.4 CNOT Gate

While we can interact with each bit separately affecting its electronic state, laser can also be used to affect the vibrational mode of the ions in the trap along the axis of alignment. Manipulating the vibrational state along with the logical electronic state allows for the implementation of the CNOT gate which along with the one qubit gates mentioned above can be used to implement any logic circuit.

I mentioned earlier that the vibrational mode acts as the modulated wave of a microwave signal. In this case, the individual ion’s electronic state acts as the carrier wave of a modulated microwave signal exhibiting a dual sideband spectrum. Figure 3-15
shows the vibrational sideband spectrum of an ion’s electronic state of frequency $\omega_0$ which is much higher than the common mode vibrational frequency $\omega_1$.

Assuming two ions where both their electronic and vibrational state of an ion have been initialized to their ground states $|0\rangle$ (i.e., $s$-orbital and no vibrational motion Figure 3-12a)

$$|q_1q_2\rangle |\nu\rangle = |00\rangle |\nu\rangle$$

In order for the ion to transition state (either electronic or vibrational), a quantized amount of energy must be injected into the system consisting of a laser beam of a specific wavelength. Application of a laser beam detuned to the blue (higher) sideband ($\omega_0 + \omega_1$) to the first ion for the $\pi$-pulse duration will transition its electron’s electronic state to

Figure 3-15 Ion’s electronic excitation state coupled to its vibrational motion. Each electronic transition at frequency $\omega_0$ has sideband frequencies as a result of the vibrational motion $\omega_1$. 

59
\[ |1 \ldots \rangle |\ldots \rangle \] due to the \( \omega_0 \) and the vibrational mode of all ions in the trap to the \( |\ldots \rangle |1\rangle \) state due to the energy of \( \omega_1 \) leaving the system in the state:

\[ |q_1q_2\rangle |\nu\rangle = |10\rangle |1\rangle \]

To implement a CNOT gate, however, we would apply the blue sideband frequency for a half the duration a \( \frac{\pi}{2} \) pulse, which would leave the system in the entangled state.

\[ |q_1q_2\rangle |\nu\rangle = \frac{1}{\sqrt{2}}(|00\rangle |0\rangle + |10\rangle |1\rangle) \]

Notice that the first ion’s electronic state and the vibrational state of all ions are both entangled. If we now address the second ion, still in ground electronic state, with a laser beam detuned to the red sideband \( (\omega_0 - \omega_1) \), the energy level of the beam is below its transition frequency \( \omega_0 \) which is not enough to transition. However, if the exact deficit in energy \( (\omega_1) \) is borrowed from the common vibrational mode, the second ion has enough energy to transition from its ground to excited state. Borrowing that phonon of energy, however, will also transition the trap’s vibrational state back to its ground state (no vibration).

\[ |q_1q_2\rangle |\nu\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) |1\rangle \]

Notice that the final electronic state of the second ion depends on the state of the electronic state of the first ion where \( q_2 \) is negated only when \( q_1 \) is \( |1\rangle \), and hence, the CNOT gate.
3.5.5 Quantum Cost Calculation

Quantum cost of a circuit represents the number of one and two-qubit gates within the circuit which, physically, represents the effect of a single laser beam that performs one of the Pauli rotations described above. The NOT gate, for example, requires a single rotation around the $x$-axis; namely, a $\sigma_x(\pi)$ rotation so the quantum cost of a NOT gate is one(1).

Although we’ve demonstrated a CNOT gate with the HZH cascade, a CNOT gate can easily be implemented with a single Controlled-$X$ rotation requiring a single pulse. A Toffoli gate, on the other hand requires five pulses as shown in Figure 3-16 which accounts for the five primitive CV and CNOT gates each requiring a single pulse (Note that, for this calculation, I am disregarding the LNNM criteria described in section 0.)

![Figure 3-16 Decomposition of the composite Toffoli gate two the five primitive 2-qubit gates (CNOT and CV gates). The Toffoli gate is assumed to have a quantum cost of 5 which is the number of one and two qubit primitive gates necessary to implement it.]

3.5.6 Ternary Quantum gates in an ion-trap system

In an ion-trap system, a ternary system can be implemented in a similar manner as I described in section 3.5.2 above where I outlined the operation of a binary system with two the basis states $|0\rangle$ and $|1\rangle$ represented as the energy levels $|g\rangle$ and $|e\rangle$. The
ternary energy states shown in Figure 3-17 can be used to implement the ternary quantum states $|0\rangle$, $|1\rangle$ and $|2\rangle$. Ternary gates can be implemented by the application of the appropriately detuned laser beam to perform state transitions from one state to the next. For example, the $[[01]]$ gate can be performed by the application of a laser beam of $w_{01}$ frequency which will transition qubits in state $|0\rangle$ to state $|1\rangle$ and qubits in state $|1\rangle$ to state $|0\rangle$ while leaving qubits in state $|2\rangle$ unchanged. The $[[12]]$ gate can be implemented in a similar manner with the application of a laser pulse of $w_{12}$ frequency.

![Diagram of ternary energy states](image)

*Figure 3-17 Implementation of ternary energy states of an ion-trap system (a) transitions between any two states is possible with a single pulse which requires three distinctly detuned laser systems for each qubit, while in (b) only transitions between neighboring energy levels are possible using two distinctly detuned laser systems but requiring two pulses for transitions between the states $|0\rangle$ and $|2\rangle$.*

Implementing the $[[12]]$ gate depends on the construction of the ion-trap. Figure 3-17a shows an ion-trap construction with three laser beams allowing a direct implementation of the $[[12]]$ gate with a single application of a laser pulse of $w_{12}$ frequency. The implementation shown in Figure 3-17b requires two pulses to implement the $[[12]]$ gate going through the $|1\rangle$ state.
3.6 Methods of Quantum Logic Synthesis

Of particular interest to our field of study are the following categories of reversible logic synthesis:

3.6.1 Heuristic methods

Although simple in concept, Heuristic methods are able to synthesize functions with large number of variables. Starting from a truth table representation of a reversible binary circuit specification, the transformation-based method proposed by Miller, Maslov and Dueck (henceforth MMD) [56] compares every input minterm to its corresponding output minterm. For every mismatched qubit, the algorithm adds an inverter (NOT gate) to correct the mismatch, and hence, construct a cascade of quantum gates to implement the specification. A single guiding principle of the algorithm prohibits any operation from altering minterms which have already been ‘corrected’, which, by design, allows this algorithm to converge for any arbitrary binary specification. Although the synthesis process is remarkably fast for large specifications of up to 16 binary variables, the algorithm’s demand for memory resources increases exponentially, which, as a result, limits its ability to process functions with larger number of variables.
Galvanized by the above limitation of the MMD algorithm, I realized and successfully demonstrated that exploring alternative sequences of the input and output minterms could yield circuits with lower quantum cost [22]. Before I delve into our discovery however, it is appropriate at this time to illustrate the mechanics of the MMD algorithm as it represents the foundation of some of our work.

Given the truth table of an arbitrary reversible specification, Figure 3-18, where the input vector \( \text{ab} \) shown in column 1 maps to the output vector \( \text{AB} \) of column 2. Since the specification is reversible by definition, the MMD algorithm, in this example, transforms the output vector to the input vector by inserting the appropriate gates between the input and output endpoints. The algorithm first examines each input/output pair starting from the top row where, in this case, it detects a mismatch between the high bits of the input and output minterms \( a \neq A \). The algorithm inserts an inverter on the \( A \) line to remedy the mismatch, and in turn, inverts the \( A \) bit of all rows of the truth table (shaded bits in column 3). A corresponding inverter is inserted below column 3 to provide a graphical

---

**Figure 3-18** Demonstration of Miller, Maslov and Dueck (MMD) Algorithm for a 2-variable binary function. The first column ‘ab’ is the input vector and the second column ‘AB’ is the output vector. Gates are placed between the input and output to transform the output vector ‘AB’ to be a replica of the input vector ‘ab’ (column 1 = column 5). The resulting cascade of gates represents the synthesized circuit. Starting from the top minterm, in column 3, an inverter is placed on the ‘A’ qubit in order to transform the output minterm \( AB=10 \) to match the input minterm \( ab=00 \). The process is repeated for all minterm pairs until column 1 = column 5. The synthesized circuit is shown below the table.
demonstration of the circuit’s construction. Once the first row has been corrected; i.e.,
the input and output minterms become an exact match, the algorithm proceeds to examine
the next set of input/output minterms of the second row. Again, the algorithm detects a
mismatch, in the upper bit, between the input minterm (01) and the recently modified
output minterm (11). To correct this difference, however, the algorithm must use the
lower bit (1) as a control line (conditional variable) while inverting the upper bit. In
essence, a Feynman gate, also known as Controlled-Not gate, is inserted where bit (a) is
inverted only if bit (b) equals 1. The algorithm continues the process until the output
vector (last column) is an exact match to the input vector (ab) signaling that the synthesis
process is complete, and that a solution (circuit) has been built which implements the
initial specification. Using the control line is necessary to preserve the previously
completed synthesis of the first pair of minterms in compliance with the guiding principle
of MMD mentioned above.

3.6.2 Cycle Decomposition

The mathematical concept of cycle permutations can be applied in quantum logic
synthesis due to the fact that quantum specifications are bijective functions where the
minterms of output vector are a permutation of the minterms of the input vector. Using
this method, any reversible specification can be decomposed into a set of 2-cycle
transpositions and each 2-cycle can be further decomposed into an ordered set of
distance-one cycles.
Figure 3-19 Cycle decomposition of 3-bit binary function shown in (a) as a table, and (b) in a Karnaugh map where the function is represented by the 3-tuple cycle (0,1,2) which is equivalent to the cycles (1,2,0) and (2,0,1). The synthesis process decomposes each of the 3-tuple cycles into their equivalent 2-tuple cycles (P1, P2, P3) as shown in (b). A cascade of 2-tuple cycles is then substituted with the corresponding gate cascade as shown in (c), (d) and (e) and the solution with the least quantum cost is selected as the best solution.

Figure 3-19a shows a specification with a single 3-cycle (0,1,2) where 0→1, 1→2, 2→0 and all other minterms are self-mapping. The 3-cycle transposition can be decomposed to a set of 2-cycle transpositions as shown in the Karnaugh map of Figure 3-19b-e. In the data flow graph of Figure 3-19c for example, the 2-cycle permutation $P_1 = (0,1)(0,2)$ first swaps the location of the minterms 0 and 1, then swaps the location of 0 and 2. Tracing the input minterms 0, 1 and 2 shows that the product of the 2-cycle transpositions (0,1) and (0,2) represents the original cycle (0, 1, 2): hence, 0→1, 1→2, 2→0. Notice that this operation is not commutative, as the product (0,2)(0,1) is not equivalent to (0,1)(0,2). The lower half of Figure 3-19c shows the cascade of gates which implement the specification through the decomposed cycles (0,1)(0,2). Figure 3-19d-e shows two possible cycle decompositions, $P_2$ and $P_3$ which implement the same specification. Notice that the $P_1$ has less number of gates than the other two. The saving
in this case could be related to the lower hamming distance of cycles (0,1) and (0,2) (HD=1) as compared to the cycle (1,2) (HD=2) found in both $P_2$ and $P_3$.

Algorithms using cycle decomposition typically explore the various sets of 2-cycle decomposition and attempt to discover a sequence of cycle permutations with the least quantum cost. The problem, however, is that the number of different 2-cycle permutations increases exponentially as the number of variables increases which makes it extremely difficult to find solutions with better quantum cost than could be discovered by some of the other methods presented herein.

### 3.6.3 Hierarchical Diagrams

Following the classical approach to logic synthesis, the hierarchical diagrams can be used to decompose a quantum circuit specification as a set of binary equations into a set of sums or products of minterms. Some of the typical approaches to hierarchical decision diagrams are the BDD [57], positive and negative Davio expansions [57, 58, 59, 60], Cosine Sine Decomposition [61], and Quantum Multiple-Valued Decision Diagrams (QMDD) [62]. Such decompositions are typically portrayed as binary tree diagram which expands exponentially at each level of the decomposition. Similar to the cycle decomposition method above, algorithms using hierarchical diagrams are limited to few numbers of variables as the amount of resources required to store and compute such diagrams becomes enormous very quickly as shown in Figure 3-20. As a result, these methods are also limited to small number of variables.
The Reed Muller Reversible Logic Synthesis algorithm [60] starts from a set of equations rather than a truth table. A solution is found by recursively factorizing the set of equations in every possible way (at each node), until all inputs and outputs are exactly the same. The number of nodes for this method grows exponentially and requires huge amounts of resources (memory, computation, and time) and, as a result, is limited to small numbers of variables.

3.6.4 Linear Nearest Neighbor Model (LNNM)

Quantum architectures which adhere to the LNNM limit interaction amongst any two qubits to their nearest neighbor. For example, 3-21 shows the de facto embodiment of the well-known Toffoli gate as a cascade of five 2-qubit gates. The first V gate, however, allows remote interaction between the qubits $a$ and $c$, which, according to the LNNM architecture, is not permissible. To calculate the LNNM quantum cost of executing such a
gate, the information of one of the qubits (a in this example) have to be transferred to the location of qubit b with the aid of swap gates (represented by the three gate block). The V gate is applied between a and c, and then a is returned back to its original position.

Figure 3-21 The universal Toffoli gate is known to have a quantum cost of 5 which is the number of primitive two-qubit gates needed to implement (a); however, the first CV gate performs remote interaction between qubits ‘a’ and ‘c’ which violates the LNNM architecture. An LNNM compliant composition of the Toffoli gate is shown in (b) at a cost of 11 two-qubit primitive gates which is further reduced to 9 two-qubit gates as shown in section 8.2 below.

3.6.5 Measurement Quality of Reversible Gates

3.6.5.1 Quantum Cost

This is the most commonly used measure to evaluate the performance of a synthesis algorithm. Quantum cost is defined as the number of primitive single and two-qubit quantum gates needed to implement the reversible specification. Primitive gates include the single qubit NOT, V and $V^\dagger$; and the two qubit Controlled counterparts: CNOT, CV and $CV^\dagger$ gates. Some of the commonly used composite gates, such as the Multiple Control Toffoli (MCT) gates, swap gate, Miller gate, ...etc. are typically decomposed to
their primitive gates and the cumulative quantum cost of the primitive gates are used as the quantum cost of such composite gates.

The majority of the literature, however, uses an idealistic quantum cost assuming that any two qubits, regardless of other qubits between them, are able to interact, and in turn, use the same quantum cost for a CNOT which spans 100 qubits as the one which involves two neighboring qubits. As I will discuss later in this dissertation, technological constraints imposed by the technical implementation of quantum gates limit interaction between qubits to their nearest neighbor. I introduce in Chapter 8 and Chapter 9 a new set of MCT quantum costs which are compliant with the Linear Nearest Neighbor Model (LNNM) and reported such benchmark yardstick to the literature.

3.6.5.2 Ancilla Bits

A typical quantum specification of $n$ variables have the same number of inputs and outputs. Some quantum algorithms require the addition of new qubits to hold intermediate calculations which are used later in the construction of the synthesized circuit. Such additional bits are typically referred to as *garbage or ancilla bits*. Some resort to the addition of ancilla bits in an effort to reduce the amount of gates necessary to implement the specification, and in turn, result in better quantum cost.

It is desirable to keep the number of ancilla bits to a minimum because of the technical cost increases with the addition of each new bit – laser beams, detectors, …etc. Some specifications require the addition of ancilla bits, however, and cannot be avoided. For
example, incomplete specifications require the addition of ancilla bits in order to convert them to complete reversible specification suitable for quantum synthesis and computation.

As I have indicated in the previous section, remote interactions between qubits might not be feasible, and as such, algorithms that add ancilla bits and allow for interaction between distant qubits provide faulty measurement of their performance. As I explain in Chapter 8, forcing compliance on such algorithms result in a huge increase in their LNNM quantum cost compared to their reported quantum cost. In an effort to normalize comparison amongst diverse algorithms, I added a secondary measurement of performance in my LNNM proposal to measure the width of the circuit. The concept of ancillary ratio is defined as the rate of increase in the circuit width relative to the original set of variables should also be reported by algorithm designers in order to provide an even comparison amongst them.
PART II

RESEARCH OBJECTIVE #1

QUANTUM LOGIC SYNTHESIS

OF

BINARY SPECIFICATIONS
Chapter 4 MMDSN and MP algorithms

4.1 Introduction

Quantum circuits are a sub category of the general class of reversible circuits which are mathematically referred to as bijection where the set of output minterms (output vector) is simply a permutation of the input vector. Mathematically, a quantum gate (also applies to a full quantum circuit) is represented as a unitary matrix which, when applied to an input quantum state, produces the resulting output quantum state. The inherent reversibility of a quantum circuit implies that, if the output state is applied to the output end of a gate then the quantum gate will yield the original input quantum state. Similar to the universal NAND gate in classical logic, physicists have been able to define a set of universal reversible quantum gates which would serve as the building blocks of quantum circuits. For example, the equivalent representation of the classical NAND gate can be constructed from the quantum Toffoli gate. Unlike classical circuits, however, quantum circuits cannot fan in or out because, doing so, would violate the no-cloning principle and would clearly render a quantum circuit irreversible. Additionally, observation, sampling or measurement of a quantum circuit is prohibited during computation as measurement disrupts the internal state of the qubit and causes it to collapse, which as a result, forces the qubit to lose its quantum properties of superposition and entanglement.

In general, automated quantum logic synthesis starts from a specification, as a table or a set of equations, mapping each possible input and its corresponding output. A fully specified reversible mapping represents a reversible function which, algorithmically, are
considered prime (or ready) for automated synthesis. A partially specified function is inherently irreversible which typically requires the addition of ancillary (extra) variables to make it reversible before it would be suitable for automated synthesis. A synthesis algorithm typically constructs a cascade of quantum gates mapping each input minterm to its corresponding input minterm. It is of course imperative that at any point in the synthesis process that the specification remains reversible.

There are currently two types of algorithms to synthesize reversible circuits: (T1) those like MMD [63, 64, 65, 66, 67, 68, 56] that start from a reversible specification, (T2) those like [37, 69, 70, 71, 11, 72] that start from non-reversible specification and create ancilla bits. The second type of methods has been successful for large functions [37, 30, 73] but solves basically a different problem. The MMD algorithm [20] (Miller, Maslov and Dueck) is currently the leading reversible logic synthesizer if no ancilla bits are used. Mathematically, the problem is to decompose a large permutation of circuit’s specification to small permutations of reversible gates that are used. MMD uses the permutation vector-like reversible function specification as its input with an internal data structure which represents to a truth table which must be stored and processed in memory. Since it is intrinsically bound by the natural binary order of minterms, and hence does not use search, MMD cannot be enhanced through better search algorithms or iterative/recursive routines. Since MMD processes only a single order of the input vector, it is reasonably fast and it distinguishes itself among other programs of this type because it achieves (theoretical) 100% convergence regardless the problem size [20].
Practically, however, it can be applied to at most 8 qubit reversible functions and very few reversible functions with more than 8 variables were presented as MMD benchmarks in the literature. It was found in our research, and by other researchers, that the complexity of both the synthesis process and the average circuit sizes synthesized by MMD grow very quickly above 8 qubits, herein “large circuits”. In our research, it was difficult to evaluate the quality of our results for large circuits from reversible specifications chiefly due to the lack of a single solution for comparison. Consequently, with this research, I set the benchmark for future research. (Observe that standard non-reversible specifications are used in recent papers [74, 72, 30], and we need reversible functions such as specified by permutations). At the time MMD program was the established benchmark for the evaluation of programs for reversible circuit synthesis with no ancilla bits. A strong asset of the philosophy used in MMD, in contrast to those used in other programs is that MMD gives a warranty of convergence if the data is small enough for MMD to be able to keep them in memory. Due to the known fact that the quality of MMD may be very low for functions where the exact minimal solution is known, several research groups are constantly attempting to improve on the MMD algorithm. Agrawal and Jha’s algorithm [63, 37] uses the number of terms in the Positive Polarity Reed-Muller (PPRM) expansion of synthesized functions as its cost function. As PPRM can be stored by an expression that is shorter than 2^n, their algorithm could, in theory, minimize larger functions. On the other hand this algorithm has to store many PPRM equations as it represents a tree-search algorithm (discussed in 3.6.3: Hierarchical
Diagrams). Also, non-factorized PPRMs may be in many cases of similar complexity to truth tables, for instance for the function \( f = a'b'c'd' \). Some of the algorithm variants from [63, 64, 60] have trouble with convergence and there is a trade-off between provable convergence and size of circuits that can be minimized. A challenge thus still exists to create an algorithm that could trade-off quality for time, but with a provable convergence for every function. I will present such an algorithm in this chapter.

After many failed attempts at creating better minimizers based on other search strategies [75, 70, 71], I decided to improve MMD. The main weakness of MMD is that it is limited to functions of the size that their truth table (exponential size) can fit in memory. This limits practically MMD's approach to about 13 variables. Because of its design principle, even with big speed penalty MMD just cannot minimize larger functions. Thus an improved algorithm has to use an entirely different representation. When it was decided to use an internal representation other than a truth table or a spectrum with \( 2^n \) minterms, the problem was “what is the best representation that would still guarantee convergence?” Kerntopf [76] used a new type of decision diagrams but did not prove the convergence and, as a result, his method only worked for 3 variables. In some of our unpublished research I used ESOPs and FPRMs rather than PPRM but I was not able to find a heuristic that would work better than the variants from [63, 64]. Other cascade types have been also proposed in the newer versions of composition-based search approaches [67, 70, 71, 75] but there were troubles with either the size of solutions or convergence. Here I present a search method that is both convergent, allows for
synthesis of large functions, and produces near minimal solutions. This algorithm includes variants which are various generalizations of MMD.

4.2 Explanation of MMDs main idea

To make this chapter self-contained I give a brief overview of MMD. More can be found in [65, 68, 56]. The main idea of all algorithms for reversible circuit synthesis of type T1 is to transform bit-by-bit a reversible function to its identity function.

**Example 4-1:** Table 4-1 illustrates the basic flow of MMD algorithm. The first column lists all input minterms of the function in the natural numerical order (linear): 0, 1, 2, 3, etc. The second column in Table 4-1 lists values of the output vectors that correspond to the input vectors from the first column. For instance, the input minterm \( \overline{a} \overline{b} \overline{c} = 000 \) is mapped to the output minterm \( \overline{A} \overline{B} \overline{C} = 000 \) and input 001 is mapped to the output minterm 100. Self-mapping minterms are minterms with matching input and output values (e.g., minterm 000 above). The synthesis process applies successive gates to the output column (ABC), bit-by-bit, to generate the corresponding minterm of the input column (abc). Recall that Toffoli and Feynman gates are also self-inverse gates (\( M^{-1} = M \)), so they process information the same way from inputs to outputs and from outputs to inputs. The MMD algorithm shown here is thus the “backward searching” or “output to input searching” algorithm. Since the first minterm is self-mapping, MMD skips to the second minterm applying a controlled- Feynman gate to bit \( c \), shaded, conditional on bit \( a \) being set, underscored. After the application of each gate, the output

77
column minterms (of intermediate functions) become more and more similar to the first column – the column of input vectors. The question is “what does it mean to be more similar?” It is an advantage of general search methods that various measures of complexity or coincidence or similarity have been used [64, 76, 67, 71]. This may lead to better and faster solutions but it is hard or impossible to prove convergence. The MMD algorithm has however a very simple and working solution to this problem. It requires that intermediate columns remain exactly the same as the input column in some subset of rows from the top. The completed rows, start from row 0, then row 1, row 2 etc. up to the minterm under construction. When some subset of rows from top are completed, they are not allowed to change (shown in shaded areas in Table 4-1) which is guaranteed by the selection of proper control bits. The final circuit is shown in Figure 4-1.

Table 4-1 MMD method illustrated with truth tables of intermediate functions. Notation $a \rightarrow c$ means $c = c \oplus a$ which “inverts $c$ if $a=1$”. Control lines are underlined and shaded minterms are completed and should not be modified. The goal of the algorithm is to insert quantum gates in order to transform the input vector (column ABC) into the input vector (column abc = column 6). Starting from the top, the algorithm skips the first row since $abc = ABC$ and processes the second row where the output minterm ‘100’ is transformed into the input minterm ‘001’ by placing two conditional inverters on qubits ‘A’ and ‘C’. The process continues until column $abc = column 6$.

<table>
<thead>
<tr>
<th>abc</th>
<th>ABC</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>001</td>
<td>100</td>
<td>101</td>
<td>001</td>
<td>001</td>
<td>001</td>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>010</td>
<td>101</td>
<td>100</td>
<td>100</td>
<td>110</td>
<td>010</td>
<td>010</td>
<td>010</td>
</tr>
<tr>
<td>011</td>
<td>001</td>
<td>001</td>
<td>101</td>
<td>111</td>
<td>011</td>
<td>011</td>
<td>011</td>
</tr>
<tr>
<td>100</td>
<td>110</td>
<td>111</td>
<td>011</td>
<td>011</td>
<td>111</td>
<td>101</td>
<td>100</td>
</tr>
<tr>
<td>101</td>
<td>010</td>
<td>010</td>
<td>010</td>
<td>110</td>
<td>100</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>110</td>
<td>011</td>
<td>011</td>
<td>111</td>
<td>101</td>
<td>101</td>
<td>111</td>
<td>110</td>
</tr>
<tr>
<td>111</td>
<td>111</td>
<td>110</td>
<td>110</td>
<td>100</td>
<td>100</td>
<td>110</td>
<td>111</td>
</tr>
</tbody>
</table>

$a \rightarrow c$  $c \rightarrow a$  $a \rightarrow b$  $b \rightarrow a$  $a \rightarrow b$  $a \rightarrow c$
This is the main idea of MMD algorithm and actually the only algorithmic idea of this method (excluding templates). The proof that this algorithm is convergent is obvious, as every step creates one more bit in a row from top that is the same in the intermediate column as in the first column. This way, after at most \( n \times 2^n - 1 \) steps (intermediate columns) the last column becomes exactly the same as the first column, and thus, the remaining function to be realized is an identity function. Obviously, the strength of this algorithm is definite convergence, but since the complexity is exponential, MMD is limited in application to a small number of bits. So far, however, MMD continues to represent the benchmark to meet as no better algorithm had been proposed. The symbol \( a \rightarrow c \) in the column 1 means that whenever \( a = 1 \) in the previous column, the bit \( c \) is flipped from 0 to 1 and from 1 to 0. Hence, this transition from column to column executes the Toffoli gate \( c = c \oplus a \). The reader may check that the number of completed rows is either the same or larger from column to column. In this example the upper complexity bound is \( n \times 2^n - 1 \) which for our 3-bit example yields \( (3 \times 2^3 - 1) = 23 \) gates. Note that our example simulation resulted in only 6 gates, which in this case, MMD happened to produce good results. But there are examples [75] where the gate number is close to the upper bound although the known minimal number of gates is lower.
The main concept of MMD, the natural binary minterm ordering was challenged in [11] as the only 100% convergent order. It was found that MMD’s minterm ordering falls into a subset of orderings that do not exhibit certain important property that was called the “control line blocking”. This observation led to the creation of the “MMDS ordering” [11]. To make this chapter self-contained, all these ideas will be defined below but first I need to motivate the new concepts. Without any backtracking, any bi-directional search or any template matching, the MMDS ordering used exhaustively were superior for 3-bit circuits [11]. The MMDS orderings can be used with any number of inputs and have larger gains compared to MMD when the number of inputs increases. However, the number of MMDS orderings is too high to use all these orderings for synthesis. In this chapter, I introduce an algorithm which uses a subset of the MMDS ordering, herein MMDSN orderings, which greatly reduces the number of terms examined while providing near minimal solution superior to MMD.

Figure 4-1 The solution circuit found from MMD in Table 4-1, drawn and created from outputs to inputs. The arrow shows the flow of signal from inputs to outputs. This method is possible because each reversible gate used in this figure is its own self-inverse.

4.3 MMDS and MMDSN Orderings
MMD stipulates that the function is arranged in a natural binary code order by inputs assignments. Each iteration adds a gate in order to correctly transform the outputs to match the inputs without changing any of the previously completed (from top row) output minterms. Other innovative algorithms utilized greedy algorithm where gates are chosen to reduce the cost function from input to output. For example, Hamming Distance determines the choice of gates to transform the output function to the original function or to identity function. Such algorithms did not always converge, unlike, MMD, which, as it might give the worst solutions, it always converges. The question however is: How can these two main ideas of natural ordered search of MMD and greedy search can be combined to improve the quality of results and always achieve the convergence. Such combination is the goal of this chapter, part of which is discussed here.

The good ordering should not conflict with the main MMD’s idea [9, 56] of not changing any previously set outputs. This idea is also what guarantees MMD’s convergence.

**Definition 4-1:** Control Line Blocking condition occurs when all control lines of the current minterm are a subset of the control lines of a previously completed minterm in the input order.

When the condition of Definition 4-1 occurs it makes it impossible to change any output bits during the current iteration without altering the output bits which have been previously completed. Occurrence of this condition hinders convergence.
control_line_blocking := false;
for terms i=1..n
   for terms j=0..i-1
      if (term[i] & term[j]) == term[i] then
         control_line_blocking := true;
      end if
   end for
end for

Example 1: minterm 101 comes after 111.
Since (101 == 101 & 111) then control line blocking exists.

Example 2: minterm 101 comes after 011.
Since (101 <> 101 & 011) then no control line blocking exists.

Figure 4-2 Algorithm and examples of control line blocking detection

Therefore, any ordering of inputs that does not lead to the occurrence of the blocking condition can be used in an improved MMD algorithm. The method to find all non-blocking permutations for any number of inputs was found in [11]. No control line blocking seems to be a very restrictive rule. For a three-input function there are initially 8! (40,320) permutations but since the 000 and 111 are assigned to the first and last location respectively, the number of permutation reduces to 6! Using the software, 48 permutations, called MMDS orders, were found to exhibit no control line blocking for all 3*3 reversible functions – see Figure 4-2. Included in this set is, of course, the original MMD natural ordering.
Figure 4-3 (a) MMDSN orders are created by first constructing a Hasse diagram where each level corresponds to the sum of digits at that level. (b) shows all possible transitions which have a single bit difference between the two minterms in the transition. (c) Finally, an MMDSN order is constructed from the Hasse diagram where all the minterms of lower levels have to be processed before any terms in the succeeding level. In this example, the order of levels is ({0}, {4,1,2}, {5,3,6}, {7}) which yields the input vector of the following order of minterms {0,4,1,2,5,3,6,7}.

The binary vectors of cells (minterms) of a 3 * 3 reversible function can be represented as a well-known Hasse diagram [77], where a bit-by-bit domination relation (1 ≥ 0, 1 ≥ 1, 0 ≥ 0) is used as an ordering relation (see Figure 4-3a, b). While binary vectors are used in Figure 4-3a, Figure 4-3b uses natural numbers being counterparts of these binary vectors. The rule says "never to take a dominating node (number) before a dominated node". Thus 5 cannot be taken before 1, for instance. As we see, MMD order satisfies these rules. Another set of good orders are shown in Figure 4-3c and Figure 4-4.

As the number of input lines increases, the number of non-blocking orderings increases exponentially. For functions with four inputs, Stedman [11] reported that 78,880 different non-blocking permutations exist. I however discovered that 1,680,382 such non-blocking permutations exist. As the amount of non-blocking orders increase so does the optimality of the MMDS orderings, and as a result, the time required to synthesize. With MMDSN order, a set of rules were created to distill the best possible control choices from the set of all possible control line choices, as follows:
• The target bit cannot be used to control the current transformation,
• Use minimal number controls bits necessary to flip the target bit,
• No past outputs can be changed,
• Process 0 \( \rightarrow 1 \) transitions first to maximize availability of control lines, and hence, guarantee convergence.

The control possibilities are then sent to the gate choice function to produce a circuit. Currently gate choice is based on Hamming Distance but it can be any cost function [63, 75, 76, 67, 69]. Using control line blocking as the only rule, a subset of all input orders can easily be found, and it can be easily proven that all non-blocking input orders will converge for all output permutations.

![Diagram](image)

*Figure 4-4 A valid MMDS order \{0,2,1,3,4,6,5,7\} for MMD-like binary synthesis which is shown to be algorithmically convergent according to MMDS convergence rule. This algorithm is outside the subset that MMDSN algorithm creates since the minterm ‘3’ (on the third level) is taken before the minterm ‘4’ on the second level.*

**Theorem 4-1:** All non-blocking input orders converge for all output permutations.

1) **Proof of Convergence:**

Convergence is guaranteed in MMD and MMDS because at any given point in the algorithms all following output bits are able to be changed without altering any
previously set outputs. This is guaranteed because the input orders do not exhibit control line blocking. With MMD and MMDS’ methodical approaches, as long as all output bits can be changed without altering any previously set outputs these algorithms will converge every time.

MMDS set of orders is a superset of MMD’s. Our improved algorithm uses multiple MMDS input orders that exhibit no control line blocking. Included in these orders is the MMD natural binary order. MMDS ordering algorithm performs the same bit manipulating strategy for all non-blocking input orders, and reduces the circuit more than the standard MMD algorithm. This outcome is obvious, given that MMD is a subset of MMDS, so it can perform no worse than MMD.

**Definition 4-2:** MMDSN order is one in which the minterm 00…0 is generated first, followed by all minterms with a single one(1) in random order, followed by all minterms with two ones (1’s) in random order, and so on, successively incrementing the number of ones (1’s) in each band until we finally reach the minterm 11…1.

**Example 4-2:** for 3 variables: MMDSN order is for instance: 000, 100, 010, 001, 110, 101, 011, 111. This is also a MMDS order but not MMD order.

### 4.4 Multi-Pass Algorithm

Earlier attempts at improving MMD algorithm resulted in very good/minimal solutions for some circuits or non-converging/incorrect circuits for others [75, 76, 67,
Thus the order of selecting outputs to be covered by gates was found experimentally to be more important than the gate heuristics to choose gates. For larger number of variables, a variant of our algorithm was created based on the following principles:

(I) Rather than maintaining a set of tables mapping inputs to outputs, the algorithm creates these columns implicitly, simulating minterms one-by-one. The simulator uses the equations from the specification together with the part of the already constructed reversible circuit. To demonstrate the concept, imagine two circuits similar to Figure 4-1 cascaded back to back and simulated from inputs at each stage of minterm transformation. The first circuit, described by equations, represents the function under synthesis, and the second circuit is the outcome of synthesis (in reverse order of gates). When the synthesis process completes, two equivalent circuits, one mirror of another exist, where the first circuit is specified by equations, and the second by reversible gates (in reverse order of gates). When we simulate this composed circuit, for every input minterm, the same minterm is obtained at the outputs of the concatenated circuits, and hence, the concatenated circuits together are a reversible identity. Since the circuits mirror one another, the solution is represented by the second circuit of the concatenated whole.

(2) A number \(k\) of randomly selected MMDSN orders are generated which represent the function under synthesis. The solution with optimal cost is selected with the possibility of backtracking if the temporary cost exceeds the minimum cost determined earlier in the process.
(3) When possible, template-matching method from MMD is used on the result for post-processing to further improve the quantum cost.

For functions of four variables, I created a set of randomly generated four-bit reversible functions, AHP1-AHP50, and synthesized them using the original MMD, MMDS and our MMDSN orders. For MMDS and MMDSN, I tested the AHP functions against all possible permutations and calculated the minimum possible gate count as shown in Figure 4-5 and Table 4-2. It is evident that our selective order consistently produces superior results compared to the single MMD order for a negligible time penalty. Notice, however, that although the MMDSN order did not generate the optimal gate count generated by MMDS, the time advantage of MP is huge at 4 bits, and would be astronomical at greater number of bits. Even at higher number of bits, MMDSN order consistently produces better results than MMD within tolerable time. For example, at 9 bits, MMDSN was able to explore 100,000 solutions within 13 minutes. Although the current implementation utilizes parallel processing on an 8 core i7 processor, the algorithm is prime for massive parallelization in a cloud infrastructure, CUDA graphic processors or HPC supercomputers. Such capability would allow for synthesizing a selecting a larger iteration variant (k) and thus produce even larger circuits. The reader should note that in this study, neither MMD nor MP used local optimization techniques, e.g. template matching, which would ideally reduce the number of gates even further. Although MP would run even slower with template matching, its inclination to parallelization would easily minimize such an impact. An additional advantage of MP
approach is that we can have a trade-off – the longer we run the new combined algorithm the better is potentially our result. This property is missing in both MMD and Agrawal/Jha approaches.

Figure 4-5 Quantum cost comparision of MMDSN to both MMD and MMDS for 50 AHP functions which are randomly generated reversibe specifications. Each function has three points corrsponding to one of the algorithms. The dotted line represents a linear trendline which demonstrates that on average, the MMDS algorithm always gives the lowest quantum cost while the MMD yields the highest quantum cost. The MMDSN results in quantum costs that are close to the MMDS and much better than MMD.

4.5 Results of the MMDSN/MP for more than four Variables

Figure 4-6 shows the results with k=100,000 for functions up to 7 variables and k=10,000 for 8 and 9 bit variable functions. The algorithm was implemented on with multiple threads running on Intel i7 processor with 8 execution cores. The application allows the user to select the value of k to any value in order to trade-off between
synthesis time and improvement of quantum cost. For example, I selected a lower value of \( k=10,000 \) for 8 and 9 variable functions because each sequence has many minterms which require a long time to synthesize, and in turn we sacrifice quality of the solution in exchange for reasonable time. With the Multiple-Pass (MP) variant of the algorithm, I am able to synthesize functions of up to 30 variables which is not possible, at this time, using either MMD or MMDSN because of the amount of resources required by both algorithms \( (2^{30} \text{ rows in memory}) \). To understand the limitation of the MP algorithm for very large functions I created a sample reversible function, AHP30_1, of 30 variables [75], which was input as separate equation for each bit as it would require a huge amount of memory to represent as a truth table. The synthesis generated a quantum array of 4496 gates and took 2 hours and 45 minutes to complete. The function was a simple cascade of Toffoli gates where each variable controls its immediate successor. Our choice of a simplistic function, at this time, sets for us a foundation for future research.

<table>
<thead>
<tr>
<th># bits</th>
<th>function</th>
<th>MMD</th>
<th>MMDSN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time (ms)</td>
<td>avg quCost</td>
<td>time (s)</td>
</tr>
<tr>
<td>3</td>
<td>0.003</td>
<td>20</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>0.004</td>
<td>165</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>0.008</td>
<td>144</td>
<td>58</td>
</tr>
<tr>
<td>5</td>
<td>0.014</td>
<td>834</td>
<td>123</td>
</tr>
<tr>
<td>6</td>
<td>0.019</td>
<td>4208</td>
<td>452</td>
</tr>
<tr>
<td>6</td>
<td>0.023</td>
<td>576</td>
<td>562</td>
</tr>
<tr>
<td>7</td>
<td>0.033</td>
<td>15206</td>
<td>987</td>
</tr>
<tr>
<td>7</td>
<td>0.032</td>
<td>17201</td>
<td>1023</td>
</tr>
<tr>
<td>8</td>
<td>0.039</td>
<td>60655</td>
<td>2345</td>
</tr>
<tr>
<td>9</td>
<td>0.042</td>
<td>251478</td>
<td>3643</td>
</tr>
</tbody>
</table>

Figure 4-6 Comparison of the performance of MMD’s algorithm to the MMDSN algorithm described in this chapter. For MMD the single sample with natural binary order was processed. For MMDSN the average quantum cost of 5 runs over 100,000 samples for each run is reported along with the standard deviation of the samples and the percentage of deviation compared to the mean. The first improvement column reports the percentage improvement/degradation of MMDSN over MMD and the final column reports the final result of MMDSN which always considers the MMD sequence in the process.
4.6 **Analysis and Conclusion**

In this chapter I presented the MMDSN/MP algorithm for quantum logic synthesis of binary specifications of large number of variables. I stated earlier that the motivation of this algorithm attempts to optimize three conflicting objectives: 1) minimizing the amount of time required for synthesis; 2) maximizing the number of variables of the specification under synthesis while 3) reducing the quantum cost of the resulting circuit. While the MMD algorithm produces results very quickly because it only operates on the single natural order of the input vector, it does not produce the least quantum cost. MMDS on the other hand explores every possible solution and produces the best quantum cost, yet, it takes a very long time to synthesize and, consequently, is severely limited to six variables. By arranging the input vector according to the Hasse diagram, the MMDSN algorithm constructs a specific subset of the entire search space which is guaranteed to be algorithmically convergent and produces better results than MMD and within a tiny fraction of the time than MMDS takes and processes functions up to 14 variables. The MP variant of the MMDSN algorithm is capable of synthesizing even larger functions, up to 30 variables, by starting from a set of equations rather than a truth table.

In Figure 4-6 the MMDSN algorithm proved to be consistent when it ran multiple times for the same function. Examining the standard deviation, we notice that the results from the 5 runs yielded quantum cost close to one another and to the average quantum cost. The column labeled *percent deviation to mean* calculates the ratio of the standard
deviation to the mean to determine how far do the samples stretch away from the average quantum cost. Except for the *hwb5* function, all the results are below 5%. Notice that the percentage improvement (in the last two columns) illustrates the superiority of the MMDSN relative to MMD except for specifications larger than seven variables. Although 100,000 samples is a large number, it is a minuscule percentage of the solution space for functions of eight variables or higher. A larger number of samples is necessary in order to discover better solutions, which would, of course, require increasing longer time as the number of variables increase.

Finally, by using the Hasse diagram as the only foundation for constructing valid solutions, the MMDSN algorithm ignores a huge swath of the solution space which includes other convergent solutions. In the next chapter, I will examine an extension to the MMDSN algorithm which allows for exploring additional subsets of the search space which also consists of solutions which are guaranteed to converge.
Table 4-2 Comparison of MMDSN to both MMD and MMDS with respect to quantum cost and duration of synthesis

<table>
<thead>
<tr>
<th>Function</th>
<th>MMDSN</th>
<th></th>
<th>MMD</th>
<th></th>
<th>MMDS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Gates</td>
<td>Q-Cost</td>
<td>Time (ms)</td>
<td># Gates</td>
<td>Q-Cost</td>
</tr>
<tr>
<td>AHP-0</td>
<td>18</td>
<td>102</td>
<td>8.393</td>
<td>20</td>
<td>144</td>
</tr>
<tr>
<td>AHP-10</td>
<td>16</td>
<td>68</td>
<td>6.991</td>
<td>29</td>
<td>209</td>
</tr>
<tr>
<td>AHP-100</td>
<td>22</td>
<td>150</td>
<td>8.04</td>
<td>25</td>
<td>149</td>
</tr>
<tr>
<td>AHP-102</td>
<td>21</td>
<td>109</td>
<td>7.653</td>
<td>28</td>
<td>192</td>
</tr>
<tr>
<td>AHP-104</td>
<td>19</td>
<td>99</td>
<td>7.408</td>
<td>28</td>
<td>192</td>
</tr>
<tr>
<td>AHP-106</td>
<td>21</td>
<td>129</td>
<td>7.567</td>
<td>24</td>
<td>116</td>
</tr>
<tr>
<td>AHP-108</td>
<td>20</td>
<td>108</td>
<td>8.078</td>
<td>21</td>
<td>129</td>
</tr>
<tr>
<td>AHP-1000</td>
<td>16</td>
<td>80</td>
<td>7.497</td>
<td>19</td>
<td>111</td>
</tr>
<tr>
<td>AHP-1002</td>
<td>21</td>
<td>113</td>
<td>7.513</td>
<td>31</td>
<td>223</td>
</tr>
<tr>
<td>AHP-1004</td>
<td>20</td>
<td>136</td>
<td>7.056</td>
<td>23</td>
<td>167</td>
</tr>
<tr>
<td>AHP-1006</td>
<td>17</td>
<td>93</td>
<td>7.495</td>
<td>24</td>
<td>172</td>
</tr>
<tr>
<td>AHP-1008</td>
<td>19</td>
<td>95</td>
<td>6.682</td>
<td>31</td>
<td>215</td>
</tr>
<tr>
<td>AHP-1010</td>
<td>18</td>
<td>74</td>
<td>6.953</td>
<td>30</td>
<td>230</td>
</tr>
<tr>
<td>AHP-1012</td>
<td>23</td>
<td>131</td>
<td>7.146</td>
<td>28</td>
<td>168</td>
</tr>
<tr>
<td>AHP-1014</td>
<td>23</td>
<td>139</td>
<td>8.069</td>
<td>27</td>
<td>179</td>
</tr>
<tr>
<td>AHP-1016</td>
<td>18</td>
<td>126</td>
<td>6.748</td>
<td>23</td>
<td>167</td>
</tr>
<tr>
<td>AHP-1018</td>
<td>17</td>
<td>105</td>
<td>6.939</td>
<td>25</td>
<td>197</td>
</tr>
<tr>
<td>AHP-1020</td>
<td>18</td>
<td>106</td>
<td>7.317</td>
<td>25</td>
<td>193</td>
</tr>
<tr>
<td>AHP-1022</td>
<td>19</td>
<td>111</td>
<td>7.697</td>
<td>24</td>
<td>156</td>
</tr>
<tr>
<td>AHP-1024</td>
<td>22</td>
<td>138</td>
<td>6.622</td>
<td>30</td>
<td>218</td>
</tr>
<tr>
<td>AHP-1026</td>
<td>14</td>
<td>66</td>
<td>7.252</td>
<td>17</td>
<td>113</td>
</tr>
<tr>
<td>AHP-1028</td>
<td>14</td>
<td>86</td>
<td>7.343</td>
<td>20</td>
<td>148</td>
</tr>
<tr>
<td>AHP-1030</td>
<td>21</td>
<td>137</td>
<td>7.776</td>
<td>27</td>
<td>167</td>
</tr>
<tr>
<td>AHP-1032</td>
<td>20</td>
<td>108</td>
<td>6.726</td>
<td>27</td>
<td>187</td>
</tr>
<tr>
<td>AHP-1034</td>
<td>19</td>
<td>123</td>
<td>7.132</td>
<td>22</td>
<td>138</td>
</tr>
<tr>
<td>AHP-1036</td>
<td>19</td>
<td>107</td>
<td>7.257</td>
<td>26</td>
<td>186</td>
</tr>
<tr>
<td>AHP-1038</td>
<td>18</td>
<td>106</td>
<td>7.927</td>
<td>18</td>
<td>106</td>
</tr>
<tr>
<td>AHP-1040</td>
<td>16</td>
<td>96</td>
<td>6.478</td>
<td>22</td>
<td>174</td>
</tr>
<tr>
<td>AHP-1042</td>
<td>22</td>
<td>146</td>
<td>7.263</td>
<td>25</td>
<td>173</td>
</tr>
<tr>
<td>AHP-1044</td>
<td>19</td>
<td>107</td>
<td>7.325</td>
<td>23</td>
<td>159</td>
</tr>
<tr>
<td>AHP-1046</td>
<td>19</td>
<td>107</td>
<td>7.739</td>
<td>23</td>
<td>147</td>
</tr>
<tr>
<td>AHP-1048</td>
<td>18</td>
<td>94</td>
<td>6.484</td>
<td>20</td>
<td>120</td>
</tr>
<tr>
<td>AHP-1050</td>
<td>23</td>
<td>123</td>
<td>7.325</td>
<td>34</td>
<td>230</td>
</tr>
<tr>
<td>AHP-1052</td>
<td>18</td>
<td>110</td>
<td>7.557</td>
<td>26</td>
<td>166</td>
</tr>
<tr>
<td>AHP-1068</td>
<td>22</td>
<td>134</td>
<td>8.606</td>
<td>21</td>
<td>97</td>
</tr>
<tr>
<td>AHP-1070</td>
<td>18</td>
<td>106</td>
<td>7.707</td>
<td>22</td>
<td>158</td>
</tr>
<tr>
<td>AHP-1072</td>
<td>20</td>
<td>112</td>
<td>7.611</td>
<td>23</td>
<td>159</td>
</tr>
<tr>
<td>AHP-1074</td>
<td>22</td>
<td>126</td>
<td>8.236</td>
<td>26</td>
<td>194</td>
</tr>
<tr>
<td>AHP-1076</td>
<td>21</td>
<td>121</td>
<td>8.644</td>
<td>28</td>
<td>184</td>
</tr>
<tr>
<td>AHP-1078</td>
<td>21</td>
<td>105</td>
<td>7.69</td>
<td>30</td>
<td>222</td>
</tr>
<tr>
<td>AHP-1080</td>
<td>21</td>
<td>145</td>
<td>7.879</td>
<td>21</td>
<td>109</td>
</tr>
<tr>
<td>AHP-1082</td>
<td>21</td>
<td>133</td>
<td>8.109</td>
<td>29</td>
<td>233</td>
</tr>
<tr>
<td>AHP-1084</td>
<td>23</td>
<td>119</td>
<td>8.797</td>
<td>31</td>
<td>187</td>
</tr>
<tr>
<td>AHP-1086</td>
<td>20</td>
<td>116</td>
<td>7.367</td>
<td>27</td>
<td>195</td>
</tr>
</tbody>
</table>
Chapter 5 Covered Set Partitions

5.1 Introduction

The algorithm developed by Miller, Maslov and Dueck (MMD) [20] led the way in logic synthesis for reversible functions which requires no additional ancillary bits. Stedman and Perkowski [26] presented an algorithm capable of producing circuits with lower number of gates by exploring permutations of input vector ordering other than the natural ordering used by MMD. Stedman’s method however stalled at large number of variables as it required an exorbitant amount of time to compute. Alhagi, Hawash and Perkowski [21, 22] followed up with a synthesis method which explores a subset of Stedman’s orderings that produce near optimal circuits within a reasonable amount of time. Mathematically, the problem of reversible function synthesis is the decomposition of large permutation of circuit’s specification to small permutations of reversible gates. MMD uses the vector-like reversible function specification as its input which corresponds to a truth table that is explicitly used in the synthesis process and, thus, is stored and processed in memory. Additionally, since it is intrinsically bound by the natural binary order of input terms, MMD’s algorithm does not utilize any search facility, and hence, cannot be enhanced through better search or iterative/recursive routines. On the positive side, MMD is reasonably fast to synthesize the function since it only processes a single ordering of the input vector, the natural ordering. In addition, MMD distinguishes itself among other methods of this type because it achieves (theoretical) 100% convergence regardless the problem size [56]. Practically speaking, however,
MMD is limited to functions of at most 8 variables due to the limitation of memory resources as clearly evident by the limited set of benchmarks presented in the literature. Clearly, MMD set the benchmark functions for small reversible functions with no ancillary bits. However, I discovered through our research that the complexity of both the synthesis process and the average circuit sizes synthesized by MMD grow very quickly above 8 qubits, herein “large circuits”. Due to the known fact that the quality of MMD may be very low for functions where the exact minimal solution is known, several research groups have constantly attempted to improve on the MMD algorithm.

Agrawal and Jha’s algorithm [63, 60] uses the number of terms in the Positive Polarity Reed-Muller (PPRM) expansion of synthesized functions as its cost function. As PPRM can be stored by an expression that is shorter than $2^n$, their algorithm could, in theory, minimize larger functions. On the other hand this algorithm has to store many PPRM equations as it represents a tree-search algorithm. Also, non-factorized PPRMs may be in many cases of similar complexity to truth tables which imposes the same resource constraints as MMD. Additionally, some variants of the algorithm [63, 64, 60] have trouble with convergence where a trade-off is stipulated between provable convergence and size of circuits that can be minimized. Although Stedman improved on the results of MMD by synthesizing all permutations of the input vector to produce optimal results, Stedman’s approach hits the ceiling for a relatively small number of bits (5) due to the time it takes to synthesize all permutations of the input vector. Alhagi, et al [21] presented the best challenge to both MMD and Stedman through synthesis of an
orchestrated subset of Stedman’s input orderings which consistently produced equal or better results than MMD and completed in a short amount of time while able to synthesize considerably large functions (30-bits). This chapter extends our effort from the previous chapter by combining the mathematical concept of covering sets with Hasse diagrams to construct new set of input vector orderings, still a subset of Stedman’s, which would produce lower cost circuits than all previous efforts, yet still able to complete within a reasonable amount of time.

5.2 MMD Style Algorithms

In their paper, *A Transformation Based Algorithm for Reversible Logic Synthesis*, Miller, et al. [56] outlined a simple, yet powerful, synthesis method of reversible circuits (herein MMD). This algorithm observes a simple, yet essential, guiding principle stating that: *A completely mapped pair can never be altered by succeeding mapping calculations*. This important rule allows MMD to always converge which is an essential principle for synthesizing arbitrary reversible circuits. Some definitions are in order before I illustrate the algorithm with an example.

**Definition 5-1:** An n-variable mapping specification is a set of n variable input/output pairs, typically represented as a table, indicating the required functionality of a logic circuit or algorithm.

**Definition 5-2:** An n-variable input/output pair describes the expected output sequence for its corresponding input sequence.
**Definition 5-3:** A completely mapped pair is a pair where, at some point in the logic synthesis, a set of logic gates have been specified to map its n-variable input to its corresponding n-variable output.

**Definition 5-4:** A self-mapping pair is a pair whose n-variable input sequence is equal to its corresponding output sequence.

Synthesizing by brute force, the MMD algorithm starts with a mapping specification of a fully specified reversible function and creates a cascade of primitive reversible gates to map all input/output pairs. Figure 5-1 shows a mapping specification of a two-variable function where the inputs are designated with \((ab)\) and the outputs with \((AB)\). The algorithm synthesizes the function as follows:

- Considering the inherent reversibility of the function, the algorithm starts synthesis from the output column \((AB)\) towards the input column \((ab)\).
- Starting with the first pair \((00 \rightarrow 10)\), the MMD algorithm realizes that an inverter on line \((a)\) would correctly map the 00 input to the 10 output. Essentially, any value presented on the \((A)\) line will be inverted, as shown in the bolded text of the third column. At this stage, the first input/output pair is completely mapped, and according to the guiding principle mentioned above, such a pair should never be modified by later transformations.
- In order to observe such a rule, the algorithm uses control lines for all subsequent synthesis as shown in the last two columns. Row two of the third
column shows the pair \((01 \rightarrow 11)\) which requires an inverter on the \((A)\) line with line \((B)\) as a control line – shaded. As a result, only the bolded digits of second and third rows are affected.

- Similarly, the third pair \((10 \rightarrow 11)\) is synthesized with an inverter on line \((B)\) which is controlled by a value of one \((1)\) on line \((A)\).
- At this stage, the algorithm realizes that the mapping circuit is complete as the first and last columns are both identical.

\[
\begin{array}{c|c|c|c|c}
ab & AB & \Phi & \Phi & \Phi \\
00 & 10 & 00 & 00 & 00 \\
01 & 01 & 11 & 01 & 01 \\
10 & 11 & 01 & 11 & 10 \\
11 & 00 & 10 & 10 & 11 \\
\end{array}
\]

Inspired by MMD, Stedman, et al. [11], were able to successfully synthesize the same functions used by MMD and yield circuits with less number of gates. Stedman, et al, questioned a limitation of the MMD algorithm requiring the use of the natural ordering for the input sequence in the mapping specification – column \((ab)\) in Figure 5-1. By analyzing all permutations of input/output pairs, Stedman, et al, realized that a finite set
of input sequences suffers from control line blocking which results in a violation of MMD's guiding principle: *never altering previously completed mapped pairs*. Such blocking sequences never converge. Figure 5-2 shows an input sequence in column (ab) which illustrates control line blocking where the first pair is easily synthesized with an inverter on line (B). However, the second pair (01 → 10) requires inverting the same output bit (B) from 0 to 1 in order to match the (b) bit of input term (01). Using (A=1) as a control line, however, would surely violate MMDs aforementioned guiding principle by altering the first *completely mapped pair* (11) back to (10). Consequently, this input sequence in figure 2 suffers from control line blocking and is rejected for it will never converge.

![Figure 5-2](image)

*Figure 5-2 Row 2 illustrates control line blocking where the output minterm ‘10’ is to be mapped to the input minterm ‘01’. Following the MMD method of synthesis, I can invert the lower bit of ‘10’ while using the upper bit for control. Had I done so, however, would alter the completed minterm in the first row ‘11’ to ‘10’ (effectively restoring the original state of the first row. Altering any completed minterm is a violation of the MMD algorithm which indicates that the algorithm is blocked and will never converge.*

Stedman, et al, were able to distill their discovery of blocking sequences to the following definition:

**Definition 5-5:** Control Line Blocking condition occurs when all control lines of the current minterm are a subset of the control lines of a previously completed minterm for a given input order.
Stedman, et al, condensed the process of detecting control line blocking of any input order to the following algorithm:

```
for terms i=1..n
    for terms j=0..i-1
        if (term[i] & term[j]) = term[i]) then
            Control Line Blocking Detected;
        end if
    end for
end for
```

Armed with such clear formula, Stedman, et al, launched a laborious quest for the optimal circuit by exhaustively synthesizing the input/output function using all valid, i.e. non-blocking, input sequences (herein, MMDS orders). For a given input/output specification, the algorithm tests all permutations of input orderings for valid sequences which are then synthesized into a set of gates. The circuit with the minimum quantum gate cost is selected as the result of the algorithm. Clearly this algorithm was always able to find solutions equal to or better than the MMD algorithm could. A mathematician trained in ordered sets would quickly realize that the concept harnessed by MMDS derives from the covering graph theory where, for convergent input orderings, earlier terms in the ordering are covered by later terms. The covering concept is enforced by the AND (\&) operation in the algorithm testing that, for convergent inputs orderings, the pattern of \textit{ones} of all completely mapped pairs are a subset of the \textit{ones} of the term under synthesis.
In the previous chapter, and in Alhagi, et al. [21], I demonstrated the use of covering graph concept to devise a mechanism for generating valid orderings. I realized that as the number of input variables of a specification increased, the level of computations for the MMDS algorithm swelled exponentially rendering it impractically slow. In the MMDSN algorithm, I focused on creating a subset of the MMDS orderings, which trades off optimality of the resultant circuit for reasonability of the time required for synthesis. Consequently, rather than tediously discovering and rejecting blocking sequences, the authors opted for creating a process to systematically build a subset of MMDS’ input orderings which assures convergence. Relying on the covering graph theory, I discovered a process to mechanically create a subset of the valid MMDS orderings by arranging the input terms in a Hasse diagram.

**Definition 5-6:** a Hasse diagram is a type of mathematical diagram used to represent a finite partially ordered set, in the form of a graph where, for the relation \{(x,y) \mid x \leq y \mid x,y \in S\}, each element of S is a vertex in the plane and draws a line segment or curve that goes upward from x to y whenever y covers.

Figure 5-3 Construction of Binary Hasse Diagram for any 3-bit function starts with the minterm with all zeros first in the first level (level 0). Every subsequent level includes all minterms which have the same sum of their digits. Level 1, for example, has the minterms \{001, 010, 100\} since the digits of each minterm add up to ‘1’. The last minterm has all digits set to ‘1’.

100
Figure 5-3 (a,b) illustrates the process of creating a Hasse diagram for a 3-bit sequence which is then used to create an MMDS compliant ordering as shown in Figure 5-3c. As demonstrated in the last chapter, I used the following procedure to create the MMDSN sequences:

- Create a Hasse diagram by – Figure 5-3:
  - Starting with the base term consisting of all zeros,
  - Draw a line from above term to new terms constructed by placing a one for each zero in the base term, (the covering property),
  - Repeat step (b) for each new term to construct a new layer of the diagram.
  - Stop when the term consisting of all ones is reached.
- Generate an MMDSN ordering by – Figure 5-4:
  - Start from the base level of the Hasse diagram which consists of the base term of all zeros,
  - Randomly permute terms of the next level consisting of a single one and place them at the end of the ordering [e.g., 4,1,2 in Figure 5-4 (c)],
  - Repeat step (b) for each level afterwards where each level includes all terms consisting of an additional one compared to the level before it [e.g., 5,3,6 in Figure 5-4 (c)],
  - Place the last term of all ones at the end of the ordering,
  - Resulting in the valid input sequence {0,4,1,2,5,3,6,7}.
The reader can easily surmise that sequences arranged in a Hasse diagram are a subset of the Stedman set of orderings, and, as a result, are convergent. The ultimate benefit of this algorithm is a circuit with less quantum gates than produced by MMD. While not as optimal as MMDS, this algorithm always completes in real time for circuits with eight variables or less, and a reasonably tolerable time for larger circuits. For a small number of variables, it is possible to compute all permutations at each level and discover the synthesis that produces the most optimal circuit with an MMD type of an algorithm. Notice that the random aspect of step (2.b) above becomes necessary as the number of variables and computation time increases, and as a result, the number of orderings is limited to a user selected upper bound \(k\) of randomly created MMDSN orderings and select the best result out of them. In summary, MMDSN devised a method of constructing a set of input term orderings which are guaranteed to converge and then synthesizing a subset of these orderings to discover the best solution out of the subset. Contrarily, MMDS had to grudgingly try each permutation of the input orderings and synthesize the ones which would converge before discovering the optimal solution.
5.3 Partial Covering Set Partitions

I realized that a partitioning of the original set of input vector permutations while maintaining properties of a covering graph could prove fruitful. I realized that the MMDSN algorithm could be sidestepping some of the Steadman search space and wanted to construct another structure which is capable of complementing MMDSN and exploring other portions of the MMDS search space; and hence, the Covered Set Partition algorithm (CSP). Figure 5-5 shows an illustration of covering set partitions for 3 and 4 bit variables. The lower half of the graph represents the lower partition of a four bit functions where the highest bit $3 = 0$, and the upper half is for the partition where $bit3 = 1$. Considering a three bit function, represented by one of the cubes, the lower plane represents the partition where the highest $bit2 = 0$, and the upper plane represents the partition where $bit2 = 1$.

In the case of the four bit function, notice that it is possible to partition on the upper bit only, or the upper two bits and process the remaining lower bits with a Hasse diagram. For example, a one bit partitioning model would split the sequence into the two halves shown and construct two Hasse diagrams for each of the remaining lower three bits, as shown by the diagonal dashed lines, following the same process as MMDSN. Using the two upper bits for partitioning, would yield four partitions with the lowest shaded plane where $(b_3b_2=00)$ followed by lower middle plane $(b_3b_2=01)$ followed by the upper middle plane $(b_3b_2=10)$ and lastly the upper plane $(b_3b_2=11)$.
The input sequence of a four variable \( (B_3B_2B_1B_0) \) can be partitioned in multiple ways. Using the upper variable \( (B_3) \) only would create two partitions (visually represented as the two cubes) and the remaining bits \( (B_2B_1B_0) \) are arranged according to a Hasse diagram (the levels of the Hasse diagram are shown as diagonal lines – described in earlier sections). Using the upper two bits \( (B_3B_2) \) for partitioning would create 4 partitions (shown as the four planes) where the remaining two bits of each partition would also be arranged according to the Hasse structure. A CSP input sequence is then constructed by arranging the minterms according to the natural order of the bits forming the partition then for each partition the remaining bits are arranged according to the MMDSN sequence described in the previous chapter.

5.4 **CSP Algorithm**

The following process outlines the steps for creating CSP sequences for an \( n \)-variable function using the \( p \) upper bits for partition:

- Create \( k=2^p \) partitions where \( p \) is the number of upper bits used with partitions \( j=0...k-1 \).
- To construct an input sequence, start from the lowest partition to the highest partition where, for each partition, use the lower \( (n - p) \) bits to create a Hasse diagram.
• For each Hasse diagram, construct and MMDSN order as follows:
  
  a. Start from the base level of the Hasse diagram consisting of all zeros,
  
  b. Randomly permute terms of the next band consisting of single ones and
     place them at the end of the ordering,
  
  c. Repeat step (b) for each band that follows in consecutive order, where
     each band has an additional one compared to the band before it,
  
  d. Place the last term consisting of all ones at the end of the sequence.

5.5 Experimental Results

To study the impact that the partition size has on the quantum cost of the synthesized
  circuit, I experimented with synthesizing all possible partition sizes for 16 benchmark
  functions of different number of variables and reported the results in
Table 5-1. I selected 16 different functions ranging from 5 variables to 13 variables, where for each function of \( n \) variables, I constructed sequences for partition sizes from zero to \( n-1 \), and for each partition size I repeated the synthesis five times for 100,000 samples each. Consequently, for a function of \( n \) variables, the number of solutions visited is \( 500,000n \) solutions.

![Figure 5-6 Normalized quantum cost for 16 benchmark functions ranging from 5 to 13 variables. In order to visualize the outcome for functions of different number of variables, I scaled the quantum cost for all functions to the range between 0 and 100% (y-axis) and scaled the partitions size between 0 and 1 (x-axis). The normalized quantum cost for functions of the same number of variables were averaged and plotted here. Notice that the lowest quantum cost (normalized) for all functions is found for normalized partitions size between 0.4 and 0.6; hence, the best quantum costs are for partition sizes which are close to the half-way point relative to the number of variables. Notice the trendlines clearly show the minimums for each dataset are close to the middle partition.](image)

Figure 6-6 shows a plot of the normalized quantum cost for the sixteen functions against the normalized partition size. For each individual function I first scaled the quantum cost to 100% and scaled the partition size to be between 0 and 1. I then grouped the functions by the number of variables and averaged the normalized quantum cost for
each *normalized partitions size* and plotted them on the graph. Notice that for each function of $n$ variables, there are $n$ points on the graph and that, for the 12 and 13 variable functions, only the first two points are visible since their normalized quantum cost drops quickly below 50% - I will discuss these functions shortly. The graph clearly shows that for all functions below 12 bits, the normalized quantum cost is lower around the midpoint of the normalized partition size. Figure 5-7 demonstrates this finding where it plots the normalized partition size where the best quantum cost occurred for the 16 functions. These points represent the lowest points of Figure 6-6 which, according to Figure 5-7, the best quantum cost occurs when the *normalized partition size* is between 0.4 and 0.6 – excluding functions with 12 or more variables.

![Graph showing normalized partition size and best quantum cost](image)

*Figure 5-7 The graph plots the normalized partition size where the best normalized quantum cost was discovered from Table 5-1, in which, the partition size and best average quantum cost are shaded in gray. Notice that for functions of 11 variables or less, the best quantum cost occurred around the midpoint of the normalized partition size (between 0.4 and 0.6) which indicates that for functions of this number of variables, a partition size in the vicinity of $\frac{1}{2}$ the number of variables typically yields the best results.*
5.6 Analysis and Conclusion

In this chapter I introduced the Covered Set Partitions (CSP) algorithm which builds on our previous work with the MMDSN algorithm and extends its reach to explore more patches of the solution space which are outside the range of MMDSN and MMD. I maintain the same objective of MMDSN where I continue to explore quantum logic synthesis for binary reversible functions of large number of variables. Our motivation remains consistent with the previous chapter where I attempt to optimize three conflicting objectives (time, quantum cost, number of variables). Compared to the MMDSN algorithm, however, the CSP algorithm sacrifices more time in favor of discovering better quantum costs. For an $n$ variable function, the CSP will take, on average, $n$ times more time to complete as it repeats the synthesis process for each partitions size ($n$ partitions). However, based on the observations above where the best quantum cost occurs around the midpoint, a possible optimization is to explore a limited number of partitions sizes which are centered around the mid-point of the number of variables. For functions of less than 12 variables, the CSP has discovered the best quantum cost in the region where the normalized partitions size is between 0.4 and 0.6. Assuming we chose $k < n$ partitions to explore, the total time would be $k$ times greater than MMDSN with a high probability of discovering better solutions than MMDSN or MMD.

The two end points of the graph where the partitions size = 0 or $n$ represent the MMDSN and MMD algorithms respectively. When the partitions size is zero, the Hasse diagram is used to construct the sequence on all the variables of the input vector, which is
exactly what the MMDSN algorithm does – see previous chapter for details. On the other extreme, when the partitions size equals to \( n-1 \), the entire input vector would be arranged according to the natural order which is exactly what the MMD algorithm does. Consequently, the two algorithms are a subset of the CSP algorithm presented in this chapter.

![Graph](image)

*Figure 5-8 Results for functions of 12 and 13 variables. Similar to the other functions, the midpoints of the partition size appears to have the best normalized quantum cost. The left edge represents the MMDSN algorithm which, in this case, suffers greatly in discovering the quantum cost.*

For functions of 12 variables or higher, the CSP algorithm also discovers lower quantum costs than on the edges, however, the left edge which represents the MMDSN algorithm, partitions size = 0, suffers the most as seen if Figure 5-8. When you examine these functions in Table 5-1, you realize that partitions sizes around the midpoint still gives low quantum cost, but not always the lowest. For the 12 bit function \( \text{plus63mod4096} \), for example, the best quantum cost, 686522.2, occurs at partitions size
10, however, at partition size 6, the quantum cost of 713982.2 is only a 4% higher than the best quantum cost discovered. The 13 bit function \textit{plus127mod8192}, however, has a higher percentage difference at the midpoint when compared to the best quantum cost.

For partition size of size \textit{zero}, the MMDSN case, the subset of the search space, which is explored by the algorithm, is considerably huge so that the random algorithm has difficulty discovering low quantum costs. The results are consistent over multiple runs, which indicates that a partition size of zero is not the best place to start when exploring functions of large number of variables. When the search space is partitioned into smaller areas, the CSP algorithm had better success in discovering lower quantum costs, since the algorithm is exploring a smaller subset of solutions for each partition. This discovery has been consistent for all functions, and the fact that partition sizes around the midpoint of the size of the function is the best place to start the hunt.
Table 5-1 Results of synthesizing 16 functions of different number of variable for every possible partitions size for the specific function. Each table includes the average quantum cost over 5 runs, along with the standard deviation for the runs and the ratio of the standard deviation to the mean (a measure of variance in results). The majority of results show consistency between runs with a variance below 3% while few exhibited higher error around 9%. The shaded number represents the partition size with the lowest average quantum cost. For functions below 12 bits, all functions witnessed the best quantum cost where the partition size is around half the number of variables.

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>517.20</td>
<td>43.02</td>
<td>8.32</td>
</tr>
<tr>
<td>1</td>
<td>480.00</td>
<td>30.14</td>
<td>6.28</td>
</tr>
<tr>
<td>2</td>
<td>465.60</td>
<td>28.65</td>
<td>6.15</td>
</tr>
<tr>
<td>3</td>
<td>647.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>834.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**hwb5 (5 bits)**

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2832.00</td>
<td>109.14</td>
<td>3.85</td>
</tr>
<tr>
<td>1</td>
<td>2786.80</td>
<td>64.12</td>
<td>2.30</td>
</tr>
<tr>
<td>2</td>
<td>2483.40</td>
<td>220.61</td>
<td>8.88</td>
</tr>
<tr>
<td>3</td>
<td>2557.40</td>
<td>86.15</td>
<td>3.37</td>
</tr>
<tr>
<td>4</td>
<td>2698.00</td>
<td>51.43</td>
<td>1.91</td>
</tr>
<tr>
<td>5</td>
<td>4208.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**hwb6 (6 bits)**

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>575.60</td>
<td>8.96</td>
<td>1.67</td>
</tr>
<tr>
<td>1</td>
<td>573.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>563.40</td>
<td>20.85</td>
<td>3.70</td>
</tr>
<tr>
<td>3</td>
<td>602.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>576.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>576.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**mod5addr (6 bits)**

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14105.60</td>
<td>409.50</td>
<td>2.90</td>
</tr>
<tr>
<td>1</td>
<td>13423.60</td>
<td>367.42</td>
<td>2.74</td>
</tr>
<tr>
<td>2</td>
<td>12567.60</td>
<td>374.13</td>
<td>2.98</td>
</tr>
<tr>
<td>3</td>
<td>12648.00</td>
<td>218.45</td>
<td>1.73</td>
</tr>
<tr>
<td>4</td>
<td>12271.40</td>
<td>391.41</td>
<td>3.19</td>
</tr>
<tr>
<td>5</td>
<td>12574.20</td>
<td>240.06</td>
<td>1.91</td>
</tr>
<tr>
<td>6</td>
<td>15206.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**hwb7 (7 bits)**

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>13874.60</td>
<td>250.96</td>
<td>1.81</td>
</tr>
<tr>
<td>1</td>
<td>12851.20</td>
<td>111.62</td>
<td>0.87</td>
</tr>
<tr>
<td>2</td>
<td>11939.80</td>
<td>327.81</td>
<td>2.75</td>
</tr>
<tr>
<td>3</td>
<td>11749.20</td>
<td>307.94</td>
<td>2.62</td>
</tr>
<tr>
<td>4</td>
<td>12315.80</td>
<td>393.93</td>
<td>3.20</td>
</tr>
<tr>
<td>5</td>
<td>12486.60</td>
<td>149.29</td>
<td>1.20</td>
</tr>
<tr>
<td>6</td>
<td>17201.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**ham7 (7 bits)**
<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>59769.00</td>
<td>1489.74</td>
<td>2.49</td>
</tr>
<tr>
<td>1</td>
<td>56143.60</td>
<td>492.84</td>
<td>0.88</td>
</tr>
<tr>
<td>2</td>
<td>54689.60</td>
<td>652.62</td>
<td>1.19</td>
</tr>
<tr>
<td>3</td>
<td>52522.20</td>
<td>254.56</td>
<td>0.48</td>
</tr>
<tr>
<td>4</td>
<td>51714.60</td>
<td>831.78</td>
<td>1.61</td>
</tr>
<tr>
<td>5</td>
<td>51910.60</td>
<td>600.00</td>
<td>1.16</td>
</tr>
<tr>
<td>6</td>
<td>53287.60</td>
<td>768.40</td>
<td>1.44</td>
</tr>
<tr>
<td>7</td>
<td>66142.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**urf2 (8 bits)**

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16847.00</td>
<td>636.93</td>
<td>3.78</td>
</tr>
<tr>
<td>1</td>
<td>17100.40</td>
<td>997.13</td>
<td>5.83</td>
</tr>
<tr>
<td>2</td>
<td>15668.60</td>
<td>833.24</td>
<td>5.32</td>
</tr>
<tr>
<td>3</td>
<td>14166.60</td>
<td>841.58</td>
<td>5.94</td>
</tr>
<tr>
<td>4</td>
<td>14532.20</td>
<td>376.44</td>
<td>2.59</td>
</tr>
<tr>
<td>5</td>
<td>14734.00</td>
<td>1390.56</td>
<td>9.44</td>
</tr>
<tr>
<td>6</td>
<td>15227.00</td>
<td>1267.06</td>
<td>8.32</td>
</tr>
<tr>
<td>7</td>
<td>14923.40</td>
<td>1318.49</td>
<td>8.84</td>
</tr>
</tbody>
</table>

**mlp4 (8 bits)**

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>62958.80</td>
<td>777.24</td>
<td>1.23</td>
</tr>
<tr>
<td>1</td>
<td>59006.40</td>
<td>693.12</td>
<td>1.17</td>
</tr>
<tr>
<td>2</td>
<td>55714.20</td>
<td>290.94</td>
<td>0.52</td>
</tr>
<tr>
<td>3</td>
<td>53742.60</td>
<td>835.05</td>
<td>1.55</td>
</tr>
<tr>
<td>4</td>
<td>53043.80</td>
<td>489.49</td>
<td>0.92</td>
</tr>
<tr>
<td>5</td>
<td>53797.80</td>
<td>648.31</td>
<td>1.21</td>
</tr>
<tr>
<td>6</td>
<td>53745.60</td>
<td>722.37</td>
<td>1.34</td>
</tr>
<tr>
<td>7</td>
<td>60655.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**ham8 (8 bits)**
<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>94677.60</td>
<td>1647.98</td>
<td>1.74</td>
</tr>
<tr>
<td>1</td>
<td>87661.60</td>
<td>1887.98</td>
<td>2.15</td>
</tr>
<tr>
<td>2</td>
<td>85417.80</td>
<td>1098.08</td>
<td>1.29</td>
</tr>
<tr>
<td>3</td>
<td>83607.00</td>
<td>870.24</td>
<td>1.04</td>
</tr>
<tr>
<td>4</td>
<td>80754.20</td>
<td>1660.82</td>
<td>2.06</td>
</tr>
<tr>
<td>5</td>
<td>81495.40</td>
<td>1622.21</td>
<td>1.99</td>
</tr>
<tr>
<td>6</td>
<td>83701.00</td>
<td>1880.12</td>
<td>2.25</td>
</tr>
<tr>
<td>7</td>
<td>86658.80</td>
<td>1028.81</td>
<td>1.19</td>
</tr>
<tr>
<td>8</td>
<td>98221.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**urf5 (9 bits)**

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>262997.60</td>
<td>2130.32</td>
<td>0.81</td>
</tr>
<tr>
<td>1</td>
<td>242097.80</td>
<td>1730.92</td>
<td>0.71</td>
</tr>
<tr>
<td>2</td>
<td>229557.20</td>
<td>1691.04</td>
<td>0.74</td>
</tr>
<tr>
<td>3</td>
<td>222400.40</td>
<td>1444.06</td>
<td>0.65</td>
</tr>
<tr>
<td>4</td>
<td>217330.00</td>
<td>1255.85</td>
<td>0.57</td>
</tr>
<tr>
<td>5</td>
<td>217870.00</td>
<td>1602.04</td>
<td>0.74</td>
</tr>
<tr>
<td>6</td>
<td>217573.80</td>
<td>1649.06</td>
<td>0.76</td>
</tr>
<tr>
<td>7</td>
<td>218820.00</td>
<td>1981.44</td>
<td>0.91</td>
</tr>
<tr>
<td>8</td>
<td>251478.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**hwb9 (9 bits)**

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>253058.20</td>
<td>1753.67</td>
<td>0.69</td>
</tr>
<tr>
<td>1</td>
<td>234130.00</td>
<td>2098.63</td>
<td>0.90</td>
</tr>
<tr>
<td>2</td>
<td>222287.20</td>
<td>1507.57</td>
<td>0.68</td>
</tr>
<tr>
<td>3</td>
<td>212919.80</td>
<td>2554.46</td>
<td>1.20</td>
</tr>
<tr>
<td>4</td>
<td>211623.00</td>
<td>3891.74</td>
<td>1.84</td>
</tr>
<tr>
<td>5</td>
<td>211727.60</td>
<td>2098.67</td>
<td>0.99</td>
</tr>
<tr>
<td>6</td>
<td>209847.80</td>
<td>994.75</td>
<td>0.47</td>
</tr>
<tr>
<td>7</td>
<td>208544.20</td>
<td>2491.61</td>
<td>1.19</td>
</tr>
<tr>
<td>8</td>
<td>247324.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**urf1 (9 bits)**

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>752606.60</td>
<td>8630.68</td>
<td>1.15</td>
</tr>
<tr>
<td>1</td>
<td>693328.20</td>
<td>6224.36</td>
<td>0.90</td>
</tr>
<tr>
<td>2</td>
<td>662626.40</td>
<td>6182.74</td>
<td>0.93</td>
</tr>
<tr>
<td>3</td>
<td>645025.60</td>
<td>2902.58</td>
<td>0.45</td>
</tr>
<tr>
<td>4</td>
<td>635811.00</td>
<td>9533.07</td>
<td>1.50</td>
</tr>
<tr>
<td>5</td>
<td>632504.00</td>
<td>3865.58</td>
<td>0.61</td>
</tr>
<tr>
<td>6</td>
<td>635001.20</td>
<td>3213.31</td>
<td>0.51</td>
</tr>
<tr>
<td>7</td>
<td>633250.80</td>
<td>5911.52</td>
<td>0.93</td>
</tr>
<tr>
<td>8</td>
<td>641412.00</td>
<td>3245.25</td>
<td>0.51</td>
</tr>
<tr>
<td>9</td>
<td>714802.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**urf3 (10 bits)**
<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4185570.60</td>
<td>3021.68</td>
<td>0.07</td>
</tr>
<tr>
<td>1</td>
<td>3759816.20</td>
<td>16189.25</td>
<td>0.43</td>
</tr>
<tr>
<td>2</td>
<td>3509468.60</td>
<td>14335.49</td>
<td>0.41</td>
</tr>
<tr>
<td>3</td>
<td>3408426.60</td>
<td>14354.58</td>
<td>0.42</td>
</tr>
<tr>
<td>4</td>
<td>3327920.60</td>
<td>19046.78</td>
<td>0.57</td>
</tr>
<tr>
<td>5</td>
<td>3270174.20</td>
<td>9570.76</td>
<td>0.29</td>
</tr>
<tr>
<td>6</td>
<td>3273673.20</td>
<td>17588.71</td>
<td>0.54</td>
</tr>
<tr>
<td>7</td>
<td>3286444.80</td>
<td>12309.80</td>
<td>0.37</td>
</tr>
<tr>
<td>8</td>
<td>3283356.00</td>
<td>4912.80</td>
<td>0.15</td>
</tr>
<tr>
<td>9</td>
<td>3297517.40</td>
<td>18489.35</td>
<td>0.56</td>
</tr>
<tr>
<td>10</td>
<td>3531787.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**urf4 (11 bits)**

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8039679.40</td>
<td>124163.35</td>
<td>1.54</td>
</tr>
<tr>
<td>1</td>
<td>7946022.80</td>
<td>123583.65</td>
<td>1.56</td>
</tr>
<tr>
<td>2</td>
<td>5564745.40</td>
<td>80074.53</td>
<td>1.44</td>
</tr>
<tr>
<td>3</td>
<td>3572569.20</td>
<td>83341.23</td>
<td>2.33</td>
</tr>
<tr>
<td>4</td>
<td>2252990.40</td>
<td>32991.30</td>
<td>1.46</td>
</tr>
<tr>
<td>5</td>
<td>1347609.40</td>
<td>6575.66</td>
<td>0.49</td>
</tr>
<tr>
<td>6</td>
<td>713982.20</td>
<td>28507.76</td>
<td>3.99</td>
</tr>
<tr>
<td>7</td>
<td>698602.20</td>
<td>24738.06</td>
<td>3.54</td>
</tr>
<tr>
<td>8</td>
<td>691560.20</td>
<td>24986.72</td>
<td>3.61</td>
</tr>
<tr>
<td>9</td>
<td>698068.80</td>
<td>3953.90</td>
<td>0.57</td>
</tr>
<tr>
<td>10</td>
<td>686522.20</td>
<td>10372.66</td>
<td>1.51</td>
</tr>
<tr>
<td>11</td>
<td>850847.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**plus63mod4096 (12 bits)**

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Average QC</th>
<th>Std. Deviation</th>
<th>% std. dev from mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40960807.60</td>
<td>398797.49</td>
<td>0.97</td>
</tr>
<tr>
<td>1</td>
<td>39614392.40</td>
<td>136085.65</td>
<td>0.34</td>
</tr>
<tr>
<td>2</td>
<td>27007772.20</td>
<td>180973.44</td>
<td>0.67</td>
</tr>
<tr>
<td>3</td>
<td>16858437.80</td>
<td>141189.68</td>
<td>0.84</td>
</tr>
<tr>
<td>4</td>
<td>10053987.40</td>
<td>51895.44</td>
<td>0.52</td>
</tr>
<tr>
<td>5</td>
<td>5682551.20</td>
<td>51999.14</td>
<td>0.92</td>
</tr>
<tr>
<td>6</td>
<td>3046166.80</td>
<td>29255.60</td>
<td>0.96</td>
</tr>
<tr>
<td>7</td>
<td>2914814.40</td>
<td>27722.12</td>
<td>0.95</td>
</tr>
<tr>
<td>8</td>
<td>2862730.60</td>
<td>32650.13</td>
<td>1.14</td>
</tr>
<tr>
<td>9</td>
<td>2786472.20</td>
<td>106196.33</td>
<td>3.81</td>
</tr>
<tr>
<td>10</td>
<td>2807440.20</td>
<td>28871.99</td>
<td>1.03</td>
</tr>
</tbody>
</table>
Chapter 6 Covered Set Partition with Evolutionary Algorithms

6.1 Introduction

The algorithm presented herein avoids the addition of extraneous output bits and does not give consideration to the LNNM model. This chapter reports our latest milestone in the chain of algorithms based on Miller, Maslov and Dueck (MMD) [56] approach to quantum logic synthesis. Stedman and Perkowski [11] presented an algorithm capable of producing circuits with lower number of gates by exploring permutations of input vector ordering other than the natural ordering used by MMD. Stedman’s method however stalls at large number of variables, as it requires an exorbitant amount of time to compute. Alhagi, Hawash and Perkowski [22] followed up with a synthesis method that explores a subset of Stedman’s orderings that produce near optimal circuits within a reasonable amount of time. Hawash, et al. [23] explored alternative convergent sets of Stedman’s orderings, dubbed Covering Set Partitions, which were able to discover solutions of lower quantum gate cost. This chapter explores the impact of partition depth on quantum cost.

The main topics of this chapter are:

- The impact on quantum gate cost of using Genetic Algorithm and Tabu search compared to random selection of valid CSP sequences,
• Comparison of the performance (with respect to quantum gate cost) of various variants of the genetic algorithm (single and double cross over) and Tabu search,

• The Impact on quantum cost of varying the depth of the CSP partition used to generate valid sequences.

6.2 **MMD Style Algorithms**

In the paper, *A Transformation Based Algorithm for Reversible Logic Synthesis*, Miller, et al. [56] outlined a simple, yet powerful, synthesis method of reversible circuits. This algorithm observes a simple, yet essential, guiding principle stating that: *A completely mapped pair can never be altered by succeeding mapping calculations*. This important rule, along with inherent attribute of natural binary ordering of the input vector, allows MMD to *always* converge which is an essential principle for synthesizing arbitrary reversible circuits. The issue of convergence has been treated fully by [21, 23, 11] and, for the sake of setting context for convergence as it relates to CSP, the reader is encouraged to review [23]. Some definitions are in order before I illustrate the algorithm with an example.

6.3 **Anatomy of Covered Set Partition Algorithm**

6.3.1 **Structure**

I hinted earlier that MMD [56] uses the natural binary order to arrange the minterms of the input vector and that such an arrangement ensures convergence. Stedman [11], Alhagi [22] and the current authors [23] documented the advantage of exploring
alternative sequencing of input vector. Stedman outlined an algorithm for detecting convergent input orderings and Alhagi devised a systematic algorithm, based on the Hasse diagram, for constructing valid input orderings for any number of bits and demonstrated the ability to produce circuits at lower quantum cost within a reasonable period of time. In our attempt to improve on Alhagi’s work, I construct a different set of sequences based on the mathematical concept of partially ordered sets described below.

**Definition 6-1:** A Hasse diagram is a type of mathematical diagram used to represent a finite partially ordered set, in the form of a graph where, for the relation \((x,y) \mid x \leq y \mid x,y \in S\), each element of \(S\) is a vertex in the plane and draws a line segment or curve that goes upward from \(x\) to \(y\) whenever \(y\) covers \(x\) (that is, whenever \(x < y\) and there is no \(z\) such that \(x < z < y\)).

Figure 6-1 displays graphical illustrations of two variants of the covering set partitions method for a function of four variables. The table to the left of the graph sets a partition depth of 1 bit, which is depicted graphically by the upper and lower regions labeled \((b3:0\text{ and } b3:1)\). The lower half of the graph represents the partition where the highest bit 3 = 1, and the upper half is for the partition where bit3 = 0. For the remaining three bits \((b2-b0)\), the algorithm uses the Hasse structure to create the sequence for each of the two halves. Notice the Hasse diagram levels are represented by the diagonal lines of the top half – see [22], [23] for more information about creating the Hasse sequence.

The following ordered set represents the order of the minterms in the sequence for a partition depth of one (underlined).

\[
\{\{0000\}, \{0001, 0010, 0100\}, \{0011, 0101, 0110\},\{0111\},
\]
Alternatively, a valid sequence could be constructed using a partition depth of 2 which is represented graphically by the four planes of the upper and lower surfaces of the cube and shown in the table on the right. In this case, terms with $b_3b_2=00$ are placed at the beginning of the sequence followed by $b_3b_2=01$, $b_3b_2=10$ and finally $b_3b_2=11$. The remaining two bits could still be taken according to the Hasse sequence. The following ordered set is a valid sequence for a partition depth of two:

\[
\{\{0000\}, \{0001, 0010\}, \{0011\}, \{0100, 0101\}, \{0111\}, \\
\{1000\}, \{1001, 1010\}, \{1011\}, \{1100\}, \{1110, 1101\}, \{1111\}\}
\]

6.3.2 Steps for creating valid sequences

**Definition 6-2:** For a binary function of $n$ variables, a band within a Hasse diagram is the set of minterms $(b_{n-1}...b_1b_0)$ which have the same number of ones; i.e., \[x = b_{n-1}...b_1b_0 | \forall x \sum_{i=0}^{n-1} b_i \text{ is the same}\].

**Corollary 1:** An $n$-variable binary function has a total of $n+1$ bands.

The following process outlines the steps for creating CSP sequences for an $n$-variable function using the $p$ upper bits for partition:

- Create $k=2^p$ partitions where $p$ is the partition depth represented by the number of upper bits resulting in the number of partitions $N=0..k-1$.
- To construct an input sequence, place all the terms sequentially according to their partition number $N=0..k-1$. 
Within each partition, use the Hasse diagram to arrange the minterms within a partition as follows:

- Start from the base level of the Hasse diagram consisting of all zeros,
- Randomly permute, i.e., shuffle, terms of the next band consisting of single ones and place them at the end of the ordering,
- Repeat step (b) for each band that follows in consecutive order, where each band has an additional one compared to the band before it, (two ones, three ones, ... ),
- Place the last term consisting of all ones (k-1 ones) at the end of the sequence.

Figure 6-1 Covering Set Partitions with partition size=1 using bit 3 to create two partitions of 3 bits each (upper and lower cubes), or partition size=2 using bits 3-2 to create 4 partitions of 2 bits each (four planes). The dark line separating the partitions is referred to as the partition boundary where minterms are not allowed to cross such boundary in the process of rearranging minterms to create different input sequences.
6.4 **Algorithmic Contest**

In section 6.3.2 above, I outlined the steps for creating a single valid sequence using the CSP algorithm. I stipulated then that there exists a number of solutions in an exponentially expanding search space. In [23] I employed a random process in constructing the sequences and maintained the ones with the best cost up to that point. It was shown then, that, for a small number of variables, the CSP performed well compared to earlier attempts by [22, 56]; yet as the number of variables increased, the ability to find better solutions became dismal at best. I concluded then that the vastness of search space hindered our ability to discover the *proverbial needle in the haystack*. In this effort, I present the results of exploring two additional alternative selection methods of the input vector sequence and compare the performance of the three methods: *Random, Genetic Algorithm and Tabu search*. It is noteworthy that I was careful to provide a fare comparison by stipulating that each method selects and synthesizes the same number of sequences and only varied the method of constructing valid sequences.

6.4.1 **Objective function using Quantum Cost**

The quality of a solution is measured by *quantum cost* which represents the number of elementary quantum gates used to implement the specification. For an arbitrary quantum circuit \( C \) with \( k \) quantum NCT gates, the quantum cost \( Q_c \) is calculated as follows:

\[
Q_c = \sum_{i=1}^{k} G_{qe}(i)
\]
where: \( G_{qc} \) is the quantum cost of each gate in the cascade and can be calculated according to Table 6-1.

**Table 6-1 Quantum Cost of elementary gates**

<table>
<thead>
<tr>
<th>Gate Type</th>
<th>Quantum Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT, C'NOT</td>
<td>1 [78]</td>
</tr>
<tr>
<td>C^2 NOT (Toffoli)</td>
<td>5 [79]</td>
</tr>
<tr>
<td>C^m NOT (3 \leq m \leq \lfloor \frac{n}{2} \rfloor)</td>
<td>12m – 22 [34]</td>
</tr>
<tr>
<td>C^m NOT (\lfloor \frac{n}{2} \rfloor + 1 \leq m \leq n – 2)</td>
<td>24m – 64 [34]</td>
</tr>
<tr>
<td>C^{n-1} NOT</td>
<td>2^n – 3 [79]</td>
</tr>
</tbody>
</table>

6.4.2 *Method 1: A Random Skip and Hop*

In order to discover a solution with the lowest quantum cost, a set of \( k \) valid solutions are created *randomly* according to the steps outlined above. Notice that because each band is shuffled in a random manner, step \( \cdot \) above, potential solutions are selected for examination in a blind manner; i.e., past solutions has no influence on the structure of new solutions. A new solution is saved only if its quantum cost is lower than the *current lowest cost*; otherwise, the solution is purged, and a new solution is randomly constructed. The following pseudo-code demonstrates the operation of random selection:

```plaintext
cost := MAX_INTEGER
k := number of solutions to examine
for i:= 1 to k
    solution := initialize();
    if (evaluate(solution) < cost)
        best_solution := solution;
    end if
end for
```

Although the search space grows exponentially, \((2^n)!\), there exists a very small possibility that a solution would be visited more than once. More dramatically, however,
is that the odds of finding solutions with low quantum cost are equivalent to the odds of hitting the jackpot of the grand lottery.

6.4.3 Method 2: Genetic Algorithm

Rather than bouncing randomly around the search space, a genetic algorithm follows a set of directed probabilistic steps where new solutions are the offspring of existing good solutions. The following block exhibits the standard structure of a genetic algorithm:

```plaintext
g := number of generations;
initialize(P(g));
do
  evaluate(P(g));
  P_1(g), P_2(g) := select(P(g));  // select set of parents
  g := g - 1;
  P(g) := recombine(P_1(g), P_2(g));  // crossover ➔ offspring
  P(g) := mutate(P(g));               // mutate selected child
while(g > 0);
```

The initialization and evaluation steps are exactly the same steps used in the random algorithm in 6.4.2 above. Roulette wheel selection process was used to randomly select two parents of the current generation for recombination. Single and double crossover operators were used to create the offspring with certain limitations on the position of the crossover, discussed below. Finally, mutation is probabilistically applied to each offspring in order to continuously maintain population diversity and avoid premature convergence to local minima.
6.4.3.1 Genotype and Valid Operators

As discussed earlier and shown in [22], [23], the band structure of definition 7.2, above, must be maintained to ensure algorithmic convergence. Consequently, recombination operators are limited to the band boundary and mutation operators are limited to intra-band alterations.

Figure 6-2 illustrates the structure of a chromosome for a three variable binary function with CSP partition depth of one (1). In order to ensure that a child is a valid CSP sequence, the crossover point(s) must happen at the band boundary position. Had the invalid crossover point been taken in Figure 6-2, the resultant child would have been invalid as it would have included the term 001 twice and lacked the term 010. Of course a repair process could have detected and corrected such a defect which would surely yield different result and could be the subject of future exploration. The reader might correctly surmise that the choice of limiting crossover to band boundary could potentially result in stale members within each band, leading to premature convergence to local minima. Such an anomaly is treated with random mutation within a band at a level higher than mutation probability suggested by standard genetic algorithms. A high level of mutation probability, I theorized, would inject diversity within children allowing them to escape such hasty local minima.
Figure 6-2 Genotype of a valid CSP input sequence showing valid and invalid mutations and cross over operations. A valid crossover can only occur at the \textit{partition boundary} for each specific partition and at the band boundary as stipulated by the Hasse diagram. Valid mutations, in this case swap, can only swap elements within the band to assure that the Hasse order is not violated.

6.4.4 \textit{Method 3: Tabu Search}

I also implemented the Tabu\textsuperscript{3} search [80, 81, 82] to examine whether it would discover solutions with lower quantum cost than either the genetic algorithm or random methods. Tabu is a meta-heuristic search algorithm which, during the selection process, \textit{forbids} search moves to solutions already visited in the past $m$ steps. As a result, the algorithm is amenable to accept, temporarily, solutions with inferior quantum cost, in order to \textit{skip}, possibly better, solutions which were \textit{just} investigated. Such an approach, I assumed, should provide protection against the trap of falling into local minima early. The following list describes the Tabu search:

\textsuperscript{3} Tabu or taboo.
Unlike the genetic algorithm where an initial population of $m$ solutions is created followed by generations of solutions through intra breeding and mutation, the Tabu search starts with a randomly begotten single solution, $\theta$. At each step of the process, $n$ mutations are serially performed to create a neighborhood of $n$ solutions, using the same probabilistic intra-band swap operator of the genetic algorithm above. The selection criteria of new solutions are:

- When a solution is selected for synthesis, it is added to the Tabu list, $T$, used to reject future encounters of the same solution.
- When a solution $\lambda$ with a better cost than $\theta$ is found:
  - Select $\lambda$ only if it is not in the Tabu list $T$ or it has not been visited for $\tau$ iterations.
  - Otherwise, select the next best solution in the neighborhood $N(\theta)$ according to the same criteria above.
- If all neighbor solutions are in the Tabu list, a new set of neighbors is generated.

\[
C := \{15, 20, 25, 30\}; \quad // \text{constant for different runs}
\theta := \text{initialize}();
\tau := \text{bands} / 2 + C(\text{runs});
\text{do}
    \text{evaluate}(\theta);
    N(\theta) := \text{sort}(\text{neighborhood}(\theta)); \quad // \text{neighborhood set}
    \lambda := \text{select}(N(\theta)) \{ \lambda \notin T(\theta) \text{ OR } T(\theta) > \tau \};
    T := \{\theta\}; \quad // \text{add to top of taboo set}
\text{while} (\text{not termination-condition});
\]
Rather than generating a fixed number of neighbors, the algorithm determines the size of the neighborhood based on the size of the band selected for mutation:

\[
\text{Neighborhood Size} = \beta \times \text{length(band)}; \quad /\!/ \beta \in \{\frac{1}{4}, \frac{1}{8}, \frac{1}{16}\}
\]

The factor \(\beta\) is inversely proportional to the number of variables and was introduced as a trade-off to speed up computation by reducing the neighborhood size as the number of variables increases. For the sake of reducing memory usage and increasing speed of comparison, I chose to store the checksum of the input vector rather than saving the entire vector in the list.

### 6.5 Experimental Results

For the purposes of this experiment, I selected five benchmark functions of 9 to 11 variables [83] and, similar to the previous chapter, collected the quantum cost for each partition size along with the standard deviation as shown in Table 7-2. In this experiment I answer the questions: what is the impact of the selection method on the quantum cost and does the partition size continue to affect the result? In order to keep a balanced comparison, the following steps were observed:

- The same synthesis algorithm was used once the input sequence was constructed.
- All algorithms processed the same number of input sequences for each partition size (100,000 sequences), regardless of the chance, that the same sequence could have been selected repeatedly.
• For each partition size, and for each flavor of the algorithm, 5 runs are executed with the average quantum cost being reported along with the ratio of the standard deviation to the mean as a measure of the variance between runs. Consequently, for a function of \( n \) variables, the algorithm explores \( 2,000,000n \) input sequences \((4 \times 5 \times 100000n)\).

• For the genetic algorithm, I selected a mutation probability of 0.2 and recombination probability of 0.8 as they appeared to provide the best results in the process of tuning the algorithm to the probability parameters.

• The Roulette wheel method is used to select two parents for the creating of new offspring.

• The comparison was performed for different partition and results for each partition size are reported separately.

![Figure 6-3 Quantum Cost for the URF9 (9 bit) function for the four selection methods at each partition point. Notice that, in general, the Tabu search performs the best compared to the other selection methods and that the Random selection method is always the worst. Also notice that the best quantum cost for all methods happens around the midpoint of the number of variables (here at partition size = 4).](image)
Table 7-2 shows a comparison amongst four different selection methods of the input sequence: *random, genetic algorithm with single crossover, genetic algorithm with double crossover*, and *Tabu search*. For a function of *n variables*, the four selection method are applied for partition sizes 0 to *n*-2. The *n*-1 partition size is not processed since it represents the MMD sequence which uses the natural order. From Table 7-2 and Figure 7-3, the Tabu search has consistently outperformed the other selection methods with few exceptions where the genetic algorithm with single or double crossover won over the Tabu search. Overall, however, utilizing an evolutionary algorithm in the selection process of input sequences has shown to be superior to random selection.

The results for the GA and Tabu search are also consistent with the results for the random selection in consideration to the partition size. For the Tabu search, partition sizes around the midpoint of the number of variables yielded the best quantum cost for all functions except for *urf1*. Even for that specific function, the quantum cost around the midpoint (193544) is only 3% off the best cost (188523).
Table 6-2 Comparison for five large functions between different selection methods (random, GA with single point crossover, GA with double point crossover, and Tabu search). By far, the Tabu search has outperformed all other methods when compared for the same partitions size as shown by the lightly shaded areas. For each function, the partition size also had an impact the outcome where similar to the random selection; partition sizes around the midpoint of the number of variables yield the best results – shaded in dark gray.

<table>
<thead>
<tr>
<th>Partition Size</th>
<th>Random Average QC</th>
<th>Single Crossover Average QC</th>
<th>Double Crossover Average QC</th>
<th>Tabu Average QC</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>94677.60</td>
<td>1.74</td>
<td>92405.34</td>
<td>1.71</td>
</tr>
<tr>
<td>7</td>
<td>87661.60</td>
<td>2.15</td>
<td>86083.69</td>
<td>2.11</td>
</tr>
<tr>
<td>4</td>
<td>85417.80</td>
<td>1.29</td>
<td>83624.03</td>
<td>1.27</td>
</tr>
<tr>
<td>2</td>
<td>83607.00</td>
<td>1.04</td>
<td>82102.07</td>
<td>1.02</td>
</tr>
<tr>
<td>7</td>
<td>80754.20</td>
<td>2.06</td>
<td>79785.15</td>
<td>2.03</td>
</tr>
<tr>
<td>5</td>
<td>81945.90</td>
<td>1.99</td>
<td>80109.98</td>
<td>2.30</td>
</tr>
<tr>
<td>2</td>
<td>83701.00</td>
<td>2.25</td>
<td>82278.08</td>
<td>2.21</td>
</tr>
<tr>
<td>7</td>
<td>86658.00</td>
<td>1.19</td>
<td>86455.58</td>
<td>1.17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>urf5 (9 bits)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25297.60</td>
<td>0.81</td>
<td>25773.65</td>
<td>0.81</td>
</tr>
<tr>
<td>1</td>
<td>24079.80</td>
<td>0.72</td>
<td>23701.35</td>
<td>0.71</td>
</tr>
<tr>
<td>2</td>
<td>22955.70</td>
<td>0.74</td>
<td>22427.38</td>
<td>0.71</td>
</tr>
<tr>
<td>3</td>
<td>22240.40</td>
<td>0.65</td>
<td>22076.40</td>
<td>0.64</td>
</tr>
<tr>
<td>4</td>
<td>21733.00</td>
<td>0.57</td>
<td>21276.67</td>
<td>0.55</td>
</tr>
<tr>
<td>5</td>
<td>21787.00</td>
<td>0.74</td>
<td>21416.62</td>
<td>0.72</td>
</tr>
<tr>
<td>6</td>
<td>21753.80</td>
<td>0.76</td>
<td>21232.32</td>
<td>0.74</td>
</tr>
<tr>
<td>7</td>
<td>21882.00</td>
<td>0.91</td>
<td>21422.74</td>
<td>0.90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nwb9 (9 bits)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25305.20</td>
<td>0.69</td>
<td>24790.92</td>
<td>0.67</td>
</tr>
<tr>
<td>1</td>
<td>23430.00</td>
<td>0.90</td>
<td>22874.01</td>
<td>0.89</td>
</tr>
<tr>
<td>2</td>
<td>22228.20</td>
<td>0.68</td>
<td>21852.89</td>
<td>0.66</td>
</tr>
<tr>
<td>3</td>
<td>21291.80</td>
<td>1.20</td>
<td>20802.64</td>
<td>1.15</td>
</tr>
<tr>
<td>4</td>
<td>21163.20</td>
<td>1.84</td>
<td>20675.65</td>
<td>1.78</td>
</tr>
<tr>
<td>5</td>
<td>21172.60</td>
<td>0.99</td>
<td>20918.67</td>
<td>0.98</td>
</tr>
<tr>
<td>6</td>
<td>20984.80</td>
<td>0.47</td>
<td>20628.39</td>
<td>0.47</td>
</tr>
<tr>
<td>7</td>
<td>20854.40</td>
<td>1.20</td>
<td>20604.67</td>
<td>1.19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>urf1 (10 bits)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>75260.60</td>
<td>1.15</td>
<td>73609.25</td>
<td>1.16</td>
</tr>
<tr>
<td>1</td>
<td>69332.80</td>
<td>0.90</td>
<td>68362.61</td>
<td>0.90</td>
</tr>
<tr>
<td>2</td>
<td>66626.40</td>
<td>0.93</td>
<td>64871.25</td>
<td>0.92</td>
</tr>
<tr>
<td>3</td>
<td>64502.50</td>
<td>0.45</td>
<td>63857.34</td>
<td>0.45</td>
</tr>
<tr>
<td>4</td>
<td>63581.10</td>
<td>1.50</td>
<td>62881.08</td>
<td>1.45</td>
</tr>
<tr>
<td>5</td>
<td>63250.04</td>
<td>0.61</td>
<td>62301.44</td>
<td>0.61</td>
</tr>
<tr>
<td>6</td>
<td>63500.20</td>
<td>0.51</td>
<td>62674.68</td>
<td>0.50</td>
</tr>
<tr>
<td>7</td>
<td>63325.80</td>
<td>0.93</td>
<td>61931.28</td>
<td>0.90</td>
</tr>
<tr>
<td>8</td>
<td>64141.00</td>
<td>0.51</td>
<td>63050.88</td>
<td>0.51</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>urf3 (10 bits)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4185571.04</td>
<td>321.68</td>
<td>411860.01</td>
<td>3026.128</td>
</tr>
<tr>
<td>1</td>
<td>37598.16</td>
<td>1618.25</td>
<td>369965.92</td>
<td>16383.29</td>
</tr>
<tr>
<td>2</td>
<td>35094.69</td>
<td>14335.49</td>
<td>34252.41</td>
<td>14243.74</td>
</tr>
<tr>
<td>3</td>
<td>34082.47</td>
<td>14354.58</td>
<td>33048.33</td>
<td>14055.13</td>
</tr>
<tr>
<td>4</td>
<td>33729.21</td>
<td>19046.78</td>
<td>32513.78</td>
<td>18385.19</td>
</tr>
<tr>
<td>5</td>
<td>32701.74</td>
<td>9570.76</td>
<td>32309.32</td>
<td>9550.83</td>
</tr>
<tr>
<td>6</td>
<td>32736.73</td>
<td>17588.71</td>
<td>31983.79</td>
<td>17747.03</td>
</tr>
<tr>
<td>7</td>
<td>32864.45</td>
<td>12309.88</td>
<td>32437.21</td>
<td>12601.05</td>
</tr>
<tr>
<td>8</td>
<td>32833.56</td>
<td>4912.88</td>
<td>32373.89</td>
<td>4909.47</td>
</tr>
<tr>
<td>9</td>
<td>32975.17</td>
<td>18489.35</td>
<td>32282.70</td>
<td>1824.22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>urf4 (11 bits)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>393562.63</td>
<td>3021.68</td>
<td>391860.01</td>
<td>3026.128</td>
</tr>
<tr>
<td>1</td>
<td>375981.26</td>
<td>16189.25</td>
<td>369965.92</td>
<td>16383.29</td>
</tr>
<tr>
<td>2</td>
<td>350946.90</td>
<td>14335.49</td>
<td>34252.41</td>
<td>14243.74</td>
</tr>
<tr>
<td>3</td>
<td>340824.17</td>
<td>14354.58</td>
<td>33048.33</td>
<td>14055.13</td>
</tr>
<tr>
<td>4</td>
<td>337291.21</td>
<td>19046.78</td>
<td>32513.78</td>
<td>18385.19</td>
</tr>
<tr>
<td>5</td>
<td>327017.45</td>
<td>9570.76</td>
<td>32309.32</td>
<td>9550.83</td>
</tr>
<tr>
<td>6</td>
<td>327367.73</td>
<td>17588.71</td>
<td>31983.79</td>
<td>17747.03</td>
</tr>
<tr>
<td>7</td>
<td>328644.45</td>
<td>12309.88</td>
<td>32437.21</td>
<td>12601.05</td>
</tr>
<tr>
<td>8</td>
<td>328335.56</td>
<td>4912.88</td>
<td>32373.89</td>
<td>4909.47</td>
</tr>
<tr>
<td>9</td>
<td>329751.17</td>
<td>18489.35</td>
<td>32282.70</td>
<td>1824.22</td>
</tr>
</tbody>
</table>

129
Table 7-2 shows a comparison amongst four different selection methods of the input sequence: random, genetic algorithm with single crossover, genetic algorithm with double crossover, and Tabu search. For a function of $n$ variables, the four selection methods are applied for partition sizes 0 to $n-2$. The $n-1$ partition size is not processed since it represents the MMD sequence which uses the natural order. From Table 7-2 and Figure 7-3, the Tabu search has consistently outperformed the other selection methods with few exceptions where the genetic algorithm with single or double crossover won over the Tabu search. Overall, however, utilizing an evolutionary algorithm in the selection process of input sequences has shown to be superior to random selection.

The results for the GA and Tabu search are also consistent with the results for the random selection in consideration to the partition size. For the Tabu search, partition sizes around the midpoint of the number of variables yielded the best quantum cost for all functions except for $urf1$. Even for that specific function, the quantum cost around the midpoint (193544) is only 3% off the best cost (188523).
6.6 Conclusion and Analysis

By utilizing heuristic based selection of future sequences based on the quality of already visited solutions, both the genetic algorithm and Tabu search methods were able to discover input vector sequences which produce superior results compared to purely random selection (~ 10%). In this experiment, for a function of $n$ variables, $2,000,000n$ solutions were explored in order to analyze each function, which is $20n$ times higher than the time needed for MMDSN alone. However, from this experiment, it is obvious that the Tabu search consistently provides the best results which are typically found around partitions sizes of $\frac{1}{2}n$. Assuming we limit our search to Tabu search and explore $k$ partitions around the midpoint, the necessary time to find the best solution is only $k$ times larger than the time required form MMDSN. This of course, is the same conclusion I reached in the last chapter when I limited the experiment to discovering the impact of partition size.

Using the random selection method, the algorithm takes random jumps around the huge search space and is able to find the best solution within the visited subset. However, by partitioning the search space into smaller sections, the random algorithm will jump within the bounds of each partition and is able to find even better solutions since the search space is more focused. By adding a genetic algorithm or Tabu search, the quality of selection increases since the selection algorithm does not randomly jump around the partitioned search space. For the genetic algorithm flavor, the two best parents are used to create the next set of solutions, which allows the algorithm to visit a set of children which are not far away from the parents. For the Tabu search, the
selection of the next set of candidates is even more constrained where a mutation operator with a small probability 0.25 is used to create all candidate solutions from the single best solution of the previous run. Additionally, due to the Tabu list, this method reduces the chance that the same solution is synthesized often which allows it to explore more unique solutions than any of the other methods. Combined, all of these qualities of the Tabu search gives it the best advantage of discovering the lowest quantum cost when compared to both the random and genetic algorithm flavors for selecting the next set of candidates.

Figure 6-4 Plots of four functions with quantum cost vs. partition size for the four selection methods. Similar to the urf5 function above, the Tabu search performs the best and all selection methods have lower quantum cost around the midpoint of the size of function.
Our attempts to discover methods for optimal logic synthesis have not all been successful. I have thus far explicated that the complexity of discovering solutions with low quantum costs increases exponentially relative to the number of variables. I thought that there might exist a method to predict the subset of input/output orderings which consistently yield better circuit cost. Consequently, I have attempted a few preprocessing methods in an effort to predict patterns which could potentially lead to good solutions. All such efforts, unfortunately, have been unsuccessful where no obvious pattern emerged as a predictor of good solutions.

### 7.1 Hamming Distance Predictor

The first method calculates the cumulative hamming distance between all input and output minterms. Such calculation can be performed extremely quickly compared to a full synthesis of a single input/output sequence. I hypothesize that the cumulative hamming distance between the input and output bits of each minterm could be a predictor of good solutions, and if so, I could quickly eliminate solutions predicted to be of high quantum cost. For example, the binary specification in Table 7-1a yields a total hamming distance of four (highlighted) between the input and output minterms. I conjectured that a possible swap of qubits locations, Table 7-1b, could yield circuits at lower cost, and by calculating the hamming distance of both situations, I could easily focus on the sequences of Table 7-1b. In this artificial example, adding a swap gate
between the two qubits brings the hamming distance to zero, as shown in Table 7-1b, and hence, the entire circuit consists of a single swap gate.

Table 7-1 Hamming distance between input (ab) and output (AB) vectors. In (a), the HD=4 when the qubits are arranged in this manner, however, if the outputs are swapped (BA), then the hamming distance=0. I assumed that it is feasible that by rearranging the qubits and synthesizing the modified function might yield better results. However, preliminary experimentation with different functions did not indicate that this would be a successful strategy.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) HD=4

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) HD=0

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

(c) HD=4

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

(d) HD=2

Figure 7-1 For the same sequence, swapping bit positions in the output and calculating the bit to bit hamming distance vs. the quantum cost of the resultant circuit. From the plot, it is clear that using a preprocessor to determine the arrangement of variables with the lowest hamming distance does not give a good predictor of the result.

Another example is shown in Table 7-1c-d where, swapping A and B results in the reduction of bit-to-bit hamming distance from four to two. For this experiment I had
hoped to find that a low cumulative hamming distance could be an indicator of sequences which yield better results by swapping variables as shown above. The results, however, did not show that either low or high hamming distances are predictors of lower cost circuits. Figure 7-1 shows the results of synthesizing a single sequence where each bit in the output sequence was swapped with all other bits followed by calculating the quantum cost.

![Figure 7-2 Cumulative Positional Distance (CPD) of minterms vs. quantum cost. I hypothesized that a preprocessor which, for a specific minterm, calculates the Absolute Distance |d| between its position in the input vector to its position in the output vector, and then accumulating all these distances to determine the CPD. The CPD would, supposedly, then could be used as a predictor of which input/output sequence could yield better quantum costs. However, as seen from the plot, no clear sign emerges to fortell such a result.](image)

7.2 Absolute Distance Predictor

I then imagined a relationship between the cumulative distances of a minterm in the input vector to its location in the output vector. Table 7-2a, for example, shows the minterm (10) in rows 2 and 3 of the input and output vectors respectively, placing a
distance of one apart. Minterm (01) also has a distance of 1 for a cumulative distance of two for all minterms of the input and output vectors. Table 7-2b, on the other hand, shows a different function with an absolute distance of 4 \([d(10) = 2, d(01)=1, d(11) =1]\). Again, I had hoped to find a clue from calculating the absolute distance to identify potential good solutions; however, again, no clear indication emerged from this exercise. Figure 7-2 shows the same function used in the Hamming Distance Predictor above with each output bit swapped with all other output bits followed by calculation of the quantum cost vs. the cumulative positional distance of minterms.

Table 7-2 Absolute Distance \(|d|\) of minterms between input and output vectors. In (a) the input minterm (01) is one position away from its location in the output vector. Similarly, the (10) input minterm is one position away from its location in the output vector, which combined, results in \(|d|=2\). Similarly for (b) where \(|d|=4\).

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) \(|d|=2\)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(b) \(|d|=4\)

7.3 Vector Length Predictor

I also hypothesized that portraying the set of \(2^n\) input and output minterms as vectors in a \(2^n\) dimension Euclidean space and that the difference in the length of such vectors could be a predictor of quantum cost. Again, however, the results in Figure 7-3 illustrate the fallacy in our assumption where the results appear to closely mimic the absolute distance predictor.
Figure 7-3 Difference in Euclidean vector length between input and output vectors as a predictor of best quantum cost does not appear to show a clear pattern of which candidate solutions could yield better quantum costs. The vector length is determined by considering that the value of each minterm as a decimal distance away from zero and calculating the Euclidean distance for the multi-dimensional input and output vectors. The absolute difference between such values is plotted vs the quantum cost.
PART III

QUANTUM LOGIC SYNTHESIS

OF

BINARY FUNCTIONS

IN COMPLIANCE WITH

LINEAR NEAREST NEIGHBOR MODEL
Chapter 8 **Linear Nearest Neighbor Model for Binary**

### 8.1 Introduction

Algorithms for automated synthesis of quantum and reversible circuits have generally measured their performance according to an established set of costs of primitive quantum gates [4, 84, 21, 56, 23, 24, 85]. The foundation for those costs, however, is built on the assumption that any two qubits are able to interact directly and independent of any other qubits. However, current realization of quantum technologies indicate that certain intrinsic physical limitations exist which inhibit such distant interaction amongst qubits [15, 16, 17]. For instance, in the One-Dimensional Ion Trap technology introduced by Cirac and Zoller [16, 15] a two qubit CNOT gate is created by entangling neighboring qubits. In quantum optics, qubits also interact by proximity using optical wires or crystals [86].

Ignoring the physical nearest neighbor limitation, some algorithms take liberty of adding many ancillary qubits with permissive interaction amongst distant qubits for the sake of reducing the overall quantum cost of the resultant circuit [57, 59, 87]. Recent attempts to account for LNN model either focused on swapping qubits for the gate to bring their qubits next to one another, or using template matching to replace non-LNN compliant sections of the circuit with their LNN compliant ones [87, 88]. In [25], I demonstrated a mathematical decomposition for determining the worst case scenario for applying LNN restriction to a quantum circuit which also considers the internal structure of MCT gates. The authors in [88] described a method for synthesizing quantum circuits...
for LNN model using template matching and qubit reordering strategies and paper [87] showed the benefit of LLN method for symmetrical layout of quantum circuits.

Regardless of the algorithm used to synthesize the circuit, a physically realizable measure of quantum cost is necessary in order to accurately determine the actual cost of implementing a quantum circuit. In this dissertation I propose a universal measure of LNN Quantum Cost which observes the cost of bringing interacting qubits next to one other for MCT gates, and the cost of enforcing the LNN restriction to the decomposition of MCT gates. In our analysis, the method only allows single \((V, V^\dagger\text{ and NOT})\) and two qubit \((CV, CV^\dagger\text{ and CNOT})\) gates. Multiple Control Toffoli (MCT) gates and Multiple Control \(V/V^\dagger\) gates are not allowed in the calculations [89]; higher levels of MCV gates require exponentially smaller degrees of rotations which are hard to implement.

I first analyze the Linear Nearest Neighbor Quantum Cost (LNNQC) of the well-known Toffoli gate. I then extend the analysis to Multiple Control Toffoli (MCT) gates and calculate the number of primitive single and two-qubit gates necessary to construct such gate with consideration to the LNN restriction. I also calculate the cost of bringing two qubit gates with separated target and control lines next to one another and introduce an algorithm for finding the optimal LNNQC for MCT gates. I introduce a new cost component, ancillary ratio, which accounts for the increase of circuit depth and finally I present the proposed LNNQC cost for a set of MCT gates and portrays the impact on the quantum cost of some of the commonly used benchmark functions.
8.2 Toffoli Gate Cost

The well-known NCT library (Not Controlled Not and Toffoli) is one of the widely used set of gates in quantum logic synthesis [4, 84, 22, 56, 23, 24]. The Toffoli gate, in particular, is considered a universal gate because, similar to NAND in classical computing, it can be used to construct any other binary gate which could be used to construct any binary quantum circuit. A set of $n$-bit Multiple Controlled Toffoli (MCT) gates are a theoretical extension to the Toffoli gate where additional control lines are used to control an XOR gate on the target qubit. Barenco et al, [84] provided a cogent mathematical construction for deriving any $n$-variable MCT gate from the set of one- and two-qubit elementary gates: $NOT$, $CV$, $CV^\dagger$ and $CNOT$. Throughout the literature, existing quantum synthesis algorithms [24, 21, 23, 20, 34, 57, 59]calculate a circuit’s quantum cost as the number of single- and two-qubit elementary gates in a synthesized circuit.

Figure 8-1 shows the de facto decomposition of the Toffoli gate currently used in the literature with a quantum cost of five (5) elementary gates [84, 68] Our disclaimer of a hypothetical model for calculating quantum cost refers to the assumption that the long range (or distant) $CV$ gate (shown within dashed rectangle in Figure 8-1) has a quantum cost of one (1) [84, 68]. The assumption that any pair of qubits separated by arbitrary distance can interact without penalty is a purely theoretical assumption. From a technological point of view, however, it is more reasonable to assume that qubits can only interact with their nearest neighbor which is consistent with existing physical realization of ion trap [90], superconducting qubits [39] or optical lattices [91].
While some technologies support interaction between qubits along a 2-dimensional plane, I limit the discussion to a linear ion trap model where a set of ions are arranged along a straight line like a tightly stretched string of beads [15]. As such, the discussion herein is limited to the application of Linear Nearest Neighbor Model (LNNM) constraint to synthesis of binary quantum circuits. When I apply the LNNM to the long range $CV$ gate of Figure 8-1, I would utilize swap gates to bridge the gap between the distant qubits (a and c) which would physically enable interaction between them. Figure 8-2 shows the most precise decomposition of the Toffoli gate where interaction between any two qubits is constrained to their nearest neighbor. In the figure, a swap gate is inserted before the $CV$ gate to bring the information of qubits a and c next to one another, and a mirror swap gate is inserted after the $CV$ gate to restore qubit a to its original position, for a total cost of eleven (11) elementary gates. Notice that the two shaded CNOT gates cancel one another, and as a result, reduce the total quantum cost for this embodiment from 11 to nine (9) elementary gates [59].

![Figure 8-1 Toffoli gate decomposed to set of 5 primitive gates. Notice the remote interaction of the first CV gate which violates the LNNM architecture.](image1)

![Figure 8-2 Toffoli gate with qubit interaction constrained to LNNM where the qubits of the first CV gates are brought next to one another through a set of swap gates. The first 3 CNOT gates act as a swap gate which brings the a qubit next to c, and the next set of CNOT gates restore the location of the a qubit back to its original location. The quantum cost here is 11 but since the two highlighted CNOT gates would cancel each other, the quantum cost of the LNNM Toffoli gate is 9.](image2)
8.3 Count of CNOT and CV Gates for MCT Gate

I now turn our attention to the quantum cost for Multiple Control Toffoli (MCT) gates with more than 2 control lines. Barenco et al. [84], Lemma 7.1, used a Gray code construction to demonstrate that an \( n \)-bit MCT gate, where \( n \) is the number of qubits of an MCT gate, requires a combination of \( 2^{n-1} \)-2 CNOT gates plus \( 2^{n-1}-1 \) 2-bit Controlled \( CV/CV^\dagger \) gates. For the sake of brevity, I shall refer to the set of \( CV/CV^\dagger \) as \( CV \) from this point forward. However elegant, the analysis did not take into account the physical requirement of LNNM and did not calculate the true cost of realizing technologies for the constraint imposed by the LNN model.

**Definition 8-1**: An \( n \)-bit multiple control Toffoli (MCT) gate is a reversible gate with the set of \( n-1 \) control line \( C = \{ c_i \in C, 1 \leq i < n \} \) and a single target line \( x \) where:

\[
    f(x) = \begin{cases} 
        \overline{x} & \forall c_i = 1 | c_i \in C \\
        x & \exists c_i = 0 | c_i \in C 
    \end{cases}
\]

To the extent of our knowledge, the vast majority of publications in the field of binary quantum synthesis have used a faulty model for aggregating the quantum cost of the resultant circuit by ignoring the LNN restriction. In this chapter, I provide a comprehensive set of equations for calculating the true cost of any MCT gate based on a) the number of control qubits and b) the gaps between the target bit and its nearest control qubit.
Figure 8.3 Decomposition of the 4-bit MCT gate without consideration of LNNM. Most algorithms assume this gate to cost 13 primitive gates, which ignores the fact that there are five gates which interact remotely in violation of the LNNM architecture.

**Lemma 8.1:** An n-Bit Multi Control Toffoli gate contains the following number of 2-qubit CV/CV+ gates:

\[
\# \text{CV, CV}^{\dagger} \text{gates for MCT}_n = \sum_{k=1}^{n-1} \text{CV}_k = 2^{n-1} - 1 \quad (eq. 1)
\]

Where:

\[
\text{CV}_k = 2^{n-k-1}
\]

n: number of qubits | n>2

k: distance between target and control qubits

**Proof:** I rely on a visual comparison between Figure 8.1 and Figure 8.3 to illustrate how to construct an MCT\(_4\) gate by starting from an MCT\(_3\) gate, as follows:

- Starting from an MCT\(_3\) gate, stretch all existing CV gates by a distance of one qubit and insert a new qubit line between the target qubit and its nearest control qubit. The new qubit will represent the new control line of an MCT\(_4\).
- Add an additional \(2^{3-1}\) CV\(_1\)/CV\(_1^{\dagger}\) gates, maintaining the alternating pattern of CV and CV\(^{\dagger}\) gates, with the newly added qubit as the control line – inside dotted rectangle.
In order to determine LNNM cost, I calculate the aggregate for each set of \( CV_k \) gates separately according to the distance \( (k) \) between the target and control qubits, as follows:

\[
CV_{[1]}: 4 = (2^4-1-1),
\]

\[
CV_{[2]}: 2 = (2^4-2-1),
\]

\[
CV_{[3]}: 1 = (2^4-3-1),
\]

Performing an analogous comparison between the MCT\(_4\) (Figure 8-3) gate and the MCT\(_5\) gate (Figure 8-4), I realize that the left half of Figure 8-4 is exactly the same as the entirety of Figure 8-3 where all CV gates have been lengthened by an additional qubit. Again, the dotted rectangle containing the right side of Figure 8-4 adds an additional \( 2^{d-1} \) CV\(_1\) gates. I now have 15 CV gates with distinct aggregates as follows:

\[
CV_{[1]}: 8 = (2^5-1-1) \quad CV_{[3]}: 2 = (2^5-3-1),
\]

\[
CV_{[2]}: 4 = (2^5-2-1), \quad CV_{[4]}: 1 = (2^5-4-1),
\]

Clearly the pattern for the total number of \( CV_k \) gates for an MCT\(_n\) gate is \( 2^{n-k-1} \) where \( k \in [1, n-1] \). Notice that the exact pattern repeats for every increment of a control line for higher orders of MCT gates. It is only a matter of mathematical dexterity to demonstrate
that the total number of $CV$ gates in equation (1) is consistent with Lemma 7.1 of Barenco et al [84] mentioned above...

Lemma 8-2: An $n$-Bit Multi Control Toffoli gate consists of the following number of 2-bit CNOT gates:

$$\#CNOT = 2 \cdot (n - 2)CNOT_1 + \sum_{k=2}^{n-2} CNOT_k = 2^{n-1} - 2 \ (eq. \ 2)$$

Where:

*n*: number of control lines

*k*: distance between target and control qubits

$$CNOT_k = (n - k - 1) \cdot 2^{k-1}$$

Proof: Taking a second glance at Figure 8-1, Figure 8-3 and Figure 8-4, notice that two additional CNOT$_1$ gates are needed for every *new* control qubit of a MCT$_n$ gate. For example, starting from an MCT$_3$ with 2 CNOT$_1$ gates, an MCT$_4$ requires 4 CNOT$_1$ gates, and an MCT$_5$ requires 6, and so on. This pattern accounts for the first term of the above equation ($2 \cdot (n - 2)CNOT_1$), as follows:

<table>
<thead>
<tr>
<th># qubits</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>...</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>#CNOT$_1$</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>2$\cdot n$</td>
<td>2$\cdot (n - 2)$</td>
</tr>
</tbody>
</table>

Let’s now consider the set of CNOT$_2$ gates within the dotted line of Figure 8-3 and Figure 8-4. In the case of Figure 8-3, we trailed every CNOT$_1$ gate with a CNOT$_2$ gate for a total of two CNOT$_2$ gates and two CNOT$_1$ gates. Let’s now focus our attention to

---

$S_n = 2^{n-1} + 2^{n-2} + \ldots + 2^{n-n} + 2^{n-n}$

$2\cdot S_n = 2 \cdot (2^{n-1} + 2^{n-2} + \ldots + 2^{n-n} + 2^{n-n}) = 2^{n} + 2^{n-1} + \ldots + 2^{n-n+2} + 2^{n-n+1}$

$2\cdot S_n - S_n = S_n = 2^n - 1$
the shaded dotted block of Figure 8-4. Notice that, in the rightmost dotted block, we added the same number of CNOT\(_2\) gates as the previous block, which is the same pattern for every additional control qubit as follows:

<table>
<thead>
<tr>
<th># qubits</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>#(\text{CNOT}_2)</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>(\left(\frac{n-2}{2} - 1\right))</td>
</tr>
</tbody>
</table>

Consider the new CNOT\(_3\) gates in the rightmost dotted rectangle in Figure 8-4. Notice that we append a CNOT\(_3\) gate to each of the CNOT\(_{\{1,2\}}\) gates in that block for a total of four (4) CNOT\(_3\) gates. Similar to the previous two cases, every new block, for higher order MCT gates, will gain 4 CNOT\(_3\) gates as follows:

<table>
<thead>
<tr>
<th># qubits</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>#(\text{CNOT}_3)</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>(\left(\frac{n-3}{2} - 1\right))</td>
</tr>
</tbody>
</table>

The recursive pattern evidently repeats for every MCT\(_n\) gate, where a number of new CNOT\(_{n-1}\) gates are added of equal quantity as of the last previously added block of CNOT gates for the preceding order of an MCT\(_{n-1}\) gate (i.e., shaded block in Figure 8-4). Obviously, the index (shown with borders) of the CNOT\(_k\) is proportional to the count of such gates. The pattern for calculating the number of CNOT\(_k\) for an \(n\)-bit MCT gate is:

\[
\text{CNOT}_k = \left(n - \left\lfloor \frac{k}{2} \right\rfloor - 1\right) \cdot 2^{k-1}
\]

which when added for all cases for \(k=2\) to \(n-2\) concludes the proof of Lemma 2. \(\blacksquare\)

Table 8-1 summarizes the number of elementary 2-qubit CV and CNOT gates for MCT\(_3\) – MCT\(_9\) where the last column provides a general formula for calculating the count for each specific gate based on the distance between the control and target qubits. The literature has been unanimous in using the numbers in the last row, labeled Gate.
Count, as the quantum cost of an MCT gate without regard to the distance between the target and control qubits. I mentioned earlier that some algorithms considered the LNNM constraint between the cascade of MCT gates, but they largely ignored the application of the same LNNM principle to the set of distant elementary gates within the decomposed MCT gate itself (gates similar to the leftmost gate in Figure 8-1) [59, 88, 89].

8.4 LNN Quantum Cost of MCTₙ Gates

**Theorem 8-1**: The maximum quantum cost of a MCTₙ gate, where n>2, is:

\[
\sum_{k=1}^{n-1} LNNQC_{CV_k} + \sum_{k=1}^{n-2} LNNQC_{CNOT_k} \tag{eq.3}
\]

Proof: In the previous section I determined the number of CV and CNOT gates needed to decompose any MCTₙ gate which I will now use to calculate the linear nearest neighbor quantum cost (LNNQC). I demonstrated earlier, Figure 8-2, that the target and control qubits of a distant gate could be brought together by a cascade of swap gates applied to one of the qubits, followed by a mirror cascade of swap gates to place it back to its original position. Figure 8-5a shows the process of bringing qubit a closer to qubit d by inserting two sets of swap gates (6 Feynman gates) followed by a mirror set to bring qubit a back to its original position. Notice that with this arrangement, the set of shaded CNOT gates will cancel each other resulting in the reduced template of Figure 8-5b for enforcing nearest neighbor constraint for any two qubit gate. The following equation
calculates the true cost of any distant CV\textsubscript{k} or CNOT\textsubscript{k} gate where k represents the gap between the control and target bit:

\[
2 - \text{qubit LNNQC}(k) = 1 + 2 \cdot [2 \cdot (k - 1)] \quad (eq. 4)
\]

Multiplying the LNNQC(k) for each qubit with the number of gates for each CV\textsubscript{k} and CNOT\textsubscript{k} proves the equations of Theorem 8-1.

Figure 8-5 LNNM equivalent of CNOT gate acting on qubits ‘a’ and ‘d’ which are two qubits apart. Rather than a quantum cost of 1, the LNNQC = 13 (a) which is further reduced to 9 (b) once the shaded gates cancel one another.

However, applying equation 4 to each of the two-qubit distant gates individually is an expensive proposition which will result in an exponential increase of LNNQC for circuits of large number of qubits [20]. In Figure 8-6 for example, the individual cost of enforcing LNN through equation 4 for the CV gates labeled 1, 2 is 17, 13 and 13 respectively. Now, I will use the same Figure 8-6 to demonstrate a better method for enforcing the LNN model to the distant CV gates within the decomposed template of any MCT\textsubscript{n} gate. A set of swap gates are first used to bring the target qubit f next to its control qubit a facilitating LNN interaction between them. Rather than returning f back to its original location through a mirror swap cascade, I only need to insert a swap gate to bring b back to its original location and f next to b. At this stage, both CV gates 2 and 3 are compliant with the LNN model as they interact with qubit b. Similarly, each dashed stage needs only a single swap gate to propagate down the target qubit f of all CV gates.
allowing interaction with the control qubit in compliance with the LNN model. According to theorem 1, independent swapping of the CV gates of Figure 8-6 requires 119 CNOT gates (at 2 CNOT per swap) while the method outlined only requires 16 CNOT gates. For any MCT\textsubscript{n} gate, the number of CNOT gates required to enforce LNN model for the CV gates is:

\[
\#\text{CNOT}(\text{MCT}_n)_{\text{CV gates}} = 4 \times (n - 2) \quad (eq. 5)
\]

I turn our attention now to the distant CNOT gates on the top right triangle of Figure 8-6, which if naively done, would require a large number of swap gates. Instead, I describe a method that would minimize the number of CNOT swap gates needed to enforce LNN model. I first note the symbol (\(\overline{\text{g}}\)) represents a reflection template (herein reflection gate), which reverses the order of the qubits of the gate (I will analyze shortly). Stage 2 has the first encounter of a distant CNOT gate whom control and target bits are brought together through a set reflection gates (represents a regular swap gates for stage 2). Two internal swap gates are also needed to allow for qubit \(b\) to interact with qubit \(c\) according to the LNN model. Stage 3 and 4 each has a pair of reflection gates of the same size as the stage number. The number of internal swap gates for stage \(k\) is:

\[
N_{\text{swap}}(k) = 2 \times N_{\text{swap}}(k - 1) + 2 \times (k - 1) \quad (eq. 6)
\]

\[\text{Figure 8-6 4-qubit swivel gate which pivots the qubits around the center point between them (abcd \(\rightarrow\)dcba). The binary swivel gate of 'n' variables requires } n^2 - 1 \text{ CNOT gates to implement.}\]
Each stage \( k \) also requires a pair of swivel gates of size \( k \). Figure 8-7 shows the construction and operation of a reflection gate. Each reflection gate is constructed from a total number of CNOT gates according to the following equation:

\[
\text{CNOT}_{\text{reflection}}(k) = k^2 - 1 \quad \text{(eq. 7)}
\]

Figure 8-7 Method for minimal swapping of distant two-qubit gates within a decomposed MCT\(_n\) gate. In stage 0, rather than adding 4 swap gates to bring qubit ‘\( a \)’ next to ‘\( f \)’ and then 4 swap gates to bring it back, I opted to bring ‘\( f \)’ next to ‘\( a \)’, and slowly bring ‘\( f \)’ back through each stage. Qubit ‘\( f \)’ is the target line and it interacts with all qubits in each of the stages. This method reduces the number of swap gates necessary to implement this MCT\(_n\) gate. Also notice that recursive patterns emerge in every stage which build on the structure of the previous stage.

**Theorem 8-2:** The number of CNOT and CV gates required to construct an optimal MCT\(_n\) gate compliant with the LNN model are:

Total Number of CV gates is given in equation 1,

and Total Number of CNOT gates =

\[
2^{n-1} - 2 + 4(n - 2) + 2 \cdot \sum_{k=3}^{n-2} 2N_{\text{swap}}(k - 1) + 2(k - 1) + 2 \cdot \sum_{k=2}^{n-2} (k^2 - 1) \quad \text{(eq. 7)}
\]

where: \( N_{\text{swap}}(2) = 2 \)
According to Theorem 8-2, for example, the MCT\textsubscript{6} gate in Figure 8-6 requires 140 CNOT gates and 31 CV gates for an LNNQC of 171; however, according to Theorem 8-1, the same MCT\textsubscript{6} gate requires 318 CNOT gates and 31 CV gates for a total LNNQC of 349.

Table 8-1 LNNQC for MCT\textsubscript{3} to MCT\textsubscript{9} gates.

<table>
<thead>
<tr>
<th>MCT</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>CV\textsubscript{1}</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>2\textsuperscript{n-2}</td>
</tr>
<tr>
<td>CV\textsubscript{2}</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>2\textsuperscript{n-3}</td>
</tr>
<tr>
<td>CV\textsubscript{3}</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>2\textsuperscript{n-4}</td>
<td></td>
</tr>
<tr>
<td>CV\textsubscript{4}</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>2\textsuperscript{n-5}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV\textsubscript{5}</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>2\textsuperscript{n-6}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV\textsubscript{6}</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2\textsuperscript{n-7}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV\textsubscript{7}</td>
<td>1</td>
<td>2</td>
<td>2\textsuperscript{n-8}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV\textsubscript{8}</td>
<td>1</td>
<td>2\textsuperscript{n-9}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Num. of CVs</td>
<td>3</td>
<td>7</td>
<td>15</td>
<td>31</td>
<td>63</td>
<td>127</td>
<td>255</td>
<td>2\textsuperscript{n-1} [54]</td>
</tr>
<tr>
<td>CNOT\textsubscript{1}</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>2(n-2) \cdot 2\textsuperscript{0}</td>
</tr>
<tr>
<td>CNOT\textsubscript{2}</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>(n-3) \cdot 2\textsuperscript{1}</td>
<td></td>
</tr>
<tr>
<td>CNOT\textsubscript{3}</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>(n-4) \cdot 2\textsuperscript{2}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNOT\textsubscript{4}</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>(n-5) \cdot 2\textsuperscript{3}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNOT\textsubscript{5}</td>
<td>16</td>
<td>32</td>
<td>48</td>
<td>(n-6) \cdot 2\textsuperscript{4}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNOT\textsubscript{6}</td>
<td>32</td>
<td>64</td>
<td>(n-7) \cdot 2\textsuperscript{5}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNOT\textsubscript{7}</td>
<td>64</td>
<td>(n-8) \cdot 2\textsuperscript{6}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Num. of CNOTs</td>
<td>2</td>
<td>6</td>
<td>14</td>
<td>30</td>
<td>62</td>
<td>126</td>
<td>254</td>
<td>2\textsuperscript{n-2} [54]</td>
</tr>
<tr>
<td>Gate Count</td>
<td>5</td>
<td>13</td>
<td>29</td>
<td>61</td>
<td>125</td>
<td>253</td>
<td>509</td>
<td>2\textsuperscript{n-3} [54]</td>
</tr>
</tbody>
</table>

Table 8-2 shows the calculated LNNQC for MCT gates (3-9) calculated according to the equations in Theorem 8-2. Notice how the LNNQC increases drastically as the distance between the control and target bits increases, which in turn, makes gates with larger gaps between the target and control qubits unfavorable. Clearly, the application of LNNQC introduces a level of normalization across various methods of automated quantum synthesis. More specifically, solutions that attempt to reduce quantum cost, as measured today, through the introduction of many ancillary qubits would suffer a penalty for each distant gate introduced. With this proposal, I hope that such a penalty would
bring uniformity to quantum cost calculation, and would allow for *evenhanded comparison* amongst various classes of synthesis algorithms.

Table 8-2 Current quantum cost used in literature compared to LNNQC.

<table>
<thead>
<tr>
<th>Circuit Depth</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Cost</td>
<td>5</td>
<td>13</td>
<td>29</td>
<td>61</td>
<td>125</td>
<td>253</td>
<td>509</td>
<td>2^n-3</td>
</tr>
<tr>
<td>LNNQC</td>
<td>9</td>
<td>31</td>
<td>77</td>
<td>171</td>
<td>361</td>
<td>743</td>
<td>1509</td>
<td>Theorem 2</td>
</tr>
</tbody>
</table>

8.5 **Circuit Depth**

Although the *near neighbor quantum cost* is, at least, many orders of magnitude larger than the currently used *quantum cost*, it represents a realistic measure of physically implementing a quantum circuit which must adhere to the *nearest neighbor* constraint. In general, the LNN model levels the field of comparison amongst the various methods used by synthesis algorithms. For example, some algorithms tend to increase the circuit depth through the addition of ancillary bits in order to gain advantage of lower number of elementary gates, and hence, *lower quantum cost*. In such cases, these algorithms assume that any 2-qubit elementary gate has a quantum cost of *one* regardless of the distance between the endpoints of the gate. Some synthesis algorithms [20, 30] produce solutions which contain gates with distances of more than 200 qubits between the target and control qubits, which, as shown in the tables below, results in an enormous increase in LNNQC, and would render such functions untenable.

Consider for a moment that, to this date, the most advanced ion trap apparatus is only capable of lining up to 8 ions within the trap, and that every additional ion (qubit) represents a milestone in technical advancement [15]. Although, in essence, the LNNM
model balances the scale for the sake of comparison, one of its most important aspects is its compliance with the physical reality of a quantum machine. To further enhance the comparison amongst different classes of algorithms, it would be useful to understand the rate of increase in the circuit width. In this dissertation, I propose that the LNNQC be accompanied by another metric to highlight the increase of a circuit’s width (I call this metric the ancillary ratio). So instead of a single number to measure quantum cost, I would use a tuple of two numbers \((LNNQC, AR)\) where a) LNNQC is the number of linear nearest neighbor elementary gates, and b), AR is the ratio of number of ancillary bits to the original width of the circuit. An ancillary ratio of 1 represents a solution without any ancillary qubits; higher ancillary ratio reflects the percentage ancillary bits added to the circuit compared to the original width of the circuit. See Table 8-3 for examples.

8.6 Experimental Results

I analyzed the impact of enforcing LNN model on all MCT library functions on Revlib \([30]\) and documented some of the results in Table 8-3 below. In the table, the "Number of qubits" column indicates the width of the circuit including any ancillary bits. The "Ancillary Ratio" column reports the ratio of the circuit width to the number of the original input variables, where, as mentioned above, an ancillary ratio of one (1) indicates no ancillary bits added to the circuit. Notice that some of the solutions quadrupled the circuit’s width to achieve a low quantum cost as shown in the column labeled "RevLib". This cost represents the number of 1 and 2-qubit elementary gates (NOT, CV or CNOT) regardless of the distance between the end points. The results of applying our LNNQC
using the naïve method of theorem 1 (LNN1) and the optimized method of theorem 2 (LNN2) are shown in the table as well.

When I compare the RevLib quantum cost to the optimized LNNQC in column 5 (LNN2), I notice that some of the functions increased drastically (e.g., 517779% for the function $frgl$). This is clearly due to the use of many distant gates or large MCT gates which when decomposed result in many large distance gates. The optimized LNN2 cost is still much better than the naïve approach of theorem 1, which results in savings up to 1264%.

8.7 Conclusion and Analysis

A huge effort has already been invested in the field of automated quantum logic synthesis that is years ahead of the feasibility of any quantum computing devices. Yet, although some notable achievements have been observed in demonstrating the feasibility of quantum computation, technical challenges and constraints exist. One of the most notable constraints is the nearest neighbor restriction to interaction amongst qubits involved in a computing paradigm. As the field of quantum logic synthesis continues to be in its infancy, solutions using different methodologies continue to emerge. In this chapter, by imposing natural technological constraints, I propose a quantum cost metrics which would level the playing field amongst diverse types of quantum logic synthesis algorithms, and would allow for a more reasonable comparison amongst them. I hope that this body of work will inspire the scientific community to bring solutions that would one day be applicable to a future computing machine.
### Table 8-3: RevLib Benchmarks with LNNQC [30]

<table>
<thead>
<tr>
<th>Function</th>
<th># of qbits</th>
<th>Non LNNM Metrics (RevLib)</th>
<th>LNNM/QC</th>
<th>Ancillary Ratio</th>
<th>% Increase LNN2/QC</th>
<th>% Savings LNN1/LNN2</th>
</tr>
</thead>
<tbody>
<tr>
<td>apex4</td>
<td>22</td>
<td>3,438</td>
<td>139,594</td>
<td>53,742</td>
<td>2.20</td>
<td>1563%</td>
</tr>
<tr>
<td>apex4</td>
<td>28</td>
<td>237,963</td>
<td>7,945,022</td>
<td>2,676,178</td>
<td>3.11</td>
<td>1125%</td>
</tr>
<tr>
<td>C17</td>
<td>7</td>
<td>99</td>
<td>407</td>
<td>337</td>
<td>1.40</td>
<td>340%</td>
</tr>
<tr>
<td>C7552</td>
<td>21</td>
<td>1728</td>
<td>12476</td>
<td>8562</td>
<td>4.20</td>
<td>495%</td>
</tr>
<tr>
<td>cm150a</td>
<td>22</td>
<td>1096</td>
<td>17937</td>
<td>7765</td>
<td>1.05</td>
<td>708%</td>
</tr>
<tr>
<td>cm152a</td>
<td>12</td>
<td>252</td>
<td>1208</td>
<td>888</td>
<td>1.09</td>
<td>352%</td>
</tr>
<tr>
<td>cm163a</td>
<td>29</td>
<td>756</td>
<td>35547</td>
<td>10991</td>
<td>1.81</td>
<td>1454%</td>
</tr>
<tr>
<td>cm42a</td>
<td>14</td>
<td>377</td>
<td>1,787</td>
<td>1,379</td>
<td>3.50</td>
<td>366%</td>
</tr>
<tr>
<td>cm85a</td>
<td>14</td>
<td>2252</td>
<td>52923</td>
<td>19537</td>
<td>1.27</td>
<td>868%</td>
</tr>
<tr>
<td>cmb</td>
<td>20</td>
<td>910</td>
<td>622786</td>
<td>98190</td>
<td>1.25</td>
<td>10709%</td>
</tr>
<tr>
<td>con1</td>
<td>9</td>
<td>206</td>
<td>1151</td>
<td>833</td>
<td>1.29</td>
<td>404%</td>
</tr>
<tr>
<td>cordinate</td>
<td>25</td>
<td>349522</td>
<td>2963275837</td>
<td>327591467</td>
<td>1.09</td>
<td>93726%</td>
</tr>
<tr>
<td>cu</td>
<td>25</td>
<td>1148</td>
<td>151076</td>
<td>33484</td>
<td>1.79</td>
<td>2917%</td>
</tr>
<tr>
<td>d1</td>
<td>11</td>
<td>416</td>
<td>1,363</td>
<td>1,549</td>
<td>2.75</td>
<td>372%</td>
</tr>
<tr>
<td>d1c</td>
<td>11</td>
<td>416</td>
<td>1,363</td>
<td>1,549</td>
<td>2.75</td>
<td>372%</td>
</tr>
<tr>
<td>d2c</td>
<td>15</td>
<td>3886</td>
<td>33563</td>
<td>133727</td>
<td>1.88</td>
<td>728%</td>
</tr>
<tr>
<td>decod</td>
<td>21</td>
<td>1,728</td>
<td>12,476</td>
<td>8,562</td>
<td>4.20</td>
<td>495%</td>
</tr>
<tr>
<td>dist</td>
<td>13</td>
<td>7,601</td>
<td>246,881</td>
<td>75,643</td>
<td>1.63</td>
<td>995%</td>
</tr>
<tr>
<td>dk17</td>
<td>21</td>
<td>1,559</td>
<td>96,013</td>
<td>26,759</td>
<td>2.10</td>
<td>1716%</td>
</tr>
<tr>
<td>dk27</td>
<td>18</td>
<td>248</td>
<td>3614</td>
<td>1792</td>
<td>2.00</td>
<td>723%</td>
</tr>
<tr>
<td>ex1</td>
<td>6</td>
<td>7</td>
<td>47</td>
<td>63</td>
<td>1.20</td>
<td>900%</td>
</tr>
<tr>
<td>ex1010</td>
<td>20</td>
<td>155534</td>
<td>18207859</td>
<td>4341197</td>
<td>2.00</td>
<td>2791%</td>
</tr>
<tr>
<td>ex2</td>
<td>6</td>
<td>141</td>
<td>617</td>
<td>407</td>
<td>1.20</td>
<td>289%</td>
</tr>
<tr>
<td>ex2</td>
<td>6</td>
<td>141</td>
<td>617</td>
<td>407</td>
<td>1.20</td>
<td>289%</td>
</tr>
<tr>
<td>ex3</td>
<td>6</td>
<td>79</td>
<td>297</td>
<td>219</td>
<td>1.20</td>
<td>277%</td>
</tr>
<tr>
<td>example2</td>
<td>16</td>
<td>5,654</td>
<td>356,857</td>
<td>90,947</td>
<td>1.60</td>
<td>1609%</td>
</tr>
<tr>
<td>f2</td>
<td>8</td>
<td>255</td>
<td>1117</td>
<td>815</td>
<td>2.00</td>
<td>320%</td>
</tr>
<tr>
<td>f1im</td>
<td>22</td>
<td>37,400</td>
<td>21,966,577</td>
<td>3,524,427</td>
<td>1.57</td>
<td>9424%</td>
</tr>
<tr>
<td>flgi</td>
<td>31</td>
<td>15,265</td>
<td>999,144,350</td>
<td>19,038,936</td>
<td>1.11</td>
<td>51779%</td>
</tr>
<tr>
<td>init</td>
<td>26</td>
<td>20031</td>
<td>14314870</td>
<td>2381782</td>
<td>1.73</td>
<td>11890%</td>
</tr>
<tr>
<td>in2</td>
<td>29</td>
<td>23,802</td>
<td>11,908,647</td>
<td>2,095,751</td>
<td>1.53</td>
<td>8805%</td>
</tr>
<tr>
<td>inc</td>
<td>16</td>
<td>2,140</td>
<td>24,229</td>
<td>11,943</td>
<td>2.29</td>
<td>558%</td>
</tr>
<tr>
<td>lifc</td>
<td>10</td>
<td>6,766</td>
<td>130,289</td>
<td>39,383</td>
<td>1.11</td>
<td>582%</td>
</tr>
<tr>
<td>majority</td>
<td>6</td>
<td>136</td>
<td>624</td>
<td>374</td>
<td>1.20</td>
<td>275%</td>
</tr>
<tr>
<td>max46</td>
<td>10</td>
<td>5444</td>
<td>151559</td>
<td>45249</td>
<td>1.31</td>
<td>831%</td>
</tr>
<tr>
<td>misec1</td>
<td>15</td>
<td>982</td>
<td>7,735</td>
<td>4,857</td>
<td>1.88</td>
<td>495%</td>
</tr>
<tr>
<td>misec3</td>
<td>28</td>
<td>119177</td>
<td>95456974</td>
<td>15130206</td>
<td>2.00</td>
<td>12696%</td>
</tr>
<tr>
<td>misec3c</td>
<td>28</td>
<td>115,190</td>
<td>61,750,817</td>
<td>11,376,499</td>
<td>2.00</td>
<td>9876%</td>
</tr>
<tr>
<td>mlp4</td>
<td>16</td>
<td>3753</td>
<td>98689</td>
<td>34415</td>
<td>2.00</td>
<td>917%</td>
</tr>
<tr>
<td>mux</td>
<td>22</td>
<td>1078</td>
<td>17835</td>
<td>7693</td>
<td>1.05</td>
<td>714%</td>
</tr>
<tr>
<td>parity</td>
<td>17</td>
<td>32</td>
<td>512</td>
<td>544</td>
<td>1.06</td>
<td>1700%</td>
</tr>
<tr>
<td>pcler8</td>
<td>21</td>
<td>327</td>
<td>4890</td>
<td>2334</td>
<td>1.31</td>
<td>714%</td>
</tr>
<tr>
<td>pm1</td>
<td>14</td>
<td>377</td>
<td>1,787</td>
<td>1,379</td>
<td>3.50</td>
<td>366%</td>
</tr>
<tr>
<td>radd</td>
<td>13</td>
<td>676</td>
<td>5096</td>
<td>3494</td>
<td>1.63</td>
<td>517%</td>
</tr>
<tr>
<td>rd3</td>
<td>3</td>
<td>29</td>
<td>67</td>
<td>95</td>
<td>1.67</td>
<td>328%</td>
</tr>
<tr>
<td>rd32</td>
<td>5</td>
<td>18</td>
<td>56</td>
<td>86</td>
<td>1.67</td>
<td>478%</td>
</tr>
<tr>
<td>rd32</td>
<td>5</td>
<td>116</td>
<td>292</td>
<td>252</td>
<td>1.67</td>
<td>217%</td>
</tr>
<tr>
<td>rd33</td>
<td>8</td>
<td>265</td>
<td>1,283</td>
<td>949</td>
<td>1.60</td>
<td>358%</td>
</tr>
<tr>
<td>rd73_66</td>
<td>10</td>
<td>1143</td>
<td>9750</td>
<td>5338</td>
<td>1.43</td>
<td>467%</td>
</tr>
</tbody>
</table>

156
Chapter 9 Multi-Dimensional LNNM Architecture

9.1 Introduction

Jordan in his thesis [55] described an algorithm for computing the ground state of infinite 2D quantum lattice system and the utility of a 2D lattice in quantum computation. Jordan described a lattice structure where neighboring qubits, arranged in rows and columns, would interact in compliance with the nearest neighbor model. Choi, et al. [92] provided an algorithm for constructing a 2-D NTC (Nearest Neighbor Two qubit gate) and demonstrated the construction of a full adder using this architecture. The proposed architecture lays out the qubits in a lattice pattern where, in addition to neighboring qubits interacting, Choi et al. allows interaction in the diagonal direction as well.

In this chapter, I chose to follow the construction of Jordan [55] allowing interaction only between neighboring qubits in the same row or column, but not across diagonals. In the description of the Paul’s trap in section 3.5.2.1, where I discussed the construction of a CNOT gate in the linear ion-trap, I described that the two qubits are entangled together through the common vibrational mode of qubits along the axis of the ion-trap. In order to accomplish entanglement in a 2-D ion-trap, a similar common mode would have to occur independently in both row and column direction in order to allow for 2-qubit gate interaction. Unless demonstrated otherwise, I believe that the likelihood of attaining two common vibrational modes in the orthogonal directions of rows and columns of a 2-D ion-trap lattice is higher than the possibility of adding two additional common vibrational modes in the diagonal directions as proposed by Choi [92].
The discussion in this chapter is limited to the layout of a Multiple Control Toffoli (MCT) gate in a two dimensional grid and the calculation of the Planer Nearest Neighbor Quantum Cost (PNNQC) using such layout.

9.2 MCT₄ in 2-D

Figure 9-1 MCT₄ in two dimensional grid layout where, now, qubits ‘a’ and ‘d’ are considered neighbors and the first CV gate is compliant with the LNNM model (a cost of 1) compared to the linear arrangement which requires 8 additional CNOT gates to bring it into compliance. Notice that the pairs ‘ac’ and ‘bd’ are still one qubit apart.

Figure 9-1 shows the internal layout of the MCT₄ gate from the previous chapter where the qubits are laid out in a 2-D grid pattern. In a linear ion-trap, the qubit pairs (a,c), (a,d) and (b,d) are considered distant pairs and require swapping of values to bring them next to one another for interaction. In the two dimensional grid layout, in accordance to the model outlined by Jordan [xx], the pairs (a,d) become neighbors and can directly interact. I calculated the Linear Nearest Neighbor Quantum Cost (LNNQC) for the MCT₄ at 31 single and double qubits gates. The two dimensional (Planer) Nearest Neighbor Quantum Cost (PNNQC) is shown in Figure 9-1 as 29 primitive gates, which, at 9%, is not a significant saving. With this arrangement, interaction between the pairs (a,c) and (b,d) require the addition of single set of swap gates to bring them next to
one another, and another set to bring their values back to position (at a cost of 4 CNOT gates).

Examining the relationship between the qubits, I noticed that qubits $a$ and $d$ interact the most with the other qubits. I also noticed that qubit $a$ always interacts as a control line and never as a target line. Qubit $d$ on the other hand is always a target line and never a control line, while all other bits in between interact as both control and target. Figure 9-2 shows another two dimensional arrangement of the qubits where I added two ancilla qubits and copied the duplicated the value of qubit $a$ into both of them. The idea here is that since qubit $a$ does not change throughout the operation, it should be safe to copy it and bring its value close to the other qubits in order for it to interact in accordance with the nearest neighbor constraint. In this arrangement, only the pair $(c, d)$ are a two qubits away requiring a set of swap gates (along with their mirrors) to interact. The $PNNQC$ in this case is 21 primitive gate which represents a 34% saving. In this case, of course, I have to add two ancilla qubits, which might not be desirable.

![Diagram](image)

*Figure 9-2 MCT$_2$ in a planer layout with qubit ‘a’ duplicated into two additional ancilla qubits in order to bring pairs ‘ac’ and ‘bd’ next one another. Notice that with this arrangement, ‘b’ and ‘c’ are now the only pair which are one qubit apart. Using this arrangement reduced the quantum cost further to 21 (rom 29) at a cost of 2 ancilla qubits.*
I now extend this line of reasoning to the MCT₅ gate as shown in Figure 9-3. In this arrangement, the target qubit \( e \) is placed at the center of a 2-D grid because this qubit has interaction with all other qubits and it is the final recipient of all transforms. This arrangement, however, puts a distance between the pairs \((a,b), (b,c), (c,d)\) which are typically neighbors in a the one dimensional arrangement of the previous chapter. This distance interaction result in a negative impact on the \( PNNQC \) (85) as compared to the \( LNNQC(77) \) – a 10% increase in quantum cost.

Figure 9-3 MCT₅ gate with a star-shaped 2-D layout centered around the target qubit \( e \). The \( LNNMQC \) for this arrangement (85) is actually worse than the linear arrangement of qubits (77) since now qubits \( a', b', c', \) and \( d' \) are each remote to one another which is not the case for the linear arrangement.

Figure 9-4 shows another arrangement of the MCT₅ gate where, similar to the MCT₄ gate above, three additional ancilla qubits are added and the \( a \) qubit value is copied into them. In this case, the \( a \) qubit is a neighbor to all other qubits and all interactions require a single primitive gate. The \( PNNQC \) cost in this case is 53 primitive gates which represents a 32% saving over the LNNM model of the previous chapter.
Figure 9-4 A modified planer arrangement of the MCT5 gate with the duplication of qubit ‘a’ into additional ancilla qubits. Since qubits ‘a’ and ‘e’ interact with all other qubits, our strategy is to bring these two qubits as close to the others as possible. Since qubit ‘a’ never acts as a target qubit, it is easy to just mirror its value upfront to other ancilla qubits which will interact in a ‘neighborly’ manner with the other qubits. This pattern reduced the quantum cost to 53 compared to the linear arrangement of 77.

In the previous arrangement I noted that qubit b interacts with both qubits c and d remotely because of the position it occupied. Once the interaction of a and b is concluded after the first few gates, qubit b assumes a similar role to qubit a where it only acts as a control. At this stage, I can transport qubit b to a new location in nearness to both qubits c and d as shown in Figure 9-5. Since the original location of qubit b is a distance 3 from the new position, it takes 6 CNOT gates to transport it to the new location. Once transported, all interactions of qubit b with both qubits c and d require a single primitive gate at a cost of one(1). At this rate, the final PNNQC of this arrangement is 43 primitive gates which represent a saving of 44% over the LNNQC. The cost of this arrangement, however, is three additional ancilla qubits.
Another improvement of the planar arrangement of the MCT5 gate where qubit ‘b’ was transported to another location which makes it next to both qubits ‘c’ and ‘d’. This shuttling of qubit ‘b’ can only occur after the first stage once qubit ‘b’ is no longer a target. The LNNM quantum cost is further reduced to 43 from 53.

For larger size of the MCT gate, a similar analysis could be followed for a layout of the gate in a two dimensional grid with the ability to relocate qubits midway in order to facilitate near neighbor interaction amongst qubits while reducing quantum cost. The number of ancilla bits to be added, however, will grow with the size of the gate which would increase the ancillary ratio discussed in the previous chapter.

9.4 Conclusion and Analysis

I introduced and analyzed the application Nearest Neighbor Model to a two dimensional grid which proved to be beneficial with respect to quantum cost. However, the most benefit of a two dimensional arrangement typically comes at the cost of adding ancilla bits which is not always desired as it would increase the cost of constructing the system. It is possible, of course, to add additional ancilla bits to the linear model of the last chapter, and also reduce the quantum cost in a manner similar to what was done using the 2-D layout. The facts that the 2-D layout has yet to be implemented in the lab and that it is also possible to benefit from the addition of ancilla bits in the linear
arrangement leads to believe that the linear arrangement is still the most useful method to pursue for future LNNMM compliant synthesis algorithms.
PART IV

QUANTUM LOGIC SYNTHESIS

OF

MULTIPLE VALUED FUNCTIONS
Chapter 10 Synthesis of Ternary Quantum Circuits

10.1 Prologue

In the realm of classical technology, the irreversibility of digital logic gates results in information loss, which manifests itself as heat dissipation. Landauer proved that using irreversible logic gates yields a rate of energy loss proportional to $kT$ [1]. Essentially, information equals energy, and the loss of it equals heat loss. Computations that preserve information are considered reversible and gates that perform reversible computation are designated as reversible gates. Bennett [3] showed that a near-zero energy dissipation is possible when a computer can operate near its thermodynamic equilibrium and further displayed that such a stasis state can be achieved through reversible components. Toffoli [4] showed that quantum logic gates are inherently reversible and demonstrated a set of universal quantum binary primitive capable of implementing any logic circuit - namely, NCT library (Not, Controlled-Not and Toffoli gates). The qubit came to represent the quantum analogy of the classical symbol of information carrier: the bit. Possibly years before the feasibility of mass production of quantum computers, researchers have been laying the foundation for manufacturing such a computing device by exploring automated synthesis algorithms of quantum logic circuits. In this chapter I tackle the problem of quantum logic synthesis for ternary quantum logic.

Ternary-valued logic represents information in a base 3 system with three base states \{0, 1 and 2\} where a qutrit (trit for short) is a quantum unit of information with three basis states. Qutrit is the ternary equivalent of the binary qubit. A ternary-valued $m$-variable reversible logic function maps each of the $3^m$ input terms to a unique output.
term; or mathematically speaking, it is an **onto** and **one-to-one** function or a **bijection**.

The problem of synthesizing a reversible circuit is the process of constructing a cascade of ternary reversible gates, which maps each, said input term to its corresponding unique output term. Mathematically, quantum circuit synthesis represents a decomposition of circuit’s specification to a number of small permutations of reversible gates.

In their work, Miller, Maslov and Dueck [10], henceforth MMD3, presented exhaustive results for all \((9!)\) permutations of a two-variable ternary reversible functions. They further illustrated a synthesis example of the inherently irreversible 3-trit full adder by adding a single ancillary trit to create a 4-trit reversible function, and then applying their synthesis algorithm to such function.


Our major contributions in this chapter are the following:

1. Demonstrate the advantage of exploring different input vector sequences on the synthesized circuit quantum gate cost.

2. Outline an algorithm for constructing valid input vector sequences using a ternary Hasse structure and provide a proof of algorithmic convergence for such sequences.
3. Establish a set of benchmark cost numbers of synthesis of large ternary functions up to 9 variables. Alhagi, et al [22], defined large binary functions as those consisting of eight (8) or more binary qubits whose information could easily be contained within 5 trits. A nine trit register carries the equivalent of 14.25 binary bits of information.

4. Introduce the set of Hidden Weighted Trit (HWT) functions into the literature as a set of benchmark functions for ternary logic synthesis. HWT functions are an extension to the Hidden Weight Bit (HWB) benchmark functions [20, 30, 96] extensively used in the literature as one of the harder binary benchmarks functions.

5. Establish an open source repository [29] for the scientific community to hold and share the set of benchmarks for ternary functions.

For the sake of self-containment, section 0 introduces the domain of ternary logic and reversible gates followed by an example of the operation of the algorithm for a two variable ternary function in section 10.4. Section 10.5 provides a detailed explanation of our ternary logic synthesis algorithm where it describes the concept of control line blocking, and demonstrates how to construct Hasse diagrams and generate input vector sequences from such diagrams. Sections 10.6 and 10.7 provide proof of convergence of the algorithm for three classes of ternary precedence orders. Section 10.8 gives details of the genetic algorithm followed by an analysis of the experimental results and conclusion.
10.2 Ternary Logic System

10.2.1 Measurement of a qubit

The theory of quantum mechanics depicts the qubit as a quantum state which could exist in a state of superposition between the two basis states of \{0, 1\}. However, upon observing the state of such a register, i.e., measure its value, the qubit loses its state of superposition and collapses to one of the two basis states. Due to the quantized nature of the particle used for computation (polarization of a photon, presence of an electron ...etc.), a detector monitoring a value of zero(0) would either observe the particle (a one) or not (a zero), but nothing in between. For example, placing a set of two orthogonal polarization filters in the path of a stream of photons would polarize some of the photons along one axis (basis state zero) and the rest along the other axis (basis state one). In essence, measurement of a qubit in the superposition state of \(\alpha|0\rangle + \beta|1\rangle\) forces the qubit to collapse to a zero with a probability of \(|\alpha|^2\) or to a one with a probability of \(|\beta|^2\).

Visually, Figure 10-1 depicts measurement as a projection of the vector representing the superposed state onto the vectors representing the two basis states \{0, 1\}. Similar to any probabilistic computation, a quantum computation is typically performed on an assembly of quantum systems consisting of a large number (N) of identical quantum circuits, all initialized to the identical set of input values. Measurement is performed by exposing half of the particles to the zero detector and the rest to the one detector, where, probabilistically, the sensor detecting the highest number of hits reflects the internal state of the quantum register. Such a binary detection system could be extended to ternary logic where three measurements are sought after, \{0, 1, 2\}.
Measurement of qubit in an optical quantum system can be performed by placing horizontal and vertical filters along with photon detectors on two orthogonal axis for detecting the number of photons on each point. One of the detectors is assigned the state $|0\rangle$ and the other state $|1\rangle$.

10.2.2 Trits and Ternary States

Ternary logic is the closed logic system with certain ternary operators that operate on three logic values $\{0, 1, 2\}$. In quantum mechanics, the three ternary values could correspond to the different polarization of a photon or alignment of nuclear spin in a uniform magnetic field. To date, Nuclear Magnetic Resonance (NMR) and Ion Trap are the most promising technologies which were used to demonstrate “quantum circuit model” of quantum computation.

Definition 10-1: A ternary quantum bit, a trit, is a ternary quantum system defined over the Hilbert space $\mathcal{H}^3$ with basis states $\{|0\rangle, |1\rangle, |2\rangle\}$, which are represented with the following Heisenberg vector notation:

$$
|0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, |1\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, |2\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
$$

A two-variable register consists of two trits which has an information capacity of $3^2 (9$ possible distinct states) represented as follows:
where the $\otimes$ symbol represents the mathematical tensor (Kronecker) product of the two trits. Consequently, an $n$-trit ternary register is a vector of n-ternary trits with a capacity of $3^n$ states that are represented by the following equation:

$$\Psi(t) = \bigotimes_{i=0}^{n} |\phi_i\rangle \quad (2)$$

where $\Psi(t)$ represents the state of the system at time ($t$) and $|\phi_i\rangle$ is the state of trit ($i$).

10.2.3 Reversible Operations

**Definition 10-2:** A $k$-variable ternary reversible gate (operator) is a bijective, one-to-one and onto mapping of every permutation of the $3^k$ input patterns.

Unlike classic logic, quantum logic circuits are inherently reversible and can only be constructed from reversible logic gates. Logical reversibility is the ability to reconstruct the input of a function from its output, and vice versa. The definition above stipulates such reversibility with the one-to-one mapping where each input term is mapped to a single element of the output, and vice versa. The onto stipulates that all elements of the output set are used, and hence, there are the same number of elements in the input and
output sets. The third requirement is of closure where the range and domain of the function are identical sets. The reader can easily deduce that, with definition 2, the set of output terms is simply a permutation of the input terms where each set includes unique set of elements.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(b)

Figure 10-2 (a) non-reversible functions f and g since separately nor combined (fg) since the value 01 is repeated twice on the output and there is no way to reconstruct the input; (b) ax is reversible function since it is possible to determine the input minterm for each output minterm.

To illustrate, Figure 10-2 (a) shows the two logical functions AND (f) and OR (g), which are, separately and jointly, irreversible. Both functions map the two inputs, a and b, to a single output, f or g, making it impossible to reconstruct the input pair from the single output. Since these gates only have a single output, one of their inputs has effectively been erased and the information it carries has been lost. Figure 10-2 (b), by contrast, represents the logical XOR function x which, taken alone, is unidirectional and irreversible. However, when x is combined with a copy of input a, the combined pair a:x represents a reversible function where each input term, a:b, maps to a single unique output term, a:x, and vice versa. ■

In binary logic, only two reversible gates (operators) exist: a wire representing identify, and an inverter representing negation. As with all reversible gates, these binary operators have the same number of input and output variables, one variable, which uniquely map each input value to an output value as shown in Figure 10-3.
Notice that, mathematically, the two functions represent the ordered set of all permutations of the input values, 0 and 1: \{\text{wire}: (0, 1), \text{inverter}: (1, 0)\}. By corollary, the ternary values \{0, 1, 2\} can be fully permuted into six unique sequences yielding a total of six unique ternary operators representing the ordered set of the ternary values: \{(0,1,2), (0,2,1), (1,0,2), (1,2,0), (2,0,1), (2,1,0)\}.

<table>
<thead>
<tr>
<th>In (a)</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wire (a)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NOT (a)</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

*Figure 10-3 Symbols for quantum binary Identity (wire) and NOT gates.*
10.3 Ternary Reversible Operators

Figure 10-4 lists the six ternary operators for a single ternary variable with their names in the first column, mathematical equation in the second column, truth table in the third column, and the symbolic notation in the last column. Clearly, the [+] 0 operator is the analogy of a wire. The [+] 1 and [+] 2 operators are ternary inverters which perform a mathematical summation of the gate's inputs modulo 3. The [12], [02] and [01] operators swap their namesake input values without affecting the third ternary value. For example, the [12] gate swaps the values one and two while leaving zeros alone.

<table>
<thead>
<tr>
<th>Gate</th>
<th>F(a)</th>
<th>Input (a)</th>
<th>Gate symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0</td>
<td>(a+0) mod 3</td>
<td>0 1 2</td>
<td>a ——— a</td>
</tr>
<tr>
<td>+1</td>
<td>(a+1) mod 3</td>
<td>1 2 0</td>
<td>a ——— a&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td>+2</td>
<td>(a+2) mod 3</td>
<td>2 0 1</td>
<td>a ——— a&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td>12</td>
<td>2(a+0) mod 3</td>
<td>0 2 1</td>
<td>a ——— a&lt;sub&gt;12&lt;/sub&gt;</td>
</tr>
<tr>
<td>02</td>
<td>2(a+1) mod 3</td>
<td>2 1 0</td>
<td>a ——— a&lt;sub&gt;02&lt;/sub&gt;</td>
</tr>
<tr>
<td>01</td>
<td>2(a+2) mod 3</td>
<td>1 0 2</td>
<td>a ——— a&lt;sub&gt;01&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

Figure 10-4 Generalized Ternary Gates with the gate name given in the first column and the mathematical equation for calculating the outputs based on the current input. The third column shows the output for the three possible inputs shown in the header, and finally the gate symbol is shown in the last column.

In an ion-trap system, a ternary system can be implemented in a similar manner as I described in section 3.5.2 above where I outlined the operation of a binary system with two the basis states |0⟩ and |1⟩ represented as the energy levels |g⟩ and |e⟩. The ternary energy states shown in Figure 10-5 can be used to implement the ternary quantum states |0⟩, |1⟩ and |2⟩. Ternary gates can be implemented by the application of the appropriately detuned laser beam to perform state transitions from one state to the next.
For example, the [[01]] gate can be performed by the application of a laser beam of $w_{01}$ frequency which will transition qubits in state $|0\rangle$ to state $|1\rangle$ and qubits in state $|1\rangle$ to state $|0\rangle$ while leaving qubits in state $|2\rangle$ unchanged. The [[12]] gate can be implemented in a similar manner with the application of a laser pulse of $w_{12}$ frequency.

Figure 10-5 Ternary energy states of an ion-trap system. In (a) transitions between any two states is possible with a single laser pulse which would require three finely detuned laser rays targeted at each ion. In (b) only transitions between neighboring energy levels are possible but only two laser rays are required.

Implementing the [[12]] gate depends on the construction of the ion-trap. Figure 10-5a shows an ion-trap construction with three laser beams allowing a direct implementation of the [[12]] gate with a single application of a laser pulse of $w_{12}$ frequency. The implementation shown in Figure 10-5b requires two pulses to implement the [[12]] gate going through the $|1\rangle$ state.

**Definition 10-3:** A controlled gate is a logic gate consisting of $n$ variables where the values of $(n-1)$ control variables enable the operation on the target variable ($n$).

**Definition 10-4:** Control lines are independent variables where a specific pattern of values on the set of control lines affects the operation on a dependent variable (target line).
**Definition 10-5:** A target line is a dependent variable where a specific operation on the line is enabled iff a set of control lines matches a specific pattern; otherwise, the signal on the line is passed through unchanged.

Figure 10-6 (a) Ternary Inverter vs. (b) binary Inverter; (c) ternary controlled op (C-OP) vs. (d) Feynman CNOT; and (e) ternary (C^2-OP) vs. (f) Toffoli gate (C^2-NOT).

Figure 10-6 (a) shows the ternary extension to the binary inverter gate of Figure 10-6(b). In the ternary case, however, the \([+1]\) and \([+2]\) act as inverters to one another which, visually, a \([+1]\) followed by a \([+2]\) operator represents a cumulative rotation of 360° around the Bloch sphere bringing the atomic particle to its original orientation. Figure 10-6 (c) shows the ternary extension to the binary C-NOT gate in Figure 10-6 (d). In this case, a value of one (1) on control line (a) activates the \([+2]\) operator on line (b) while other values on line (a) would pass (b) unchanged. Of course any of the five ternary operators could be used on the target line (b) resulting in five representations
of the controlled gate. Additionally, the control line (a) could theoretically utilize any of the ternary values \{0, 1, 2\} as a control signal resulting in a total of fifteen (15) representatives of the Feynman gate in the ternary domain. Similarly Figure 10-6 (c) shows a ternary equivalent of the Toffoli gate where the operator $[12]$ affects the target line (c) only if (a) and (b) are both one (1). The reader can easily deduce, through the same argument above, that a total of 45 representatives of the Toffoli gate exist in the ternary space. Al-Rabadi [95] and Khan [93] considered several multi-valued (MV) and ternary gates including the mod-sum gates used in this chapter. Miller et al [10] used ternary gates identical to the gates shown in Figure 10-6. and they limited the control line to the value of one (1).

10.4 Synthesis by Example

Before I delve into the details of the ternary synthesis algorithm, it would be helpful to start with an illustration of the process as described by Miller, et al [10]. Figure 10-7 shows a reversible ternary function of two variables composed of 9 terms ($3^2$). Column (ab) represents the input vector and column (AB) represents the corresponding output vector. The objective of the synthesis is to create a cascade of primitive reversible ternary gates to map all input minterms to their corresponding output minterms. The algorithm terminates when all terms of the output vector (AB) map to their corresponding terms of the input vector (ab) – compare column 8 to column 1. The algorithm processes the terms one trit at a time and places gates only when the input/output trits of the same position mismatch. The algorithm observes a simple, yet essential, guiding principle stating that: A completely mapped pair should never be altered by succeeding mapping
This important rule assures that the algorithm will always converge, which is an essential criterion for synthesizing arbitrary reversible circuits.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>A</th>
<th>B</th>
<th>+1</th>
<th>+2</th>
<th>+1</th>
<th>+2</th>
<th>2</th>
<th>+2</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

**Figure 10-7** Ternary Synthesis Example transforming output vector AB (column 2) into input vector ab (columns 1 and 8). The \([+1]\) in the header represents the ternary gate inserted at that point which is also shown pictorially in the circuit below the column. The shaded values indicate the values impacted by the gate; the \([+1 1]\) in the header of column 5 indicate that the lower qutrit is used as a control of value ‘1’ while the upper bit has the \([+1]\) gate as shown in the pictorial below the column.

Figure 10-7 illustrates the step by step synthesis process along with the circuit diagram for each transformation, as follows:

- Considering the inherent reversibility of the function, the algorithm starts synthesis from the output column (AB) towards the input column (ab).
- Starting with first pair (00 \(\rightarrow\) 12), the algorithm realizes that a \([+1]\) inverter on line (a) would correctly map the upper trit 0 to 1. Essentially, any value presented on the (a) line will be incremented by 1 modulo 3, as shown in the shaded text of the third column.
• A second gate, $\llbracket +2 \rrbracket$ is placed on line (B) bringing its value from two to zero matching the corresponding input line (b). Notice that due to their unconditional nature, the above two gates affect all the terms of the output vector as demonstrated by the shaded values.

• The synthesis process continues with the second term where the upper trit of input term (01) mismatches the newly realized output term (11) in column 4. The algorithm places a $\llbracket +1 \rrbracket$ gate on line (A) to perform a $1 \rightarrow 0$ transformation only if line (B) has the value of one; hence, a controlled gate. As stated before, the control values (highlighted with thick borders) are used to ensure that completely mapped pairs are never modified by a later step.

• The third input term (02) now maps to (22). A $\llbracket +2 \rrbracket$ gate on line (a) controlled by a value of (2) on line (b) remedies the mismatch of the upper trit.

• The fourth input term (10) now maps to (22) requiring two gates to correct. The first gate is a controlled $\llbracket +2 \rrbracket$ on line (b) controlled by a value of (2) on line (a).

• To correct the upper bit, I realize that a $\llbracket [12] \rrbracket$ swap gate on line (a) would correctly map that line and the remaining minterms of the function.

• Realizing that column 1 is identical to column 8, the synthesis algorithm terminates with a successful synthesis of the specification.

Taking a deeper look at the synthesis example, the reader would surely discover a discordance between the operator indicated on top, e.g., $\llbracket +1 \rrbracket$, and what the table indicates as the result of such an operation (shaded). For example, with the pair (00, 12), performing $\llbracket +1 \rrbracket$ on the trit value of ($A=1$) would surely yield a value of ($A=2$) not the
(A=0) indicated in the first row of the table, column 3. To alleviate this disagreement, remember that I have started our synthesis process from the output vector (AB), column 2, heading towards the input vector (ab), replicated in column 8. And why not, this is a reversible circuit after all! So, in truth, the operation $[+1]$ answers the question, “what do I need to make the input trit of value ($a=0$) into the output trit of value ($A=1$)?” A $[+1]$ of course! And that is exactly what I have specified. I could have easily started from the input vector (ab) and synthesized the circuit by adding gates to match the output in a similar manner and would have surely ended up with a circuit functionally equivalent to the one shown above.

10.5 Ternary Logic Synthesis Algorithm

I was inspired by our research in the binary domain [22] which revealed that the size of the resulting circuit is greatly influenced by the arrangement of terms of the input vector. In the above example, the terms of the input vector (ab) are arranged in their natural ternary order. The remainder of this chapter answers the questions: Is the size of the circuit influenced by the arrangement of the input vector? Do such arrangements affect algorithmic convergence?

I will demonstrate below that the answer to both questions is in the affirmative where the arrangement of the input vector influences the number of gates required to implement the circuit. I will also demonstrate through an example the concept of control line blocking where, for a subset of such input orderings, the algorithm becomes trapped in an endless loop. It is worthy to mention, for the sake of completeness, that our algorithm for
synthesizing reversible circuits adheres to the basic assumption for quantum circuits as outlined by Toffoli [4] below:

1. No fan-out is permitted between gates,
2. Loops are not permitted,
3. Permutations of connections between gates are permitted.

10.5.1 Control Line Blocking

Some minterm orderings of the input vector violate the bedrock principle of never altering previously completed mapped pairs, and forces the algorithm into an infinite loop. Using the same steps of section 10.4 above, Figure 10-8 lists the synthesis of the first three minterms of the input sequence: 00 22 02. Once the second minterm is mapped correctly, trying to map the third pair (20 \(\rightarrow\) 02) becomes impossible without altering the first two completely mapped pairs. For example, in order to map the lower trit correctly (0 \(\rightarrow\) 2), I would normally use the upper trit (2) as a control signal and possibly apply swap gate \([02]\) to provide the correct mapping. However, such an operation would surely alter the completely mapped pair of the second row (22 \(\rightarrow\) 20), and in effect violates the aforementioned principle. The reader can easily deduce that going back and attempting to correct the term of the second row will result in an infinite loop. Similarly, attempting to use the lower trit (0) as a control signal is also destructive where, in this case, the first completely mapped pair would be altered.
**Definition 10-6:** Control Line Blocking condition occurs when all control lines of the current minterm are a subset of the control lines of a previously completed minterm for a given input order.

It is possible, of course, to programmatically detect when a previously mapped pair has been altered, and, consequently, reject the input sequence. Going through all possible permutations of input arrangements would surely guarantee the discovery of the most optimal solution. Even for functions with a small number of ternary variables, however, attempting all permutations is impossible. A lowly 3-variable ternary function consists of 27 \( (3^3) \) minterms resulting in 27! \( (10^{28}) \) possible permutations: not an easy feat even for our most powerful computing machines. The question then is: would it be possible to only focus the search on sequences which are guaranteed to converge?

![Table](image)

**Figure 10-8** When the algorithm reaches the shaded minterm which needs to be transformed from 20 to 02. If the upper trit is used as control to avoid changing the second minterm, any gate applied to change the lower trit from 0 to 2 will also change the first minterm. The same thing will happen if I use the lower trit (0) as control where in this case, the second minterm will be altered. Either way, this is a violation of the algorithm and the algorithm is blocked at this point.

10.5.2 Ternary Hasse Input Sequence

Mathematics comes to the rescue! It is possible to construct a subset of all possible convergent sequences using the mathematical concept of Hasse diagrams and covering graphs. Rather than cycling through the entire set of permutations, I could easily
construct a number of such valid input sequences and discover the ones which provide
the circuits with the lowest quantum cost.

**Definition 10-7:** a Hasse or Poset diagram is a type of mathematical diagram used to
represent a finite partially ordered set, in the form of a graph where, for the relation
\( \{ (x, y) \mid x \leq y \mid x, y \in S \} \), each element of \( S \) is a vertex in the plane and draws a line
segment or curve that goes upward from \( x \) to \( y \) whenever \( y \) covers \( x \) (that is, whenever \( x < y \) and there is no \( z \) such that \( x < z < y \)). The relations \(<\) and \(\leq\) represent a precedence
hierarchy between the operands and not necessarily analogous to the mathematical
inequality relations on real number.

I will start with a demonstration of constructing a Hasse diagram for a two variable
ternary function – see Figure 10-9, Starting from smallest valued minterm (00), I draw a
line to each of the two minterms which satisfy the relation of partial ordered sets: \( \{ (x, y) \mid x \leq y \mid x, y \in S \} \). Loosely speaking, I find all minterms which are trit-wise larger
than the minterm at hand. For 00, adding a 1 to the lower trit yields 01, and adding a one to
the upper trit yields 10 – shown in Figure 10-10(a).

In a similar fashion, the 01 minterm would yield 02 by adding a 1 to the lower trit, and
a 11 by incrementing the upper trit – Figure 10-10 (b). The process repeats for all the
terms in the set until the highest number 22 is reached. Consider for a moment the upper
trit of term 02 in Figure 10-10 (c). Notice that for the branch (00\(\rightarrow\)01\(\rightarrow\)02), each
transition only affects a single trit and that, for the sake of maintaining closure within the
ternary domain, the process stops once a trit reaches the value of 2. Now that the lower
trit for minterm 02 has reached the ceiling of 2, the upper trit can transition through its stages (02 → 12 → 22).

Figure 10-9 Structure of the ternary Hasse diagram for a 2-variable function. Each level (band) contains the set of minterms which has the same mathematical sum of digits. The sum of digits represents the band number shown on the right hand side.

10.5.3 Construction of Input Sequence

Once I have constructed the Hasse structure, I group all minterms at the same level together, within a set of bands, as shown graphically in Figure 10-9.

**Definition 10-8:** A Hasse Ternary Band is the set of terms at the same level in a ternary Hasse diagram where the sum of trits of each term equals the zero-based numerical order of the band.

The column on the right of Figure 10-9 shows the sum of each trit in a band. For example, band 3 has the terms 12 and 21 which both add up to 3, and hence, they are in band 3.
Corollary 2: A ternary function with $n$ variables has $2n+1$ Hasse Ternary Bands.

Since the highest band has the single minterm of $n$ number of the digit 2, the sum of all digits is clearly $2n$. Since the sum is zero based, according to Definition 10-7, the number of bands is $2n+1$.

At this stage, I am able to use the Hasse Ternary Bands to construct input vectors which are guaranteed to converge. The following pseudo-code outlines the process:

```plaintext
inputSequence := {}
for index := 0 to 2*n
    bandSequence := Permutation (Band[index])
    Append (inputSequence, bandSequence)
end for
```

The above pseudo-code can be described by the following steps:

- Step 02: Start at the lower band consisting of all zeros (0…0), and stop at the upper band consisting of all twos (2…2),

Figure 10-10 Construction of the Ternary Hasse Diagram for a 2-variable ternary function. (a) Starting from the bottom with 00, I add two new minterms by adding 1 to each digit resulting in (01, 10). (b) for each new minterm (e.g. 01), I repeat the process where I add 1 to each digit resulting in (02, 11). (c) For each digit, the process is repeated until the upper value ‘2’ is reached.
• Steps 03, 04: For each band, append any permutation of the terms within the band to the end of the sequence,

Observe that, for a two variable function, the combined permutations of the bands will result in 24 valid input sequences (2!·3!·2!). The following two vectors are examples of valid input sequences:

• \( S_1 = \{00, 01, 10, 20, 02, 12, 22\} \)
• \( S_2 = \{00, 10, 20, 02, 12, 21, 22\} \)

The alert reader will readily notice that, in constructing the input sequence, the \( 0 \rightarrow 1 \rightarrow 2 \) precedence order, defined below, is not necessarily obeyed as prescribed in section Ternary Hasse Input Sequence above. For instance, the vector \( S_2 \) includes the term \( 10 \) with a high trit of one (1), followed by \( 01 \) with a high trit of zero (0); yet, I consider this sequence valid. Notice that since the precedence criteria (of \( 0 \rightarrow 1 \rightarrow 2 \)) applies only to the construction of the Hasse diagram, and not to the construction of the input vector from the Hasse diagram. The only restriction for constructing the input vector from a Hasse diagram is: All minterms of a lower band must be used before any minterms in the next higher band. Clearly the algorithm described above satisfies both conditions.

**Definition 10-9:** Precedence order refers to the mathematical binary relation between a set of elements in a partially ordered set (Poset) where one element precedes the other. “Partial orders” reflect the fact that not every pair of elements of a Poset need be related.
Now that an input vector has been constructed, I apply the same synthesis process detailed in the “Synthesis by Example” section above. What are the advantages of this algorithm then? With this algorithm, I am now able to: 1) systematically construct multiple input vector arrangements which are guaranteed to converge, 2) without searching through every possible permutation of the input vector. This allows us to examine a large number of input vector arrangements in order to discover the circuit with the best quantum cost among the input vectors. Most likely, however, such a solution will not be the optimal circuit realization as the number of possible input vector arrangements grows exponentially.

10.5.4 Hasse Precedence Quandary

In our attempt to construct Hasse diagrams for the ternary space, I was confronted with a dilemma regarding precedence in the form of the question: *What precedence exists amongst the three ternary values {0, 1 and 2}?* Figure 10-11, for example, shows three possible arrangements of the ternary values establishing the precedence of one constant over the other(s). *Do I treat the literal one (1) as equal, less than or greater than the other two?* Intrinsically, there is no natural precedence among the three constants, but rather, a symbolic primacy born out of our choice of mathematical manipulations. Figure 10-11 clearly demonstrates that I conveniently, yet arbitrarily, designated the symbol $|0\rangle$ to align at $0^\circ$, while the other two symbols are $120^\circ$ away, and hence, there are no natural physical phenomena dictating precedence amongst the three constants. In the algorithm described herein, I have artificially set precedence for the convenience of implementation.
where, as described shortly, I opted to perform *low-to-high* trit transitions first followed by *high-to-low* transitions, and as a result, introduced an artificial algorithmic prejudice in favor of the value *two* (2). Our choice was largely driven by the procedure for constructing the Hasse diagram, described above, where I have favored the constant *two* over *one* and the latter over *zero*. Such a favorable treatment results in delaying the appearance of terms containing the constant *two* while forcing the terms of containing *zeros* to appear earlier during the synthesis process. Such delay allows us to avoid control line blocking by relying on the fact that the value *two* will always appear later in the sequence, and hence, can be used as a control variable. I shall revisit this topic further in the discussion about convergence below.

Figure 10-11 Three possible Ternary Value Precedence diagrams where the ternary values can be arranged with different precedence orders. (a) indicates a natural mathematical order of precedence. (b) indicates that the values 0 and 1 are equivalent in precedence and are lower than 2. (c) indicates that 1 and 2 are at the same level and are both above 0. The precedence order plays a role in the structure of the Hasse diagram and the consequent construction of ternary input sequences.

Figure 10-11 (b) shows that the constant *two* (2) has precedence over the other two values, and Figure 10-11 (c) shows that the two constants *one* (1) and *two* (2) are of equal precedence and that both are higher than the value *zero* (0). Algorithmically, it is feasible, of course, to swap the symbolic values *zero* (0) with *two* (2) which would grant preference to the value *zero* (0) over the value *two* (2). Following the same thought
process, the reader can quickly ascertain that, at most, there exist 12 unique precedence orders:

1. Six for precedence order (a) representing the six unique permutation of \{0,1,2\}

   \[ O_a = \{0 \rightarrow 1 \rightarrow 2, 0 \rightarrow 2 \rightarrow 1, 1 \rightarrow 0 \rightarrow 2, 1 \rightarrow 2 \rightarrow 0, 2 \rightarrow 0 \rightarrow 1, 2 \rightarrow 1 \rightarrow 0\} , \]

2. Three for precedence order (b), \( O_b = \{2 \rightarrow \{0, 1\}, 1 \rightarrow \{0, 2\}, 0 \rightarrow \{1, 2\}\} , \)

3. Three for precedence order (c), \( O_c = \{\{1, 2\} \rightarrow 0, \{0, 1\} \rightarrow 2, \{0, 2\} \rightarrow 1\} . \)

Mathematically speaking, members within each of the three sets \( O_a, O_b, O_c \) are described as equivalent classes where a single element acts as a representative of the entire group. In this chapter I limit the discussion to the three precedence orders shown in Figure 10-11 and treat them as representatives of the twelve possible precedence orders. The precedence order of Figure 10-11(a) will be explained immediately after the discussion about algorithmic convergence.

**Definition 10-10:** Given a set \( S \) and an equivalence relation \( \sim \) on \( S \), the equivalence class of an element \( a \) in \( S \) is the subset of all elements in \( S \) which are equivalent to \( a \), represented as: \([a] = \{ x \in S | x \sim a\}\).

### 10.6 Convergence of Algorithm

The bedrock rule for all MMD-like algorithms states that: *in order for it to converge, the algorithm should never alter any of the completely mapped pairs.* I have adequately demonstrated that control variables must be employed in order to satisfy this rule and
illustrated the concept of control line blocking. I have thus far stipulated that a Hasse style ordering could remedy control line blocking but have yet to corroborate our supposition. Proving that our synthesis algorithm converges for any of the input sequences generated through the Hasse structure is the underpinning of the algorithm, and without the certitude of convergence, the algorithm would not have any advantage over a search of the entire set of permutations. Now that the bar has been set, I initially ventured through a brute force programming approach to demonstrate the convergence of ternary input sequences constructed through a ternary Hasse diagram. The effort entailed synthesizing every valid ternary Hasse input vector against all possible permutations of output vectors. I was able to demonstrate convergence for all two variable input vectors, 24 in total, against all permutations of the output vectors, a total of 362,880 (9!). However, as I stepped up to ternary circuit of three variables, the number of output permutations quickly inflated to 27!; an astronomical number even for the fastest super computer. Proving convergence of a two variable ternary problem, however, contributes nothing over the existing effort of MMD [10]. Our experience in the binary domain [22] fueled our determination to prove convergence through the old fashioned constructive proof with a convincing logical argument. I shall first provide a proof for input vectors based on the first precedence order of Figure 10-11 (a) and defer the proof of the other two precedence orders to a later section.

**Theorem 10-1:** All input vectors concordant with the Ternary $0 \rightarrow 1 \rightarrow 2$ precedence order will converge for all possible output permutations.
**Foundation for proof:** The theorem asserts that any input vector constructed through a Hasse diagram with the $0 \rightarrow 1 \rightarrow 2$ precedence relationship amongst the ternary values will converge for all permutations of the output vector. Notice that the stipulation about *all possible output permutations* validates that the algorithm is able to process all possible circuits (input/output mappings). It is worthy at this time to highlight the following points:

1. The algorithm blindly processes each input/output pair and does not maintain history nor peeks ahead in the course of synthesis.
2. For an $n$ variable circuit, the algorithm processes one *trit* at a time: the *target trit*, and uses the remaining $n-1$ *trits* as control variables. Hence, any $n-1$ control pattern will be encountered *exactly three* times in the course of synthesis.
3. For each input/output pair transformation, the algorithm processes *low to high* transitions first followed by *high to low* transitions. This stipulation makes the maximum number of control lines available which allow us to avoid altering any completely mapped pair which, in turn, facilitates convergence.
4. The algorithm uses the three ternary values $\{0, 1, 2\}$ for control variables.
5. For any band $b$, the algorithm processes all terms within the band before processing any terms in band $b+1$.

**Lemma 10-1:** For any control patterns, the minterms exhibiting the control pattern will always appear in three consecutive bands according to the $0 \rightarrow 1 \rightarrow 2$ precedence order of their target trit.
Figure 10-12 Precedence order of low and high bits in a ternary Hasse diagram. This Hasse structure follows the natural precedence order (0→1→2) which is also evident for each specific digit as it moves from one level to a higher level. The lower digit exhibits the same pattern in the NW direction, while the upper bit exhibits the pattern in the NE direction.

**Proof of Lemma 1:** The second point above embodies the essence of the proof of the Lemma 1, and consequently, the proof of convergence of all input sequences constructed on the basis of the 0→1→2 precedence order. I will utilize Figure 10-12 of the ternary Hasse diagram for a two variable circuit to illustrate the second point. In this discussion, I will use the symbol \( x \) to represent a value of *don’t care*. Now, suppose that we use the high order trit as a control variable during the synthesis process, then each of the patterns \( 0x^5 \), \( 1x \), and \( 2x \) will only repeat three times throughout the synthesis process as clearly shown by the dark solid lines travelling in the north western direction. For example, the control pattern \( 0x \) includes the terms \( 00, 01, 02 \) which ascend on the left branch of the Hasse diagram ascending in the north western direction. Notice that the three terms appear in three consecutive bands (band 0, 1, 2) and they appear in the order of the target trit (lower trit in this case). Similarly the terms of pattern \( 1x \) appear in the three consecutive bands 1, 2 and 3, and the terms for pattern \( 2x \) appear in bands 2, 3 and 4. Similarly, patterns \( x0 \), \( x1 \), and \( x2 \) where the lower trit is used as a control variable, each

---

5 Underlined digits are control line values.
of the 3-tuple terms appear in in three consecutive bands as shown by the gray lines travelling to the north east in Figure 10-12. Table 10-1 lists the band number where each midterm appears according to the control pattern used. The third row, for example, shows that the terms 20, 21 and 22 appear in bands 2, 3 and 4 consecutively. ■

*Table 10-1 Shows the band number where each minterm appears where the value of x is shown in the upper header. For row \((2x)\), the minterms \((20, 21, 22)\) appear in bands \((2, 3, 4)\) consecutively which is guaranteed by the structure of the Hasse diagram. I rely on this ordering for the proof of convergence.*

<table>
<thead>
<tr>
<th>(x)</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x)</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(1x)</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>(2x)</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>(x0)</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(x1)</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>(x2)</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

**Proof of theorem 1**: In the discussion above, notice that the expressions *appearance of terms* and *target trit* both referred to the terms of the *input* vector. The reader might still recall from the synthesis example of section 10.4 above that the synthesis process transforms the output vector to match the input vector, and in turn, making the input vector the destination (or target) of the transformation. With this in mind, I provide the following constructive proof of theorem 1:

1) At any point in the synthesis, all completely mapped pairs are identical; i.e., input and output minterms are equal and are arranged according to the ternary Hasse diagram,

2) At the \(k^{th}\) term in the synthesis process, the \(k^{th}\) input term could not have appeared in the previously completely mapped pairs because, for a bijective function, it should only exist once in the input vector,
3) For any mismatched $j^{th}$ trit of the $k^{th}$ input term, the corresponding $n-1$ trits of the output term represent the control pattern,

\[
\begin{array}{|c|c|c|c|c|}
\hline
Ci_{n-1} & Ci_{n-1} & \ldots & 0 & \ldots & Ci_1 \\
\hline
\end{array}
\quad
\begin{array}{|c|c|c|c|c|}
\hline
Co_{n-1} & Co_{n-1} & \ldots & 1 & \ldots & Co_1 \\
\hline
\end{array}
\]

4) The first appearance of the input control pattern ($Ci_{n-1} \ldots Ci_1$) appears when the target trit $Ti_j = 0$ – see Lemma 10-I. I now consider three possible situations based on the output control pattern ($Co_{n-1} \ldots Co_1$):

a) $\forall \ y = 1..n-1 \ | \ Ci_y = Co_y$: With exception to the target trit, all other trits are identical in both the input and output minterms. In this situation, I can safely use the control pattern without altering any completely mapped pairs since, according to Lemma 10-I, a zero in the target trit points to the first encounter of the control pattern.

b) $\exists \ y = 1..n-1 \ | \ Ci_y \neq Co_y$ and the output control pattern $Co_{n-1} \ldots Co_1$ has not been encountered before (in terms 1..k-1): Again, I can safely use this control pattern without altering any completely mapped pairs,

c) $\exists \ y = 1..n-1 \ | \ Ci_y \neq Co_y$ and the output control pattern $Co_{n-1} \ldots Co_1$ has been encountered before: This situation will occur only if the previously encountered completely mapped term ($Fo$) with the similar control pattern has a zero in the $j^{th}$ position of the target trit, i.e. $Fo_j(k)=0$, which makes $To_j(k) \in \{1,2\}$. Trying to match $To_j(k)$ to its corresponding input trit $Ti_j(k) = 0$ would surely alter
the trit $F_0(k)$, and hence, violating the bedrock rule of *preserving previously completely mapped pairs*. In section *Foundation for proof* above, stipulation 3 guarantees that $T_0(k)$ will not change on the first pass of *low to high* transitions – since I want to transform a \{1 or 2\} into a \{0\}. However, as other trits within term $k$ are transformed within the first pass from a lower to a higher value, the control pattern $C_{o_{n-1}} \ldots C_{o_1}$ of the term $k$, on the second pass (high to low), is guaranteed to be different and that it would have *larger trit values* which would, most likely, match minterms in some of the later bands. Now that the control pattern the $k^{\text{th}}$ term $\{C_{o_{n-1}} \ldots C_{o_1}\}^{\text{pass II}}$ is different than the control pattern of term $F_0$, I can safely alter $T_0(k)$ without altering any previously mapped minterm (situation 4)b) above), and hence, remain in compliance with MMD’s bedrock rule for convergence. Figure 10-13 shows an example where the control values $C_0 = 11x$ are skipped in pass 1 allowing trit 2 to transform $1 \rightarrow 2$, and on the second pass, a new control value $C_0 = 12x$ is used to safely transform $T_0(1 \rightarrow 0)$.

d) Upon full synthesis of the $k^{\text{th}}$ term, the output term will be identical to the input term.

5) The second encounter of the control pattern $C_{i_{n-1}} \ldots C_{i_1}$ will immediately follow in the next consecutive band, at the $m^{\text{th}}$ term, with a value of *one(1)* in the *target trit*. In the case of a mismatch, the corresponding output *target trit* would surely be a *two(2)* since the *zero(0)* have already appeared step 4) above. A controlled $[12]$ operation would safely do the transformation without affecting any completely mapped terms.
6) As a result of the above construction, it is clear that the bedrock lemma of *never altering a completely mapped pair* holds true throughout the above process; and hence, the algorithm will *always converge* for any Hasse compliant input sequence constructed using the \(0 \rightarrow 1 \rightarrow 2\) precedence order. This concludes the proof of theorem 1.

![Figure 10-13 Example for case 4c. In an attempt to change 111 to 120, the two lower trits have to be modified one at a time. Were I to attempt changing the lower trit from 1 to 0 first, I would had to use the ‘11’ control pattern which was encountered before, and would have modified the first minterm (110). But by always doing ‘low to high’ transitions first (pass 1), in this case the middle trit (from 1 to 2), the output minterm will change properly to allow for new control values to emerge which would not affect previously completed minterms. In pass 2, I can safely modify the lower trit from 1 to 0 since the ‘new controlling value (12)’ will be guaranteed not to modify any previously completed minterms.](image)

10.7 **Convergence of Triangular Hasse Precedence Orders**

**Theorem 2:** All input vectors concordant with the Ternary \(0 \rightarrow \{1,2\}\) and \(\{0,1\} \rightarrow 2\) precedence orders will converge for all possible output permutations.

**Proof:** The theorem asserts that any input vector constructed through a Hasse diagram with one of the triangular precedence orders \(0 \rightarrow \{1, 2\}\) or \(\{0, 1\} \rightarrow 2\) will converge for all permutations of the output vector. Figure 10-14b show the banded Hasse diagram for the \(0 \rightarrow \{1,2\}\) precedence order. Mathematically speaking, notice that the ternary values 1
and 2 belong to an equivalent class (see definition 14 above) as they both have an equal precedence over the zero but no precedence between them. As a result, I could easily select one of the two values to represent the behavior of the entire equivalent class as shown in Figure 10-14 (a). Arbitrarily designating the constant one (1) as the representative of the equivalent class, I can remap the banded Hasse diagram with its binary representation as shown to the right of Figure 10-14 (b). For this example, the ternary band number one \{01, 10, 02, 20\} simply becomes \{01, 10\}, and the set in band two simply becomes \{11\}. Since I have already proved convergence for reversible circuit synthesis using a binary Hasse diagram in [22], I can easily conclude that all ternary Hasse diagrams with the triangular precedence order 0 \(\rightarrow\) \{1, 2\} will also converge. Using the same reasoning for the last triangular order \{0, 1\} \(\rightarrow\) 2, the values \{0, 1\} are members of the same equivalent class which, again, reduces the problem to the binary Hasse diagram as seen above. This concludes the proof of theorem 2. ■

Figure 10-14 (a) One of the triangular precedence orders \{(0) \(\rightarrow\) \{1, 2\}\} can be represented by the binary equivalent class \{(0) \(\rightarrow\) \{1\}\} since the \{1, 2\} are of the same precedence level. (b) The ternary Hasse diagram based on this triangular precedence order can be reduced to the binary equivalent Hasse diagram shown to the right.
10.8 Selection through Genetic algorithm

A ternary function with \( n \) variables has \( 3^n \) minterms in the input vector which makes the number of possible permutations of an input vector an astounding \( 3^n! \) !

Our method of constructing convergent input vector sequences constructs a subset of all convergent sequences. According to Corollary 2, a ternary Hasse diagram consists of \( 2n+1 \) bands where each band consists of \( N(b) \) minterms – see derivation in Section 10.12.

\[
N(b) = \sum_{j=0}^{m=b/2} \left( \begin{array}{c} n \\ m-j \end{array} \right) \cdot \left( \begin{array}{c} n-(m-j) \\ 2j+(b \mod 2) \end{array} \right)
\]

where \( \binom{n}{k} \) is the combination operator and \( \mod \) is the modulo operator.

In the process of constructing the input vector, step 0 above of the pseudo-code in section 10.5.3 above, selects a single sequence of any permutation of all minterms in a band. Consequently, the total number of possible input vectors is the product of all permutations of all bands, stated as:

\[
T(n) = \prod_{b=0}^{2n} N(b)!
\]
Figure 10-15 Number of permutations for all possible input vectors (all possible solutions) vs. Hasse based sequences (valid solutions constructed by this algorithm).

Clearly, as the number of variables increases, Table 10-2, the number of possible input vectors generated by our algorithm still grows exponentially, despite at orders of magnitudes slower than generating all possible input vectors – see Figure 10-15. For a 3 trit function, I could easily examine all 6,494 Hasse based input vectors and select the one which yields the best quantum cost. Functions of 4-trits or more, however, suddenly become beyond the capacity of our best computers. Confronted with such daunting roadblocks, and borrowing from our experience in the binary domain [24], I opted to employ a genetic algorithm to construct potential input vector arrangements based on the results of previously synthesized input vectors.
Table 10-2 All possible permutations vs. total permutations for Hasse based sequences where all possible solutions are shown in column 3, while the number of solutions constructed using the Hasse structure are shown in column 4.

<table>
<thead>
<tr>
<th>Number of Variables (n)</th>
<th>Number of Minterms ((3^n))</th>
<th>Input Vector Permutations ((3^n))</th>
<th>(T(n)) (\prod_{b=0}^{2^n} N(b))</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>9</td>
<td>362,880</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>(1 \times 10^{28})</td>
<td>6,494</td>
</tr>
<tr>
<td>4</td>
<td>81</td>
<td>(5.8 \times 10^{120})</td>
<td>(1.21 \times 10^{17})</td>
</tr>
<tr>
<td>5</td>
<td>243</td>
<td>(5.7 \times 10^{475})</td>
<td>(1.55 \times 10^{56})</td>
</tr>
<tr>
<td>6</td>
<td>729</td>
<td>(1.4 \times 10^{1772})</td>
<td>(1.9 \times 10^{243})</td>
</tr>
<tr>
<td>7</td>
<td>2,187</td>
<td>(3.3 \times 10^{6266})</td>
<td>(4.1 \times 10^{850})</td>
</tr>
<tr>
<td>8</td>
<td>6,561</td>
<td>(1.1 \times 10^{22,196})</td>
<td>(1 \times 10^{8,391})</td>
</tr>
<tr>
<td>9</td>
<td>19,683</td>
<td>(8 \times 10^{75,974})</td>
<td>(2 \times 10^{8,615})</td>
</tr>
</tbody>
</table>

10.8.1 Objective function using Quantum Gate Count

In their analysis of the simple 2 trit function, the authors of [10] arbitrarily assigned a cost of \textit{one} for unconditional ternary gates, and a cost of \textit{two} for controlled ternary gates. Practically, however, there are no existing physical implementations of ternary quantum systems, and hence, no realistic cost could be assumed. For the purposes of this chapter, I use the number of gates as the measure of fitness, or objective function, for the genetic algorithm. For an arbitrary ternary quantum circuit \(C\) with \(k\) quantum gates, the quantum cost \(Q_c\) is calculated as follows:

\[
Q_c = \sum_{i=0}^{k} G_{qc}(i)
\]
where: $G_{qc}(i)$ is the quantum cost for each gate which I assume to be one for the purposes of our analysis. Actual gate cost would be used once physical implementation of ternary gates is realized.

10.8.2 Genetic Algorithm

Rather than bouncing randomly around the search space, a genetic algorithm follows a set of directed probabilistic steps where new solutions are the offspring of existing good solutions. The following block exhibits the standard structure of a genetic algorithm:

```
g ← number of generations;
initialize(P(g));
do 
    evaluate(P(g));
    P1(g), P2(g) ← select(P(g)); // Set of parent pair
    g ← g - 1;
    P(g) ← recombine(P1(g), P2(g)); // crossover ➔ children
    P(g) ← mutate(P(g)); // Mutate children
  while (g > 0);
```

The initialization step ($initialize(P(g))$) randomly creates a set of valid input vector sequences, initial population, using the Ternary Hasse Diagrams where, for each band, the set of minterms are randomly shuffled, and the resulting band arrangement is concatenated to the input vector under construction – step 0 of section 10.5.3 above. Synthesis of the initial population gives the fitness, quantum cost, of each input vector arrangement which is used to determine the next generation, offspring, of solutions to examine. Roulette wheel selection process is then used to randomly select two parents of the current generation for recombination. For this research, I studied both single and double crossover operators to create the next generation, with special consideration for
the position of the crossover - discussed shortly. The final step of the genetic algorithm applies a mutation operator in order to continuously maintain population diversity and avoid premature convergence to local minima.

10.8.3 Genotype and Valid Operators

As discussed earlier and shown in [22, 23], the band structure, defined above, must be faithfully preserved in order to assure algorithmic convergence. As a result of the banded structure of the algorithm, recombination operators are limited in their application to the boundaries of a band in order to avoid a minterm jumping from one band to another. In a similar fashion, mutation operators are constrained to swapping minterms intra-band which will also preserve the certitude of convergence.

Figure 10-16 illustrates the structure of a chromosome, i.e. input vector arrangement, for a two variable ternary function consisting of five bands. As hinted earlier, in order to ensure that an offspring is a valid input vector sequence, the crossover point(s) must occur at either end of a band, but not in the middle of a band. Had the invalid crossover point been taken in Figure 10-16, the resultant child would have been invalid as it would have included the minterm 01 twice and lacked the term 10. Of course a repair process could have detected and corrected such a defect which, depending on the repair process, could yield a different, yet valid, input vector. The reader might correctly surmise that the choice of limiting crossover to band boundary could potentially result in stale members within each band, leading to premature convergence to local minima. In general, genetic algorithms introduce mutations as a remedy for premature convergence where mutation typically acts as a background operator at a low probability of
occurrence. For this study, however, I intentionally elevated the probability of applying the mutation operator, at a level higher than suggested by standard genetic algorithms, in order to counteract the limitations imposed on the recombination operator (band boundary only). A high level of mutation probability, I theorized, would inject diversity within children allowing them to escape such hasty race to the nearest local minima.

![Figure 10-16 Genotype of a valid input sequence showing valid and invalid mutations and cross over operations. A valid crossover can only occur at the band boundary as stipulated by the Hasse diagram. Valid mutations, in this case swap, can only swap elements within the band to assure that the Hasse order is not violated.]

### 10.9 Experimental Results

For the purposes of this study, I have limited our experiment to the set of Hidden Weighted Trit (HWT) benchmark functions which are introduced, for the first time, into the literature in my papers [26, 27]. HWT functions are an extension to their binary counterpart Hidden Weighted Bit (HWB) functions which were first introduced by Prasad et al [97] and are heavily cited as one of the harder benchmark for reversible binary logic synthesis [21, 24, 23, 20].
Definition 10-11: Hidden Weighted Trit (HWT) functions are reversible ternary functions where the output minterm is generated by circularly shifting the input minterm by the number of its non-zero trits.

For the sake of a balanced comparison, I used the same synthesis algorithm for both the natural and Hasse based input sequences and the same method for calculating the cost. The only independent variable, in this case, is the input vector arrangement which represents the crux of our experiment reported in this chapter. The use of genetic algorithm is merely an aid for discovering Hasse based input vectors with lower circuit cost.

Table 10-3 Comparison between using the natural order of the input vector vs. using the Hasse structure to construct valid input vector arrangements. As the search space increases, the probability of discovering better solutions decreases while the time required to discover such solutions increases drastically.

<table>
<thead>
<tr>
<th>Function</th>
<th>Natural</th>
<th>Hasse</th>
<th>Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gates</td>
<td>Time</td>
<td>Gates</td>
</tr>
<tr>
<td>Hwt4</td>
<td>215</td>
<td>6x10⁶</td>
<td>86</td>
</tr>
<tr>
<td>Hwt5</td>
<td>782</td>
<td>6x10⁶</td>
<td>504</td>
</tr>
<tr>
<td>hwt6</td>
<td>2877</td>
<td>6x10⁶</td>
<td>2285</td>
</tr>
<tr>
<td>hwt7</td>
<td>10151</td>
<td>4x10⁶</td>
<td>8993</td>
</tr>
<tr>
<td>hwt8</td>
<td>35122</td>
<td>3.5x10⁶</td>
<td>32268</td>
</tr>
<tr>
<td>hwt9</td>
<td>119421</td>
<td>4x10⁶</td>
<td>111323</td>
</tr>
</tbody>
</table>

Table 10-3 shows the results of synthesis of the HWT functions of 4 to 9 trits using the natural and the Hasse based input vector ordering. Clearly the Hasse based ordering has produced better results for all functions with a 60% saving for the HWT-4 function down to 7% for the HWT-9 function. Of course the results should not be surprising as
the genetic algorithm processed a total of 600,000 arrangements of the input vector consisting of 100 generations of 100 individuals each for two variants of recombination methods (single & double crossover), and 30 combinations of probabilities of recombination and mutation. The fact that I am able to freely construct convergent input vector sequences, at our whims, is the strong point of this algorithm, and hence, our main contribution to the research area. Notice that the percentage of savings shrinks dramatically as the number of variables increase which can be easily explained with a quick glance at Table 10-3. Even though the search space is greatly reduced with the Hasse based algorithm, a nine trit function has a search space of the order $10^{9.615}$, for which, an exploration of 600,000 elements is like a drop in a colossal ocean.

To exasperate matters further, the time to synthesize functions with larger number of variables increases exponentially. Although our implementation of the genetic algorithm took advantage of multithreading on an 8 core Intel® i7 processor, the nine variable function consumed more than two hours to yield 7% improvement by visiting 600,000 solutions Figure 10-17 demonstrates an exponentially increasing curve depicting time vs. number of variables for class of HWT functions. For a four trit function, the 600,000 visited solutions represent a $6 \times 10^{-10}$% of the Hasse based search space where a saving of 60% is a great achievement. For a nine variable function, however, covering a similar ratio of the search space requires visiting close to $1 \times 10^{9606}$ potential solutions which is beyond the possibilities of all existing computing power on earth.
In order to accelerate the time required for finding Hasse sequences with the lowest cost, several experiments using the GPU - CPU mapping have been performed. The results of these experiments are shown in Table 10-4 and Table 10-6 for the HWT5 and HWT6 functions respectively. The columns in both tables represent in order: \textit{case} represents the type of experiment, \textit{cores} is the number of cores (CPU or GPU) the algorithm is running on, \textit{repetitions} is the number of times the sequences are computed, \textit{samples} is the number of different sequences computed, \textit{Total Time} is the time required for the whole computation and \textit{Time/Sample} is the unit time required to compute and evaluate a single sequence.
Table 10-4 Performance times for the HWT5 function which shows that, for CUDA, at full thread capacity, it is able to execute at 55 microseconds per sample compared to 304 microseconds for the CPU.

<table>
<thead>
<tr>
<th>Case</th>
<th>cores</th>
<th>repetitions</th>
<th>samples</th>
<th>Total Time</th>
<th>Time/Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>8 Core Host CPU</td>
<td>8</td>
<td>153600</td>
<td>153600</td>
<td>46.70</td>
</tr>
<tr>
<td></td>
<td>CUDA</td>
<td>512</td>
<td>300</td>
<td>153600</td>
<td>14.50</td>
</tr>
<tr>
<td>1</td>
<td>Single sequence running 1000 times on a single CUDA core (loop inside CUDA core)</td>
<td>1</td>
<td>1000</td>
<td>1000</td>
<td>40.00</td>
</tr>
<tr>
<td>2</td>
<td>Same as above, plus a call to __syncthreads() outside the loop.</td>
<td>1</td>
<td>100</td>
<td>100</td>
<td>4.10</td>
</tr>
<tr>
<td>3</td>
<td>Single sequence specified duplicated across the specified number of cores. Loop in CPU</td>
<td>1</td>
<td>1000</td>
<td>1000</td>
<td>22.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>100</td>
<td>200</td>
<td>2.27</td>
</tr>
<tr>
<td>4</td>
<td>Passthrough 2 devices</td>
<td>1024</td>
<td>512</td>
<td>524288</td>
<td>28.98</td>
</tr>
</tbody>
</table>

The first case in Table 10-4 and Table 10-6 shows the Genetic Algorithm running on an i7 Intel 960 Processor with 8 cores at 3.2GHz each. The system has 12 GByte DDR3 memory. The GA runs through 72 sets of parameters of 300 generations each, where each consists of 512 individuals. For HWT5, I realize a 317% speedup on the GPU relative to the CPU. For the HWT6 function, however, such a speedup diminishes to a mere 10% advantage for the GPU. This big difference in performance and the decrease of the performance when 512 are compared to 1024 cores is a result of the transfer time required to send the sequences from the CPU to the GPU. This can be confirmed by observing that the time/sample remains unchanged between the two scenarios (i.e., the processing happens at the same speed). The reason for observing a speedup for HWT5 is that the algorithm is CPU bound, and having more CPU cores helped demonstrate the value of CUDA (when we are CPU bound). This is because the difference in the number of minterms to be synthesized between the five and six variable function is very large which takes a lot of time to transfer between the CPU and GPU. As a result, the GPU
based approach is ideal when the amount of data transfer between the host and GPU is minimal.

Table 11-5 shows the distribution of work between CPU and GPU (column 2) as compared to a CPU only approach (column 1).

<table>
<thead>
<tr>
<th>CPU</th>
<th>CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>for 300 generations</td>
<td>CPU: For 300 generations</td>
</tr>
<tr>
<td>for 512 individuals</td>
<td>CUDA: Synthesize 512 individuals</td>
</tr>
<tr>
<td>AddToQueue(individual)</td>
<td>CPU: Bread &amp; Cull</td>
</tr>
<tr>
<td>Bread &amp; Cull</td>
<td>CUDA: Synthesize, etc.</td>
</tr>
<tr>
<td>8 Threads (Kernel):</td>
<td></td>
</tr>
<tr>
<td>while(queue.empty())</td>
<td></td>
</tr>
<tr>
<td>Synthesize(queue.pop(Individual))</td>
<td></td>
</tr>
</tbody>
</table>

In order to understand the detailed operation of the algorithm I ran few experiments to verify our conclusion about data transfer issues impeding performance. Cases 1 and 2 in Table 10-4 and Table 10-6 show the result for running a single thread on the GPU and the loop over the available sequences are directly implemented in the GPU. In case 1, a single sample is fed to a single CUDA thread, and the same sample is synthesized 1000 times. Case 2 is the same, with the addition of `syncthreads()` outside the loop and only for 100 samples. They both gave essentially the same results for synthesizing a single sample. From this experiment I conclude that the GPU did not require explicit synchronization with the host and that our time measurement is accurate where I am measuring the time it takes the GPU to compute all sequences.

In case 3 I specified the loop inside the CPU and made the call to CUDA 1000 times calling a single thread to do the synthesis. As expected, adding a second thread (effectively synthesizing two sequences) did not affect the time, as both CUDA threads
are working in parallel. Similarly, doing 512 threads (the number of cores on a single device) was all done in the same time as a single thread. This particular configuration shows the efficiency and the usefulness of the GPU acceleration. The pseudo code of case 3 is shown below:

```
CPU: for 1000 times
CUDA: Synthesize(sample)
```

Case 4 is the same algorithm as Case 3, where 1024 sequences are fed to two GPU devices (two distinct graphic cards) each with 512 threads. The loop is repeated 512 times. For the HWT5 function, you can see that it took about the same time per sample which effectively did not affect the time per sample. For the HWT6, however, the time per sample almost dropped by 30% indicating issues related to data transfer.

Table 10-6 Performance times for the HWT6 function where, at full capacity, CUDA takes 376 microseconds per sample compared to the CPU at 645 710 microseconds per sample. The CUDA speedup is lower than for the HWT5 function since the dataset for HWT6 is larger forcing some of the data to exist in CUDA memory buffers shared amongst multiple threads. For the HWT5 function, all the data was able to fit in local memory buffers each exclusively dedicated for each processor core.

<table>
<thead>
<tr>
<th>Case</th>
<th>cores</th>
<th>repetitions</th>
<th>samples</th>
<th>Total Time</th>
<th>Time/Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>8 Core Host CPU</td>
<td>8</td>
<td>153600</td>
<td>153600</td>
<td>109</td>
</tr>
<tr>
<td>CUDA</td>
<td>512</td>
<td>300</td>
<td>153600</td>
<td>99</td>
<td>0.000645</td>
</tr>
<tr>
<td>1</td>
<td>Single sequence running 1000 times on a single CUDA core (loop inside CUDA core)</td>
<td>1</td>
<td>1000</td>
<td>1000</td>
<td>424</td>
</tr>
<tr>
<td>2</td>
<td>Same as above, plus a call to _synchthreads() outside the loop.</td>
<td>1</td>
<td>100</td>
<td>100</td>
<td>42.7</td>
</tr>
<tr>
<td>3</td>
<td>Single sequence duplicated across the specified number of cores. Loop in CPU</td>
<td>512</td>
<td>100</td>
<td>51200</td>
<td>27.2</td>
</tr>
<tr>
<td>4</td>
<td>Pass through 2 devices</td>
<td>1024</td>
<td>512</td>
<td>524288</td>
<td>197</td>
</tr>
</tbody>
</table>
In order to get both devices to work in parallel, a special memory mode in the host memory was used. It is called the pinned host memory mode which page locks a region of memory on the host and makes it visible to the CUDA device. The locked memory holds the input and output sequences while the results were kept on CUDA’s local memory. I ensured that the algorithm on the CUDA device avoids memory conflicts between threads by coping the input/output vectors to local memory, and allocates the output buffers on local memory as well. When shared memory was used for the output buffers, it took much longer to synthesizes the sequences due to bank conflicts between the threads trying to read and (especially write) to the shared memory.

Notice that cases 1 and 2 produced unexpected results. I expected that placing the loop inside the CUDA core would yield the best results; however, I measured a 100% degradation in performance when the loop was place inside the CUDA compared to placing it in the CPU (case 3). I observed the same anomaly for both the HWT5 and HWT6 functions.

Finally, to summarize the results of the GPU acceleration, considerable acceleration of circuit computation is achieved if the following conditions are satisfied:

- Minimize the CPU-to-GPU transfer data,
- Minimize the GPU global-to-local memory transfer,
- Split the data so that different running GPU cores do not obstruct each other by blocking the global memory access.
10.11 Conclusion and Analysis

In this chapter I compared the synthesis process using a natural ternary arrangement of the input vector versus a subset of all possible arrangements and successfully demonstrated the benefit of the latter. In the process, I introduced a synthesis algorithm capable of synthesizing any arbitrary ternary function with a large number of variables. Since the option of exploring the entire search space is unfeasible, our unique method of constructing input vector arrangements, with guaranteed convergence, becomes an essential component of the search algorithm.

Building a Hasse based ternary structure provides a protection against the trap of control line blocking and allows our synthesis algorithm to access any element within the limited Hasse based search space in a random manner. Still, rather than randomly hopping throughout the search space, I constructed new sets of candidates (offspring) based on the best solutions found up that point. I have shown in section 6.5 that genetic algorithm and Tabu search has consistently resulted in better quantum cost, compared to random selection, and I believe that it is the case here as well. The search space for the ternary domain grows at a larger magnitude than in the binary domain, which makes it even harder for random selection method to discover better solutions. One question that has not been answered by our work in the ternary domain, however, is whether partitioning the search space, similar to our CSP algorithm, would yield better results for different partitions.

Although the CUDA performance for the $hwt5$ and $hwt6$ functions have shown superior results compared to the CPU, the benefit in performance is only marginal
considering that 1024 processing cores were used compared to the 8 cores of the CPU. Considering that a CUDA core runs at 1GHz compared to the CPU core at 3.7 GHz, I had expected a performance boost of 30 to 50 times what the CPU can do. Both the CPU and CUDA devices utilize DDR3 memory at 3GHz which does not explain the slowdown. Memory contention is always an issue with any multiprocessing algorithm, so I made sure that for the CUDA implementation, every core has access to completely separate memory spaces, including constants and stack. Since CUDA provides three memory levels, *local, shared and global*, I made sure to fit data as close to the processing core as possible (local first, shared next, then global). For the *hwt5* function, all dataset was able to fit within the *local* memory which is exclusive for the specific core which resulted in a descent speedup compared to the CPU. However, for the *hwt6* function, the *shared* memory had to be used in order to keep the data as close to the processing core as possible. The shared memory, however, is slower and requires more time to execute.

Of course, the CPU L1 and L2 cache infrastructure is superior (in size and speed) to what CUDA provides where the entire dataset for both functions can easily fit within the L1 cache. Even though the CPU is running 8 cores, its access to the L1 cache is extremely fast that it makes up the time when it faces off with 1024 CUDA cores.
10.12 Derivation of Equation

Equation 3 calculates the number of minterms in each band in a ternary Hasse diagram using the precedence order $0 \to 1 \to 2$. According to definition 8, for any minterm in a band $(b)$, the mathematical sum of its trits equals the band number. To demonstrate the derivation of equation 3, shown below, I will first start with an example where, for a function of five variables, I calculate the number of minterms at band levels 4 and 5. Notice that, according to corollary 1, a function with five variables has 11 bands (0 to 10).

\[
N(b) = \sum_{j=0}^{m=b/2} \frac{n!}{(m-j)! (n-(m-j))!} \cdot \frac{1}{(2j + (b \mod 2))}
\]

**Band 4 (even):** To calculate the number of elements in a band, I find all combinations where the sum of digits is equal to 4, which is the sum of the following arrangements:

<table>
<thead>
<tr>
<th>$j$</th>
<th>(Combination of selecting $2$ twos of 5 variables)×(Combination of selecting $0$ ones of $m$ variables)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$(\binom{2}{2}) \cdot (\binom{3}{0}) = 10$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$(\binom{1}{1}) \cdot (\binom{4}{2}) = 30$</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$(\binom{0}{0}) \cdot (\binom{5}{4}) = 5$</td>
<td></td>
</tr>
</tbody>
</table>

**Band 5 (odd):** Similarly, I find all combinations where the sum of digits is equal to 5, which is the sum of the following arrangements:
<table>
<thead>
<tr>
<th>$J$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(Combination of selecting 2 twos of 5 variables) · (Combination of selecting 1 ones of 3 variables) $\binom{2}{2} \cdot \binom{3}{1} = 30$</td>
</tr>
<tr>
<td>1</td>
<td>(Combination of selecting 1 twos of 5 variables) · (Combination of selecting 3 ones of 4 variables) $\binom{1}{1} \cdot \binom{4}{3} = 20$</td>
</tr>
<tr>
<td>0</td>
<td>(Combination of selecting 0 twos of 5 variables) · (Combination of selecting 5 ones of 5 variables) $\binom{5}{0} \cdot \binom{5}{5} = 1$</td>
</tr>
</tbody>
</table>

In equation 3, a band $b$ in an $n$ variable function has $m = \frac{b}{2} + 1$ terms which are added together ($\sum_{j=0}^{m=b/2}$). For each index $j=0..m$, I first select the $(m-j)$ digits of twos out of $n$ possible digits $\binom{n}{m-j}$. Since the sum of digits at each band equals the band level $b$, then the remainder must be selected from digits which have a value of one. The total number of such digits would be:

$$Number\ of\ ones = b - 2 \times (m-j) = b - 2 \times \frac{b}{2} + 2j = 2j$$

Since I have already used $(m-j)$ digits for selecting the twos (in the first term above), I have $n-(m-j)$ digits to select the ones from. Consequently, the second term of equation 3, $\binom{n-(m-j)}{2j+(b \mod 2)}$ represents the combination of selecting $2j$ of ones out of $n-(m-j)$ digits. Notice that when the band number $b$ is even, $b \mod 2 = 0$, while for an odd band number $(b)$, the term $(b \mod 2) = 1$ which essentially selects an additional one out of the remaining digit. ■
Chapter 11 **MV Benchmarks and Extensible Quantum Specification (XQS)**

11.1 **Introduction**

Existing literature in multiple-valued reversible circuit synthesis tends to focus on ternary circuits with rather small number of variables using an array of different specifications [13],[16], [8], [17], [14], [10], [19], [15], [9], [18], [20], [23], [11],[12], [22]. In turn, it becomes hard, if not impossible, to formulate an objective comparison amongst various algorithms. Binary logic synthesis, on the other hand, has an established set of benchmark functions which are commonly used by most researchers to measure the performance of their algorithms. Some of the most common binary functions are supported by online resources such as RevLib [24] and Maslov’s Benchmark Specifications [5] where quantum specifications of different classes along with some of the best discovered circuits are available.

Binary reversible functions and circuits already use the standard Programmable Logic Array (PLA) file format to describe the specification as input/output pairs and the set of gates which represent the synthesized circuit. Both RevLib and Maslov’s websites contain various reversible benchmark functions using the PLA format and some functions are specified with a truth table. RevLib, for example, uses the truth table format (called SPEC) for specifying embedding of irreversible functions, while the PLA format is used for specifying larger functions [24]. Extending the classical logic PLA specification to quantum binary logic has proven successful since the level and complexity of computation is somewhat similar to their classical binary counterparts.
Delving into multiple-valued computation, however, increases the complexity of representing functions. It is feasible for the quantum unit of computation, the qubit, to utilize multiple basis of computation within a single implementation, e.g., hybrid quantum circuits (HQC). It is also feasible to design a circuit where different qubits utilize different basis of computations throughout the circuit. The PLA specification is not well suited for representing multiple-valued functions with such complexity and would require a major overhaul in order to do so.

In this chapter I present a foundational framework for an initial set of multiple-valued benchmark functions which will standardize the yardstick for measuring performance amongst algorithms in the MV domain. I also propose a new file format that is designed specifically for multiple-valued and hybrid functions. The new eXtensible Quantum Specification (XQS) is based on the universal YAML file format [6] which is a human friendly data serialization standard for modern programming languages. The YAML file format, and hence XQS, allows for encapsulation of data structures (name spacing), inheritance, expressivity and extensibility. YAML is supported by the majority of modern programming languages where packaged libraries for the specific language are available on the Internet [6]. Both benchmark functions and XQS specification are available on [4].

11.2 Reversible Multiple-Valued Logic Functions

A reversible multiple-valued function is a bijection \( B : I \rightarrow O \) from \( N^k \) to \( N^k \), with \( N \) being the number of inputs (outputs) and \( k \) is the radix of computation basis. Table 11-1 shows the truth table of a two variable multiple-valued reversible function of radix 3 (i.e., ternary). Similar to their binary counterparts, quantum
Table 11-1 Example of multiple-valued (ternary) reversible function where AB are the inputs and PQ are the outputs.

<table>
<thead>
<tr>
<th>AB</th>
<th>PQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>02</td>
<td>02</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>20</td>
</tr>
</tbody>
</table>

circuits for multiple-valued specifications are constructed from a set of quantum logic reversible gates which are capable of evolving the state of the qubit between the MV states (based on the radix of operation). For example, a ternary quantum gate transitions the state of the qubit amongst the states \( |0\rangle, |1\rangle, \text{and } |2\rangle \); therefore, these gates are referred to as multiple-valued reversible logic gates. Figure 11-1 shows the quantum circuit (in this case a single quantum gate) that realizes the function from Table 11-1. The gate shown in the figure is the Controlled-[02] and belongs to the class of the Controlled-U (C-U) reversible logic gates. The C-U family is a well-known class of quantum logic gates which have been physically implemented and demonstrated for both binary and ternary logic. In Boolean quantum circuits, the C-U gate applies the function \( U \) on qubit \( B \) if state \( |A\rangle = |P\rangle = |1\rangle \), hence \( |Q\rangle = U|B\rangle \). Otherwise, the value of qubit \( |B\rangle \) is unchanged, \( |Q\rangle = |B\rangle \). The Controlled-U set of gates for the ternary computational basis have been introduced in [21].
**Figure 11-1** Example of multiple-valued reversible function represented as a circuit. In this illustration, the A=P qubit is used as a controlling value for the gate [02]. When A is set, the [02] gate is active, otherwise, it is a passthrough.

**Definition 12-1:** *(Multiple-Valued Controlled-U Gates)* The multiple-valued controlled-U gates are the radix gates: when $|A⟩ = |P⟩ = |r - 1⟩$, (with r being the radix of the control qubit) the target qubit is modified according $|Q⟩ = U |P⟩$. Such function is illustrated in Table 11-1; the target qubit is modified using a multiple valued operator so that $|22⟩ = U |20⟩$, and $|20⟩, U |22⟩$. The matrix of this U operation is shown in eq. 1(a).

As can be expected, because of the higher radix in multiple-valued quantum reversible functions, the number of available operators is relatively larger when compared to the binary cases. For instance, while in the binary case only one CNOT gate exists (Table 11-2(d), in ternary circuits at least three possible equivalent gates exists Table 11-2 (a-c).

| Table 11-2 Matrix representation of ternary (a-c) and binary (d) quantum gates. |
|---------------------------------|---------------------------------|---------------------------------|
| $[02] = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$ | $[12] = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ | $CNOT = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ |
| (a) | (b) | (d) |

(eq. 12.1)
**Definition 12-2:** (Multiple-Valued Gates) For multiple-valued logic for the basis states of radix $r$, there exists a number of unitary quantum gates equal to the number of unordered $r$-subsets (each $r$-subset is a permutation of the $r$-1 values) of the set \{0,1,2,...,r−1\} and the number of basis $r$ permutation gates = $r!$

For example, the set of permutations for binary gates, $r = 2$, are the unordered 2-element set \{(0, 1), (1, 0)\} which represent the wire and inverter respectively. By corollary, the set of ternary values \{0, 1, 2\} can be fully permuted into the six elements of the 3-tuple set \{(0, 1, 2), (0, 2, 1), (1, 0, 2), (1, 2, 0), (2, 0, 1), (2, 1, 0)\} which represents the single-qubit ternary gates symbolically shown in Table 11-3.

This increase in the resources in the multiple-valued circuit design as compared to the binary case, require extra care in the design process, because selection of a particular set of operators is in-fact a selection of a specific algebra. The ternary operators described here are an example of the Galois Field algebra which defines the *modulo* function for level 3 [7].
Table 11-3 Generalized Ternary Gates based on Galoi Field 3 (GF3) with the gate name given in the first column and the mathematical equation for calculating the outputs based on the current input. The third column shows the output for the three possible inputs shown in the header, and finally the gate symbol is shown in the last column.

<table>
<thead>
<tr>
<th>Gate</th>
<th>F(a)</th>
<th>Input (a)</th>
<th>Gate symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0</td>
<td>(a+0) mod 3</td>
<td>0 1 2</td>
<td>a – a</td>
</tr>
<tr>
<td>+1</td>
<td>(a+1) mod 3</td>
<td>1 2 0</td>
<td>a +1 a-1</td>
</tr>
<tr>
<td>+2</td>
<td>(a+2) mod 3</td>
<td>2 0 1</td>
<td>a +2 a+2</td>
</tr>
<tr>
<td>12</td>
<td>2(a+0) mod 3</td>
<td>0 2 1</td>
<td>a 12 a12</td>
</tr>
<tr>
<td>02</td>
<td>2(a+1) mod 3</td>
<td>2 1 0</td>
<td>a 02 a02</td>
</tr>
<tr>
<td>01</td>
<td>2(a+2) mod 3</td>
<td>1 0 2</td>
<td>a 01 a01</td>
</tr>
</tbody>
</table>

In the binary case, the number of operators (gates) is increased when the qubit is placed into a state of superposition with a V gate – referred to as the state of superposition where the qubit is effectively halfway between the states, |0⟩ and |1⟩. For instance, an equivalent decomposition to the binary [CNOT] gate = [V ]*[V ] = [V† ]*[V†] where the application of a single V or V† gate rotates the qubit into the superposition state. The V gate applies a rotation of 90° in one direction while the V† applies rotation of 90° in the opposite direction around the qubit real axis – as envisioned by the Bloch sphere. Similar decomposition can be envisaged for multiple valued gates.

For example, the ternary gates of eq. 1 above can be decomposed into V=V† gates as shown in eq. 12-2 below:

\[
V02 = \sqrt{[02]} = \begin{pmatrix} \frac{1+i}{2} & 0 & \frac{1-i}{2} \\ 0 & 1 & 0 \\ \frac{1-i}{2} & 0 & \frac{1+i}{2} \end{pmatrix} \quad V12 = \sqrt{[12]} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix} \quad (eq. 12.2)
\]

(a) \quad (b)
\[
V_{01} = \sqrt{[01]} = \begin{pmatrix}
\frac{1+i}{2} & \frac{1-i}{2} & 0 \\
\frac{1-i}{2} & \frac{1+i}{2} & 0 \\
\frac{2}{0} & \frac{2}{0} & 1
\end{pmatrix}
\]
\[
V = \sqrt{\text{CNOT}} = \begin{pmatrix}
\frac{1+i}{2} & \frac{1-i}{2} \\
\frac{1-i}{2} & \frac{1+i}{2}
\end{pmatrix}
\]

Definition 2 clearly states that the number of possible permutative quantum gates, using the radix \( r \) as the basis of computation, is exponentially proportional to \( r \). For the binary basis state there exists a single inverter represented by the NOT gate Figure 11-2(d). In the ternary computation basis, however, there exists three self-inverters ([01], [02], and [12]) which when pair of each are placed back to back yields the identity matrix (self-reversible gates) Figure 11-2 a-c. The other two ternary gates ([+1] and [+2]) are supplementary inverters where placing one of each restores the original state Figure 11-2d. The last two gates can be implemented with the self-inverting gates where, in Figure 11-2e, the gate [+1] = [12]·[02] and the gate [+2] = [02]·[12]. I mentioned that, in the binary basis of computation, the square root of NOT gates, \( V \) and \( V^\dagger \), represent the quantum states of superposition (Eq. 12-2d) which is a state that rests halfway between the \( |0\rangle \) and \( |1\rangle \) basis states. Similarly, it is feasible to define an equivalent set of gates, for the ternary basis of computation, which suspend the qubit in a superposition state between the three basis states \( |0\rangle, |1\rangle, \text{and} \ |2\rangle \). Notice that, in this case, the square-root of a ternary quantum gate, creates superposition between coefficients that are not on the diagonal of the unitary matrix representing the gate. Thus, the matrix representation for \( [V02] = \sqrt{[02]} \) shown in eq. 12-2a) creates a superposition for the input states \( |0\rangle \text{ and} \ |2\rangle \) while if the input state is \( |1\rangle \) the output is an observable deterministic
quantum state. However, the gates [+1] and [+2] create a different type of superposition given by the unitary matrices shown in eq. 12.3.

\[
V_{+1} = \sqrt{[+1]} = \begin{pmatrix}
\frac{2}{3} & -\frac{1}{3} & \frac{2}{3} \\
\frac{2}{3} & \frac{2}{3} & -\frac{1}{3} \\
-\frac{1}{3} & \frac{2}{3} & \frac{2}{3}
\end{pmatrix}
\]
(a)

\[
V_{+1} = \sqrt{[+1]} = \begin{pmatrix}
\frac{2}{3} & \frac{2}{3} & -\frac{1}{3} \\
-\frac{1}{3} & \frac{2}{3} & \frac{2}{3} \\
\frac{2}{3} & -\frac{1}{3} & \frac{2}{3}
\end{pmatrix}
\]
(b) \hspace{1cm} (eq. 12.3)

Table 11-5 Ternary Square Root of [+1] and [+2] gates.

Figure 11-2 Operation of Ternary Inverters (a,b,c) shows three self inverters, while (d) show the two complementary inverters (+1 and +2). (e) shows substitution of +1 to (12, 02) and +2 to (02,12) where the middle 02 gates cancel one another, and then the 12 gates cancel one another yielding identity; hence, +1 and +2 are complementary invertors.

These quantum gates have similar properties to the V and V\(^\dagger\): for instance the [V02] and [V02]\(^\dagger\) postulate that:

\[
[V02] [V02] = [V02]^\dagger [V02]^\dagger = I
\]  \hspace{1cm} (eq. 4)

Finally, the consequence of the higher variety of the multiple-value reversible gates is that the possible functions can be more complex as well as they can be represented in more complex or user-convenient manners. For instance, a ternary majority function can
take three values such as 0, 1 and 2 to express the comparison of the output value to a
particular threshold \( t \) such that:

\[
\text{maj}_3(a, b, c) = \begin{cases} 
0 & \text{if } \text{maj}_3(a, b, c) < 1 \\
1 & \text{if } \text{maj}_3(a, b, c) = 1 \\
2 & \text{if } \text{maj}_3(a, b, c) > 1 
\end{cases}
\]

(eq. 5)

11.3 Extensible Quantum Specification (XQS)

Binary quantum functions specifications and circuits have been adequately
represented through existing classical standard specifications such as the PLA file
format (used by Espresso [3]), Berkeley Logic Interchange Format - BLIF [1], [2],
and RevLib [24] file format. RevLib, which is used primarily to describe binary
input specifications of quantum circuits, outlines two methods for describing a
quantum specification, the PLA Sum of Product (SOP) format shown in Figure 11-3,
and the truth table format known as SPEC. The RevLib website gives great details
about the REAL format and how it can be used for describing the components of a
quantum circuit as shown to the right of Figure 11-3.

The complexity of describing quantum circuit for multiple valued logic increases
where a quantum qubit could be in more than two states (multiple valued) or, in the
case of a hybrid circuit, could have qubits using different basis of computation
(binary, ternary, quaternary, …etc.) at the same time. It might be feasible to
shoehorn additional information to the existing PLA and SPEC formats in an effort
of representing higher basis of computation; however, such extensions to these
specifications would become complex, repetitive and highly incompatible.
In order to deal with complex specifications and increased complexity of functions, as in the case of the multiple-valued logic, I chose to introduce a new extensible file format specifically designed for multiple valued quantum logic. The new format is based on the extensible YAML file format specification [6] which is a human friendly data serialization standard for programming languages which allows for encapsulation of data (scoping or name spacing), inheritance, expressivity and extensibility. YAML support for the majority of modern programming languages exist through packaged libraries available on the Internet.

Figure 11-3 (a) Example of binary reversible quantum circuit and (b) its specification in the PLA format.
11.3.1 Structure of YAML

YAML\(^6\) supports three basic primitives: mappings (hashes and dictionaries), sequences (arrays and lists), and scalars (strings and numbers). Each of these data types can be nested within any of the supported data types to any depth (or branches) which is a necessary vehicle for scoping or name-spacing specification elements. Table 11-6 shows examples of the three data types in YAML format along with their parsed abstract representation. Notice in the last column that I have a set of sequences nested within a mapping element. The strings and numbers represent the scalars in YAML.

<table>
<thead>
<tr>
<th>YAML Input</th>
<th>Type</th>
<th>Parsed into language data structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Mark McGuire</td>
<td>Array</td>
<td>[“Mark McGuire”, “Sammy Sosa”, “Ken Griffey”]</td>
</tr>
<tr>
<td>- Sammy Sosa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Ken Griffey</td>
<td>Sequence</td>
<td></td>
</tr>
<tr>
<td>Home runs: 65</td>
<td>Hash</td>
<td>{“Home runs”: 65, “Average”: 0.89, “Total Runs”: 147}</td>
</tr>
<tr>
<td>Average: 0.89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Runs: 147</td>
<td>Mapping</td>
<td></td>
</tr>
<tr>
<td>American:</td>
<td>Sequences</td>
<td></td>
</tr>
<tr>
<td>- Red Sox</td>
<td>nested with Mappings</td>
<td></td>
</tr>
<tr>
<td>- Tigers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Yankees</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Canadian:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Mounties</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 11-6 Basic Data Types supported by YAML language which can be nested to as many levels as needed. The first column shows the YAML syntax of the type shown in the second column. The last column represents the parsed YAML in most modern programming languages.

11.4 Example of Extensible Quantum Specification

The XQS specification in Figure 11-4 for a 2 digit ternary full adder has four primary sections:

\(^6\) YAML: stands for Yet Another Markup Language or Yaml Ain’t Markup Language
**Signature**: Holds the signature of this function for human identification and reporting basis.

**Defaults**: This section represents a base set of parameters which are shared among sections in the specification and implementation subsections. For example, the `defaults` section is inherited by the `specification::inputs` section, which is, itself, inherited by the `specification::outputs` section. In this case, both inputs and outputs have the value `radix = 3`.

**Specifications**: This section holds one or more representations of the same specification. In this example, a truth table representation is used to represent a 2 digit full adder ternary function.

**Implementation**: This section holds the set of quantum gates implementing the specified function. Similarly, this section allows for different presentations of the same implementation.

![Figure 11-4 Pseudo-code of an XQS specification describing a Full Adder with 2 inputs and 1 output. The implementation section shows a solution for this adder.](image)
The example described in Figure 11-5 shows a complete specification represented with a truth table mapping each input minterm to its corresponding output minterm. With XQS format, it is easily possible to include multiple representation of the same specification in the same file (i.e., PLA, Blif, ..etc.). For example, a new element (e.g., Equation Format) can be nested under the specification element and use ternary logic equations to describe the same function. Similarly, multiple implementation formats could be specified as shown in the figure, where both the RevLib and Typed formats are specified. The Typed format, for example, will be parsed into an array of terms for each gate, while the RevLib format will be parsed as a list of strings. Such flexibility provides a vehicle of interoperability between applications by allowing different algorithms to parse the specifications in a manner compatible to their internal data structures.

Element inheritance is shown on the line \[\text{inputs: &defaults}\]. In this case, all the attributes defined in the defaults elements are used (inherited as the base definition of the inputs element, which then can be overridden by a specific definition within the inputs element. In this case, the inputs element will contain both the \text{radix = 3} and \text{variables = 2} parameters when parsed by the YAML parser. The outputs element will contain \text{radix = 3} and \text{variables = 1}. 
Hybrid Function Specification

<table>
<thead>
<tr>
<th>signature:</th>
</tr>
</thead>
<tbody>
<tr>
<td>function: hybrid circuit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>specifications:</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputs:</td>
</tr>
<tr>
<td>variables: abc</td>
</tr>
<tr>
<td>radix: 322</td>
</tr>
<tr>
<td>outputs: &amp;inputs</td>
</tr>
<tr>
<td>variables: --f</td>
</tr>
<tr>
<td>radix: --2</td>
</tr>
</tbody>
</table>

Figure 11-5 XQS specification of a binary/ternary hybrid function where the radix is defined for each qubit (322 is ternary for the upper qubit, and binary for the other two). “Don’t care” is shown as ‘-’ in the output variable definition.

11.4.1 Hybrid Multiple-Valued Reversible Function

The XQS specification provides the ability to represent hybrid quantum circuit (HQC) specifications utilizing multiple basis of computation (e.g., binary and ternary). Figure 11-5 shows the input and output definition of a hybrid circuit using both binary and ternary registers (qubits). There are three input variables (abc) with variable a uses ternary logic while b and c are both binary. The output also defines three variables where the least significant digit represents a binary function while the other digits are insignificant (don’t care).

HQC implementation where gates of different basis states operate on the same qubit are presented in the same sequential format used to specify a cascade of gates representing the implementation. In this case, gates of different basis would be listed back to back representing the specific implementation.
11.4.2 *Multiple-Valued Finite State Machine*

As a final example of the XQS I describe a small Finite State Machine (FSM). Figure 11-6, Figure 11-7 show an *odd number of ones* detector for both Moore and Mealy machines respectively. The FSM is a two-state machine with a single input \((x)\), which detects whether an odd number of ones came through a sequence of binary digits. Notice, that the FSM is specified by the two major sub-headings under the specification section: the transitions and the implementation. The implementation represents the state encoding while the transitions define the state transition and output generation function (for the Mealy FSM). For the case of the Mealy FSM, the output is defined for each next state - as shown in brackets. Notice that in the transitions section, you can define any arbitrary function, such as a threshold function, by following the same methods of defining classical state machines.

---

**Moore Machine Specification**

**signature:**
function: odd number of ones detector

**specifications:**
inputs: \([x]\)
transitions:
- A: 0, B, Output: 0
- B: 0, B, 1: A, Output: 1

**implementation:**
- A: 0
- B: 1
In this section, I introduce a set of multiple-valued benchmark function generators through an online website found on http://quantumlib.cecs.pdx.edu. The site currently generates the list of functions found in.

Table 11-7 Sample benchmark functions available on MV Benchmark Repository: http://quantumlib.cecs.pdx.edu.

<table>
<thead>
<tr>
<th>Function</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_vgte</td>
<td>v digits</td>
<td>1</td>
<td>binary variable</td>
<td>3_4gte5: four variable ternary function where the output is 1 whenever the value &gt;=5.</td>
</tr>
<tr>
<td>r_vgtev2</td>
<td>v1 + v2 digits</td>
<td>1</td>
<td>binary variable</td>
<td>3_4gte2: six variable ternary function where the output is 1 whenever the value of the upper 4 digits &gt;= the value of the lower 2 digits.</td>
</tr>
<tr>
<td>r_vcountr</td>
<td>v digits</td>
<td>(r-1)*ceil(log,(v))</td>
<td>Counts the number of repeated instance of the digits 1 to r-1.</td>
<td>3_4count12: ternary digit counter of 4 variables counting the number of ones and twos.</td>
</tr>
<tr>
<td>r_vfulladder</td>
<td>v digits</td>
<td>ceiling(log,(v*r))</td>
<td>Adds all digits</td>
<td>3_4fulladder: adder of 4 ternary digits</td>
</tr>
<tr>
<td>r_vSwivel</td>
<td>v digits</td>
<td>v digits</td>
<td>Reverse the order of digits where abc =&gt; cba</td>
<td>3_5Swivel: ternary swivel gate of five variables.</td>
</tr>
</tbody>
</table>
The web site allows users to select and group benchmarks according to their criteria with the goal of making the evaluation and comparison of results easier. The following outlines some of the features and functionality of the site:

- Function Generation On Demand:
- Allows for specification of **radix, number of variables and function type.** Unlike other web sites, the benchmark functions are generated in real time and allows the user to build functions suited for his/her needs. For instance a user might want to synthesize adders for 5,6,7 bits, and all such functions are generated in real time and provided to the user in the XQS file format.
- Select whether the function should be fully reversible, even for incomplete functions. For such functions, the web site will add the necessary number of **ancilla** bits to both input output vectors and produces a reversible specification representing the same function.

The resulting format of the functions is the XQS but other formats can be requested and will be generated in real-time. Additional functions of the web site provide methods for ordering and retrieving previous results.

- Sorting:
  a. Functions can be sorted by the number of **ancilla bits or cost of the gates.** Additional methods of sorting allow the user to cascade sorting functions to obtain a best function realization based on specified criteria.
For instance, a search for a full adder on $n$ input variables, ordered in the increasing order of the gate cost and of the ancilla bits.

11.6 Conclusion

In this chapter I introduced a set of multiple-valued benchmark functions for general use in automated synthesis algorithm that are generated on the QuantumLib web site. I also introduced an extensible file format specifically designed for MV quantum specifications and circuits which allows for simple specification of both fixed- and mixed-radix functions. Since existing formats are definitions of binary classical circuits conveniently used for the quantum binary domain, they are limited in their structure to define some of the new concepts of quantum computing including multiple valued and hybrid logic. The XQS format, however, is specifically designed to accommodate these new concepts and allows a level of flexibility for the algorithm designer to define ancillary information as they see fit.

The extensibility of the XQS definition can also be a danger to interoperability between different algorithms if each algorithm defines its own structure. At this stage of the maturity of quantum computing, the extensibility allows freedom in defining the structure, however, as the field matures, the structure has to be defined by the research and professional community to allow interoperability between disparate systems and algorithms.
11.7 Quantum Lib Function Generators

The function generator on the quantum lib website provides a list of functions already generated along with the parameters used to generate it. The site also provides a search, sorting and pagination functionality to make it easy to find classes of functions based on their names, radix or number of variables.

![Index page of quantum function generator.](image)

The site allows the user to view existing functions or generate new functions with the ability to download the function specification in the new Extended Quantum Specification (XQS) format.
Figure 11-9 details of function along with downloadable specification
Chapter 12 Generalized Multiple Valued Swivel Gate

12.1 Introduction

The majority of quantum logic synthesis algorithms have typically sited their performance according to an established set of quantum cost numbers counting the number of single- and two-qubit primitive quantum gates [1,2,3,4,5,6,7]. Recently, a growing number of researchers are taking into account the implications of LNNM physical constrains imposed by technology used to construct quantum circuits. So rather than calculating a quantum cost which ignores such technological constraints, algorithms have utilized the well-known universal swap gate to bring interacting qubits next to one another, and in turn, enforcing the LNNM constraint [8, 9, 10, 11] entry. In a typical application of the LNNM constraint, the information of a qubit is typically transported over a long range of qubits through as set of swap gate in order to bring it next to the associated qubit of the gate. Moreover, the same number of swap gates are used to restore the qubit to its initial location.

Khan, et al [13] introduced the set of ternary gates based on Galois Field logic. This branch of logic is a subset of the finite field logic where the field order is either a prime GF(p) or power of prime GF(p'). For the purposes of this ternary logic, the GF(p) is typically used where the GF(p) is the finite field of residue classes modulo p. In their paper, Khan et al, also introduced the ternary swap gate which I will discuss later and build upon to derive the set of multiple valued ternary and swivel gates.
The chapter starts with a discussion of LNNM logic and binary swap gates and demonstrates the extension to binary swivel gates in section II and calculates the cost of binary swivel gate in section III. Analogous construction of the ternary swivel gate is presented next followed by the derivation of cost of the ternary swivel gate, sections IV through VI. Finally, sections VII through IX show the derivation of multiple-valued swap gate, swivel gate and the calculation of number of gates for each.

12.2 Binary Swivel Gate

Figure 12-1 (a) shows a set of swap gates which bring the distant qubits (a and d) next to one another facilitating the LNNM interaction. A second set of swap gates immediately follow (mirror) to restore the a qubit back to its original place in the circuit. Due to introduction of the mirror set of swap gates, the gray CNOT gates effectively cancel one another, and hence it is possible to use 2 CNOT gates to represent the functionality of a swap cascade as shown in Figure 12-1(b).

![Diagram of binary swivel gate](image)

*Figure 12-1 (a) LNNM rendition of distant CNOT gate (on far left) which is brought into LNNM compliance with the aid of swap gates (block of 3 CNOTs), (b) minimized version of (a) where the shaded CNOT gates cancel one another.*

The swivel gate operates on a number of qubits (n) and it transforms the set of qubits by symmetrically swapping the content of qubits around the center. In the case of an odd number of qubits, the qubit in the center represents the pivot point, and in the case of an
even number of qubits, the gap between the two central qubits represents the pivot point – see Figure 12-2.

**Definition 1**: An n variable swivel gate $S_v(n)$ swaps operating on the qubits $(x_1, x_2, ...x_n)$ swaps the values of the qubits around a central pivot point in a symmetrical arrangement. In essence, the swivel gate transforms the input $(x_1, x_2, ... , x_{n-1}, x_n)$ to $(x_n, x_{n-1}, ... x_2, x_1)$.

Figure 12-3 shows the block diagram of a swivel gate presented in [12]. Notice that the well-known binary swap gate (shown within the dotted box) is a special case of the generalized swivel gate introduced in this chapter. The 4-bit binary swivel gate shown in the figure exhibits a regular recursive structure of CNOT gates which repeats in multiple stages. The pattern of repetition can be visualized as a set of down ladders which start from the left of the gate where each successive CNOT ladder has one less qubit. Once the last pair of qubits interacts through a two gate CNOT ladder, a rising ladder completes the gate.

![Figure 12-2 Pictorial of swivel gate with (a) odd number of qubits, (b) even number of qubits.](image)

The down ladder pattern propagates the upper qubit (a) down to the lowest line (4) while shifting all other qubits (c-d) up by one position. A similar pattern applied to lines
1-3 propagates the qubit on line 1 (b) down to the lowest line (3). Lastly, the last pattern applied to lines 1-2 represents the 2-qubit reduced swap gate pattern seen in Figure 1 above. At this stage, line 1 has the value of qubit (d) while all other lines hold their final qubit value in addition some remainder term as a result of the down ladder. In order to remove the remainder term, I successively apply a set of CNOT gates between every two neighboring qubits. Notice that the remainder term for each qubit is exactly the same as the value of the qubit immediately above it. For example, the last two lines hold the terms:

\[ \begin{align*}
\text{line } 3 &= b \oplus c \oplus d \\
\text{line } 4 &= a \oplus b \oplus c \oplus d \\
\end{align*} \]

By applying a CNOT operation between from line 3 to line 4, I eliminate the remainder term as follows:

\[ \text{line } 4 = (a \oplus b \oplus c \oplus d) \oplus (b \oplus c \oplus d) = a \]

Repeating the same process for each pair of qubits will remove all remainder terms and yield the result expected by the swivel gate.

![Figure 12-3](Image)

*Figure 12-3 Four bit binary swivel gate composed of a set of CNOT gates. The upper right triangle shows the well-known swap gate composed of three CNOT gates.*
12.3 Quantum cost of binary swivel gate

The quantum cost of a binary swivel gate can be easily calculated from the number of CNOT primitive gates used to construct this composite swivel gate.

\[
\# \text{CNOT}_n = 2 \cdot \sum_{k=1}^{n-1} k + (n - 1) = (n)(n - 1) + (n - 1) = n^2 - 1 \quad (1)
\]

Equation (1) also represents the binary LNNM quantum cost of the swivel gate.

12.4 MV Ternary SWAP Gate

I now consider construction of a swivel gate for multiple valued logic and I start by demonstrating the concept to the ternary domain. Table 12-1 shows the set of ternary gates based on the Galois Field Sum of Product Logic (GFSOP) for basis 3 GF(3).

<table>
<thead>
<tr>
<th>Input (X)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>GF(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_{00})</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>(X)</td>
</tr>
<tr>
<td>(X_{11})</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>(X + 1)</td>
</tr>
<tr>
<td>(X_{22})</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>(X + 2)</td>
</tr>
<tr>
<td>(X_{12})</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>(2 \cdot X)</td>
</tr>
<tr>
<td>(X_{02})</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>(2 \cdot X + 1)</td>
</tr>
<tr>
<td>(X_{01})</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>(2 \cdot X + 2)</td>
</tr>
</tbody>
</table>

And the following set represents the GF(3) basic literals for variable \(x\):

\[
\text{GF(3)Literals} = \{1, 2, X_{40}, X_{41}, X_{42}, X_{01}, X_{02}, X_{12}, X^2\}
\]

![Figure 12-4 ternary swap gate requiring 5 primitive gates to implement and using Galois Field 3 algebra.](image)
Khan et al [13] discusses GFSOP gates in further details. In the same paper, Khan introduced the ternary swap gate as shown in Figure 12-4. The symbol $\oplus$ representing the XOR gate in binary logic is extended to represent the modulo function, which in the ternary domain, the equation $A\oplus B$ represents $(A+B) \ mod\ 3$. In figure 4, line 2 has the value $a\oplus b$ after the first CMOD3 (Controlled Modulo 3) gate. Were I to follow the exact same pattern of the binary swap gate and place an inverted CMOD3 gate from line $b$ to $a$, I would have ended up with $a\oplus a\oplus b = 2a\oplus b$ on upper qubit which is not desired.

In order to remedy this condition, I place the gate $[[X_{1,2}]]$, symbolized as $2x$ in [13], which performs a doubling operation on qubit (a). Now when I place the inverted CMOD3 gate after the $[[X_{1,2}]]$ gate, I end up with the value of (b) on the upper qubit as follows:

$$ upper\ qubit = [2 \cdot a + (a + b)] \mod\ 3 $$

$$ = (3a + b) \mod\ 3 = b $$

Similarly, the lower qubit has the term $a\oplus b$ instead of simply $a$. By adding the last two CMOD3 gates will eliminate the $b$ term as follows:

$$ lower\ qubit = [(a + b) + b + b] \mod\ 3 $$

$$ = (a + 3b) \mod\ 3 = a $$

12.5 Ternary Swivel Gate

Figure 12-5 shows a 4-variable ternary swivel gate which follows a pattern similar to its binary counterpart. I start by constructing the left hand down ladder which carries the first qubit down to line 4 while shifting all other qubits (b-d) one position up. Notice the placement of the $[[X_{1,2}]]$ operator, shown by its mathematical equivalence as $[[2x]]$, 239
which, analogous to the ternary swap gate, will eliminate the remainder term returned by the inverted gate. The recursive down ladder pattern repeats in a number of stages until the qubit value of the lowest line propagates to the top line. Similar to the binary swivel gate, for a function of \( n \) variables, there exists \( n-1 \) down patterns.

![Figure 12-5: 4-qubit ternary swivel gate which has similar structure as its binary counterpart with the addition of the 2x gate and the double return gates at the far right.](image)

At this stage of construction, the ternary swivel gate has a similar structure to its binary counterpart, and each line has the same GF\((3)\) value on its line. Line 4, in Figure 5, has the value \( a \oplus b \oplus c \oplus d \) instead of the desired value of qubit \( a \). In order to eliminate the remainder term \( b \oplus c \oplus d \), I place two \( \text{CMOD}_3 \) gates rather than one which will remove the remainder term according to GF\((3)\) logic as follows:

\[
\text{line } 4 = (a \oplus b \oplus c \oplus d) \oplus 2 \cdot (b \oplus c \oplus d) = (a + 3b + 3c + 3d) \mod 3 = a
\]

A similar up ladder with double gates is applied to the remaining pair of qubits to eliminate the rest of reminder terms.

12.6 Quantum Gate Count of Ternary Swivel Gate

In addition to the gates found in the binary case, I introduced a set of \([X_{12}]\) to correct for the returning remainder term, and another set of up ladder \( \text{CMOD}_3 \) gates to eliminate
the remainder terms at the end of the cascade in accordance with GF(3) logic. For a swivel gate of \( n \) variables, the number of required gates is:

\[
\text{number of gates} = (n^2 - 1) + \frac{n^2 - n}{2} + (n - 1)
\]

\[
= \frac{3n^2 + n - 4}{2}
\]  

(2)

12.7 MV Swap Gate

I now extend this concept further to higher computational basis following the same construction and using GF(r) mathematics where \( r \) represents the basis of computation. Notice that, according to the definition of GF logic, the value of \( r \) must be prime number. Figure 12-6 shows a multiple valued swap gate using the GF(r) logic. Notice that in order to eliminate remainder terms in the final values of the qubits, a set of \((r-1)\) CMOD\(_r\) gates are needed for each rung of the up ladder.

12.8 MV Swivel Gate

Figure 12-7 shows a generalization of the ternary swivel gate for multiple valued logic of basis \((r)\). Building on the MV swap gate, a set of multiplication gates with a constant multiplier \((r-1)\) is used to eliminate remainder terms for moving qubits up in the down ladder. The up ladder requires a cascade of \((r-1)\) controlled GF(r) gates between each two neighboring lines in order to remove the final remainder terms.
12.9 Quantum Gate Count of Ternary Swivel Gate

I can now calculate the number of gates necessary for the MV swivel gate as follows:

\[
\text{number of gates} = 3 \cdot \sum_{k=1}^{n-1} k + (r - 1) \cdot (n - 1) \\
= 3 \left[ \frac{n(n-1)}{2} \right] + (r - 1) \cdot (n - 1) \\
= (n - 1) \left( \frac{3n}{2} + r - 1 \right)
\] (3)

Table 12-2 shows the number of quantum gates along with the general formula for calculating the number of quantum gates needed to construct a swivel gate of any basis.

**Table 12-2 Number of gates of swivel gates for binary, ternary and multiple valued radix-\(r\) basis of computation.**

<table>
<thead>
<tr>
<th># qubits</th>
<th>Binary</th>
<th>Ternary</th>
<th>MV Basis ((r))</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>(r + 2)</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>13</td>
<td>(2r + 7)</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>24</td>
<td>(3r + 15)</td>
</tr>
<tr>
<td>5</td>
<td>24</td>
<td>38</td>
<td>(4r + 26)</td>
</tr>
<tr>
<td>6</td>
<td>35</td>
<td>55</td>
<td>(5r + 40)</td>
</tr>
<tr>
<td>(n)</td>
<td>(n^2 - 1)</td>
<td>(\frac{3n^2 + n - 4}{2})</td>
<td>((n - 1) \left( \frac{3n}{2} + r - 1 \right))</td>
</tr>
</tbody>
</table>
12.10 Conclusion

In this chapter I introduced the generalized multiple valued swivel gate for $n$ number of variables which is applicable to any basis of computation. I have also calculated the number of GF(r) logic gates necessary to implement this logical block. The family of binary swivel gates has proven useful for implementation of LNNM compliant MCT$_n$ gates as shown in section 8.4. I believe that this family of composite gates will be useful for other applications because it follows a regular structure and is scalable in respect to the number of variables and to the basis of computation.
Chapter 13 Final Conclusion

My earliest thought as I first started to learn about quantum computation and quantum mechanics was one of bewilderment and skepticism at the fact that such phenomena are feasible. Einstein’s famous references to quantum mechanics as “Spooky actions at a distance” and “God don’t play dice with nature” provided a level of affirmation to my skepticism; yet at the same time, piqued my curiosity to dive deeper into this subject and learn about the new understanding of nature. As I started to learn about reversible logic and quantum computing and quantum logic synthesis, I initially had trouble understanding the advantages that such a technology could hold over our increasingly fast processors. I also wondered whether this path of research is mostly theoretical and would never have any value or, possibly, one day, my research would be useful in advancing this burgeoning field. Reversible logic was also a new topic which I had not encountered in my academic or professional career and I was not sure why reversible logic is a topic of importance since a typical circuit has a one directional flow of information: inputs to outputs.

I was already aware by Gordon Moore’s famous prophecy and I was convinced that such prophecy has an end date that was approaching quickly. Issues of heat dissipation from processors, hard drives, and all our technologies was common knowledge as well. I started to grasp the importance of reversibility and its relation to heat dissipation from Landauer’s [1] who proved that binary logic circuits built using traditional irreversible gates inevitably lead to energy dissipation, regardless of the technology used to realize
the gates. He equated that the loss of information in a classical gate manifests as heat because information is energy and energy cannot be lost, rather, it transforms from one form to another. For example, a classical OR gate has two inputs and a single output, where, assuming that it takes an amount of energy $\mu$ to maintain the information for each input, the total energy fed into the OR gate equals to $2\mu$. However, since a single output is produced by the gate at an energy level of $\mu$, the remaining energy exits the OR gate in the form of heat.

As modern silicon technology packs more and more transistors within smaller volumes, heat dissipation becomes even more of a problem. Zhirnov et al. [98] showed that power dissipation in CMOS technology will eventually make it impossible to remove the heat efficiently and foretold of 2020 as the milestone when classical CMOS technology will reach such a roadblock. Bennett [3] also declared that the use of reversible gates will eliminate or greatly reduce the loss of heat and confirmed the earlier prediction by Landauer that the preservation of information will preserve power and eliminate heat loss. Reversibility is truly a simple concept to comprehend where a reversible computing block exhibits a one-to-one mapping between its inputs and outputs. In theory, one can trace back the input value if the output value is known, and vice versa. Permutative specifications are the most common example of reversible circuits where all minterms of output vector are permutations of the minterms of the input vector.

Quantum gates exhibit such reversibility where quantum computation utilizes quantum mechanics phenomena to perform computations that are typically performed in a closed system in which the total energy of the system must be preserved. Specifically, the principle of preservation of angular momentum of an electron, which represents an
electron spin, must be preserved – an electron spin (up or down) represent the basis states $|0\rangle$ and $|1\rangle$ of a quantum system. Nielsen and Chuang [78] showed that a binary permutative specification complies with these quantum principles which as I noted earlier, a permutative specification is reversible, which confirms Bennett [3] conclusion, that such circuits will not exhibit heat loss.

Still, is the field of quantum logic theoretical or do I have empirical evidence that such principles are feasible, technologically, for future computations. Nielsen and Chuang [78], Stock [90], Cirac [16], Holzscheiter [15] and others described real implementation of technologies based on quantum mechanics used to perform actual computations and measurements in the lab. The most leading technologies are Ion-Trap and Nuclear Magnetic Resonance (NMR) which have been used to initialize qubits with information and then perform computation using either laser or EMF pulses to perform the computations.

In conclusion, the future of transistor-based technologies is reaching its limits and the urgency to discover new methods of computations is real. Quantum computation is one of the promising technologies, which skirts the issues of heat loss, inherently suitable for massively parallel computations, and have been shown to be feasible. As with existing technologies, automated logic synthesis is one of the essential tools for implementing large systems, and I chose to focus my research on the developing field of automated quantum logic synthesis of permutative specifications for large number of variables. Although other research has been done in this space, I found that the focus has been on small circuits, on methods that add many ancillary qubits, and methods that do not account for the physical implementation of quantum gates. I wanted to provide
algorithms to automate the synthesis of large circuits and redefine the yardstick used to measure the performance of an algorithm in order to account for the physical constraints imposed by the technology – specifically, linear nearest neighbor model. I also wanted to explore logic synthesis algorithms in the ternary and other multiple valued logic domains.

13.1 Accomplishments of my research

13.1.1 Automated Logic Synthesis of Large Binary Specifications

I created a succession of algorithms that accept a permutative binary specification and automatically generate a quantum cascade of gates that implement the circuit. Although similar algorithms existed at the time with the same general goal, they lacked the focus on large number of variables, took a long time to synthesize, added a large number of ancillary qubits or did not explore other solutions in the search space. The following table provides a comparison of my contributions (shaded) to others:
Table 13.1 Comparison of my binary synthesis algorithms, MMDSN and CSP, to the work of others in the field. My algorithm offers the best compromise for this multiple optimization problem, where the number of variables is maximized while the time of synthesis, quantum cost are minimized without the addition of any ancillary variables.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Quality of Results</th>
<th>Speed</th>
<th>Size of Function</th>
<th>Scalability</th>
<th>Complexity</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMDSN [21]</td>
<td>Low quantum cost, no ancilla bits</td>
<td>Slow (hours for 12+ bits)</td>
<td>30</td>
<td>Multithreaded, Encodes values in bits</td>
<td>$kO(2^n)$</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>CSP with Tabu Search [24]</td>
<td>Lowest quantum cost, no ancilla bits</td>
<td>Slow (hours for 11+bits)</td>
<td>30</td>
<td>Multithreaded</td>
<td>$nkO(2^n)$</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>MMD [56]</td>
<td>Medium, Only explores a single solution</td>
<td>Fast (minutes for 12+ bits)</td>
<td>12</td>
<td>None</td>
<td>$O(2^n)$</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>Agrawal/Jha [63]</td>
<td>Medium, storage space quickly exhausted, and adds ancilla bits.</td>
<td>Very slow (hours for 6+ bits)</td>
<td>16</td>
<td>Limited by the amount of memory.</td>
<td>$nO(2^n)$</td>
<td>Not guaranteed</td>
</tr>
<tr>
<td>Shende [61]</td>
<td>Exact for small functions</td>
<td>Very slow</td>
<td>3</td>
<td>Limited by memory and time</td>
<td>$nO(2^{n+1})$</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>Patino [99]</td>
<td>Good, some backtracking</td>
<td>Very slow</td>
<td>18</td>
<td>Limited by memory and time</td>
<td>$nO(2^n)$</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>Stedman [11]</td>
<td>Best MMD style search solution</td>
<td>Very Slow (days for 7+ bits)</td>
<td>5</td>
<td>Limited by time</td>
<td>$O(2^{n+1})$</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>Wille/Dreschler [57, 59]</td>
<td>Medium, Adds many ancillary variables</td>
<td>Slow (hours for 9+ bits)</td>
<td>16</td>
<td>Limited by memory</td>
<td>$nO(2^n)$</td>
<td>Guaranteed</td>
</tr>
</tbody>
</table>

13.1.2 **MMDSN Algorithm:**

I collaborated with Alhagi on the implementation of this algorithm which, unlike MMD, explored a large subset of the search space, and unlike Stedman, finished in a short amount of time and is capable of performing up to 30 bit functions. Compared to MMD, MMDSN is able to achieve improvements of up to 55% for smaller functions. However, similar to all other algorithms in the space, as the number of variables increases, the performance becomes lower in quality of quantum cost, and higher in the time it takes to synthesize. Stedman on the other hand who synthesizes every possible solution in the search space performs well compared to MMDSN, but quickly halts at 5
variables, as the number of solutions equals to $2^n!$. MMDSN establishes a balance between the time it takes to synthesize and the number of solutions it explores. In my experiments, the MMDSN algorithm is able to synthesize 10,000 samples for the 9-variable hwb9 function in approximately 1 hour. In comparison, Stedman takes almost 12 hours to synthesize a function of 5 variables. MMDSN also does not add any ancilla bits like the algorithms by Wille and Agrawal.

I published this work in ISMVL 2010 conference and in the Facta Universitatis journal.

13.1.3 Covered Set Partition Algorithm

Although the MMDSN has provided better results than similar algorithms, its performance lagged as the number of variables increased. I realized that, although, the Hasse structure provides a means of constructing converging solutions, that structure only allowed for a subset of the valid solution space to be explored. I developed the Covered Set Partition (CSP) algorithm which includes the MMD sequence and all MMDSN sequences and which also includes only valid solutions. The CSP algorithm partitions the search space into a set of partitions where the partition boundary is adjustable to any of the bit boundaries. By partitioning the search space in this manner, the CSP algorithm has a better chance of discovering solutions of lower quantum cost, since each partition is smaller by a number of magnitudes when compared with the entire search space. I discovered that partition sizes which are in the vicinity of the half point of the number of variables typically result in the best quantum cost. I also discovered that the CSP algorithm provided the best results with the aid of Tabu search.
One of the drawbacks of this algorithm is that it takes longer to execute, however, by focusing on the few partition sizes around the center point, it is most likely that I can discover solutions better than MMD, MMDSN and some of the other similar search algorithms. For functions below 12 bits, the CSP algorithm performed better than MMDSN and MMD by 25% while for 12- and 13-bit functions, the CSP performed more than 1000% better than MMDSN. This of course shows that breaking the search space into smaller subsets allows the CSP to find better solutions.

I also studied the impact of changing the selection process of candidate solutions and compared the results amongst random selection, two variants of a genetic algorithm and Tabu search. I discovered that the use of Tabu search combined with partitioning produce better results than each on its own.

I reported my findings in the International Technology New Generations (ITNG) conference in 2010 and the IEEE Innovation in Information & Communication Technology (ISIICT) conference in 2011.

13.1.4 Redefinition of Benchmarks Measurement

As I learned about some of the physical implementation of quantum computing using Ion-Trap and NMR, I realized that the de facto quantum cost metric used in the literature is not realistic. I decided to explore the Linear Nearest Neighbor Model as a realistic alternative which would bring compliance of benchmark measurement closer to physical implementation. Although I found several attempts by other researchers to address the nearest neighbor constraint, I realized that their contribution was still incomplete since they did not address the internal structure of the composite gates used to construct the
circuit. In addition, none of them provided a comprehensive set of equations to calculate the LNNM quantum cost.

I initially derived a set of equations which calculate the number of swap gates necessary to bring any binary quantum into LNNM compliance. Inserting swap gates was the common method of enforcing LNNM in the literature and everyone ignored the internal structure of the MCT gates. I then derived equations to calculate the number of swap gates necessary to bring any of the MCT gates into compliance with LNNM. Finally, I implemented a program to calculate the LNNM quantum cost for all existing benchmark functions on RevLib [30] and Maslov’s [20] websites.

I then derived a set of equations which optimize the LNNM structure for the most commonly used family of MCT gates by reducing the number of CNOT gates needed for enforcing LNNM architecture. I also investigated arranging the qubits on a 2-D grid and calculate the quantum cost of some of the MCT gates when arranged in a 2 dimensional plan (rather than a straight line).

I published a paper in ULSI 2012 where I reported my results and proposed that the LNNQC should be used for comparison because it is compliant with technology and that the LNNQC brings evenness to comparison amongst different methods of synthesis. I am in the also in the process of preparing a paper to report the improved quantum cost and the 2-D model.

13.1.5 Ternary Quantum Synthesis

I also extended my work into the ternary domain with the hope that using the same number of qubits to hold a higher concentration of information could potentially be
important in the development of quantum computing. I realized that the research in the ternary domain was minimal compared to its binary counterparts, and that few researchers addressed logic synthesis for large functions. I decided to extend my work in constructing valid solutions using the Hasse diagram into the ternary domain and reported the results of my findings.

The ternary domain introduced a new set of challenges that were not applicable in the binary domain. The issue of precedence amongst the three digits extended the potential search space by multiple magnitudes and introduced many possibilities of constructing the Hasse diagram. I addressed this issue and demonstrated the construction of the Hasse diagram using multiple precedence orders and provided a proof that for each order the synthesis algorithm will find a solution.

Miller, Maslov and Dueck [MMD] reported their about their logic synthesis for a 2 variable ternary function using the natural ternary order for their input sequence. AlRabadi reported various methods of synthesis without a particular set of benchmark measures. As I did not find a set of benchmarks for ternary logic for larger number of variables, I introduced the \textit{Hidden Weighted Trit} family of functions which defined a general set of functions for up to 9 variables. I used my ternary synthesis algorithm and demonstrated that it is capable of synthesizing up to 9 ternary variables and that the results are superior to using the single natural order.

One of the challenges with any logic synthesis algorithm is the amount of time it takes to synthesize, and this is particularly important in the ternary domain since the spaced grows at the rate of $3^n$! I constructed a set of experiments to study the performance benefit from using the CUDA graphic processor with 1024 processing cores vs. an Intel
i7 processor with 8 cores. For functions of five ternary variables, CUDA provided a 300% speedup compared to the Intel processor which dropped to 100% speedup for a sixe ternary variable function. Despite of the number of execution cores, CUDA might not be ideal for quantum logic synthesis as the number of variables increase because the local memory buffer (fastest memory access in CUDA) is smaller than such functions and would eventually result in huge degradation of performance. One of the main advantages of the Intel processor is the structure and the size of its L1 and L2 caches which are large enough to hold the local dataset, and its data streaming and prefetching policies are ideal for quantum logic synthesis.

I reported my findings in ISMVL 2012 and published a chapter about the work on CUDA in the book “GPU Computing with Applications in Digital Logic”, TICSP 2012.

13.1.6 Extended Quantum Specification and MV function generator

In addition to introducing the HWT family of functions, I introduced six more multiple valued functions into the literature and created an online website to generate such functions. The function generator is available from the Portland Quantum Logic Group website: quantumlib.cecs.pdx.edu. Using the website, you can request the generation of one of the following functions: Counter of non-zero literals, Reflection (Swivel) gate, Full Adder, Comparator against a constant, Comparator against a variable, and the Hidden Weighted Digit. The user can specify the radix of computation (binary, ternary, ..etc.), the number of variables and whether they want the function to be converted to a reversible specification by adding ancillary digits.
In the process of developing these multiple valued functions, I realized that the current file formats (PLA, REAL) used to specify binary quantum specifications is insufficient for specifying multiple-valued and hybrid quantum functions. In turn, I introduced the new file format, the Extended Quantum Specification (XQS), to accommodate the necessary definition of such functions. The XQS file format is a specific implementation of the commonly used YAML file format.

I will be reporting this work in the upcoming conference, ISMVL 2013.

13.1.7 Generalized Multiple Valued Swivel Gate

In the process of optimizing the Multiple Control Toffoli gates to the Linear Nearest Neighbor Model, I came across a specific pattern that swaps the values of a set of qubits around a pivot point. I realized that this pattern also applies to the commonly used binary two variable swap gate. After spending time to understand the structure of such a pattern, I was able to generalize the construction of such gate to any number of variables in the binary domain and derived a set of equations to calculate the cost of such pattern, which I dubbed as the Swivel gate.

I then explored extending the same pattern to multiple-valued logic and was able to create a generalized construction of the composite swivel gate to any radix of computation. I also provided a generalized set of equations to calculate the cost of this composite gate based on the radix and the number of variables. The swivel gate is LNNM compliant by design that makes it ideal in synthesis algorithms which adhere to the LNNM architecture.

I will be introducing this family of gates in the Reed-Muller workshop in May 2013.
13.2 **Future Work**

I am currently collaborating with Dr. Martin Lucak in Japan on a post processing algorithm aiming at further reduction of the synthesized circuit. This effort is attempts to use intermediate ancillary qubit to share control lines amongst neighboring Multiple Control Toffoli (MCT) gate within a circuit. A secondary post processing stage will further reduce these added ancillary qubits through template matching and substitution of common gate patterns with more efficient representations.

I also plan on pursuing further work in the LNNM binary domain and revise the CSP algorithm to use the LNNM architecture to calculate its quantum cost. I also plan on researching the potential of automated synthesis which takes account of LNNM during the process of synthesis rather than a post processing or calculation stage.
List of Publications


Bibliography


Portland, OR, 2011.


International Symposium on Multiple-Valued Logic (ISMVL'09), Naha, Okinawaw, 2009.

[94] A. Al-Rabadi, L. Casperson, M. Perkowski and Z. Song, "Multiple Valued Quantum Logic".


Logic (ISMVL’04), Toronto, 2004.