3-2021

Commuting Compositions for Quantum Circuit Reduction

Brenna R. Cole

COMMUTING COMPOSITIONS FOR QUANTUM CIRCUIT REDUCTION

THESIS

Brenna R. Cole, Captain, USAF

AFIT-ENG-MS-21-M-023

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-21-M-023

COMMUTING COMPOSITIONS FOR QUANTUM CIRCUIT REDUCTION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Cyberspace Operations

Brenna R. Cole, B.S.

Captain, USAF

March 25, 2021

AFIT-ENG-MS-21-M-023

COMMUTING COMPOSITIONS FOR QUANTUM CIRCUIT REDUCTION

THESIS

Brenna R. Cole, B.S.
Captain, USAF

Committee Membership:

Laurence D. Merkle, Ph.D
Chair

Scott R. Graham, Ph.D
Member

Lt Col Patrick J. Sweeney, Ph.D
Member

AFIT-ENG-MS-21-M-023

Abstract

Quantum circuit simplification improves program execution on quantum hardware by reducing error from prolonged environmental interaction and noisy gate operations. One simplification technique is template matching, which repeatedly conducts local optimization by replacing small sequences of gates within a circuit by optimized versions. Underlying this method is the problem of identifying sequences matching templates. This is challenging because some, but not all, gates can commute within a circuit. This means there may not be a subcircuit that matches a template in the original circuit specification, but a match may exist in an equivalent rearrangement of gates. In such cases, certain reductions are possible only after the consideration of alternative gate orderings. This research focuses on the identification of commuting gate sequences in support of circuit reduction. In particular, this work generalizes the notion of commuting gates and layers to n-layer commuting compositions and identifies all three-layer commuting compositions composed of Toffoli, CNOT, and NOT gates for circuits with three to five qubits.

Acknowledgements

To my parents—thank you for your love, support, and encouragement. You taught me the foundational lessons of hard work, integrity, and humility that form the bedrock of any academic endeavor.

To my advisor, Dr. Merkle—thank you for your guidance, patience and assurance. You taught me how to press forward when unsure of the destination. The Albert Einstein quote you so often shared exemplifies this: "If we knew what is was we were doing, it would not be called research, would it?"

To my committee members, Dr. Graham and Lt Col Sweeney—thank you for the classroom instruction, thesis feedback, and mentorship. Your passion for learning and teaching encourages me to continue exploring new ideas.

To my friends and fellow students—thanks for going on this journey with me. It's been fun!

Brenna R. Cole

Table of Contents

Page

Page

List of Figures

List of Tables

COMMUTING COMPOSITIONS

FOR QUANTUM CIRCUIT REDUCTION

I. Introduction

1.1 Motivation

The United States is in a great power competition with near-peer adversaries [38]. Establishing a strategic edge in this competition is crucial to national defense. A key component to reaching that state is technological superiority. As quoted in the *U.S. Air Force's Science and Technology Strategy*, former Secretary of the Air Force Heather Wilson said, "The advantage will go to those who create the best technologies and who integrate and field them in creative operational ways that provide military advantage" [63]. Leading the race to develop and utilize game-changing technologies is vital to securing a military and economic advantage over adversarial nations.

Quantum computing is one of the developing technologies that will contribute to this competitive edge. The United States has recognized it as so with the enactment of the National Quantum Initiative (NQI) Act in December 2018 [54]. This Act creates a coordinated federal approach to quantum development in order to ensure U.S. leadership in quantum sciences for the nation's security and economic prosperity [47].

In addition to the United States, China and Russia have recognized quantum sciences as a technology leading to a strategic edge. China is building a national quantum science research center, leads the world in number of quantum communication and cryptography patents, and is investing generous funding towards quantum development [21]. Russia, while lagging behind in the early years of quantum re-

search, has joined the race by investing $790 million for quantum research and development [20, 57]. The active pursuit by near-peer adversaries of quantum development increases the urgency for the United States to develop a quantum computer capable of solving real-world problems and using it to strengthen military capabilities.

The foreseen advantage of quantum computers is due to the novel method in which they processes data. Rather than simply speeding up classical computation techniques, it fundamentally alters the manner in which information is processed by harnessing properties of quantum mechanics [45]. As a result, some problems that are classically infeasible to compute have known quantum solutions. This promises huge implications to fields such as data security and materials engineering.

One of the first contributions heralding the power of quantum computing is the algorithm created by Peter Shor in 1999 that efficiently solves the problem of factoring large prime numbers, which is believed to be classically intractable [59]. The difficulty of factoring such numbers is the foundation of the widely-used RSA encryption algorithm. This algorithm is used for securing data in many applications, ranging from bank transactions to military communications. Once a quantum computer exists that is capable of running non-trivial instances of Shor's algorithm, this prominent encryption algorithm will no longer be secure.

Another example of quantum computing potential is its foreseen capability to simulate atomic-level molecular chemistry [19]. Classical computers struggle to precisely simulate molecular behavior of anything larger than a few atoms due to computational constraints [45]. The ability to simulate the structure of complex molecules will enable a more robust understanding of chemical interactions [54]. This could be applied to a range of defense-oriented material engineering problems to create stronger equipment or weapon systems, for example.

While known quantum algorithms such as Shor's exist, hardware capable of run-

ning such algorithms is still being developed. The current state of quantum computers is termed the Noisy Intermediate Scale Quantum Computing (NISQ) era. This refers to quantum computers that have low qubit counts, short coherence times, and high error rates [6]. This is a promising step towards developing useful quantum computers, but it is not enough to achieve the anticipated uses of quantum computing technologies. In order to do so, the physical quantum devices must be improved. While this hardware is being developed, however, there is still much software-oriented research that can be done to improve the state of existing technology and accelerate the advent of quantum computers useful for solving real problems. The research presented in this work contributes to that endeavor by presenting a new way to decrease circuit cost, thus making programs execute more efficiently and yield better results in the current noisy environment.

1.2 Problem Background

The initiative to improve the state of quantum computing spans a wide berth of scientific fields and research areas. Among others, physicists, mathematicians, and computer scientists are working to investigate physical qubit technologies, develop quantum algorithms, and improve computations via quantum error correction codes. The focus of this research—reversible and quantum circuit optimization—is another such field dedicated to the advancement of quantum computing.

The goal of circuit optimization is to minimize the costs associated with circuit execution. This is an objective of both classical and quantum computing—the more efficiently a circuit runs the better. However, the low coherence times and high noise present in current quantum computers add to the importance of streamlining the execution process. Coherence time relates to the length of time a qubit can be expected to remain in an excited state, which is key to harnessing the potential of

quantum computers [24]. The longer a circuit takes to execute, the more likely a qubit is to decohere, rendering the computation useless. Coherence times are low in current devices due to noise, or disruption from interaction with the environment, that disturbs the fragile qubit state. Every gate operation in a circuit induces additional noise; therefore, fewer gates will lead to not only faster execution time, but also less error.

One of the methods designed to reduce the cost of a circuit is template matching. This technique searches a circuit for a subcircuit matching a template of gates, then replaces the subcircuit with an optimized version. An underlying challenge in this approach is pattern matching, or finding a sequence of gates that matches a given template. A significant difficulty of this problem lies in the fact that some, but not all, gates can commute in a quantum circuit. This means that there may exist alternate orderings of gates that realize the same function, and a template may be found in the rearranged order of gates but not the original.

This leads to the main premise of this research: to identify equivalent alternative orderings of gates so that pattern matching algorithms may discover more template matches and corresponding simplifications, leading to greater reductions in circuit costs.

1.3 Quantum Circuit Commuting Compositions

Commutations of quantum gates within a circuit can result in multiple gate sequences realizing the same function. Alternative orderings could yield gate sequences that match templates with known reductions. Previous researchers of quantum circuit simplification, such as Rahman, et al. and Iten, et al., account for pairwise commutation of gates in their algorithms [53, 28]. The latter of these works presents the idea of considering commutations of more than two gates. It states that "in general,

it could happen that in a circuit $C = (C_1, C_2, C_3)$, no gates commute pairwise, but the unitary corresponding to (C_1, C_2) could commute with the unitary corresponding to C_3. Hence, one could bring the circuit C into the form (C_3, C_1, C_2), which could help matching in principle"[28].

The present work generalizes and formalizes that postulate with the proposal of the commuting composition property.

A **commuting composition** is defined as a sequence of $n \geq 2$ elements such that the product (i.e. composition) of the first k elements $(0 < k < n)$ commutes with the product (i.e. composition) of the remaining $(n - k)$ elements.

An **n-layer commuting composition** is a commuting composition of n elements where no subsequences of $(n-1)$ or fewer elements create a commuting composition. In other words, n is the minimal number of elements in this sequence required for a commutation to exist.

In regards to quantum circuits, the elements can be gates or layers (where a layer can contain multiple concurrent gates). The composition of elements corresponds to the product of the unitary matrices corresponding to each element. The identity gate is included as an element: if $k = 1$ or $(n - k) = 1$, the single layer can be viewed as a composition in the sense that it is equivalent to itself composed with the identity. More detailed explanations of these concepts are given in Sections 2.1.2 and 2.1.3.

1.4 Research Objectives

This goal of this research is to answer the following research question and test the associated hypotheses:

Research Question:

> How can commuting compositions of layers within a quantum circuit be used to make circuit transpilation more effective?

Hypotheses:

1. Three-element commuting compositions for circuits composed of NOT, CNOT and Toffoli gates exist.

2. Rearrangements of quantum gates in a circuit can yield reductions not otherwise captured by state of the art optimization tools.

3. Consideration of commuting layers will yield more three-element commuting compositions than accounting for gates alone.

The idea the research question addresses is whether alternative orderings of quantum gates realizing the same function as the original circuit could result in improved program execution after undergoing transpilation techniques. It has already been shown that accounting for pairwise commutation of gates within template matching algorithms can yield better results than searching the original gate specification alone [53, 28]. Pairwise commutation, however, is a small subset of the greater problem that the research question poses. This work further investigates the topic by expanding the search from two to three elements and examining commutations of groupings of gates rather than commutations of single gates. The hypotheses are the propositions tested in support of this investigation. They are designed to prove whether such commuting circuits exist, whether they can improve current reduction techniques, and whether it is beneficial to consider commutations of layers of gates.

1.5 Scope

1.5.1 Three Layers

This work searches for and analyzes commuting compositions of three layers. This extends previous research in two respects. First, previous work considered only single

gates per layer, not the more general layers considered here. Second, it considered only sequences of two gates, not three.

1.5.2 Three to Five Qubits

This work analyzes circuits composed of three to five qubits. It begins with circuits of three qubits because that is the minimum required to allow all gates from the NOT, CNOT, and Toffoli gate set to be in the circuit. It ends with five qubits because this number allows for an initial exploration of three-layer commuting compositions that will help inform whether it is worthwhile to search for commuting compositions in circuits containing more than five qubits. The $O(n!)$ complexity of the problem (See Appendix A) makes it computationally expensive to search for high-qubit count circuits. This five-qubit analysis lays the groundwork to determine whether to expend the resources to consider greater numbers.

1.6 Document Overview

The remainder of this document is structured as follows. Chapter II discusses background information and previous work related to quantum circuit reduction and template optimization techniques. Chapter III describes the methodology employed to identify and analyze three layer NCT commuting compositions. Chapter IV presents the results of the investigation of the commuting sequences. Finally, Chapter V concludes the document by summarizing the contributions of this research and proposes future work.

II. Background and Literature Review

This chapter presents an overview of concepts related to quantum circuit optimization via template matching. It lays the groundwork for understanding the commuting composition problem and its role in quantum circuit reduction. Section 2.1 reviews fundamental concepts of quantum computing; Section 2.2 discusses the process of realizing a given function as a quantum circuit; Section 2.3 describes cost metrics used to analyze the efficiency of a circuit; Section 2.4 presents the template matching optimization technique; and 2.5 describes Qiskit, which is the primary toolset used in this research.

2.1 Quantum Computing Preliminaries

Quantum computing is a computational paradigm using quantum mechanics to process information. This section describes key elements of quantum computation and how they differ from classical computation. It begins with explaining the quantum bit (qubit) in Section 2.1.1, proceeds with how qubits are manipulated via gates in Section 2.1.2, and explains how gates are combined to make a circuit in Section 2.1.3. Finally, this section ends by describing ways in which circuits are specified in Section 2.1.4.

2.1.1 Qubits

The basic unit of information for quantum computing is the qubit. Unlike a classical bit which can exist in only the 0 or 1 state, a qubit can exist in a linear combination of 0 and 1. The state of a qubit is represented as $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, with $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. This state can also be represented as a vector, $\binom{\alpha}{\beta}$. When a qubit is in a linear combination in which both α and β are nonzero,

it is in a state of *superposition*. When a qubit is observed, or measured, it collapses from the superposition to either 0 or 1 [45]. The state corresponding to 0 is written as $|0\rangle = 1\,|0\rangle + 0\,|1\rangle = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ and the state corresponding to 1 is written as $|1\rangle = 0\,|0\rangle + 1\,|1\rangle = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$.

The state space of a quantum system is a Hilbert space over the complex numbers [17]. Specifically, a quantum system's state is described by a unit vector within this complex space [45]. For a single qubit, $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, the amplitudes α and β span a 2-dimensional Hilbert space that comprises every state in which the single qubit quantum system can exist. This 2-dimensional state space is often represented by the Bloch sphere, as shown in Figure 1.

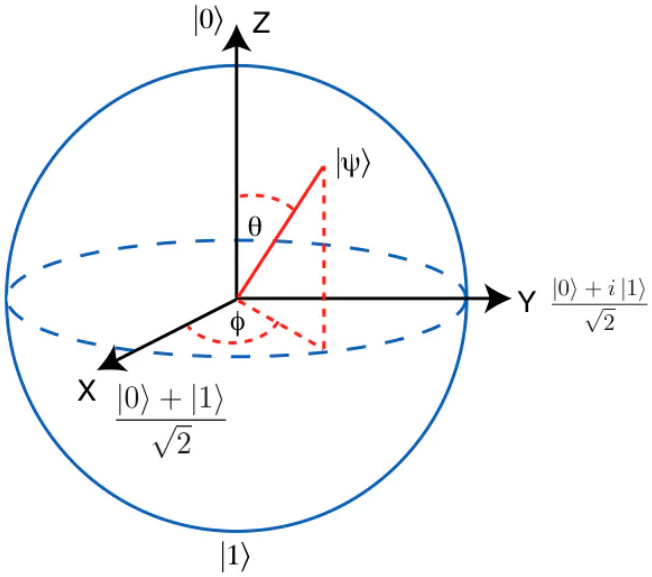


Figure 1: Bloch sphere. Adapted from QuTech [48].

For a system with n qubits, the state space corresponds to a 2^n-dimensional Hilbert space, given by the tensor product of the individual qubit state vectors [27]. For example, a quantum system with two qubits $|\psi_1\rangle = \alpha_1\,|0\rangle + \beta_1\,|1\rangle$ and $|\psi_2\rangle = \alpha_2\,|0\rangle + \beta_2\,|1\rangle$ spans a 4-dimensional Hilbert space, given by

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = \begin{pmatrix} \alpha_1 \\ \beta_1 \end{pmatrix} \otimes \begin{pmatrix} \alpha_2 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_2 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{pmatrix} \tag{1}$$

Qubits are currently realized by a variety of physical technologies, to include ion traps, superconductors, linear optic tools, diamonds, and quantum dots [55]. It is an ongoing area of research to determine which existing or new technologies will emerge as the primary materials used, as each have advantages and disadvantages. For example, ion trap-based systems have seen higher coherence times, lower error rates, and more connectivity between qubits, but have lower qubit counts due to the difficulty of controlling them and their high susceptibility to noise [33, 43]. Superconducting quantum computers, which use the transmon qubit, have higher qubit counts and are easier to control, but have lower coherence times, experience higher error, and have fewer adjacent qubits [33, 34, 43]. The concept of adjacent qubits is described in Section 2.2.3.

2.1.2 Quantum Gates

To perform a computation on a qubit, a quantum gate is applied to it. This is a physical operation, such as a microwave pulse, that is performed on a qubit to alter its state. All quantum gates can be represented as unitary matrices [9]. A unitary matrix is a matrix U such that $UU^\dagger = I$, where $U^\dagger$ is the adjoint of U (complex conjugate of U transposed) [45]. Since $UU^\dagger = I$, the inverse of U ($U^\dagger$) exists and thus every quantum gate is reversible. A gate operation on a single qubit can be visualized by a rotation about the origin on the bloch sphere.

For example, Equation 2 shows the matrix corresponding to the Pauli X gate, also known as the NOT gate [25]. The X gate functions similarly to the classical NOT gate: as the classical NOT gate inverts a bit, the quantum NOT gate inverts a qubit. Visualized on the Bloch sphere, this gate rotates the state vector by 180° around the x-axis.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2}$$

Thus, the X gate inverts a qubit $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ by switching the positions of α and β to give $|\psi\rangle = \beta\,|0\rangle + \alpha\,|1\rangle$. This can be verified by calculating the product of the the X gate and the qubit state, as shown in Equation 3.

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} \tag{3}$$

Most quantum computing devices support gates that operate on one or two qubits [6]. In the two-qubit case, one of the qubits is specified as the control qubit and the other is specified as the target qubit. When the control qubit is in the $|1\rangle$ state, the operation is applied to the target qubit. For example, Figure 2 shows the two-qubit CNOT gate, which applies the X gate to the target qubit ($|q_1\rangle$) when the control qubit ($|q_0\rangle$) is $|1\rangle$. The matrix representation of this gate is in Equation 4.

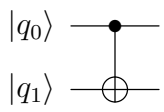


Figure 2: CNOT gate

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{4}$$

2.1.3 Quantum Circuits

A quantum circuit is a sequence of operators applied to qubits in a quantum system. The circuit corresponds to a unitary matrix that is the product of the operator for each layer in the circuit multiplied in reverse order [27]. As an example, consider the circuit generating a Bell State as shown in Figure 3. This circuit entangles two qubits, which means that their states cannot be isolated from one another—the state of the quantum system cannot be written as the product of two individual qubit states [45]. To compute the final circuit unitary, the matrices corresponding to the individual layers must first be computed. This is done by calculating the tensor product of the matrices corresponding to the gates operating on the individual qubits in the system for each layer.

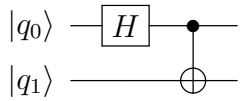


Figure 3: Bell State circuit

In the Bell State example, let L_1 refer to the layer in which the Hadamard gate is applied to $|q_0\rangle$. In this layer, no state-changing operations are applied to $|q_1\rangle$, so the identity matrix can be used to represent the gate for that qubit. The unitary matrix corresponding to the operator for the entire layer is therefore computed as shown in Equation 5.

$$L_1 = H \otimes I = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \tag{5}$$

The matrix representation of the second layer of operations, which in this case will be denoted L_2, is the matrix for the CNOT operation, as shown in Equation 6.

$$L_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{6}$$

The unitary corresponding to the entire circuit can then be calculated by taking the product of L_2 and L_1. As previously mentioned, to calculate the circuit unitary, the order of operands is multiplied in reverse order relative to how it appears in the circuit diagram. A state analysis of the quantum system after the operand is applied for each layer shows the reason for this. The original state is $|\psi_0\rangle$. The state of the system after application of the first layer operand is $|\psi_1\rangle = L_1 |\psi_0\rangle$. The final state of the system after application of the second layer operand is $|\psi_2\rangle = L_2 |\psi_1\rangle = L_2 L_1 |\psi_0\rangle$. Therefore, to calculate the state of the final system, the operator for the entire circuit can be applied to the original state. In this case, that operator is $L_2 \cdot L_1$. The matrix analysis for this is shown in Equation 7.

$$L_2 L_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{pmatrix} \quad (7)$$

Using the operator for the circuit, the resulting state of the quantum system can be computed by multiplying the circuit operator and the quantum system's state vector (by which it is meant to left-multiply the state vector by the matrix of the circuit operator). For example, Equation 8 shows the result of a quantum system in the state $|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ with the Bell State circuit applied to it is $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \left[\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right] = \frac{1}{\sqrt{2}} \left(|00\rangle + |11\rangle \right)$.

$$U|00\rangle = L_2 L_1 |00\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad (8)$$

This means that, when measured, the system will collapse to either $|00\rangle$ or $|11\rangle$ with a 50% probability for each possibility.

2.1.4 Circuit Representations

Quantum circuits can be specified in multiple ways. Three of the most common methods are circuit diagrams, instruction lists, and directed acyclic graphs (DAGs).

A quantum circuit diagram is a figure showing "wires" that represent the logical

qubits in the quantum system. Each gate that is applied to a single qubit is represented with a symbol, commonly a boxed letter or a crossed circle. Gates that operate on more than one qubit are shown with a dot on the control qubit(s) connected to a gate symbol on the target qubit. The representation of the Bell State circuit in Figure 3 is an example of defining a circuit via a diagram.

In addition to diagrams, circuits can be described in text by defining a list of instructions operating on qubits, called a netlist [44]. This is similar to writing a classical program using assembly language. Netlists can be created in multiple formats. Example languages are OpenQASM, Qiskit, and REAL [14, 25, 60]. Circuits represented in these languages are text files that describe the gates in the circuit and the qubit registers on which the gates act. Figure 4 shows an OpenQASM instruction list format specifying a Bell State circuit. OpenQASM is defined in more detail in Section 2.5.2.

```
1  OPENQASM 2.0;
2  include "qelib1.inc";
3
4  qreg q[2];
5  creg c[2];
6
7  h q[0];
8  cx q[0],q[1];
9  measure q[0] -> c[0];
10 measure q[1] -> c[1];
```

Figure 4: OpenQASM code for Bell State circuit

In addition to diagrams and netlists, quantum circuits can be represented as directed acyclic graphs (DAG)s. In this representation, the nodes of the graph correspond to gates, quantum registers, and classical registers, and the edges correspond to logical qubits or classical bits that are the input or output for each of the nodes [25, 44].

As an example, a DAG representation of the Bell State circuit is shown in Figure 5.

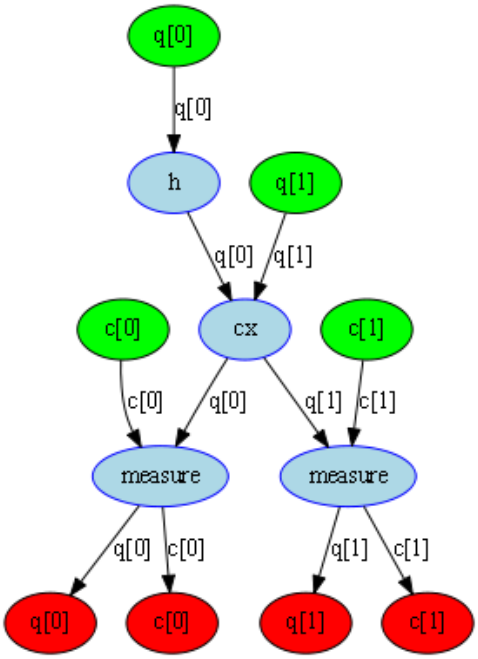


Figure 5: Bell State circuit DAG

Circuit diagrams, netlists, and most DAGs show a specific sequence of gates in the circuit, but that sequence is not necessarily the only one that will compute to the same overall matrix operator. For any two gates A and B, if $AB = BA$, the gates commute and can switch execution order in the circuit without altering the function. In other words, the output after the execution of both gates does not change depending on which executes first.

One example of this is the *moving rule*, formally defined as the property that "two adjacent gates g_1 and g_2 with controls c_1 and c_2, and targets t_1 and t_2 can be interchanged if $c_1 \cap t_2 = \emptyset$ and $c_2 \cap t_1 = \emptyset$" [53]. The two gates depicted in Figure 6a show an example. In this case, each gate's control lines are disjoint from the other gate's target line. This means the gates can be interchanged, yielding the equivalent circuit in Figure 6b.

In contrast to this example is the circuit given in Figure 7, in which the two gates

(a) (b)

Figure 6: Gate commutation via moving rule

share a common line between the target of the first gate and the control of the second gate. In this case, the gates cannot commute and must be executed in the order shown in order to realize the desired function.

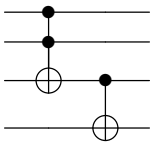
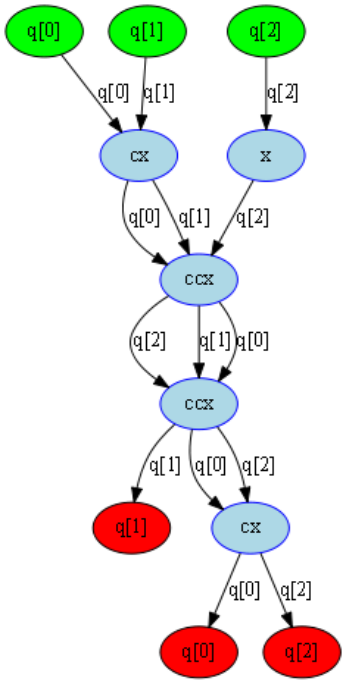


Figure 7: Non-commuting gates

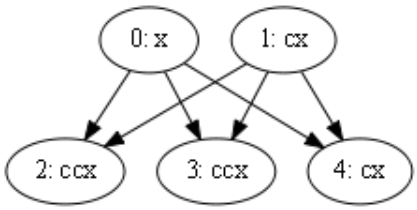
The moving rule shows one instance of when two gates can commute within a circuit. It is not sufficient, however, to account for all gate commutations. In particular, it does not generalize to more than two gates or to gates that are not directly adjacent. The research conducted in this work considers problems that are not captured by the moving rule.

Rahman et al. presented a type of DAG that incorporates the moving rule, known as the *canonical form* or *DependencyDAG* [25, 28, 53]. Unlike the DAG shown in Figure 5, a DependencyDAG only has an edge between gate nodes if one of those nodes depends on the other.

The difference between the previously-mentioned DAG and the DependencyDAG is shown in Figure 8. Both graphs represent the same circuit, but unlike Figure 8a, Figure 8b shows the existence of a dependency between gates by an unlabelled edge



(a) DAG circuit representation



(b) Canonical circuit representation

Figure 8: DAG circuit representations

between nodes. Where there is not an edge, no dependency exists. In contrast, the edges in Figure 8a correspond to the logical qubits upon which the gates are operating. Those edges give no indication of whether the gates can commute. For example, the two CCX gates in this circuit can commute. Figure 8a has a directed edge between the first and second of these gates, corresponding to the gate order in which the circuit was originally specified. It does not indicate whether this order is required for the circuit to realize the desired function. Figure 8b does not have an edge between these gates (nodes 2 and 3). This indicates that they are independent of one another and that the order in which they execute will not impact the functionality of the circuit.

While the canonical form shows pairwise commutations that allow for some gate reorderings (two gates can commute if they are not dependent on one another), it does not depict all possible gate sequences that can realize the original function. It considers dependency between neighboring gates, but there could exist a case in

which pairs of gates could move together in the circuit without compromising the dependency between them or changing the circuit's function. This would yield an alternative gate sequence not captured by the canonical form.

2.2 Circuit Synthesis

The process of realizing an algorithm or function on a physical quantum computer can be decomposed into three steps: synthesizing the function into a reversible circuit (Section 2.2.1), mapping gates from the reversible to quantum level (Section 2.2.2), and modifying the quantum-level circuit to account for physical constraints (Section 2.2.3). Understanding these abstraction levels is important for circuit optimization as simplification can be conducted during any of the steps in this process. Depending on the method used, it may be more effective to run the technique at a higher or lower abstraction level. Furthermore, understanding the decomposition of a high-level reversible gate into quantum gates lends insight into which gates to target for reduction in order to achieve the best performance when the circuit is run on quantum hardware.

2.2.1 Reversible Circuit Synthesis

The first step in implementing a function on a quantum computer is synthesis into a reversible circuit. A reversible circuit is a bijective mapping of inputs to outputs, composed of cascades of reversible gates, and has no fanout or feedback [15]. Unlike a classical gate, a reversible gate must have an equal number of inputs and outputs and a one-to-one mapping between them [56].

Any function, reversible or irreversible, can be implemented as a reversible circuit [2]. If the function is irreversible, ancilla inputs and garbage outputs are used to embed the function into a reversible circuit. Ancilla lines are added to the origi-

nal inputs to enable the function, and garbage outputs are lines for which the final output values are not considered in the result [56]. For example, Figure 9 shows a reversible embedding of a one-bit full adder function (a one-bit full adder includes both a carry-in input and a carry-out output) [61]. The one-bit full adder function is

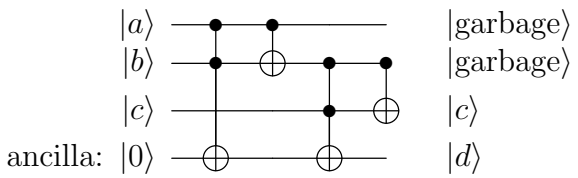


Figure 9: NCT 1-bit adder circuit. Bits a, b, c are added and the result is dc

not inherently reversible, but the addition of the ancilla line and the exclusion of the two garbage lines from the result allows for the reversible embedding.

Although all quantum gates are reversible, not all reversible circuits are composed of quantum gates. Often, abstract reversible gates, which can operate on arbitrarily many inputs, are used. While these gates work well to embed a function into a reversible circuit, they cannot be run directly on a quantum computer, as most quantum devices are only able to implement gates that operate on one or two inputs. There are several universal gate libraries used for reversible circuit synthesis: among the most common are the Mixed-Polarity Multiple-Control Toffoli (MPMCT) gates, Multiple-Control Toffoli (MCT) gates, and NCT gates [2, 5]. These gate sets are listed below from higher to lower levels of abstraction.

- The **MPMCT gate library** is the set containing MPMCT gates. A MPMCT gate is defined as a gate $g(C, t)$ where C is the set of controls lines and t is the target. Each control element is either a positive or negative control. The X operation is performed on the target line if and only if the state of the control line is asserted in accordance with its specification of a positive or negative

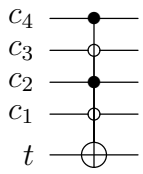


Figure 10: MPMCT gate with four control lines

control [1]. An example of a MPMCT gate is shown in Figure 10. In this example, an X operation is performed on the target if control lines c_1 and c_3 are in the $|0\rangle$ state and control lines c_2 and c_4 are in the $|1\rangle$ state.

- The **MCT gate library** is a subset of the MPMCT gate library for which all controls are positive. This means that the control lines are asserted, and the X is performed on the target, if and only if the control lines are in positive states [2]. An example of a MCT gate is shown in Figure 11. In this example,

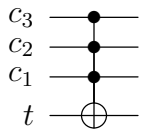


Figure 11: MCT gate with three control lines

an X operation is performed on the target if c_1, c_2, and c_3 are in the $|1\rangle$ state.

- The **NCT gate library** is a subset of the MCT library in which the number of control lines is two or fewer. In other words, it is exactly the set containing the NOT, CNOT, and Toffoli gates [2]. The NOT, CNOT and Toffoli gates are also referred to as the X, CX and CCX gates, respectively. The NCT gates are shown in Figure 12. The matrices corresponding to these gates are shown in Equations 9–11.

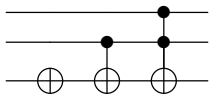


Figure 12: The NOT, CNOT, and Toffoli gates

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{9}$$

$$CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{10}$$

$$CCX = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{11}$$

2.2.2 Quantum Circuit Mapping

Once a function has been synthesised into a cascade of reversible gates, those gates are decomposed into sequences of quantum gates. The quantum gate decomposition is based on a chosen gate library. Two of the common universal quantum gate libraries are the NCV library and the Clifford+T library [2].

The NCV gate library is composed of the Pauli-X gate, the CNOT gate, the Controlled-V (CV) gate, and the Controlled-$V^\dagger$ gate [30]. The V gate is also known as the square-root-of-not gate. The matrix representations for the X and CX gates are shown in Equations 9 and 10 in Section 2.2.1; the remaining gate matrices are shown in Equations 12 and 13.

$$CV = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix} \tag{12}$$

$$CV^\dagger = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \end{pmatrix} \tag{13}$$

The NCV gate library was used frequently in early work on quantum circuit mapping.

Recently, the Clifford+T gate library has been used more often than the NCV gate library due to its role in fault tolerant computing [46]. In particular, the set of Clifford gates has seen promising results in research on quantum error correction using stabilizer codes [12]. The T gate is added to the Clifford set in order to achieve universality. The Clifford+T gate library is composed of the Clifford gates (X, CNOT, H, S, and $S^\dagger$), along with the T and $T^\dagger$ gates [41]. The matrix representations of these gates (other than the X and CNOT gates shown previously in Equations 9 and 10,

respectively) are shown in Equations 14–18.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{14}$$

$$S = \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{i} \end{pmatrix} \tag{15}$$

$$S^{\dagger} = \begin{pmatrix} 1 & 0 \\ 0 & -\mathrm{i} \end{pmatrix} \tag{16}$$

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix} \tag{17}$$

$$T^{\dagger} = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{-i\pi}{4}} \end{pmatrix} \tag{18}$$

Regardless of the gate set used for mapping, this step produces a logical circuit composed of quantum gates that is at the low-level instruction set required to run on an ideal quantum computer. However, there is one final step before it can be run on an actual device—the logical circuit must be converted into a physically realizable circuit by accommodating for any hardware-imposed constraints.

2.2.3 Hardware Constraints

Once an algorithm has been mapped to a logical quantum circuit, the final step is to modify that circuit to account for architecture-specific physical constraints. These are restrictions due to the the construction of the quantum computer rather than the abstract laws of quantum physics.

NISQ-era quantum computers have limited connectivity between qubits. The

physical connectivity of the qubits on a quantum device is referred to as its topology. A topology diagram shows what connections exist between physical qubits. Qubits corresponds to the nodes and an edge indicates they qubits connected [25]. Two qubits are *adjacent* if they are physically connected, which means two-qubit operations are allowed between them. Examples of toplogies for two IBM quantum computers are shown in Figures 13 – 14.

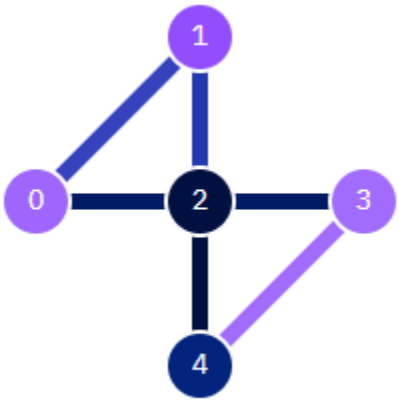


Figure 13: Topology diagram for 5-qubit ibmq_5_yorktown

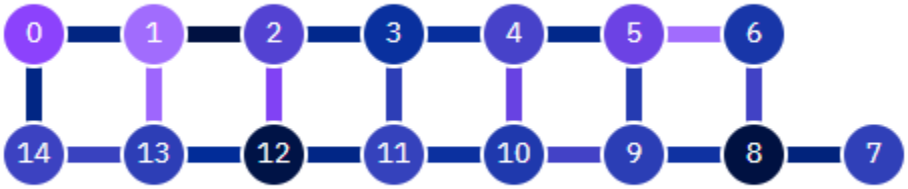


Figure 14: Topology diagram for 15-qubit ibmq_16_melbourne

The linear nearest neighbor (LNN) constraint states that a two qubit gate can only be applied to physically adjacent qubits [31]. For example, physical qubits 0 and 2 in Figure 13 can be used in a two-qubit operation, but physical qubits 0 and 3 cannot. In order to achieve adjacency when a pair of physical qubits involved in a two-qubit operation are not in positions allowing it to occur, SWAP operations are

applied to interchange the positions of logical qubits on the physical topology. SWAP operations are costly and noisy, so an open field of research is how to minimize the number of SWAP operations required to implement a quantum circuit.

One example of such work is presented by Bataille and Luque, in which they analyze the mathematical properties of the set of circuits generated by SWAP and CNOT gates to find minimization techniques [9]. These are incorporated into an algorithm that optimizes gate selection for the circuit, which works so long as the qubit topology is a complete graph. Another example is by Rahman et al., which presents a method to account for the nearest neighbor constraint during the reversible circuit synthesis step as opposed to a post-processing step [52].

A problem related to minimizing the number of SWAP operations is mapping logical qubits to physical qubits. If the logical qubits are mapped to physical qubits such that the two physical qubits involved in a two qubit operation are already adjacent or able to be made adjacent with a relatively small number of SWAPs, the circuit execution will be more efficient and output better results.

An example of this type of work can be seen in the research presented by Murali et al., which inputs an optimization problem formulated from hardware constraints and circuit characteristics to the Z3 satisfiability modulo theory (SMT) solver to output a logical to physical qubit mapping that minimizes execution time by reducing number of inserted SWAPs [42]. Another example of work on qubit mapping is presented by De Almeida et al., which focuses specifically on CNOT restrictions present in IBM quantum hardware. In this research, the authors find optimal qubit permutations adhering to IBM CNOT gate restrictions by formulating it as an integer linear programming problem that accounts for different mapping costs [16].

2.2.4 Previous Work on Circuit Decomposition

This Section describes previous research that has been done on reversible circuit synthesis. It focuses on converting high-level reversible circuits into quantum circuits. Section 2.2.4.1 describes decomposing MCT gates into NCT gates, and Section 2.2.4.2 describes methods for mapping reversible circuits into equivalent quantum-level realizations.

2.2.4.1 Reversible Level

One of the major works used for decomposition of a reversible circuit into a quantum circuit is *Elementary gates for quantum computation*, in which Barenco et al. propose techniques for decomposing MCT gates with large numbers of control lines into NCT gates (MCT gates with two or fewer control lines) [8]. This is commonly referred to as Barenco decomposition or basic Toffoli gate decomposition [58]. An example of this decomposition is shown in Figure 15. This decomposition algorithm is

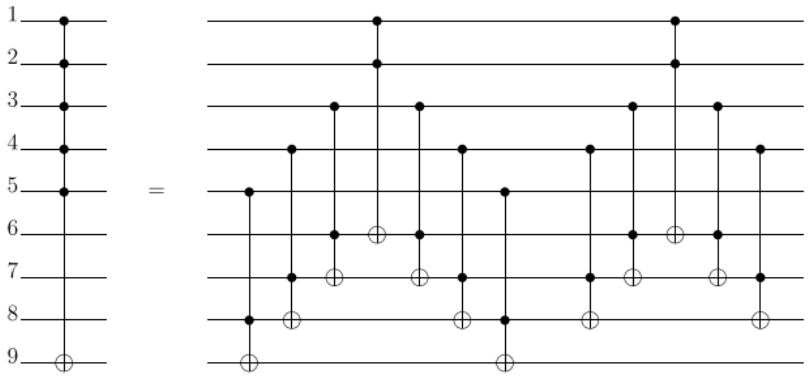


Figure 15: Barenco decomposition example. Reproduced from Barenco et al. [8].

often used as a starting point for researchers mapping reversible circuits to quantum circuits. Barenco decomposition is used to convert high-level reversible circuits to circuits containing only NCT gates. From there, the NCT gates are decomposed into a quantum-level gate set. It is also important to note that while Barenco decomposi-

tion provides an effective method for breaking large MCT gates into smaller ones, it is not the only or necessarily the most efficient way. An ongoing area of research is finding other methods for large MCT gate decomposition.

2.2.4.2 Reversible to Quantum Circuit Mapping

Many methods are employed for synthesis and optimization of reversible circuits into cascades of quantum gates. This present research focuses on template matching, which is described in greater detail in Section 2.4. This section presents alternative approaches so that the reader has a broader understanding of the field and how template matching compares to other solutions.

Much research towards mapping a reversible circuit to a Clifford+T based quantum circuit first maps the reversible circuit to an NCV circuit before transforming the NCV circuit to a Clifford+T circuit. One example is by Miller et al., in which the authors map NCV circuits to Clifford+T circuits by replacing all V and $V^{\dagger}$ gates with Clifford+T equivalents, then rearranging the placement of the T gates to enable possible gate cancellations, and finally optimizing the subcircuits between the T gates by looking for possible CNOT reductions [41]. For each step of this process the authors provide the algorithm for how they accomplished it.

Another example is by Abdessaied et al., which presents an approach to map MPMCT Toffoli gates (as opposed to starting with only MCT or NCT gates) into Clifford+T based circuits [1]. The method presented defines four cases to which a gate can belong. It then considers a reversible circuit gate by gate, and for each gate maps it to a Clifford+T gate cascade based on which of the four cases it matches.

An example of a template-based mapping approach is shown in the work by Biswal et al. [11]. In this paper, the authors use a tool called Colorado University Decision Diagram (CUDD) to transform a function into a binary decision diagram (BDD).

The resulting BDD graph is traversed node by node, and each node is replaced with the quantum subcircuit from a template library that corresponds to the specific node structure. The node is mapped to a reversible circuit, which is then mapped to a circuit comprised of elementary gates from the NCV library, and finally all gates that are not in the Clifford+T library are exchanged for equivalent sequences of fault tolerant gates. After this, the new circuit is traversed for redundant gates that can be eliminated and for patterns in which T gates can be restructured to cancel each other (reducing the T-count) or be executed in parallel (reducing the T-depth).

Another technique used to map reversible circuits to quantum circuits is exclusive sum of products (ESOP). An example of this method for the design of Clifford+T based circuits is by Meuli et al. [39]. This paper first uses a k-input lookup table (k-LUT) mapping technique to map a single large gate into sequences of smaller gates. It then uses ESOP decomposition to map the smaller gates to a Clifford+T network. Following this, they use a post-synthesis optimization method based on graph matching to further reduce the T-count and T-depth.

2.3 Logical Circuit Optimization

Circuit optimization is a major area of research within the quantum computing field. This is particularly important in the current NISQ-era because of the low resource availability and high error rates. Circuit simplification is not only a matter of making a program run more efficiently, but of making it capable of executing correctly at all. Current per-qubit and per-gate error rates are high, meaning that more efficient execution translates to lower overall error rates. Furthermore, the circuit optimization problem is complicated by the fact that current and anticipated quantum computers only implement gates between selected pairs of qubits (See Section 2.2.3). Three primary cost metrics considered in evaluating quantum circuits are gate count, circuit

depth, and ancilla line count [7]. These are discussed in Sections 2.3.1, 2.3.2, and 2.3.3, respectively.

2.3.1 Gate Count

The number of gates in a circuit is a key component in the overall circuit efficiency. In addition to requiring more processing time, the application of a quantum gate to one or more qubits will result in some error, which contributes to decoherence and can affect the outcome of the computation [18]. Therefore, the lower the gate count, the faster the circuit will run and the higher the likelihood it will produce usable results.

The total number of quantum gates that are in the circuit is known as as the *quantum cost*. Often, variations of total quantum cost are considered based on the difficultly of implementing a certain type of gate or the gate type that is preferred for a specific architecture.

For example, most optimization techniques for circuits composed of gates from the Clifford+T set (discussed in Section 2.2.2) aim to reduce the *T-count*, or number of T and $T^\dagger$ gates, as the cost of a fault-tolerant implementation of a T gate is significantly higher then the rest of the gates in the set [1, 11, 46]. This has been the emphasis of most Clifford+T based optimization research in the past, although more recently the number of CNOT gates in a Clifford+T circuit is being considered as well.

Reduction of CNOT gates is important because two-qubit gates have higher error rates than single-qubit gates [23]. Additionally, they can be more costly to implement due to the LNN constraint, which requires the physical qubits involved in a two-qubit operation to be adjacent [10]. In order to achieve that requirement, SWAP operations, which exchange the location of two qubits and are composed of three CNOT gates, are repeatedly used to move the control and target qubits until they are adjacent so

that the logical operation can be executed [6]. This means that in order to execute one logical CNOT operation, many physical CNOTs may be required.

2.3.2 Circuit Depth

Circuit depth refers to the number of distinct layers that exist in the circuit. A layer is a set of quantum gates that can be executed concurrently. The more gates that can be executed concurrently, the smaller the total execution time of the circuit [2].

As with gate count, circuit depth has variations that depend on the gate set used for circuit construction. For example, a Clifford+T adaptation of circuit depth is the *T-depth*, or number of layers containing one or more T or $T^\dagger$ gates [46]. Again, the strong focus on reducing T-depth is due to the high cost of the T-gate implementations. If the T or $T^\dagger$ gates are in the same layer, processing time is used more efficiently. This is because the slot of time used for the layer will be the length required to execute the T gate. If the other gates that need to be executed in this layer are also T gates, their execution times overlap. If the other gates take less time to execute, the processor is doing nothing on the non-T gate lines while waiting for the T operation to finish. The goal, therefore, is to group the T gates into as few layers as possible to minimize the number of layers with long execution times.

2.3.3 Ancilla Lines

Ancilla lines are extra control or data lines added to a circuit [2]. They are required to embed a non-reversible function into a reversible circuit [56]. Additional ancilla lines may also be necessary in the decomposition process from a reversible circuit to a quantum circuit [62].

There is often a trade off between the number of ancilla lines in a circuit vs the other costs, particularly gate count and circuit depth. An example is the approach

taken by Biswal et al., which achieves very low T-depth at the cost of using many ancilla qubits [11]. In this work, the number of ancilla qubits can outnumber the number of non-ancilla of qubits in the computation. This is infeasible with current devices, as existing hardware technologies have low qubit counts.

2.4 Template Matching for Circuit Synthesis and Optimization

This section describes the circuit optimization method of template matching. Section 2.4.1 defines template matching and explains the difference between identity and library templates. Section 2.4.2 explains the main challenges inherent to template matching.

2.4.1 Template Matching Overview

Template matching is an optimization method that reduces circuit cost by repeatedly conducting local optimization of subcircuits [35]. This technique traverses a circuit searching for subcircuits matching known templates, then replaces each identified subcircuit with the corresponding lower-cost but functionally equivalent gate sequence [37].

Research into template matching for reversible circuit simplification started in 2003 with the work done by Miller et al. [40]. This work presents the idea of reducing circuit cost by replacing a sequence of gates with an equivalent but smaller sequence for circuits composed of gates from the NCT set. Building upon this, Maslov et al. published an article in 2005 on template matching specifically for quantum circuits [35]. It extends the earlier work by creating NCV gate templates and presenting the idea of a stand-alone template called an identity template, discussed in Section 2.4.1.1. Since then, there have been many implementations of template matching to improve cost metrics of quantum circuits. Variations include abstraction level, gate

set, number of qubits, and type of template used. There are two primary types of templates for circuit reduction: identity templates and library templates.

2.4.1.1 Identity Templates

As defined by Abdessaied et al., identity templates are circuits realizing the identity operator that are composed of "m gates such that each subcircuit of size less than $m/2$ cannot be replaced by another template" [3]. Since the matrix corresponding to the template is equal to the identity, if the complete sequence of gates in the template is found, the entire sequence can be removed from the circuit. An example of an identity template is shown in Figure 16.

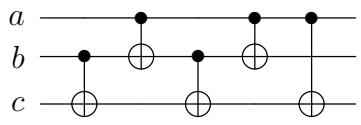


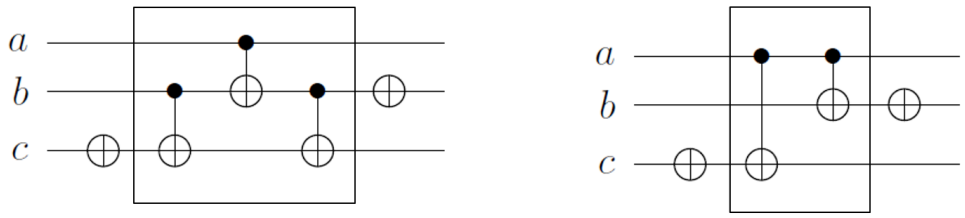
Figure 16: Identity template example. Reproduced from Maslov et al. [35]

Identity templates have the useful property that was presented as Lemma 2 in *Toffoli network synthesis using templates*: "If a network $G_0G_1...G_m$ realizes the identity function, then for any k-shift, $G_kG_{(k+1) \bmod m} \cdots G_{(k-1) \bmod m}$ realizes the identity" [36]. This property is useful in that the template can be thought of as a cycle—it can start with any of the gates present in the template and the sequence of gates going around the ring from the starting gate to the ending gate (directly preceding the starting gate) is an equivalent identity template. Therefore, a sequence matching the template does not necessarily have to start at the first gate specified, but rather can start at any gate so long as the subsequent gate order remains the same.

Furthermore, identity templates have the advantage that the entire gate sequence does not need to be present in order to yield a reduction by template application.

The only criterion to result in a reduction is that more than half of the gates in the template match, with a greater number matching meaning a greater reduction. Formally, this is the following: "a series of gates in a network that matches the sequence of gates $G_i G_{(i+1) \bmod m} \cdots G_{(i+k-1) \bmod m}$ of the template $G_0 G_1 \cdots G_{m-1}$ exactly, is replaced with the sequence $G_{(i-1) \bmod m} G_{(i-2) \bmod m} \cdots G_{(i+k) \bmod m}$ without changing the network's output, where $k \in \mathbb{N}$, $k \geq \frac{m}{2}$" [36].

For example, given the $m = 5$ gate template $G_0 G_1 G_2 G_3 G_4$ shown in Figure 16, at least $\frac{5}{2}$ gates need to match for a reduction. Therefore, if three gates are matched, say $G_0 G_1 G_2$, they can be replaced with $G_4 G_3$, thereby reducing the gate count by one. This can be seen in Figure 17. The boxed subcircuit in Figure 17a can be replaced with the boxed subcircuit in Figure 17b.



(a) Circuit prior to template application (b) Circuit after template application

Figure 17: Application of identity template

A disadvantage of approaches based on identity templates is that they cannot be used to convert between gate sets: the template and its application must be done at the same level of abstraction. One use of template application techniques is to combine the decomposition of a high-level reversible circuit to a quantum-level circuit with the optimization step. Identity templates do not allow this—the decomposition and optimization steps must be done independently.

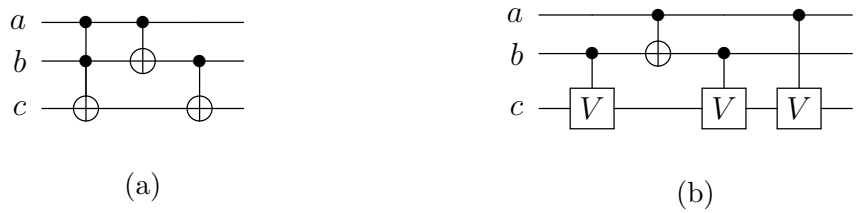
The previously discussed work by Maslov et al. is an example of template matching using identity templates [35]. This work presents NCV gate templates and applies them to quantum-level realizations of MCT gates with up to 11 lines. The results

are compared to the best-known quantum realizations at the time of the writing to evaluate differences in gate count. The results ranged from 76.56% to 93.75% fewer gates in the circuits simplified by the NCV templates.

2.4.1.2 Library Templates

Library templates are a type of template composed of two sequences of gates of differing lengths but equivalent operator matrices. These are referred to as library templates in this work to distinguish them from identity templates. In literature, only the term "template" is used; the specific type is determined by context.

Cost savings via library templates can therefore be realized by replacing occurrences of one element of each pair by occurrences of the corresponding element. These templates do not necessarily compute to the identity; the only requirement is that the correlated templates realize the same operator. An example of a library template is shown in Figure 18.



(a) (b)

Figure 18: Library template. Subfigure (b) shows the optimal NCV version of the circuit in subfigure (a). This template is designed to combine the optimization and decomposition steps so that decomposing the NCT circuit into the NCV circuit will produce better results. Reproduced from Arpita et al. [7]

Two advantages of library templates are that they can realize any operator and that the paired sequences can be composed of different gate sets. Lifting the restriction that the gate sequence must compute the identity offers more flexibility in the construction of templates; any sequence with an optimized corresponding version can be used. This significantly increases the number of templates available. The fact that

the template and its pair can be composed of different gate sets is advantageous in that these templates can be used to combine the optimization and mapping procedures into one step. Rather than decomposing the circuit to a lower level gate set, then applying an optimization scheme, the optimization can occur at the same time as decomposition by mapping small sequences of a higher level gate set to a known optimized sequence of lower level gates.

A disadvantage of library templates is that, for the majority of cases, the entire sequence of gates in the template must be matched. This makes it more difficult to find template matches within the larger circuit, and leads to the templates more often being composed of a smaller number of gates than that of the identity templates.

An example of research using this method was conducted by Scott and Dueck, in which the researchers considered pairs of MCT gates operating on three or fewer qubits on up to four lines and identified the optimal NCV gate realization of each pair [58]. Therefore, the templates in this case were composed of two NCT gates and their equivalent NCV realization. The authors report on the results of computational experiments comparing the effectiveness of decomposing a large circuit gate by gate, as had previously been done, to doing so two-gates at a time. Their experiments use their template matching technique on all 3-bit reversible functions and a set of benchmarks from Revlib [60]. They discovered that of the 40,310 circuits tested for the 3-bit reversible case, 91.45% of them were improved, with an average improvement of 19.565% reduction in gate count. Of the 16 benchmark functions, all were improved. The improvements ranged from a 11.6% to 63.2% reduction in gate count.

Following on to this work is that done by Arpita et al., in which the researchers create the same type of templates as Scott and Dueck, but increase the number of gates in the template to three [7]. Using benchmarks from RevLib, they apply Barenco decomposition to decompose the large MCT gates into NCT gates, then decompose

the NCT circuit into a NCV circuit using their triple-gate templates. They compared the gate count of the resulting circuits to that of Scott and Dueck for eight common benchmarks to find that five circuits had better results using triple gate templates and three circuits had better results using template pairs.

2.4.2 Template Matching Challenges

Although template matching techniques have been shown to improve circuit costs, the challenges of generating templates and identifying template-matching sequences currently limit the impact these techniques can have.

2.4.2.1 Template Search

The paper presented by Rahman and Dueck demonstrates the need for more quantum templates in order to achieve better optimization results [51]. Their research analyzes how template matching reduction compares to proven optimal sequences for 3-qubit circuits. To do so, all three-qubit optimal NCV circuits are generated by an exhaustive search. Next, Barenco decomposition is used to map known MCT realizations of three-qubit functions to NCV circuits. These are simplified via template application with previously published NCV identity templates. The results show that the circuits simplified by template matching rarely produced optimal circuits. The authors analyzed the template library and concluded that it did not contain all templates that would have been applicable in the circuit simplification. From this, they conjectured that finding more templates would improve the results.

The method Rahman and Dueck used to find optimal circuit realizations—an exhaustive search—quickly becomes infeasible with the addition of more qubits [49, 51]. Therefore, their test serves as a case study of the need to find more templates rather than suggesting an exhaustive search is the best way to find optimal circuits. They

concluded their paper with the remark that a fuller set of templates with a corresponding template matching algorithm is required for template matching to produce optimal results [51].

Rahman and Dueck published another paper that tackles the problem of generating templates by finding all identity circuits with three qubits and up to eight gates [50]. While this contribution is beneficial in providing a complete set of templates according to the accepted definition, the authors found that there could exist identity circuits that are not by definition a template, but could result in circuit reduction regardless. Specifically, they noted that the restriction of an identity circuit containing an identity subcircuit could be a limitation in finding optimal circuit realizations. They left the exploration of this idea as future work.

2.4.2.2 Pattern Matching

Finding patterns matching templates within a quantum circuit is challenging because of the property that some of the gates within the circuit can commute, but not all of them. As Iten et al. point out, "If all gates in a circuit commute, pattern matching is straightforward: we can simply check whether all gates in the pattern can be found in the circuit. The other extreme case is to assume that none of the gates in a circuit commute" [28]. In the second case, the list of gates can be searched in-order to determine whether a match exists. Since quantum gates can commute in only some instances, neither of those cases apply. Rather, all of the possible gate orderings must be considered to evaluate the existence of a match.

One work that addresses this problem is presented by Abdessaied et al. [3]. In this work, the authors encode the pattern matching problem into a boolean satisfiabilty problem, which they use a SMT solver called metaSMT to answer. To account for gate commutation, they encode the moving rule (described in Section 2.1.4) into the

SMT solver. They apply to their approach to a set of reduced benchmarks from RevLib, and obtain an average of 11.42% reduction in cost from the original circuits.

Two additional works that address this problem are by Rahman et al. and by Iten et al. [28, 53]. Both of these works present exact template matching algorithms that traverse the canonical representation of a circuit forwards and backwards to identify matching sequences. Both of these works provide heuristic options for reducing the search space at the expense of possibly missing a match. The work by Iten et al. offers more granularity over the heuristics employed. This algorithm has been incorporated into Qiskit, which is discussed in the next section. Both the Rahman and Iten algorithms show positive results when tested on a set of MCT benchmarks.

All three of these works account for gate commutation, but they only consider commutation of adjacent gates. As discussed in the paper by Iten et al., it is possible that commutations other than pairwise exist within a circuit that would not be captured by these algorithms.

2.5 Qiskit Software

The primary software used in this research is IBM's quantum development framework Qiskit [4]. Written in Python, Qiskit is "an open source SDK for working with quantum computers at the level of pulses, circuits and algorithms" [24]. Four components comprise Qiskit: Qiskit Terra, Qiskit Aer, Qiskit Ignis, and Qiskit Aqua [25]. The focuses of the elements are circuits and pulses, simulators, noise and errors, and applications, respectively [26].

As the foundational code base for Qiskit and the element geared towards circuit development between abstraction levels, Qiskit Terra is the component used in this work. Background information on Qiskit Terra is presented in Section 2.5.1. Integrated into Qiskit is the Open Quanutm Assembly Language, which is discussed in

2.5.2.

2.5.1 Qiskit Terra

The Qiskit Terra classes most relevant to this work are the `QuantumCircuit`, `Operator`, `DAGCircuit`, and `DAGDependency` classes. Section 2.5.1.1 describes them in more detail. Additionally, modules for transpilation and conversion are used. The transpilation modules are `template_optimization` and `preset_passmanagers`. The conversion modules are `circuit_to_dag` and `dag_to_circuit`. These are discussed in Section 2.5.1.2.

2.5.1.1 Circuit Representation Classes

The `QuantumCircuit` class is used to represent quantum circuits and contains methods for creating, modifying, and analyzing them. Among other functions, a `QuantumCircuit` object can run on real or simulated backends, pass through transpilation routines, and compute the output expected in a perfect environment. To create a `QuantumCircuit` object, the IBM QX circuit composer, Qiskit language, or OpenQASM string can be used.

Objects from the `Operator` class represent a matrix operator that will evolve a state vector or density matrix [25]. To initialize an `Operator`, a quantum circuit can be passed into the `Operator` constructor as a parameter. This has the convenient result that the unitary matrix corresponding to the circuit can be accessed with the data property of the `Operator` object. The `Operator` class also contains the `compose` method, which calculates the product of two operator matrices.

The `DAGCircuit` class represents a quantum circuit as a directed acyclic graph (DAG). Before undergoing any transpilation routine, a circuit is first converted into a DAG. In this form, the nodes correspond to inputs, outputs, and gate operations,

and the edges correspond to the qubits or bits that are the input or output of the node [25].

Closely related to the `DAGCircuit` is the `DAGDependency` class. This class also represents circuits as directed acyclic graphs, but does so in a way that shows pairwise dependency between gates. In a `DAGDependency` object, the nodes represent gates in the circuit and the edges represent dependencies between two operations. In other words, if there is an edge between two nodes, it means that those nodes do not commute. This class corresponds to the canonical form described in [28] and is used in that work to find sequences within a circuit matching given patterns.

2.5.1.2 Transpiler Passes

The Qiskit transpiler is responsible for circuit transformations. It chains together algorithms that modify a circuit (transpiler passes) via a pass manager [25]. A pass manager consists of one or more passes and schedules the order in which the passes will be applied to the circuit. Qiskit offers `preset_passmanagers` that consist of a defined set of transpiler passes. It also allows for custom creation of passes and pass managers.

2.5.1.3 Preset Pass Managers

The `preset_passmanagers` consist of pipelines of passes corresponding to various optimization levels [25]. These are beneficial when the user requires control over the rigor of optimization applied to the circuit, without configuring the specific passes. There are four optimization levels that range from zero to three. A higher level corresponds to more aggressive optimization. A trade-off exists between level of optimization and transpilation time: a higher level of optimization results in a longer transpilation time [25]. The descriptions of each level are given below [4]:

- **Level 0:** "No explicit optimization other than mapping to backend"
- **Level 1:** "Light optimization by simple adjacent gate collapsing"
- **Level 2:** "Medium optimization by noise adaptive qubit mapping and gate cancellation using commutativity rules"
- **Level 3:** "Heavy optimization by noise adaptive qubit mapping and gate cancellation using commutativity rules and unitary synthesis"

"Adjacent gate collapsing" in Level 1 refers to cancellation of adjacent CX gates in the circuit. "Gate cancellation using commutativity rules" in Levels 2 and 3 is similar, but it considers a larger set of gates and cancels the gates if a gate and its adjoint are adjacent. The "unitary synthesis" conducted in Level 3 means that some gates are composed into a single operator that is returned as a unitary gate defined by a corresponding matrix [4].

2.5.1.4 Template Optimization Pass

The `template_optimization` pass implements the pattern matching and template reduction algorithms presented in [28]. The pass is instantiated with a set of identity templates, then traverses the circuit (considering pairwise commutation of gates) for the longest sequences matching the templates. It then replaces the longest sequences with the corresponding shorter versions. If no templates are specified upon pass instantiation, the default templates of two X gates, two CX gates, and two CCX gates are used. The application of these default passes can be summarized as traversing the circuit for cancellation of adjacent identical gates with consideration of pairwise commutation.

2.5.2 OpenQASM

Integrated with Qiskit is Open Quantum Assembly Language, OpenQASM, which is used to specify quantum circuits [14]. A Qiskit quantum circuit can easily be converted into an OpenQASM string. The language syntax is straightforward, which makes it simple to manually or automatically read a circuit specified as an OpenQASM string. Furthermore, the file format is integrated into tools outside of IBMQX, such as the JKQ Quantum Functionality Representation [13]. This makes it beneficial for quantum computing research regardless of whether Qiskit is the software being used.

2.6 Summary

This chapter covers background information and previous work related to quantum circuit simplification via template matching. Section 2.1 discusses preliminaries on quantum computing. Section 2.2 explains how an arbitrary function is embedded in a quantum circuit. Section 2.3 explains the goals of logical circuit optimization. Section 2.4 presents the technique of template matching for circuit reduction. Finally, Section 2.5 describes Qiskit, which is the primary software used in this work. Previous research related to these topics is presented throughout the respective sections.

III. Methodology

3.1 Overview

This chapter describes the methodology employed to answer the question of whether, within a quantum circuit, there exist triplets of layers such that directly adjacent layers cannot commute, but the first or last layer may commute with the remaining pair of layers. Section 3.2 describes the goals and questions of the research. Section 3.3 describes design decisions for the investigation. Section 3.4 describes the process to find matching sequences, and Section 3.5 describes the analysis of the found matches.

3.2 Goals

The goal of this research is to identify the existence of sequences of three quantum layers with the property that no directly adjacent layers commute, but a single layer may commute with the remaining pair of layers. More formally, this property can be described as a circuit with layers $L_1L_2L_3$ such that $L_1L_2 \neq L_2L_1$ and $L_2L_3 \neq L_3L_2$, but $L_1L_2L_3 = L_3L_1L_2$ or $L_1L_2L_3 = L_2L_3L_1$.

The significance of such sequences is that, should they exist, they could yield completely new realizations of a given gate netlist. This would impact circuit optimization techniques that rely on finding patterns of gates within a circuit, such as template matching [28]. By introducing a new arrangement of gates, pattern matching algorithms could potentially identify longer gate sequences that would allow further reduction of quantum cost or depth within the circuit.

The specific research question and hypotheses guiding this work are presented in Section 1.4.

3.3 Design Decisions

This section describes the decisions made in searching for three-layer quantum circuit commuting compositions.

3.3.1 NCT Gate Set

The search for quantum circuits that are three-layer commuting compositions can be done with any gate set. This work focuses on the NCT gate set, which, as described in Section 2.2.1, consists of the Toffoli, CNOT, and X gate. This set is chosen because it is a useful abstraction level for improving circuits to run on quantum hardware, but is not specific to a single type of device.

High-level reversible circuits, such as those composed of MPMCT or MCT gates, are often first decomposed to NCT gates. There are several well-known algorithms for this, such as Barenco decomposition or Nielsen and Chung mapping [2], [8], [45]. From the NCT gate set, the circuit is decomposed to a quantum circuit composed of one or two qubit gates that can be run on hardware. For example, the three qubit Toffoli gates must be decomposed to machine level gates prior to being run on a device. Figure 19 shows how IBM does this, by decomposing the Toffoli gate into Clifford+T gates. The Clifford+T gate set is currently used in IBM devices, but should that change the NCT gates could just as easily be decomposed into the new low-level gate set implemented on the machine.

By using the NCT gate set to search for matching sequences, circuit simplification

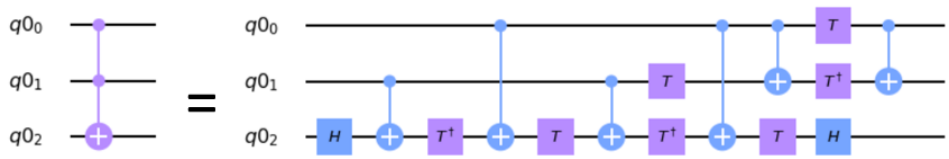


Figure 19: Decomposed Toffoli gate

is done at a low enough level that most functions embedded into a quantum circuit will at some point be decomposed into these gates, but high enough that it will still be relevant should the machine-level gate set change based on differing physical technologies.

A property of the NCT gates set is that, as show in Section 2.2.1, the matrices corresponding to each gate will be a 2 x 2 (X gate), 4 x 4 (CX gate), or 8 x 8 (CCX gate) matrix where each row and column has exactly one '1', and the remaining entries are '0'. Since the matrix corresponding to a layer is calculated by taking the tensor product of the gates in the layer (including the identity matrix), the unitary corresponding to the entire layer will maintain the property that each row and column will contain a single one with zeros as the remaining entries. Furthermore, the unitary corresponding to the entire three-layer sequence will maintain this property.

3.3.2 Layers vs Gates

This work looks at layers of gates rather than single gates. This is a broader scope than single gates alone, as each layer can contain one or more gates. For example, a layer with three qubits could consist of both a CNOT gate and an X gate. The primary motivation for this is that the inclusion of multi-gate layers yields more three-layer sequences to check for commuting compositions, which means that there are more options to find matching sequences, thus greater potential for circuit reordering and cost reduction.

3.3.3 Qiskit Software

Qiskit is the primary toolset selected for this research. It is not the only software that could be used for this purpose: other quantum programming languages such as Jaqa, TriQ, or Quipper would work [22], [32], [43]. Furthermore, a quantum

specific language is not strictly necessary to accomplish the main goal of finding three-layer NCT commuting compositions. For that purpose, all that is needed is software capable of large matrix multiplication. However, Qiskit is chosen because it offers features that smoothly integrate the various elements of this research. These include methods for converting between gates and matrices, statistics about each circuit, hardware compatibility, and straightforward extensibilty for future work.

In particular, the circuit representation and transpilation classes described in Section 2.5.1 led to the design decision of using Qiskit for this work. The application of these classes to this research is described below.

1. `QuantumCircuit`: Stores single-layer and three-layer gate sequences. This allows for conversion to DAGCircuits and Operators, obtaining circuit cost metrics, and running on quantum hardware to evaluate result differences.

2. `DAGCircuit`: Separates three-layer circuits into single layers, then converts each layer into distinct `QuantumCircuit` objects to pass through the CheckMatch algorithm or transpiler passes.

3. `Operator`: Returns the unitary matrix corresponding to a circuit and composes matrices for use in determining whether they are commutative.

4. Preset Pass Managers: Applied to circuits with and without rearranged commuting gates to determine if current techniques yield different results based on gate order.

5. `TemplateOptimization` Pass: Applied to circuits with and without rearranged commuting gates to compare transpiled circuit outputs. Specifically, returns whether the alternative gate orderings result in matched templates that would not be found within the original circuit specification.

In addition to having a well-documented and supported codebase that is directly applicable to the work done in this research, Qiskit was chosen because it is widely used across the quantum community and contains state of the art techniques for quantum computing developments. This makes it a valuable choice for incorporating new ideas with what is currently being done and posturing follow-on work for seamless integration with both this research and the wider quantum computing field.

3.4 Search For Three-Layer NCT Commuting Compositions

The first step in investigating three-layer NCT commuting compositions is to determine if such sequences exist, and if so, to find them.

A brute force search is the chosen method for this. The remainder of this section describes how the brute force search is implemented. Section 3.4.1 presents the initial step of generating all possible unique layers composed of NCT gates given n qubits. It then describes how these layers are combined, with repetition allowed, in sequences of three to obtain all possible three-layer circuits. Once all possible circuits are generated, they are checked to identify whether they meet the criteria to be a three-layer commuting composition. This is described in Section 3.4.2.

3.4.1 Layer and Sequence Generation

To identify the unique layers, gate combinations are broken into cases that correspond to the possible qubit operations on one layer (i.e. CX only, CX and X, CCX Only, CCX and X, etc..). Each case is analyzed to determine the possible layers using those gates. The qubit ordering does matter, as gates with different target qubits will correspond to a different matrices. For example, the three cases for the Toffoli gate acting on three qubits is shown in Figure 20.

Once the unique layers are identified, they are constructed as `QuantumCircuit`

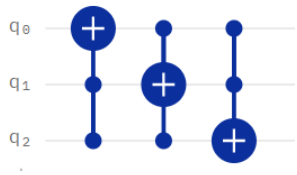


Figure 20: Toffoli gate layers

objects in Qiskit. This allows them to be converted to the `Operator` class, which is useful for algebraic operation and obtaining the circuit's unitary.

The code used to create QuantumCircuits corresponding to each layer type is slightly different for each gate composition. For each case, two functions are utilized: CreateParams and MakeQuantumCircuit. The CreateParams function returns a list of integers corresponding to the target and control qubits. The MakeQuantumCircuit function takes in the result of CreateParams to create the `QuantumCircuit` object. An example of the CreateParams and MakeQuantumCircuit code is shown in Algorithm 1 and Algorithm 2. This example is for the CCX only case for five qubits. For the CCX only case with fewer qubits, the *num_qubits* variable is set to three or four. Similar code is created for the additional gate cases.

Algorithm 1 CreateParams_CCX_5

```
num_qubits ← 5
params ← []
for i in range(num_qubits) do
    for j in range(num_qubits) do
        if j ≠ i then
            for k in range(j, num_qubits) do
                if k ≠ i and k ≠ j then
                    params ← [i, j, k]
                end if
            end for
        end if
    end for
end for
return params
```

Algorithm 2 MakeQuantumCircuit_CCX_5

Parameters: *params* – List of integers corresponding to [target, control1, control2]

```
num_qubits ← 5
Instantiate QuantumCircuit object qc with num_qubits qubits
qc.ccx(params[1], params[2], params[0])          ▷ Add CCX gate to circuit
return qc
```

When a layer is composed of two or more gates, CreateParams uses the lists of single-gate layers to create the multi-gate case. It then iterates through all the elements in each list. For each iteration, it checks if any of the gates are operating on the same qubits. If all the qubits being operated on are distinct, it combines the two lists and appends the newly combined list to the return structure. An example of this is shown in Algorithm 3 for the CCX and X gate case with five qubits.

Algorithm 3 CreateParams_CCX_X_5

```
params ← []
ccx_list ← CreateParams_CCX_5()
x_list ← CreateParams_X_5()
for ccx in ccx_list do
    for x in x_list do
        match ← False
        for qb in ccx do
            if qb in x then
                match ← True
                break
            end if
            if match == False then
                new ← ccx + x
                params.append(new)
            end if
        end for
    end for
end for
return params
```

The MakeQuantumCircuit function corresponding to each multi-gate case makes a quantum circuit, then adds the two or more gates to the circuit according to the

order of integers appearing in the list. For example, in the CCX and X case for five qubits, the first three integers correspond to the CCX gate, and the remaining one or two integers correspond to X gates. The pseudocode for this is shown in Algorithm 4.

Algorithm 4 MakeQuantumCircuit_CCX_X_5

Parameters: *params* – List of integers corresponding to [target, control1, control2, x1, x2 (optional)]

```
Instantiate QuantumCircuit object qc with five qubits
qc.ccx(params[1], params[2], params[0])        ▷ Add CCX gate to circuit
qc.x(params[3])                                 ▷ Add X gate to circuit
if len(params) == 5 then
    qc.x(params[4])                             ▷ Add second X gate to circuit if it exists
end if
return qc
```

To generate all possible three-layer sequences of the identified layers, Python's Itertools module is used in a CombineLayers function. This function is shown in Algorithm 5. It takes in the set of unique layers and combines them in sets of three. It does this by generating the Cartesian Products of three instances of the set of single layers and returning the result. In other words, it returns all three-layer combinations with repetition allowed. The returned list contains tuples of `QuantumCircuit` objects corresponding to layers 0, 1, and 2 in the three-layer circuit.

Algorithm 5 Combine Layers

Parameters: *single_layers* – List of single-layer `QuantumCircuit` objects

```
three_layers ← []
for i in itertools.product(single_layers, repeat=3) do
    three_layers.append(i)  ▷ Stores cartesian products of three single_layer sets
end for
return single_layers
```

The result of the CombineLayers function is then fed into the CheckMatch algorithm. The results of the sequence and layer generation are presented in Section 4.2.1.

3.4.2 CheckMatch Algorithm

The CheckMatch algorithm checks if a three-layer sequence matches the target criteria of having no adjacent commuting layers, but a single layer that can commute with a pair of layers. The pseudocode for this algorithm is shown in Algorithm 6.

The algorithm first checks if any two neighboring layers in the three-layer sequence are the same. If so, it is returned as not a match. This is because the two layers are redundant as they compute the identity. They can be removed from the circuit entirely and the sequence is not of use to this problem. This would be caught in the next step, but checking for sameness first eliminates the need to compute the matrices.

After checking for repeated layers, the algorithm calculates the products of the unitary matrices corresponding to neighboring layers to see if $L_1L_2 = L_2L_1$ or $L_2L_3 = L_3L_2$. If so, the tuple is returned as not a match as it does not match the desired property.

Finally, the algorithm calculates the product of the matrices corresponding to a pair of layers and the remaining single layer, then the product of those terms in reverse. If the result is the same forwards and backwards, the sequence matches the target criteria and the tuple is returned as a match.

All tuples that return true when fed into the CheckMatch algorithm are stored as OpenQASM strings.

Section 4.2.3 presents the results after all three-layer sequences for three to five qubit NCT circuits are run through the CheckMatch algorithm.

Algorithm 6 CheckMatch

Parameters: *sequence* – Tuple of three `QuantumCircuit` objects corresponding to Layers 0, 1, and 2 in the three-layer sequence to check

```
C ← Operator(sequence[0])      ▷ Create Operator objects from QuantumCircuit objects
B ← Operator(sequence[1])
A ← Operator(sequence[2])
if sequence[0] == sequence[1] then      ▷ Check if layers 0 and 1 are the same
    return False
end if
if sequence[1] == sequence[2] then      ▷ Check if layers 1 and 2 are the same
    return False
end if
if CB == BC then   ▷ Check for pairwise commutation between layers 0 and 1
    return False
end if
if BA == AB then   ▷ Check for pairwise commutation between layers 1 and 2
    return False
end if
if ABC == BCA then ▷ Check if layer 3 commutes with composition of layers 1 and 2
    return True
end if
if ABC == CAB then ▷ Check if layer 1 commutes with composition of layers 2 and 3
    return True
end if
```

3.5 Sequence Analysis

At this point, all sequences of NCT gate layers operating on three to five qubits that are three-layer commuting compositions have been found. The next step is to analyze the sequences to see what observations can be made. Section 3.5.1 describes how the results are assessed to determine the existence and frequency of three-layer NCT commuting compositions. Section 3.5.2 outlines the procedure for determining whether the identified commutations can make a difference in circuit reduction using current state of the art techniques. This procedure includes creating a circuit that contains the commuting composition (Section 3.5.2.1), rearranging the gates according to

the allowed commutation (Section 3.5.2.2), running the original and modified circuits through existing optimization algorithms (Section 3.5.2.3), analyzing the resulting circuit costs (Section 3.5.2.4), and evaluating the performance differences on actual quantum hardware (Section 3.5.2.5). Next, Section 3.5.3 explains the methodology for determining whether the commuting compositions are in reduced form. Following this, Section 3.5.4 presents how to determine whether any of the identified sequences contain a single gate per layer. Finally, Section 3.5.5 described how the number and type of operations in each circuit are analyzed.

3.5.1 Number of Matching Sequences

The initial observation to make is the existence and quantity of three-layer commuting compositions using the NCT gate set.

The first part of this—existence—is to determine whether such patterns of NCT gates can be found. If not, this vein of research may not be worth pursuing and efforts would be better spent looking at two-layer commuting compositions (pairwise commutations) or three-layer commuting compositions of different gate sets.

If three-layer NCT commuting compositions do exist, the next question to answer is how many of them there are. This helps determine whether it's worthwhile to invest the computational resources to parse a circuit in search of sequences matching commuting compositions. If there are very few sequences of this type, the likelihood of finding a match within a circuit leading to gate reductions is small. This could imply that, for the typical scenario, the possible gate reduction is not worth the additional transpilation time. Such cases are when a "close enough" result is adequate, or when the fidelity of the computer is high enough that a fewer number of additional gates is unlikely to affect the outcome of the computation. For scenarios where obtaining the greatest circuit simplification is more important than transpilation time, however, it

may be worth the extra resources to search for the additional reductions commuting compositions could yield. This could include cases where accuracy is more important than the speed of obtaining the result, or where resources are very limited (as is the case in the NISQ-era).

If there are many three-layer NCT commuting compositions, it may be worth searching for them in most scenarios, with the exception being when compilation time greatly exceeds circuit reduction in importance.

After finding the number of such sequences, the percentage of commuting compositions compared to all possible circuits is determined. This is to identify if, as qubit count increases, the proportion of three-layer commuting compositions increases, declines, or stays the same. This informs whether searching for matching sequences composed of a greater number of qubits is likely to produce fruitful results.

The results corresponding to this section are presented in Section 4.3.1.

3.5.2 Possibility of Circuit Reduction

Once the existence (or lack thereof) of three-layer NCT commuting compositions has been established, the next consideration is whether they can result in greater circuit reductions than that which are found with current state of the art techniques.

To determine this, an existence proof is used to show whether there does exist a case in which this is true. To do so, the following steps are performed for three, four, and five qubits:

1. Create a circuit containing a three-layer commuting composition subcircuit
2. Modify the circuit by commuting the subcircuit
3. Reduce the original and modified circuits with current optimization techniques
4. Compare the circuit costs following the reductions

5. Compare performance of the original, modified, and reduced circuits on current quantum hardware

The pseudocode for these steps is shown in Algorithm 7.

Algorithm 7 Circuit Reduction Experiment

Parameters: $L_1L_2L_3$ – Three-layer NCT commuting composition
C_{pre} – sequence of quantum gates
C_{post} – sequence of quantum gates

1: $C_{orig} \leftarrow C_{pre}L_1L_2L_3C_{post}$
2: $C_{mod} \leftarrow C_{pre}L_2L_3L_1C_{post}$ **or** $C_{mod} \leftarrow C_{pre}L_3L_1L_2C_{post}$
3: $pm \leftarrow PassManager(TemplateOptimization())$ ▷ Instantiate template optimization pass
4: **for** $circ$ in $[C_{orig}, C_{mod}]$ **do**
5: $pm.run(circ)$ ▷ Apply template optimization pass
6: **for** i in $[1, 2, 3]$ **do** ▷ Apply preset pass managers
7: $transpile(circ, optimization\_level = i)$
8: **end for**
9: $circ.size()$ ▷ Obtain cost metrics
10: $circ.count\_ops()$
11: $circ.depth()$
12: **end for**
13: $C_{red} \leftarrow$ transpiled circuit with lowest costs ▷ Select most reduced circuit
14: **for** $circ$ in $[C_{orig}, C_{mod}, C_{red}]$ **do** ▷ Execute circuits on quantum hardware
15: Run $circ$ on ibmq_vigo
16: Run $circ$ on ibmq_santiago
17: **end for**

The results of these tests are presented in Section 4.2.3.1 for three qubits, Section 4.2.3.2 for four qubits, and Section 4.2.3.3 for five qubits.

3.5.2.1 Circuit Creation

The first step is to create a circuit containing an NCT three-layer commuting composition as a subcircuit. This corresponds to Line 1 in Algorithm 7.

The circuit created in this step is of the form $C_{orig} = C_{pre}L_1L_2L_3C_{post}$, where C_{pre} is the subcircuit containing all gates before the three-layer NCT commuting

composition, $L_1L_2L_3$ are the layers of the commuting subcircuit, and C_{post} is the subcircuit containing all gates after the commuting composition.

These circuits are designed to show possible gate reductions rather than implement a specific functionality. While the circuits do not serve known useful purposes on their own, they could occur within a larger circuit. For example, they could exist as part of the objective function in the quantum approximate optimization algorithm (QAOA).

3.5.2.2 Circuit Modification

To modify the circuit, the commuting subcircuit is replaced with its alternate gate sequence. The gates before and after the subcircuit remain unchanged. The original and modified circuits implement the same function, which can be confirmed by comparing the unitary matrices corresponding to the operator of each circuit.

This step corresponds to Line 2 in Algorithm 7. In some cases, it may be true that $L_1L_2L_3 = L_2L_3L_1 = L_3L_1L_2$. In such a case, either $L_2L_3L_1$ or $L_3L_1L_2$ is selected for use in the modified circuit. As this is an existence proof, as long as a reduction from the modification occurs, only one of the two options is necessary for the algorithm. In this research, the circuits under consideration are small and can be scrutinized by hand. Therefore, the option chosen is based on manually viewing the circuit and selecting the commutation that will result in the cancellation of more high-cost gates.

3.5.2.3 Circuit Reduction

Then next step is to reduce the original and modified circuit using current optimization techniques. These techniques are implemented as transpilation passes in Qiskit. Four passes are used: the first is the template optimization pass; the remaining three are preset pass managers with optimization levels one through three.

The application of the template optimization pass corresponds to Line 5 in Algo-

rithm 7. This experiment uses the default templates of two X gates, two CX gates, and two CCX gates. The application of the preset pass managers corresponds to Line 7. Optimization levels one through three are used. Optimization level zero is omitted as it does not perform any explicit optimization; rather, it only makes the circuit runnable by mapping it to a backend [25].

3.5.2.4 Cost Metric Comparison

After both the original and modified circuits are run through the four transpilation passes, the resulting circuits are analyzed for total number of gates, number of each type of gate, and depth. This corresponds to Lines 9 through 11 in Algorithm 7. These metrics are compared to identify cost differences. The transpiled circuit with the lowest cost is saved as the "reduced modified" circuit for use in the final step.

3.5.2.5 Hardware Performance Comparison

Finally, to compare the difference in performance of the original, modified, and reduced circuits, they are run on two IBM quantum computers: ibmq_vigo and ibmq_santiago. Both are five qubit devices, but with different qubit topologies. The two topologies are shown in Figures 21 and 22. Given that the qubit and link reliabilites for a given machine are not identical, it is possible that either the original circuit or the modified circuit benefits from a "lucky" mapping from logical qubits to physical qubits. Ideally, this would be addressed by gathering statistics over a large number of mappings for each circuit.

Before running the circuits, the device error rates are recorded from that day's calibration. This informs prediction and understanding of the results. A higher error rate will result in the correct answer being returned a lower percentage of the time.

Every time a circuit is executed on a device, it is run 1,024 times. The results

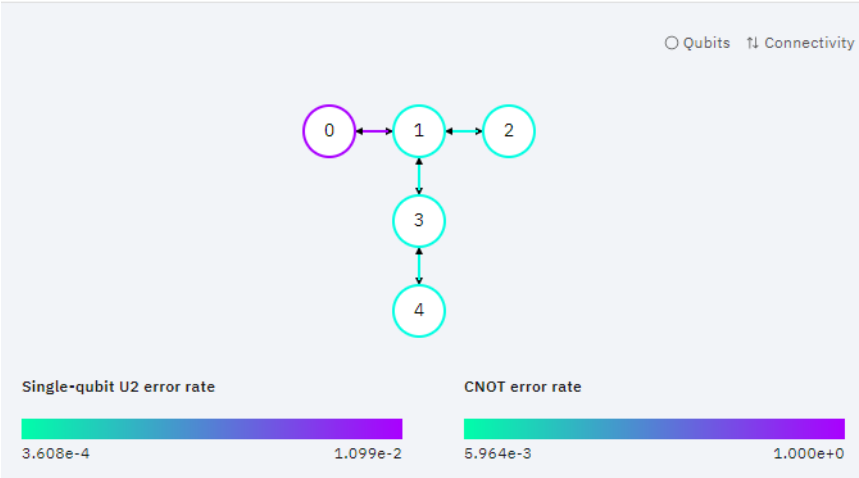


Figure 21: Qubit layout for ibmq_vigo

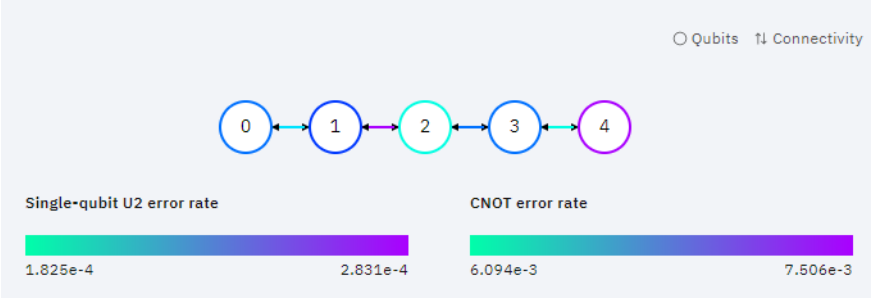


Figure 22: Qubit layout for ibmq_santiago

are returned as measurement probabilities, which correspond to how many of the runs return each output. The correct answer is calculated by applying the unitary operator corresponding to the circuit to a quantum system in the state $|0..0\rangle$. The measurement probabilities corresponding to the correct answer for the original, modified, and reduced modified circuits are compared to determine the impact of circuit reduction in outputting the correct answer.

3.5.3 Removing Redundant Gates

An additional consideration of the identified sequences is whether they are in reduced form, meaning that they cannot be further optimized. Some of the sequences may have trivial reductions, such as the example in Figure 23. In this case, the

adjacent X gates cancel, and the circuit can be reduced to the one shown in Figure 24.

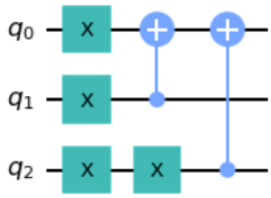


Figure 23: Circuit with redundant X gates

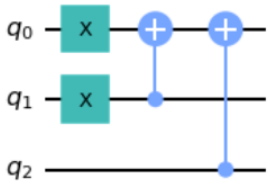


Figure 24: Circuit with redundant gates removed

The cancellation of two adjacent gates will affect the unitary matrix of the layer, so the layer products must be recomputed to determine if the sequence is still a three-layer commuting composition after it is reduced.

A likely use case of commuting compositions is further simplification of already reduced circuits. If a circuit has already been reduced, it is likely that adjacent gates of the same type (at least the X and CX gates) have been eliminated. For example, redundant CX cancellation is included in optimization levels one through three for IBM's preset transpilation passes. Redundant Toffoli gates are not cancelled with the preset pass managers, although they are with the template optimization pass. If a circuit has already been reduced with these techniques, commuting triplets that contain redundant gates will not be found. In such a case, it would be advantageous to eliminate such circuits from the sequences list, as searching for them would consume computational resources but would not result in any further reductions.

To eliminate redundant gates, Qiskit's template optimization pass is used on the NCT commuting compositions.

First, each sequence is composed as a `QuantumCircuit` object with a depth of three. Next, a template optimization pass is created with the three default templates: two X gates, two CX gates, and two CCX gates. After that, the pass is added to a pass manager, and the pass manager is run for each circuit. The output is the transpiled circuit with any redundant gates removed. Finally, all of the transpiled circuits are run through the CheckMatch algorithm to identify if they are still three-layer NCT commuting compositions. If so, they are stored as OpenQASM strings.

The results of this analysis are presented in Section 4.3.2.

3.5.4 Existence of Single-Gate Layer Sequences

The three-layer commuting compositions can be composed of one or more gates per layer. It is worth identifying whether any of the sequences contain a single gate per layer. Firstly, this is because it answers the original question posed by Iten et al. (quoted in Chapter I), which motivated this research [28]. Additionally, single-gate layer sequences could be simpler to find in the follow-on problem of searching a larger circuit for a commuting composition subcircuit. It would require looking at the precursors or successors of a single node in the DAG for a specific sequence of three nodes, rather than the precursors and successors of multiple nodes.

To determine whether any of the three-layer NCT commuting compositions identified for circuits of three, four, or five qubits contain a single gate per layer, the circuits are analyzed with Qiskit's size() function. This function returns the total number of operations in the circuit. For there to be a single gate per layer, this means there are exactly three gate operations. Therefore, the size() function is run on all identified sequences. If the size is exactly equal to three, the sequence is saved as a single-gate

layer sequence, otherwise the sequence is ignored.

The above step will show whether there exist three-gate circuits with three to five qubits that are three-gate commuting compositions. If any are returned as such, the existence question is answered. However, if no three-gate NCT commuting compositions are identified, the question remains open. To fully answer it, all possible three-gate NCT circuits are examined.

Circuits with up to seven qubits are evaluated for this. The seven qubit upper bound is because the sets of qubits operated on by any two adjacent gates cannot be disjoint. If they were, the gates could commute as they have no qubits in common and therefore are not dependent on one another. Thus, the circuit would not satisfy the criteria to be a three-element commuting composition.

NCT circuits with eight or more qubits cannot meet the requirement that adjacent gates operate on at least one shared qubit. The most qubits operated on by a gate from the NCT set is three (the Toffoli gate). If adjacent gates share at least one qubit, each Toffoli gate after the first adds a maximum of two previously unused qubits to the circuit. This gives $3 + 2 + 2 = 7$ total qubits in the circuit. An example distribution for this is shown in Figure 25. A circuit with eight qubits would require that at least two consecutive gates have no qubits in common, so the circuit fails to meet the requirement to be a three-gate commuting composition. An illustration of this is shown in Figure 26.

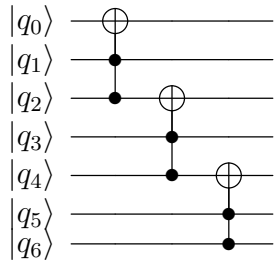


Figure 25: Seven qubits allows for adjacent gates sharing at least one qubit

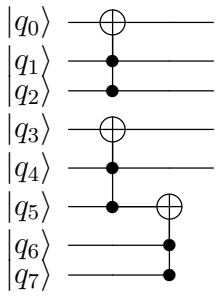


Figure 26: Impossible to use all eight qubits and satisfy the condition that adjacent gates must operate on a shared qubit

Therefore, to prove via exhaustion whether three-gate NCT commuting compositions exist, one, two, six and seven qubit circuits need to be checked in addition to circuits with three to five qubits.

A similar methodology as that to find three-layer commuting compositions is used to search for three-gate commuting compositions. First, all combinations of control and target qubits are generated for the NCT gates. Second, they are combined in sequences of three. Next, the sequences are checked to see if all qubits in the circuit are utilized (if not all the qubits are utilized, that circuit would have already been checked by one of the checks for circuits of a lower qubit count). If all qubits are in use, the intersections of the qubits for the first and second and second and third gates are checked. If they are not empty, that sequence moves to the final step. Lastly, the sequences are run through the CheckMatch algorithm to check if they are three-gate commuting compositions.

The results of this search is presented in Section 4.3.3.

3.5.5 Number and Type of Operations Per Match

A final consideration for the identified sequences is the number and type of gate operations in each circuit. This informs the type of reductions that could be possible

by rearrangements of commuting composition subcircuits within a circuit. For example, one approach of how to use these matches may have the goal of eliminating as many CCX gates as possible. In that case, it could be helpful to know which commuting compositions have higher counts of CCX gates because they may be more likely to provide possible CCX cancellations. Alternatively, it could be possible that it is more common to find X or CX gate cancellations after rearranging the commuting subcircuit. Should that be true, it would be beneficial to know what circuits contain high numbers of X or CX gates. Either way, knowing the number of each type of operations in a circuit could create a rough gauge of possible reductions and help select commuting composition subcircuits to search for in the larger circuit.

To determine these counts, the size() and count_ops() functions from Qiskit are used on each commuting composition `QuantumCircuit` objects. These methods return the total gate count and number of each type of gate in the circuit, respectively. The results are saved in a file, which can then be parsed to select the circuit to study, observe averages, and identify trends.

These results are presented in Section 4.3.4.

3.6 Summary

This chapter covers the methodology used to evaluate the existence, quantity, and properties of three-layer NCT commuting compositions. It describes the design decisions and why they were chosen, the process of generating layers of NCT gates and using those layers to create circuits of depth three. It then presents the algorithm that determines if a circuit is a three-layer NCT commuting composition and explains how the identified commuting composition circuits are analyzed.

IV. Results and Analysis

4.1 Overview

This chapter presents the investigation results and analysis for three-layer NCT commuting compositions. Section 4.2 describes the results from the search for such circuits, to include how many unique three-layer NCT circuits exist for three to five qubits, how many of those are three-layer commuting compositions, and whether the identification of additional commutations can affect the result of circuit execution on quantum hardware. Section 4.3 analyzes the identified circuits for the properties discussed in Chapter III. Finally, Section 4.4 concludes the chapter.

4.2 Search for Three-Layer NCT Commuting Compositions

This section describes the results of searching for three-layer NCT commuting compositions. Section 4.2.1 presents the results of generating all possible layers and layer triplets. Section 4.2.2 uses one of the three-qubit sequences to show an example of the whole process of determining if a sequence meets the target property. Section 4.2.3 presents the results of running the triplets through the checkMatch algorithm. For each qubit count, it gives an example of one of the identified matches and how it improves the circuit transpilation and execution. Finally, Section 4.2.4 gives the time it took to search the circuit sets.

4.2.1 Generation of Sequences to Check

This section describes the generation of all unique layers and three-layer circuit possibilities for three, four, and five qubits operated on by gates from the NCT set.

4.2.1.1 Three Qubit Layers

Circuits with three qubits are the first evaluated. The set of unique layers of NCT gates operating on three qubits contains 22 elements. There are three CCX only layers, six CX only layers, seven X only layers, and six CX and X layers. A layer with only identity gates is not considered due to the fact that a three-layer circuit with an identity layer will not meet the target criteria. The identity layer will commute with its adjacent layers, which disqualifies it from being a three-layer commuting composition.

The breakdown of the 22 layers is below:

- **CCX Gates Only:** Three qubits gives three target options. For each target, there is only one choice of controls: the two remaining qubits. The total number of layers is equal to the number of targets times the number of options for each target. This gives $3 \cdot \binom{2}{2} = 3 \cdot 1 = 3$ unique layers.

- **CX Gates Only:** Again, three qubits gives three target options. For each target, any one of the remaining lines can be selected for the control. With three qubits, this means there are two options for control selection. This gives $3 \cdot (3 - 1) = 6$ unique layers.

- **X Gates Only:** The options to apply X gates to a three-qubit circuit are as follows: all lines have X gates; two of the three lines have X gates; one of the three lines has an X gate. Thus, the number of unique layers with X gates is $\binom{3}{3} + \binom{3}{2} + \binom{3}{1} = 1 + 3 + 3 = 7$.

- **CX and X Gates:** There are six options to apply a CX gate to a three qubit circuit. For each CX gate, there is a single line remaining on which an X gate can be applied. This gives $6 \cdot 1 = 6$ layers.

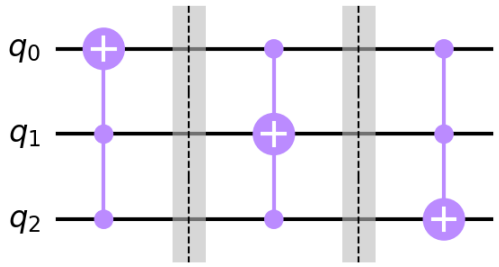


Figure 27: Three-qubit CCX gate layers

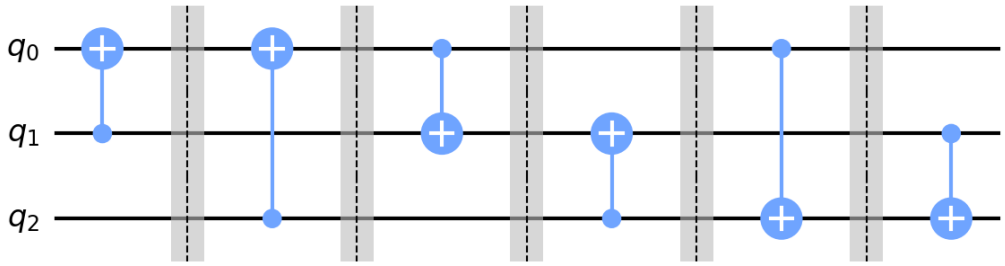


Figure 28: Three-qubit CX gate layers

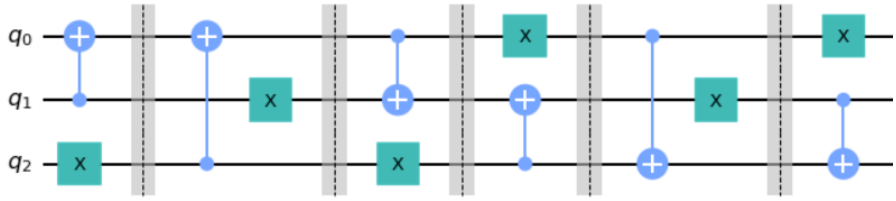


Figure 29: Three-qubit CX and X gate layers

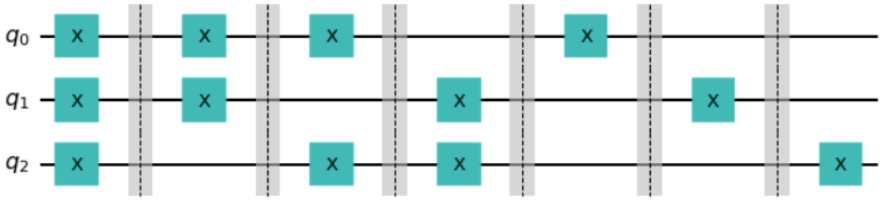


Figure 30: Three-qubit X gate layers

Visualizations of these layers are shown in Figures 27 to 30.

The unitary matrix for each of these layers corresponds to an 8 by 8 matrix. With 22 unique layers, there are $22^3 = 10,648$ combinations of three layers.

4.2.1.2 Four Qubit Layers

The next case considered is four-qubit circuits. There are 99 unique layers for four-qubit circuits operated on by NCT gates. The case of CCX only gates contains 12 layers, the CX only case contains 24, the X only case contains 15, the CCX and X case contains 12, and the CX and X case contains 36. Theses cases are described below:

- **CCX Gates Only:** Four qubits gives four options for target selection. Once a target is selected, there are three remaining lines, from which two controls are selected. Thus, the number of target options times the number of control options for each target yields $4 \cdot \binom{3}{2} = 4 \cdot 3 = 12$ unique layers.
- **CX Gates Only:** Four qubits allows for two cases of layers with only CX gates: one CX gate applied or two CX gates applied.
  - **One CX Gate:** There are four options for target selection. For each target, one control is selected from the three remaining lines. Thus, the number of layers is $4 \cdot \binom{3}{1} = 4 \cdot 3 = 12$.
  - **Two CX Gates:** For each single CX gate applied, there are two remaining lines on which another CX gate can be applied. There are two options for that CX gate (each line can be either the target or control). This gives $12 \cdot 2 = 24$ possibilities. This number accounts for each layer twice. For any two-gate layer composed of gates g_1 and g_2, there is one instance where g_1 is chosen first and g_2 second, and another where g_2 is chosen first and g_1 second. Since the order does not matter, the total number is divided in half, giving $\frac{24}{2} = 12$ unique layers.
- **X Gates Only:** The layers with only X gates can have gates applied to all

qubits, three qubits, two qubits, or one qubit. This gives $\binom{4}{4} + \binom{4}{3} + \binom{4}{2} + \binom{4}{1} = 1 + 4 + 6 + 4 = 15$ unique layers.

- **CCX and X Gates:** For each CCX gate applied, there is one remaining line on which an X gate can be applied. This gives $12 \cdot 1 = 12$ layers.

- **CX and X Gates:** For each single CX gate applied, there are two remaining lines on which one or two X gates can be applied. If two CX gates are applied, there are no remaining lines. Therefore, the number of single CX layers times the X options for each of those gives $12 \cdot 2 = 36$ unique layers.

Each layer corresponds to a 2^4 x 2^4 = 16 x 16 matrix. Combining the layers yields $99^3 = 970,299$ unique three-layer sequences.

4.2.1.3 Five Qubit Layers

The last case considered is a five-qubit circuit. There are 491 unique layers for circuits of five qubits operated on by NCT gates. These layers correspond to the following: 31 options for the X only case, 200 options for the CX and X case (140 possibilities for one CX gate and one to three X gates and 60 possibilities for two CX gates and one X gate), 80 possibilities for the CX only case (20 for one CX gate and 60 for two CX gates), 60 options for the CCX and CX case, 90 options for the CCX and X case, and 30 options for the CCX only case.

- **CCX Gates Only:** For each of the five target options, there are four remaining lines from which to select two controls. This gives $5 \cdot \binom{4}{2} = 5 \cdot 6 = 30$ unique layers.

- **CX Gates Only:** As with the four-qubit case, a five-qubit circuit allows for CX only layers containing one or two CX gates.

- **One CX Gate:** There are five options for target selection. This leaves four qubits from which to select one control. Therefore, there are $5 \cdot 4 = 20$ unique layers.
- **Two CX Gates:** For each CX gate applied, there are three remaining lines for the second CX gate. As shown in Section 4.2.1.1, there are six unique options for applying a CX gate to three lines. This gives $20 \cdot 6 = 120$. Again, this accounts for each layer twice, so the total number of unique layers is $\frac{120}{2} = 60$.

- **X Gates Only:** Layers with only X gates can operate on five, four, three, two, or one of the five qubits. This gives $\binom{5}{5} + \binom{5}{4} + \binom{5}{3} + \binom{5}{2} + \binom{5}{1} = 1+5+10+10+5 = 31$ layers.

- **CCX and CX Gates:** For each application of a CCX gate, there are two lines remaining. That give two options to apply a CX gate to each layer. The yields $30 \cdot 2 = 60$ layers.

- **CCX and X Gates:** For each CCX gate, there are two remaining lines that give 3 options to apply X gates. This yields $30 \cdot 3 = 90$ layers.

- **CX and X Gates:** Layers consisting of CX and X gates can be divided into two cases: one CX gate and one to three X gates or two CX gates and one X gate.
  - **One CX Gate:** When one CX gate is applied in a layer, three lines are remaining to apply X gates. As shown in Section 4.2.1.1, there are seven ways to apply X gates to three qubits. This gives $20 \cdot 7 = 140$ distinct layers.

– **Two CX Gates:** When two CX gates are applied to a layer, there is one line remaining to apply an X gate. This gives $20 \cdot 1 = 20$ layers.

Each of these layers corresponds to a 2^5 x 2^5 = 32 x 32 matrix. The 491 layers are combined to form $491^3 = 118,370,771$ unique three-layer sequences.

4.2.2 Example of Matching Three-Qubit Sequence

Once all of the possible three-layer sequences composed of NCT gates operating on three, four, and five qubits have been generated, the next step is to identify which are three-layer commuting compositions. This section show the process used to determine this by providing a three qubit example. The four and five qubit sequences follow the same process, with the exception that the matrices are 16x16 and 32x32, respectively. The example circuit is shown in Figure 31.

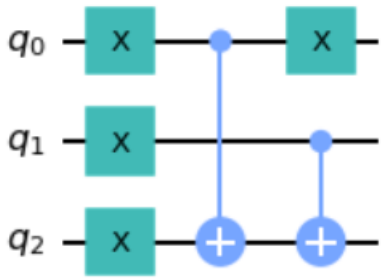


Figure 31: Diagram of example three-qubit circuit

The matrix analysis to determine if the sequence is a three-layer commuting composition is shown in Figures 32 through 35. To start, the matrices corresponding to each layer are shown in Figure 32. The L_1 layer corresponds to the layer of three X gates, L_2 to the layer with a CX gate controlled by q_0 and targeting q_2, and L_3 to the layer with an X gate acting on q_0 and a CX gate controlled by q_1 targeting q_2.

After verifying there are no identical adjacent layers, L_3L_2 and L_2L_3 are computed to check for equality. Their products are shown in Figure 33. Similarly, L_2L_1 and

$$L_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad L_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$L_3 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 32: Matrix representations of layers 1, 2, and 3

L_1L_2 are computed to check for equality (shown in Figure 34). Since the compositions are not equivalent, this gate sequence does not have pairwise commutation of layers.

$$L_3L_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad L_2L_3 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 33: Layers 2 and 3 do not commute

Finally, after validating that the sequence does not commute pairwise, the final step is to check if a single layer commutes with the composition of two layers. This is done by computing $L_3L_2L_1$ (the unitary for the original sequence), $L_1L_3L_2$ (moving layer 1 to the back) and $L_2L_1L_3$ (moving layer 3 to the front). Figure 35 shows this result. As shown in the figure, the matrices are equivalent. Therefore, both the first

$$L_2L_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \quad L_1L_2 = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 34: Layers 1 and 2 do not commute

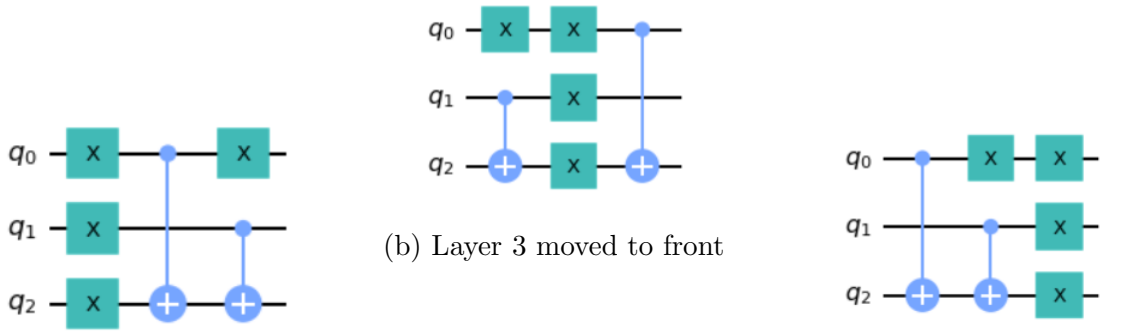
and the last layer commute with the composition of the remaining two layers. This means that all three circuits in Figure 36 yield the same results.

$$L_3L_2L_1 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$L_1L_3L_2 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \quad L_2L_1L_3 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Figure 35: Matrix representations of commuted circuits

A benefit of this circuit is that the commuted realizations of it contain adjacent gates that can be eliminated. In the original subcircuit, this reduction would not have been found since the neighboring layers do not commute. However, changing gate order according to the three-layer commutation yields a new flow of gates that allows for reduction in quantum cost. Specifically, the gate count is lowered from



(a) Original sequence

(b) Layer 3 moved to front

(c) Layer 1 moved to back

Figure 36: Equivalent circuit realizations

six to four. This type of reduction can be used as a library template: if the original sequence of layers is found in a circuit, that sub-circuit can be replaced with the less-costly version to reduce the overall circuit cost.

4.2.3 Identifying Matching Sequences

This section shows the results of applying the algorithm described in Section 3.4.2 and Section 4.2.2 to the identified three-layer NCT circuits to determine which are three-layer commuting compositions. For each three, four, and five qubit case, an example is given of how rearranging the gates within a circuit according to the identified commutation can reduce circuit cost using existing techniques. The three, four, and five qubit cases are in Section 4.2.3.1, Section 4.2.1.2, and Section 4.2.1.3, respectively. An overview of the total number of three-layer NCT commuting compositions found compared to the total number of sequences is shown below in Figure 37.

4.2.3.1 Three Qubit Layers

When run through the checkMatch algorithm, the three qubit circuits yielded 72 of the original 10,648 sequences to be three-layer NCT commuting compositions.

One of the sequences is shown in Figure 38a. In this circuit, either the first or third

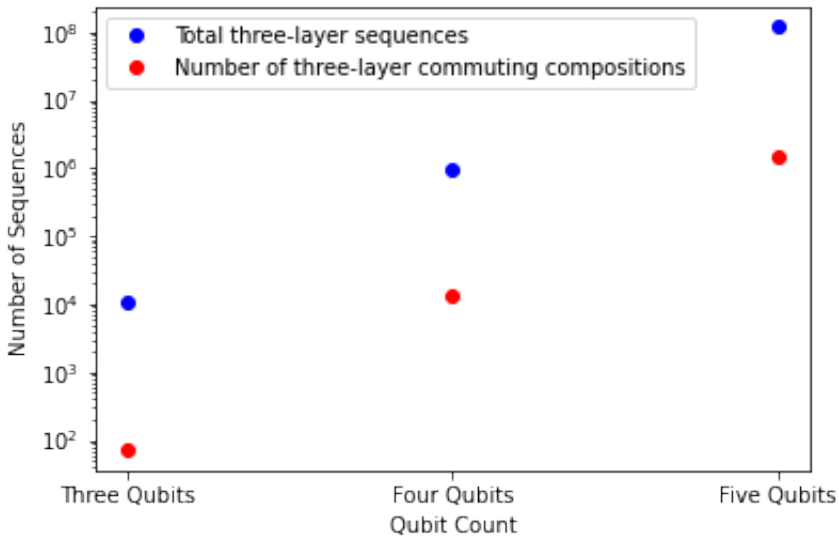


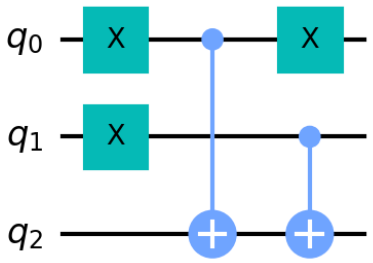
Figure 37: Number of three-layer NCT commuting compositions relative to total sequences

layer can commute with the composition of the two other layers. These commutations are shown Figures 38b and 38c.

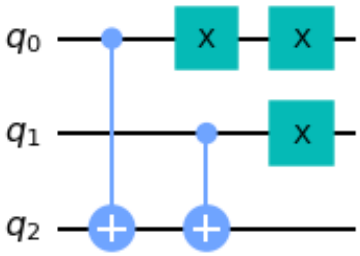
The impact of the sequence's commuting property can be seen in the circuit example shown in Figure 39. This circuit is deliberately created to highlight the potential cost reduction commuting compositions can bring to quantum circuit reduction; it is not taken from an existing benchmark. It is left to future work to create a method to search existing and future circuits for the presence of such sequences.

When the three-layer subcircuit, outlined in red on the circuit diagram, is replaced with its commuted version (first layer commuted with the second and third layers), the circuit changes to that shown in Figure 40.

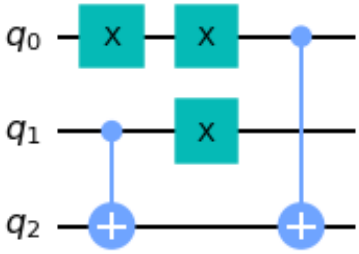
The new sequence of gates presents further cost reduction opportunities. In particular, adjacent gates of the same type can be eliminated. The reduced circuit after adjacent gate cancellation is shown in Figure 41. Comparison of the unitary matrices corresponding to the orignial, modified, and reduced circuits show that they



(a) Three qubit three-layer commuting composition



(b) First layer commuted with compositions of second and third layers



(c) Third layer commuted with composition of first and second layers

Figure 38: Equivalent three-qubit circuits

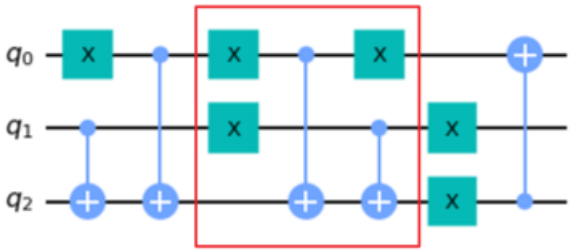


Figure 39: Circuit containing three-layer commuting composition subcircuit

implement the same function.

To show that current optimization techniques would not result in this reduction without the gate rearrangement achieved via the subcircuit commutation, the circuit is transpiled with Qiskit's preset pass managers and the recently added template optimization pass.

Table 1 and Table 2 show the cost metrics post-transpilation for the original circuit

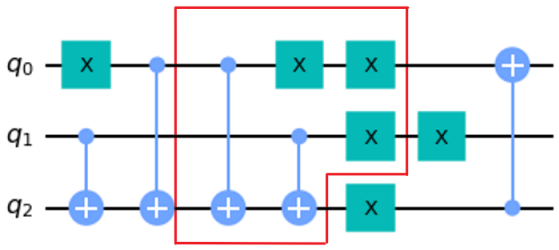


Figure 40: Original three-qubit circuit replaced with commuted subcircuit

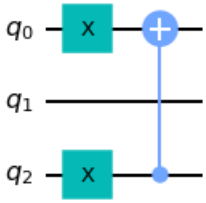


Figure 41: Modified three-qubit circuit after elimination of redundant gates

and modified circuits, respectively.

	Pre-Transpiled	**Level 1**	**Level 2**	**Level 3**	**Template**
Total Gates	11	11	11	7	11
CX Gates	5	5	5	3	5
X Gates	6	6	6	3	6
Synthesized U Gates	n/a	n/a	n/a	1	n/a
Depth	7	7	7	5	7

Table 1: Original three-qubit circuit costs post-transpilation

	Pre-Transpiled	**Level 1**	**Level 2**	**Level 3**	**Template**
Total Gates	11	7	3	3	3
CX Gates	5	1	1	1	1
X Gates	6	6	2	1	2
Synthesized U Gates	n/a	n/a	n/a	1	n/a
Depth	6	4	2	3	2

Table 2: Modified three-qubit circuit costs post-transpilation

As can be seen from the results, these reduction techniques do not output the

optimized circuit when applied to the original circuit. In fact, three of the four techniques tested resulted in no cost improvements to the original circuit. The preset pass manager with optimization level three did see cost reductions, but the most-reduced circuit resulting from the original circuit still had a higher cost than the most-reduced circuit resulting from the modified circuit.

In contrast to the original circuit, all of the optimization techniques reduced circuit cost when applied to the circuit modified by rearranging commuting layers. Even the preset pass manager with optimization level one, while outputting the maximal reduction, achieved better cost metrics than that of level three with the original circuit. Of particular importance is the reduction in CX count in these circuits, as CX gates result in more error than X gates. Another interesting note is that, even without transpilation, simply re-ordering the gates in the circuit led to a reduction in circuit depth. While this is not necessarily standard, it does show the importance of considering alternate ordering of gates within a circuit. Simple reductions may exist that could result in lower cost circuits and therefore more accurate results.

To see the difference in execution results, the original, modified, and reduced modified circuits were run on two IBM quantum computers: ibmq_vigo and ibmq_santiago [23]. The test was run on 24 November 2020.

On ibmq_vigo, the error rates ranged from 3.608×10^{-4} to $1.099e^{-2}$ (averaging $2.582e^{3}$) for single qubit gates and $5.964e^{-3}$ to 1.000 (averaging $2.556e^{-1}$) for CNOT gates. The 1.000 error rate is abnormal and corresponds to the unreliability of the physical qubit labeled as q_0 on the topology graph. A CNOT operation performed between q_0 and its neighbor q_1 could return up to a 1.000 error.

On ibmq_santaigo, the error rates ranged from $1.825e^{-4}$ to $2.831e^{-4}$ (averaging $2.282e^{-4}$) for single qubit gates and 6.094^{-3} to $7.506e^{-3}$ (averaging $6.631e^{-3}$) for CNOT gates.

The matrix corresponding to the entire circuit is shown in Figure 42. To calculate the expected results, this operator is applied to a three qubit system $|q_2 q_1 q_0\rangle$ in the state $|000\rangle$ to show that, in a perfect environment, the circuit would output the result $|100\rangle$.

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

Figure 42: Unitary corresponding to the three-qubit circuit

When the original, modified, and reduced modified circuits were run on the quantum computers, the impact of reducing the circuits with the commuting composition is apparent. The circuits were executed 1,024 times on both computers.

On the ibmq_vigo computer, the original circuit had a measurement probability of returning the right answer of 37.988%. The modified circuit had a measurement probability of returning the correct answer of 22.266%. The reduced circuit run on the same computer had a measurement probability of returning the correct answer of 72.754%. The histograms showing the full results are shown in Figures 43 to 45. These graphs show the percentage of total runs that each output value was returned.

When run on the ibmq_santiago computer, the measurement probabilities of returning the correct answer were 83.691% for the original circuit, 85.156% for the modified circuit, and 93.848% for the reduced circuit. The histograms showing the full results are displayed in Figures 46 to 48.

A summary comparing the percentage of runs for which the correct answer was

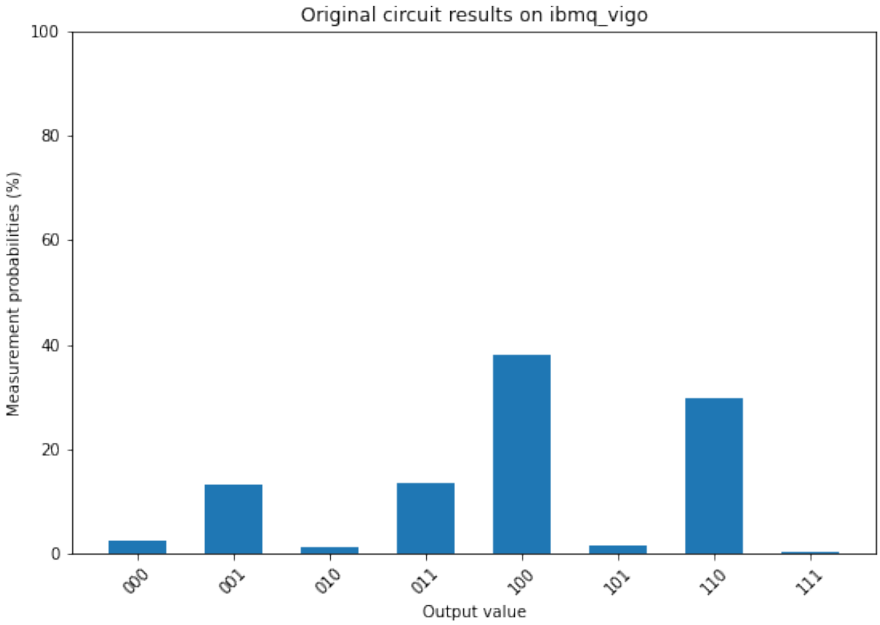


Figure 43: Three-qubit original circuit results - ibmq_vigo

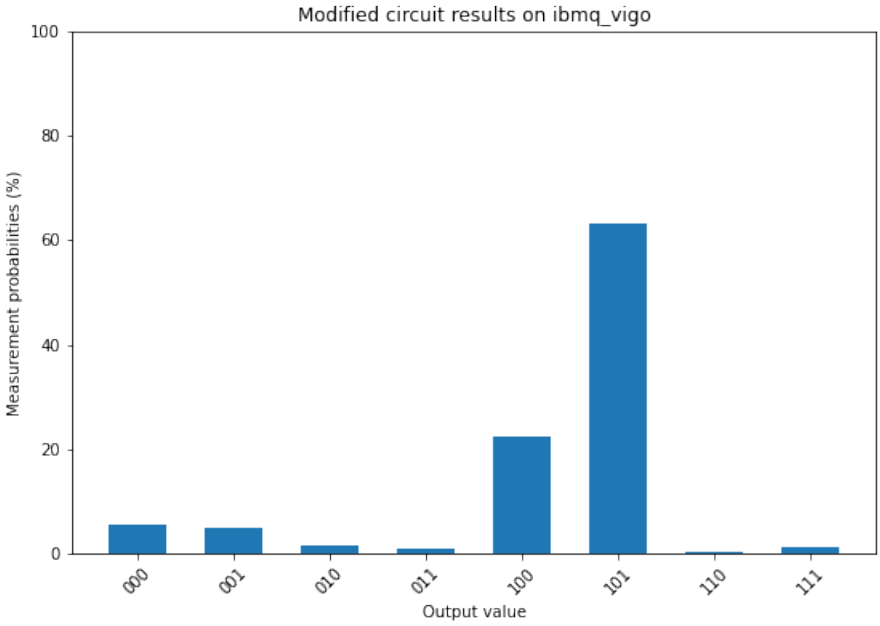


Figure 44: Three-qubit modified circuit results - ibmq_vigo

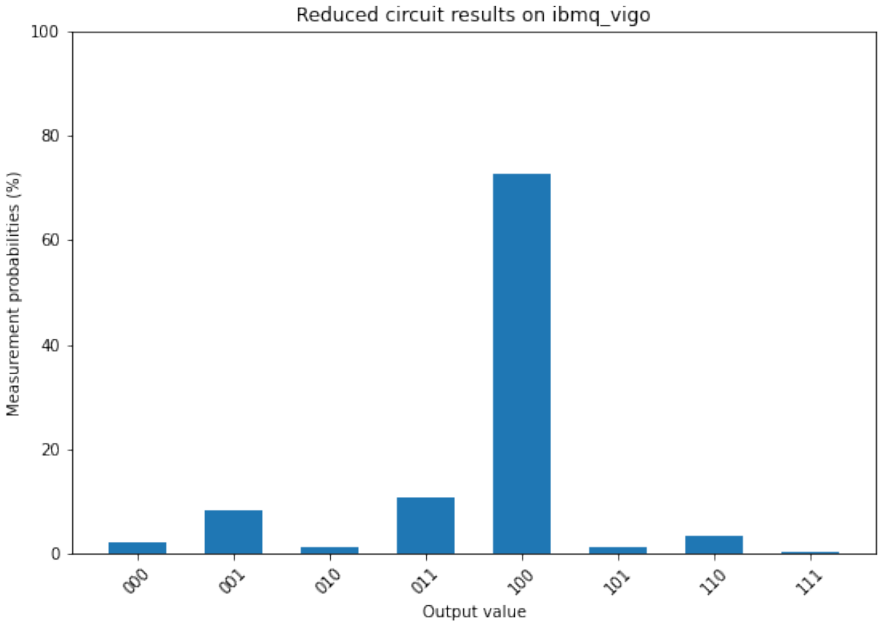


Figure 45: Three-qubit reduced circuit results - ibmq_vigo

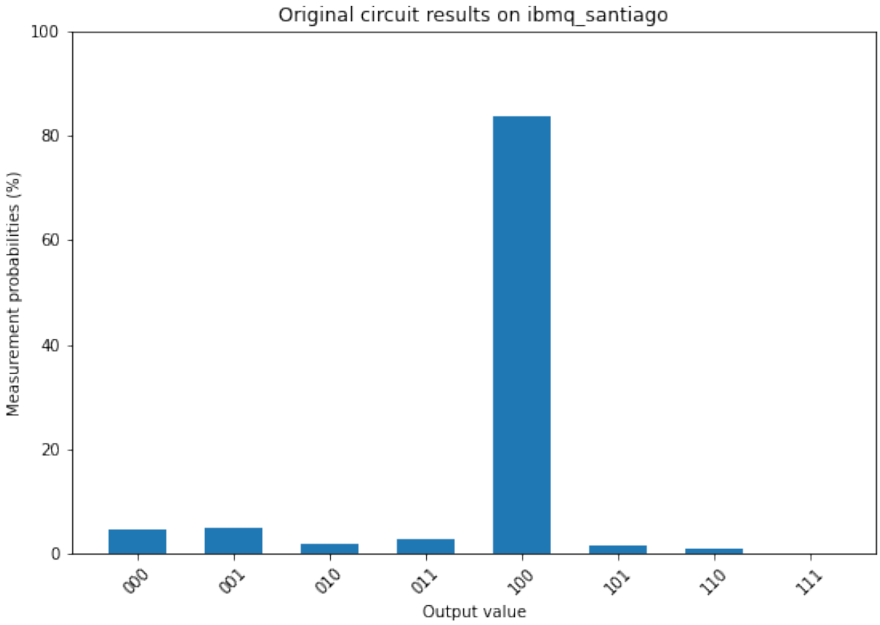


Figure 46: Three-qubit original circuit results - ibmq_santaigo

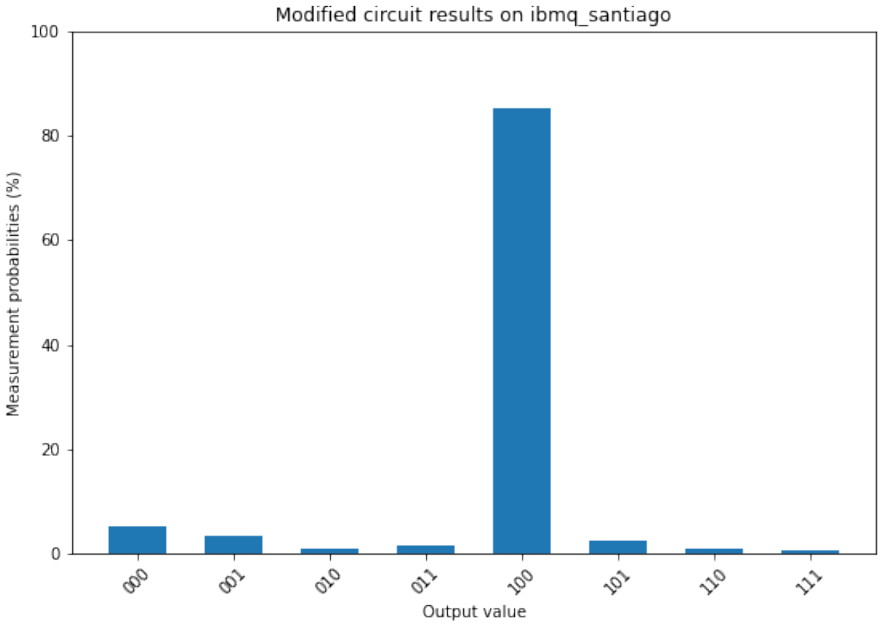


Figure 47: Three-qubit modified circuit results - ibmq_santaigo

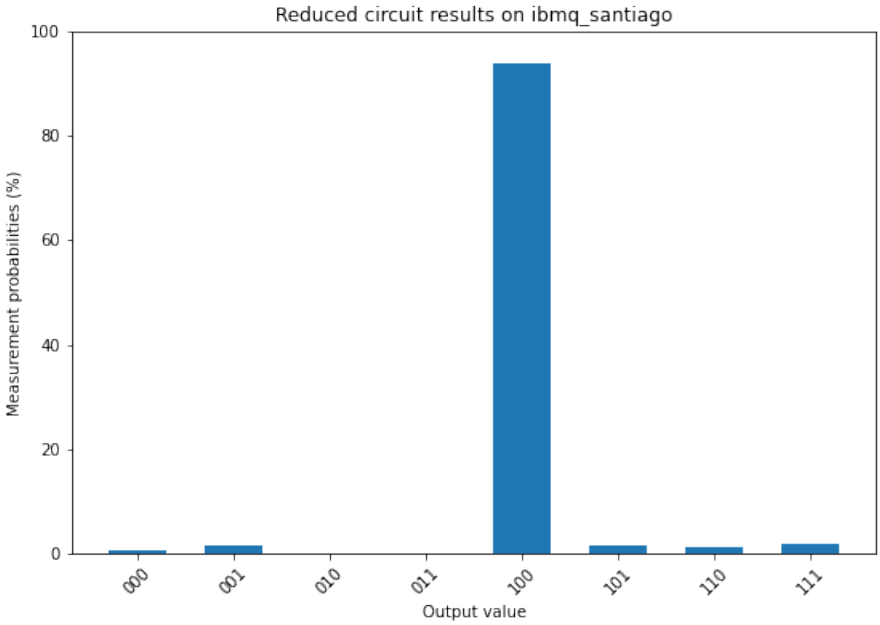


Figure 48: Three-qubit reduced circuit results - ibmq_santaigo

outputted for each test is shown in Figure 49. In both cases, the reduced circuit returned the correct result a higher percentage of the time. This shows that the reductions found after the gate rearrangement were able to create a circuit implementing the original function that returns better results when run on quantum hardware.

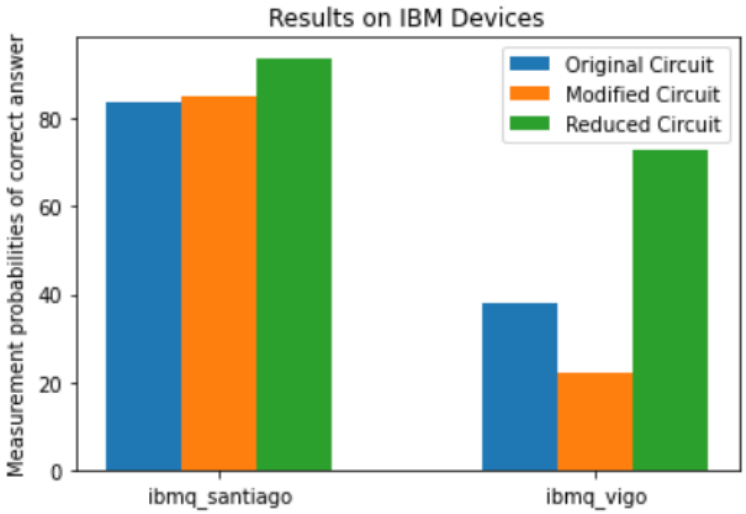
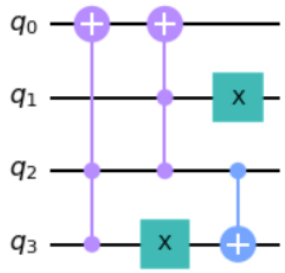


Figure 49: Measurement results on ibmq_vigo and ibmq_santiago

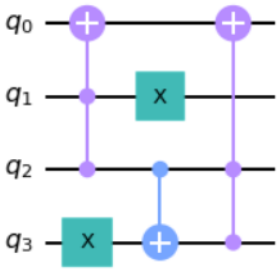
4.2.3.2 Four Qubit Layers

Of the 970,299 four-qubit circuits, 13,536 are three-layer NCT commuting compositions. An example of one of the circuits and its commutations are shown in Figure 50. As with the three qubit example, the first layer can commute with the composition of the second and third layers, or the third layer can commute with the composition of the first and second layers.

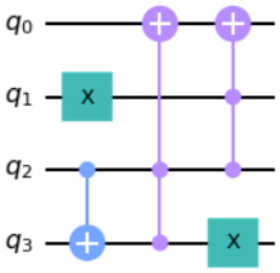
To demonstrate cost reduction impact from the application of this commuting composition, the circuit shown in Figure 51 is reduced with and without the rearranging gates within the three layer commutation. Again, as in Section 4.2.3.1, this circuit was designed to prove the existence of potential cost reductions via commuting compositions. It was not taken from a benchmark or algorithm known to the author.



(a) Four qubit three-layer NCT commuting composition



(b) First layer commuted with compositions of second and third layers



(c) Third layer commuted with composition of first and second layers

Figure 50: Equivalent four-qubit circuits

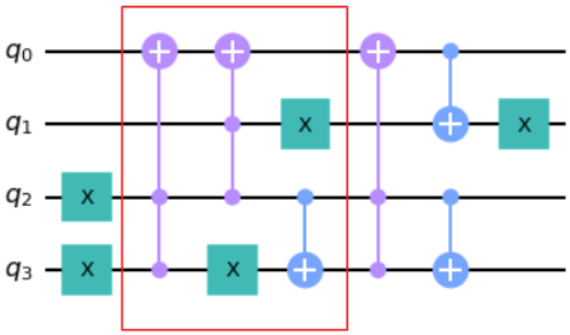


Figure 51: Four-qubit circuit containing three-layer commuting composition subcircuit

The modified circuit after the match is replaced with one of its commuted versions is shown in Figure 52.

Both the original and modified circuits were run through the preset pass managers for optimization levels one, two, and three, as well as the template optimization pass. The results from those transpilation passes are shown in Tables 3 and 4.

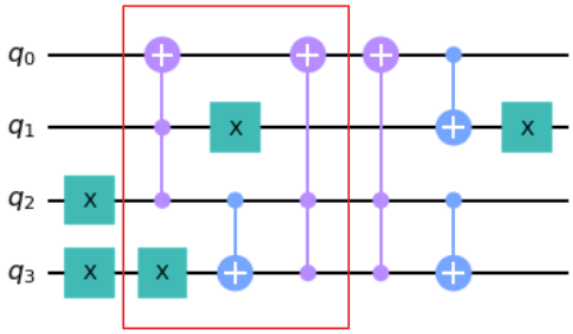


Figure 52: Original circuit replaced with commuted subcircuit

	Pre-Transpiled	**Level 1**	**Level 2**	**Level 3**	**Template**
Total Gates	11	11	9	43*	9
CCX Gates	3	3	3	n/a	3
CX Gates	3	3	3	18	3
X Gates	5	5	3	3	3
T gates	n/a	n/a	n/a	11	n/a
$T^\dagger$ gates	n/a	n/a	n/a	8	n/a
H gates	n/a	n/a	n/a	2	n/a
Synthesized u gates	n/a	n/a	n/a	1	n/a
Depth	7	7	6	32	7

*Clifford+T (hardware level) decomposition

Table 3: Original four-qubit circuit costs post-transpilation

	Pre-Transpiled	**Level 1**	**Level 2**	**Level 3**	**Template**
Total Gates	11	11	7	41*	5
CCX Gates	3	3	3	n/a	1
CX Gates	3	3	3	18	3
X Gates	5	5	1	1	1
T gates	n/a	n/a	n/a	11	n/a
$T^\dagger$ gates	n/a	n/a	n/a	8	n/a
H gates	n/a	n/a	n/a	2	n/a
Synthesized u gates	n/a	n/a	n/a	1	n/a
Depth	7	7	6	32	4

*Clifford+T (hardware level) decomposition

Table 4: Modified four-qubit circuit costs post-transpilation

Unlike the three-qubit case, the output of the optimization level three transpilation results includes T, $T^{\dagger}$, and H gates. This is because 3,792 of the four qubit matches contain one or two CCX gates, and the CCX gate must be decomposed to the quantum level Clifford+T gates prior to running on IBMQX hardware. The Clifford+T decomposition of the CCX gate is shown in Figure 19 in Section 3.3.1.

Another difference between the four-qubit and the three-qubit case is that the preset pass managers have worse results than the template optimization pass when run on the modified circuit. Again, this difference is due to the existence of CCX gates. The preset pass managers do not cancel adjacent CCX gates. CommutativeCancellation—the optimization pass used in the preset pass managers—only considers H, X, Y, Z, CX, CY, and CZ gates [25]. In contrast, the default template optimization pass does cancel redundant CCX gates because it is initialized with adjacent, X, CX, and CCX gates for the identity templates.

The circuit with the least cost is the modified circuit after the template optimization pass. This circuit is shown in Figure 53 and is used as the reduced modified circuit for the hardware test.

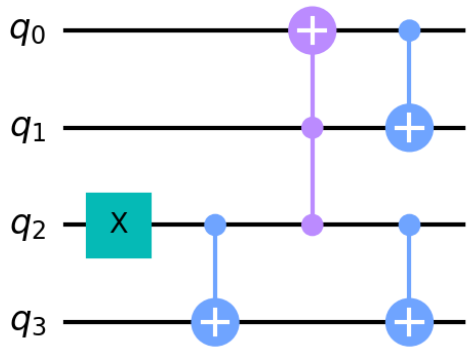


Figure 53: Modified four-qubit circuit after undergoing template optimization pass

To evaluate the circuit reduction impact on computational performance, the origi-

nal, modified, and reduced modified circuits were run on ibmq_vigo and ibmq_santiago. The correct result of this circuit operating on four qubits $|q_3q_2q_1q_0\rangle$ in the $|0000\rangle$ state is $|0100\rangle$ with 100% probability. The two circuits were run on each device 1,024 times. The test was performed on 8 Dec 20, on which day the error rates averaged $4.986e^{-4}$ for single qubit and $8.520e^{-3}$ for two qubit gates on ibmq_vigo, and $2.798e^{-4}$ for single qubit and $1.009e^{-2}$ for two qubit gates on ibmq_santiago.

The histograms showing the detailed results for ibmq_vigo are shown in Figures 54 to 56.

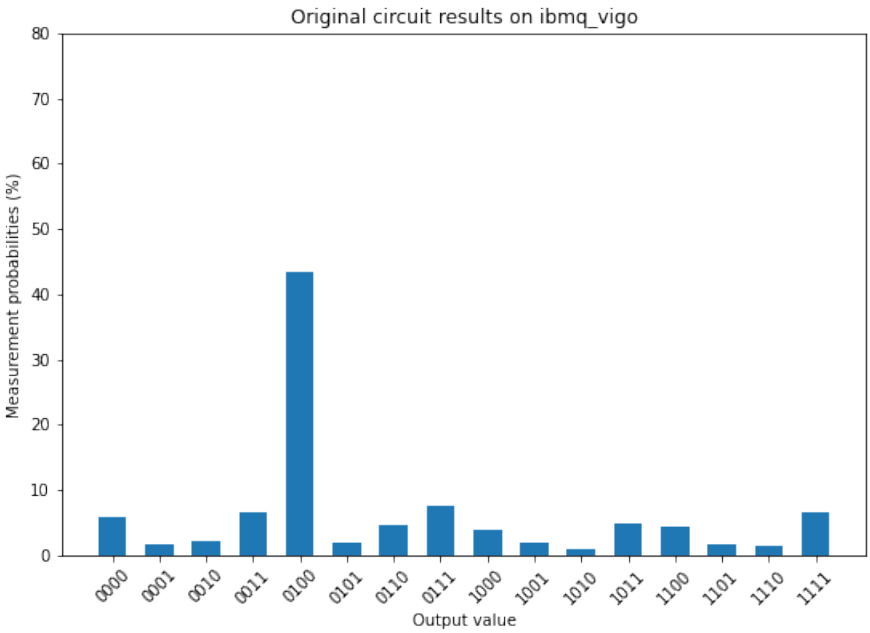


Figure 54: Four-qubit original circuit results - ibmq_vigo

The histograms showing the detailed results for ibmq_santiago are shown in Figures 57 to 59.

Just as in the three qubit case, the reduced circuit yielded the best results on both ibmq_vigo and ibmq_santiago. On ibmq_vigo, the reduced circuit produced the correct result 68.262% of the time, as compared to 43.457% and 39.844% for the original and modified circuits, respectively. On ibmq_santiago, the reduced circuit

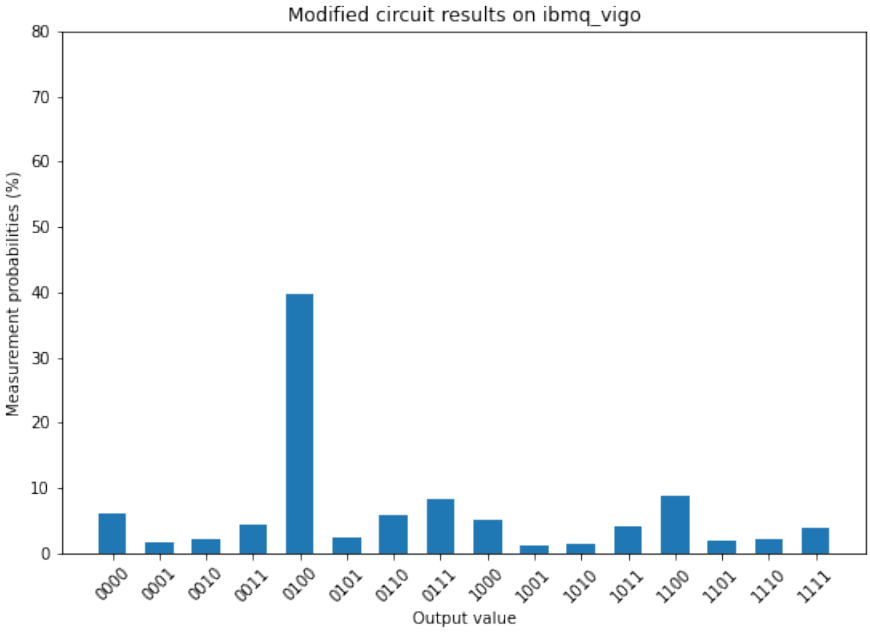


Figure 55: Four-qubit modified circuit results - ibmq_vigo

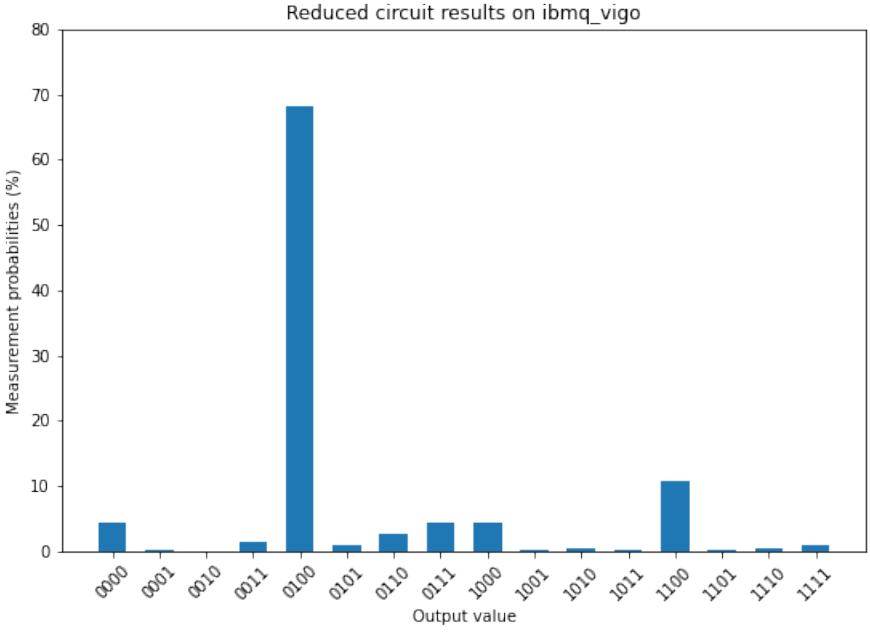


Figure 56: Four-qubit reduced circuit results - ibmq_vigo

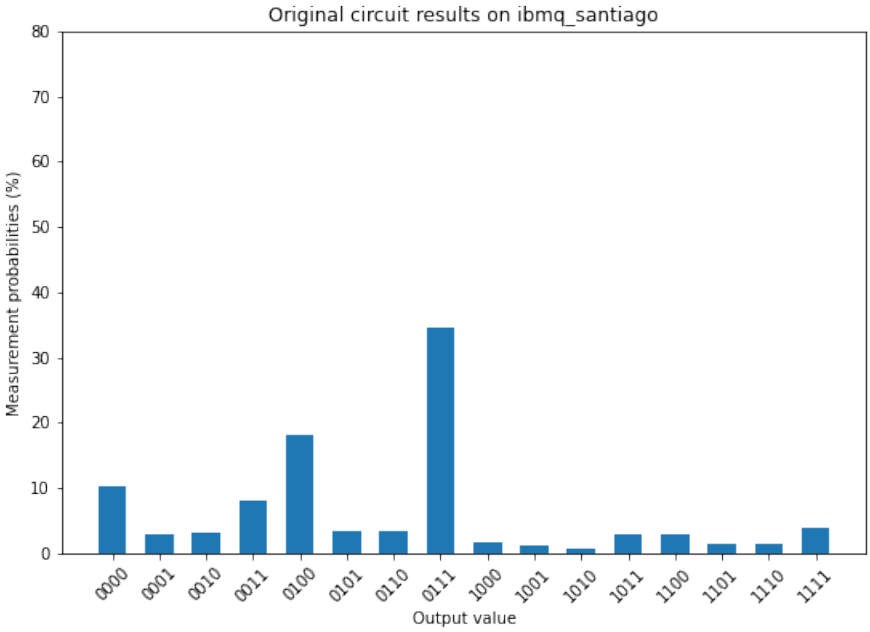


Figure 57: Four-qubit original circuit results - ibmq_santaigo

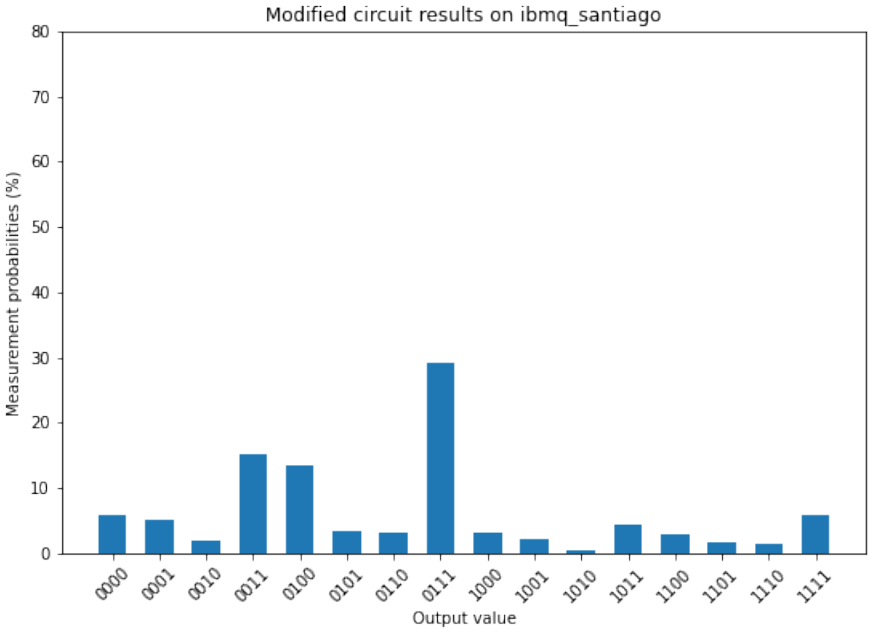


Figure 58: Four-qubit modified circuit results - ibmq_santaigo

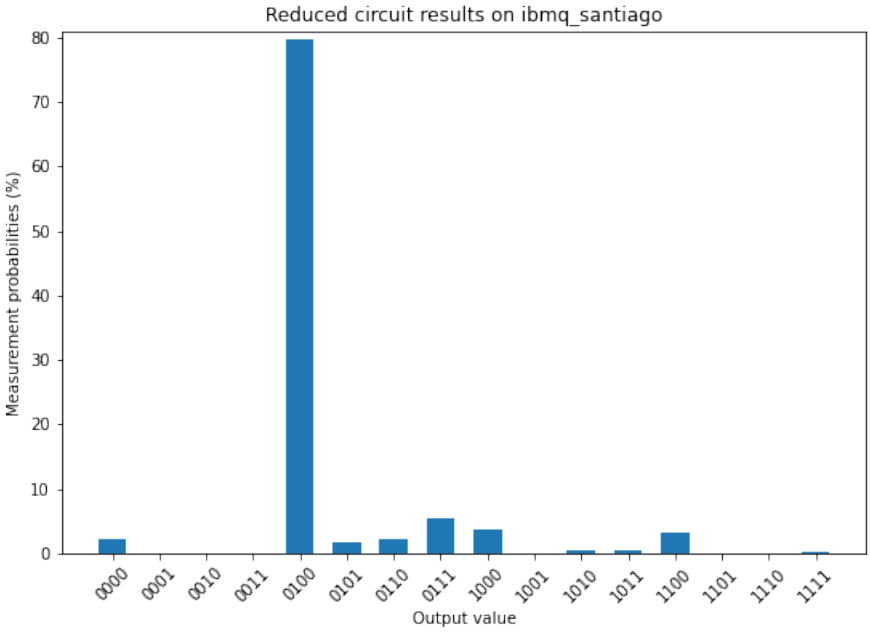


Figure 59: Four-qubit reduced circuit results - ibmq_santaigo

output the correct results 79.688% of the time, compared to 18.262% and 13.574% for the original and modified circuits, respectively. The summary of correct results for both devices is shown in Figure 60.

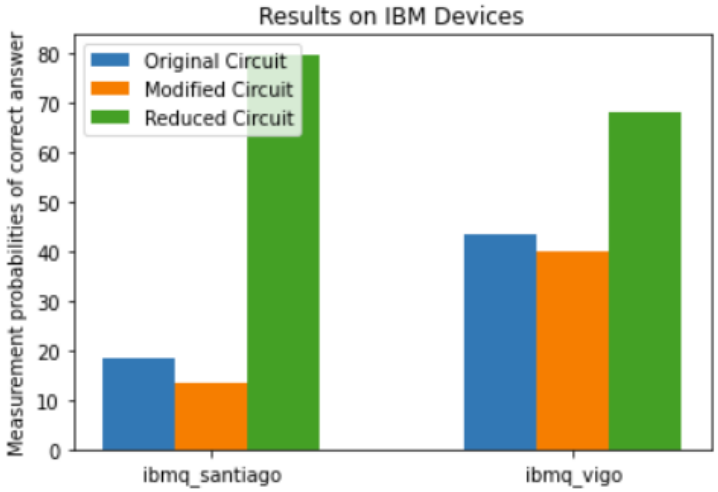
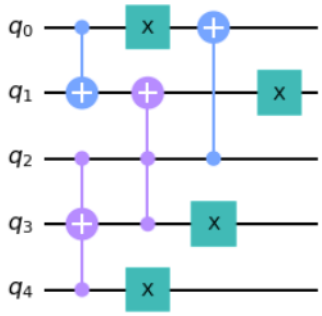


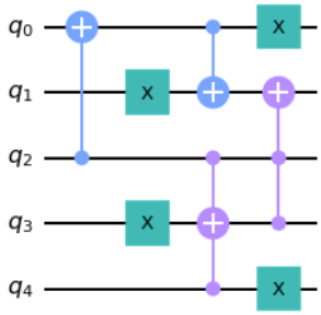
Figure 60: Four-qubit circuit measurement results

4.2.3.3 Five Qubit Layers

Of the 118,370,771 five-qubit circuits, 1,518,480 are three-layer commuting compositions. One of the matches is shown in Figure 61a. This sequence is different than the sequences chosen as examples for the three and four qubit cases in that the first layer commutes with the composition of the second and third layers, but the third layer does not commute with the composition of the first and second layers. The allowed commutation is shown in Figure 61b.



(a) Five qubit three-layer NCT commuting composition



(b) Third layer commuted with the composition of the first and second layers

Figure 61: Equivalent five-qubit circuits

As with the three and four qubit cases, to prove a cost reduction is possible using existing techniques when accounting for the commuting composition but not without the commutation, a circuit containing the match was created. This circuit is shown in Figure 62 (with the three layer commuting composition boxed in red). The circuit

modified by rearranging the gates according to the commuted sequence is shown in Figure 63.

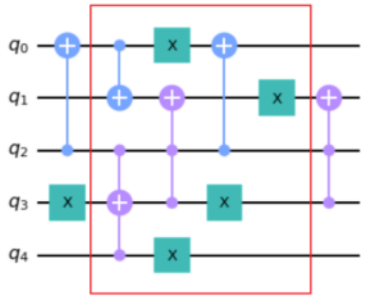


Figure 62: Five qubit circuit containing three layer commuting composition subcircuit

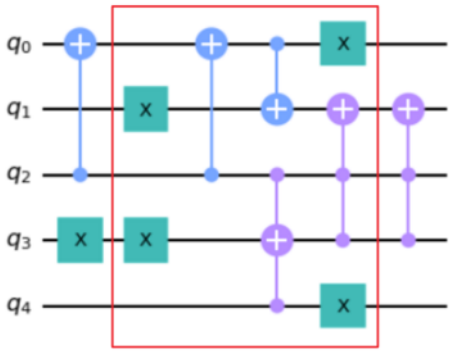


Figure 63: Original five-qubit circuit replaced with commuted subcircuit

The original and modified circuits were run through the Qiskit preset pass managers with optimization levels one, two, and three, and the template optimization transpilation pass. The results from the transpilation passes are shown in Tables 5 and 6. As seen in the tables, the original circuit did not have any reductions resulting from a transpilation pass. The modified circuit had reductions with every pass. The greatest cost reduction once again occurred with the template optimization pass on the modified circuit. This reduced circuit is shown in Figure 64.

The Clifford+T decomposition of the original and modified circuits resulting from the level three transpilation pass had no difference in the number of T and $T^\dagger$ gates.

	Pre-Transpiled	**Level 1**	**Level 2**	**Level 3**	**Template**
Total Gates	11	11	11	51*	11
CCX Gates	3	3	3	n/a	3
CX Gates	3	3	3	21	3
X Gates	5	5	5	5	5
T gates	n/a	n/a	n/a	9	n/a
$T^\dagger$ gates	n/a	n/a	n/a	9	n/a
H gates	n/a	n/a	n/a	6	n/a
Synthesized u gates	n/a	n/a	n/a	1	n/a
Depth	5	5	5	32	5

*Clifford+T (hardware level) decomposition

Table 5: Original five-qubit circuit costs post-transpilation

	Pre-Transpiled	**Level 1**	**Level 2**	**Level 3**	**Template**
Total Gates	11	9	7	46*	5
CCX Gates	3	3	3	n/a	1
CX Gates	3	1	1	19	1
X Gates	5	5	3	3	3
T gates	n/a	n/a	n/a	9	n/a
$T^\dagger$ gates	n/a	n/a	n/a	9	n/a
H gates	n/a	n/a	n/a	4	n/a
Synthesized u gates	n/a	n/a	n/a	2	n/a
Depth	5	5	4	30	3

*Clifford+T (hardware level) decomposition

Table 6: Modified five-qubit circuit costs post-transpilation

As discussed in Section 2.3, the T-count is a heavily considered metric when evaluating quantum circuits because of the high cost to implement T gates. While the T-count was not reduced by the optimization level three pass on the original and modified circuits, it was reduced when that pass was applied to the reduced circuit (circuit outputted from template optimization on the modified circuit). Table 7 shows the comparison of gate counts for the original, modified, and reduced circuits run through the level three transpilation pass.

This demonstrates an example of where optimizing circuits at a higher level of

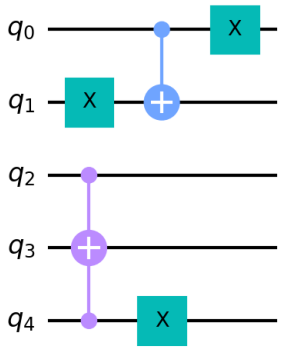


Figure 64: Modified five-qubit circuit after template optimization pass

	Original	Modified, not previously reduced	Modified and reduced by template optimization
Total Gates	51	46	19
CX Gates	21	19	7
X Gates	5	3	3
T gates	9	9	4
$T^\dagger$ gates	9	9	4
H gates	6	4	2
Synthesized u gates	1	2	0
Depth	32	30	12

Table 7: Level three optimization results for five-qubit circuit

abstraction prior to decomposing and optimizing at the hardware-level gates could yield greater reductions than optimizing only at the low level.

To see how the circuits compare when run on real hardware, the original, modified, and reduced circuits were again run on IBMQX machines. The circuit chosen for the reduced circuit is the result of the template optimization pass on the modified circuit, and is shown in Figure 64. All three circuits were run on ibmq_vigo and ibmq_santiago 1,024 times on 14 December 2020. On this day, the ibmq_vigo error rates averaged $4.278e^{-4}$ for single qubit gates and $7.897e^{-3}$ for CNOT gates, while ibmq_santiago error rates averaged $4.296e^{-4}$ for single qubits and $1.060e^{-2}$ for CNOT gates. The

expected result of the circuit operator acting on a quantum system $|q_4q_3q_2q_1q_0\rangle$ in state $|00000\rangle$ is $|10011\rangle$ with a 100% probability.

The histograms showing the detailed results for ibmq_vigo are shown in Figures 65 to 67. The histograms showing the detailed results for ibmq_santiago are shown

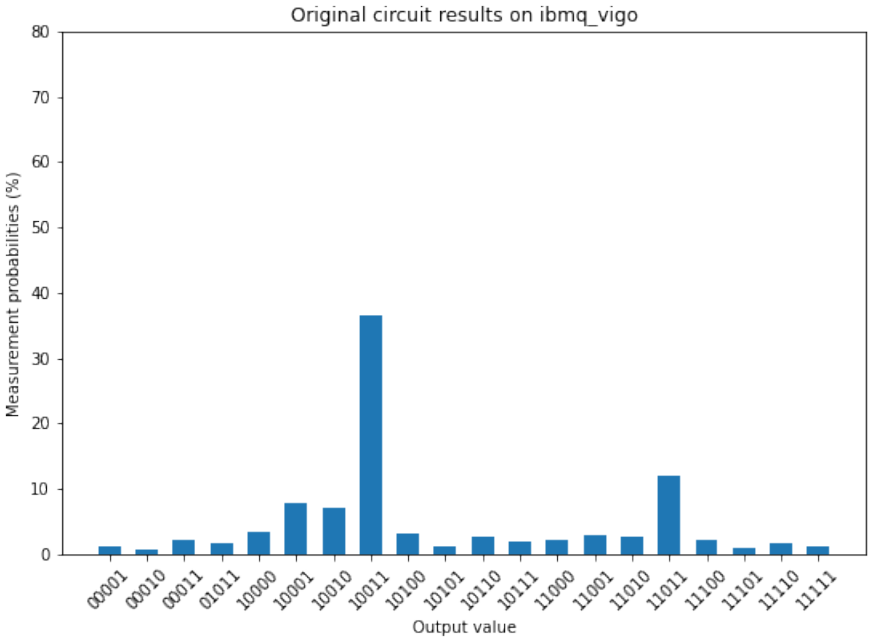


Figure 65: Five-qubit original circuit results - ibmq_vigo

in Figures 68 to 70.

Figure 71 shows the percentage of times that the correct result was returned from the circuit execution. As can be seen from the table, higher device error rates lead to a greater difference in the percentage of correct results.

4.2.4 Note on Computational Time to Find Matches

The computational time required to find three-layer commuting compositions increases with the number of qubits. The asymptotic time complexity of this problem is $O(n!)$. The analysis of this is shown in Appendix A. The factorial complexity is due to the increasing number of possible layers per qubit count.

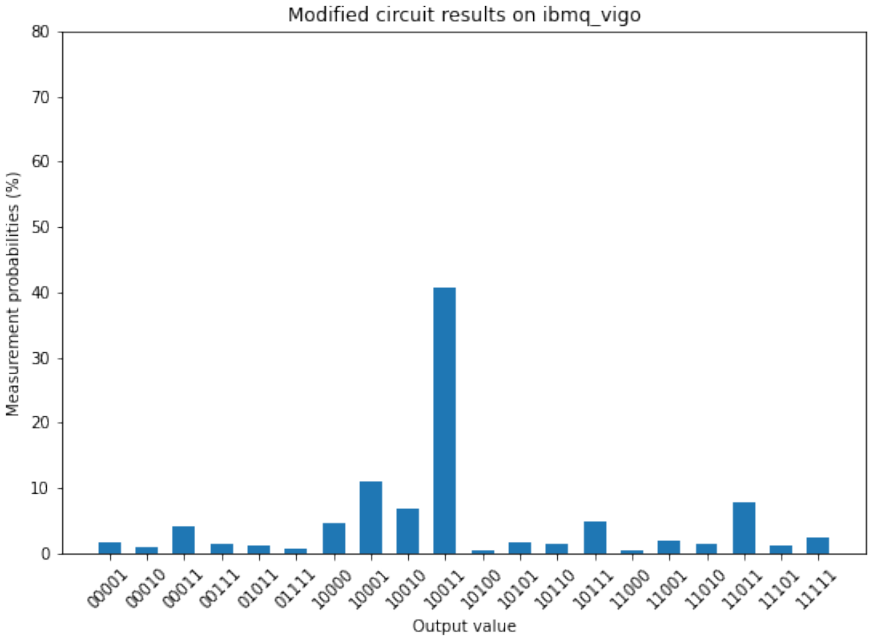


Figure 66: Five-qubit modified circuit results - ibmq_vigo

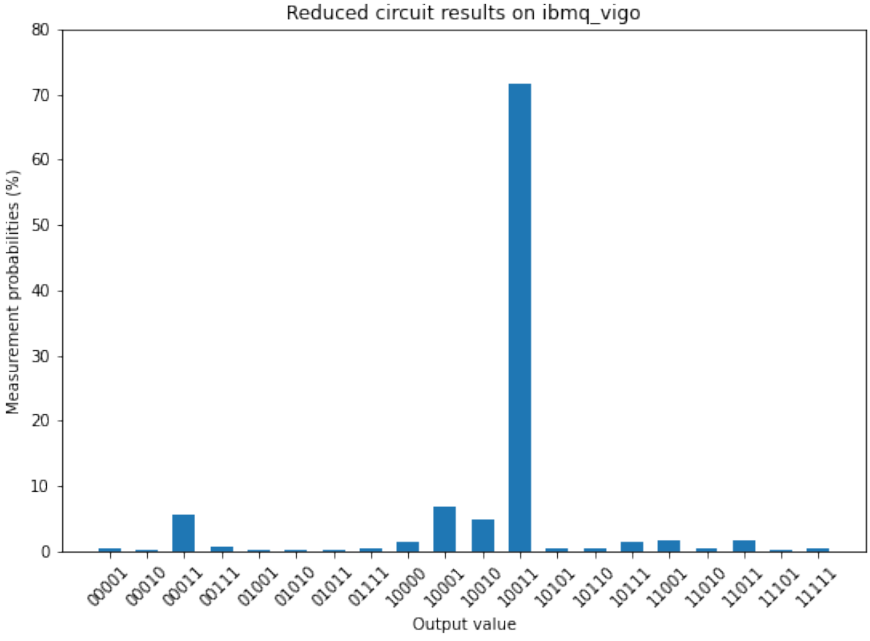


Figure 67: Five-qubit reduced circuit results - ibmq_vigo

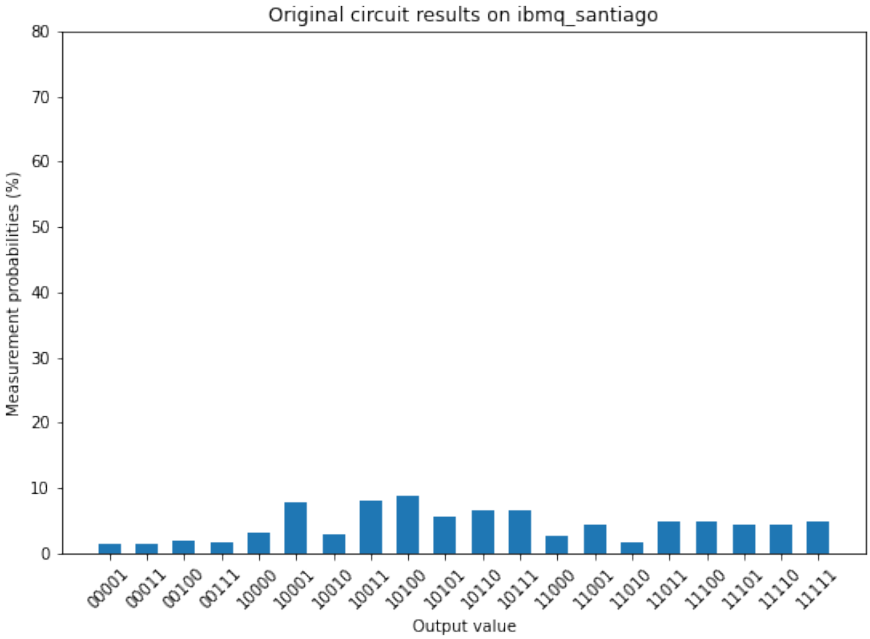


Figure 68: Five-qubit original circuit results - ibmq_santaigo

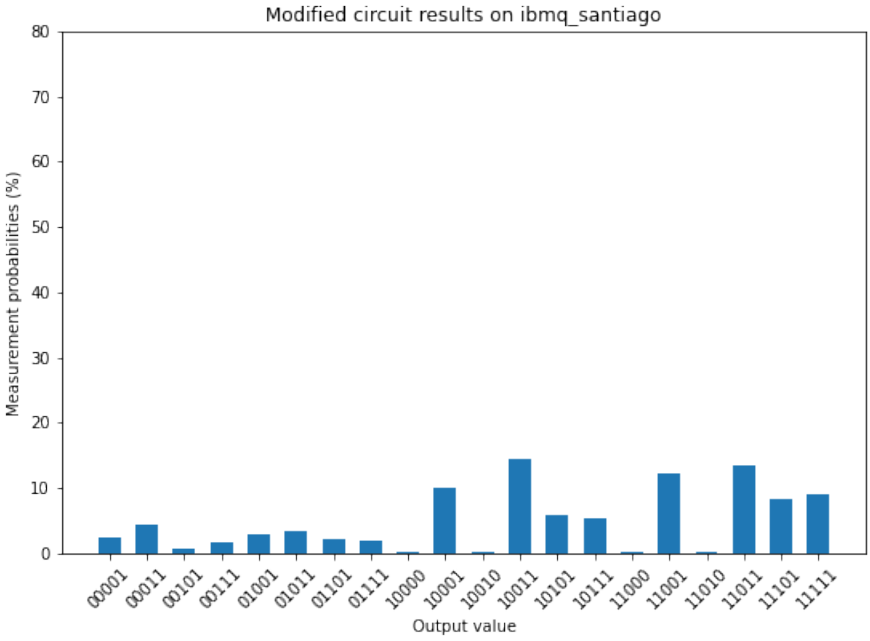


Figure 69: Five-qubit modified circuit results - ibmq_santaigo

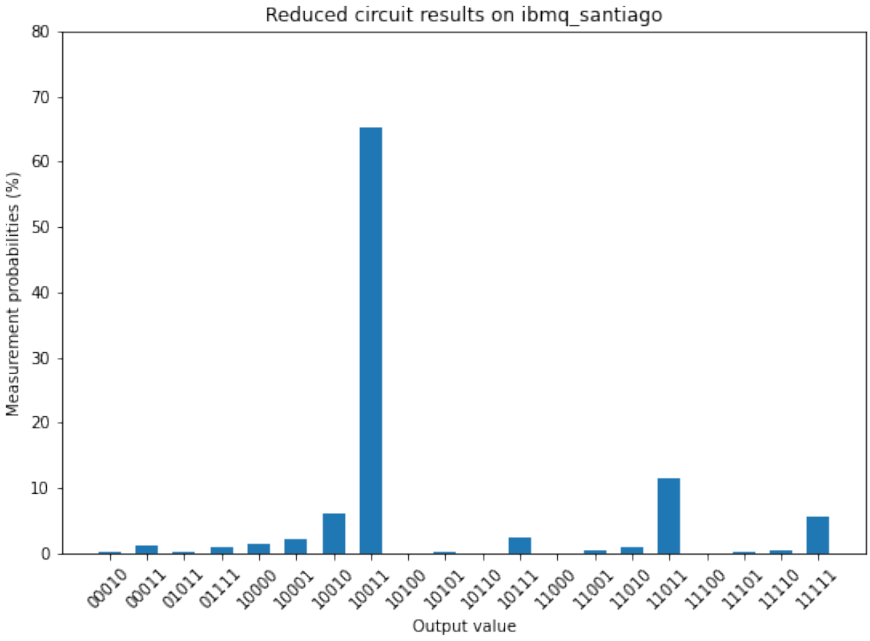


Figure 70: Five-qubit reduced circuit results - ibmq_santaigo

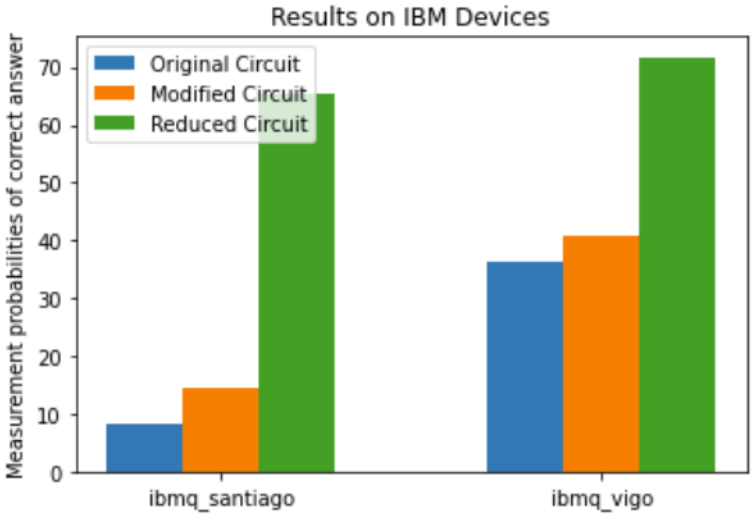


Figure 71: Five-qubit circuit measurement results on ibmq_vigo and ibmq_santiago

In addition to the growing number of sequences to evaluate, the size of the matrices corresponding to each layer also grows. For a circuit with n qubits, the matrix will be 2^nx2^n. Therefore, the check to find sequences matching the three-layer commuting

composition property requires the resources to compute matrix multiplication for exponentially growing matrices.

The python code for the algorithm to check for three-layer commuting compositions was run on an Intel Xeon 2.30GHz processor with 256 GB RAM running the Windows 10 operating system. In wall clock time, it took 1 minute and 17 seconds to check all 10,648 three-qubit three-layer sequences for commuting compositions, 1 hour, 4 min and 44 seconds to check all 970,299 four-qubit sequences, and 5 days, 6 hours, 49 minutes and 48 seconds to check all 118,370,771 five-qubit sequences. The other workload on the machine was minimized so as not to detract resources from the checkMatch algorithm.

4.3 Sequence Analysis

4.3.1 Existence of Three-Layer Commuting Compositions

As seen in Section 4.2.3, there do exist three layer sequences of NCT gates with the property that no adjacent layers commute, but a single layer commutes with a pair of layers. Therefore, there can exist alternative gate orderings within a larger circuit that are not considered with current techniques that consider only pairwise commutation.

While such three-layer commuting circuits do exist, the amount of them is small in proportion to the number of all possible sequences. For the three qubit case, 0.676% of all three-layer combinations meet the three-layer commuting composition property. The percentage is slightly higher but still small for the four and five qubit case: 1.395% and 1.28% respectively.

With only three cases, a trend cannot be definitively identified as to whether there is a correlation between number of qubits in the circuit and percentage of all possible circuits that are three-layer commuting compositions. Increasing the number

of qubits from three to four raised the proportion by .719%, which initially seemed to indicate more qubits would result in a higher percentage of three-layer commuting compositions. However, the five qubit case had a lower proportion of such sequences, proving that hypothesis to be false. Circuits with more qubits could be checked; however, commuting compositions with higher numbers of qubits would become more specific and less generic. The more specific the sequence, the less likely it will be found in a larger circuit. Therefore, there may not be a practical benefit to finding sequences of gate combinations acting on many qubits.

4.3.2 Elimination of Redundant Gates within Sub-Circuits

After finding the three-layer commuting compositions for three, four, and five qubits, they were processed to determine how many could be simplified via elimination of adjacent gates. To do this, they were run through the template optimization transpiler pass with Qiskit to find instances of identical adjacent gates and replace with the identity. Next, they were run through the checkMatch algorithm which maintained the three-layer commuting composition property.

This is to determine how many three-layer NCT commuting compositions exist such that there are no trivial reductions within the sub-circuit itself. Such subcircuits have a higher likelihood of existing within already reduced circuits and resulting in additional cost reductions.

The results of this test are shown in Table 8. They show that the number of non-reducible commuting compositions is much fewer than the total number of commuting compositions. However, despite the relatively low numbers, the existence of such sequences shows promise for the possibility of pre-reduced circuits to have further reductions via the use of commuting compositions.

Two of the three qubit reduced sequences are shown in Figure 72. The twelve

	Possible Sequences	Matching Sequences	Non-reducible Sequences
Three Qubits	10 648	72	12
Four Qubits	970 299	13 536	3888
Five Qubits	118 370 771	1 518 480	474 420

Table 8: Non-reducible three-layer NCT commuting compositions

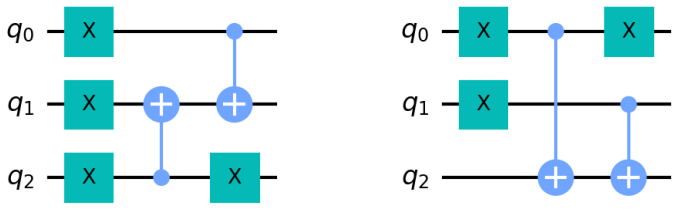
total possibilities for three qubit circuits are permutations of these two sequences.

4.3.3 Single Gate Layer Sequences

As described in Section 3.5.4, all of the identified matches were analyzed to see if they contain a single gate per layer by checking if any circuits contain exactly three gates. None of the identified three-layer NCT commuting compositions had that property; all had five or more gates. Therefore, three-gate NCT commuting compositions with three to five qubits do not exist.

To determine whether three-gate NCT commuting compositions exist at all, one, two, six and seven qubit circuits were evaluated according to the experiment described in the latter part of Section 3.5.4. Circuits of three gates were created. They were checked for all qubits to be in use and that sequential gates operate on at least one similar qubit.

The one qubit case has only the circuit of three X gates that meets these criteria.



(a) (b)

Figure 72: Three-qubit three-layer NCT commuting compositions in reduced form

The two qubit case has 64 circuits that meet these criteria: each layer contains a single X gate or a CNOT gate.

The six qubit case has 84,240 circuits to check. There are 729,000 three gate circuits in total. After eliminating the circuits that do not use all six lines, there are 171,900 circuits remaining. Finally, after eliminating circuits with adjacent gates operating on disjoint sets of qubits, there are 84,240 circuits remaining to check if they are three-gate commuting compositions.

The seven qubit case has 51,030 circuits to check. In total, there are 1,157,625 three-gate NCT circuits with seven qubits. Of those, 153,090 utilize all seven qubits. That number is further reduced to 51,030 after checking that adjacent gates operate on at least one shared qubit.

All of the circuits meeting the requirements were then fed through the CheckMatch algorithm to see if any met the conditions to be a three-gate commuting composition. None did. As circuits with greater than seven qubits containing three NCT gates will contain pairwise commuting gates, this final check of one, two, six and seven qubit circuits proves that three-layer NCT commuting compositions do not exist.

4.3.4 Number and Types of Gates in Sequences

As described in Section 3.5.5 the three-layer NCT commuting compositions were analyzed for total number of gates and how many of each type of operation. The purpose of this is to have a better understanding of the makeup of the identified matches. Additionally, it could inform which commuting compositions to search for when evaluating a given circuit.

The results of the number and types of operations in the three, four, and five qubit circuits are shown in Tables 9 through 11, respectively.

	Maximum	Minimum	Average
Total Gates	6	5	5.500
CCX Gates	0	0	0
CX Gates	2	2	2
X Gates	4	3	3.500

Table 9: Number and types of gates in three-qubit circuits

	Maximum	Minimum	Average
Total Gates	9	5	6.390
CCX Gates	2	0	1.350
CX Gates	5	0	2.480
X Gates	7	0	3.550

Table 10: Number and types of gates in four-qubit circuits

	Maximum	Minimum	Average
Total Gates	12	5	7.170
CCX Gates	3	0	0.950
CX Gates	6	0	2.860
X Gates	10	0	3.360

Table 11: Number and types of gates in five-qubit circuits

4.4 Summary

This chapter presents the results from the experiments run in this research. The main contribution is the identification of three-layer NCT commuting compositions. In support of that, this chapter also describes the results of the steps taken to generate all possible sequences for three, four, and five qubit circuits. Furthermore, it presents analysis of the found circuits to inform decisions on when to search for the commuting compositions, how they can help reduce a circuit, and properties they contain.

V. Conclusions

This chapter concludes this research by describing its contribution to the field of quantum computing and offering areas of future work. Section 5.1 overviews how this research fits into the broader problem of logical quantum circuit reduction. Section 5.2 presents the contributions of this work to the field. Section 5.3 identifies areas of future work. Finally, Section 5.4 offers concluding remarks.

5.1 Overview

Quantum circuit optimization is a broad field encompassing many focus areas and techniques. One of the techniques is circuit optimization via templates. One of the computational problems underlying this method is that of finding sequences of gates that match templates within a circuit. The difficulty of this problem lies in the fact that a template may not match a sequence of gates in the original specification of the circuit, but may match if the gates in the circuit are rearranged with allowable commutations. This research analyzes such commutations to identify alternative gate sequences that could realize a circuit without changing its functionality.

The broad research question presented in Section 1.4 of how commuting compositions of layers within a quantum circuit can be used to make circuit transpilation more effective drives this work. Results of computational experiments testing three hypotheses are reported to inform that question. Those hypotheses are:

1. *Three-element commuting compositions for circuits composed of NOT, CNOT and Toffoli gates exist.*

    - This hypothesis is **true for three-layer** NCT commuting compositions (See Section 4.2.3).

- This hypothesis is **false for three-gate** NCT commuting compositions (See Section 4.3.3.)

2. *Rearrangements of quantum gates in a circuit can yield reductions not otherwise captured by state of the art optimization tools.*

    - This is **true** using Qiskit's preset optimization passes and template matching pass (See Sections 4.2.3.1 through 4.2.3.3). Qiskit is actively managed by experts across IBM and the quantum computing community. The template optimization pass is the implementation of the 2020 algorithm by Iten et al., which claims to "further improve practically relevant quantum circuits that were already optimized with state-of-the-art techniques" [28]. While not the only optimization tools existent, these are solid representations of the simplifications current research is capable of coducting.

3. *Consideration of commuting layers will yield more three-element commuting compositions than accounting for gates alone.*

    - This is **true for the NCT gate set** — considering gates alone resulted in no three-element commuting compositions while consideration of layers yielded 72, 13,536, and 1,518,480 three-element commuting compositions for three, four, and five qubit layers, respectively (Sections 4.2.3 and 4.3.3).

These results offer insight into the usefulness of application of commuting compositions in circuit reduction. In addition to strictly determining whether the tested hypotheses are true or false, this research identifies characteristics of the commuting compositions that may be helpful in future work.

5.2 Contributions

5.2.1 Presentation of the Commuting Composition Problem

The first contribution this research makes to quantum computing advances the theory of the field by providing the definition of the commuting composition property. This generalizes previous work on circuit commutations to account for layers instead of single gates and compositions of layers rather than each considered individually. The formalism of this property paves the way for an entirely new area of research within logical circuit reduction. Work in this area could yield new methods to analyze a circuit for alternative gate orderings that compute to the same operator matrix. This allows for the possibility of finding sequences of gates that could be reduced, thereby lowering the overall cost of the circuit.

5.2.2 Investigation of Three-Layer NCT Commuting Compositions

The second contribution of this research advances the state of the quantum computing research infrastructure through the identification, presentation, and analysis of commuting compositions composed of three layers of NCT gates operating on three, four, and five qubits. The examples identified are ready to be incorporated into transpilation software such as Qiskit, which will both facilitate further research and accelerate the arrival of practical quantum computing. This sub-area within the broader commuting composition problem proves the existence of commutations composed of more than two single-gate layers and that they can result in greater reductions to circuit cost. It serves as an initial work investigating alternative commutations of gates in a circuit.

This answers whether, within a NCT circuit, there exist sequences of three layers such that no adjacent layers commute but a single layer commutes with the remaining pair of layers. It turns out that such sequences do exist, and that they can result

in reductions to a quantum circuit that are not captured in current state of the art optimization techniques.

In addition to establishing the existence of such circuits, this research performs an exhaustive search of all possible three-layer NCT circuits for three, four, and five qubits to find which circuits contain this property. All of the identified circuits are recorded as OpenQASM strings so that they can be used in future work.

Finally, this work analyzes the results to identify which of the three-layer commuting compositions are optimized, if there are any with a single gate per layer, and the number and type of gate operations in each. These metrics offer insight to the situations in which the application of the identified commuting compositions may result in an overall reduction. Optimized sequences will be useful in circuits that have gone through initial simplification passes. In such cases, neighboring same gates will have already been eliminated and the commuting compositions containing redundant gates will not exist in the circuit. Therefore, it would waste resources to search for them. Single gate layers may be easier to find within a larger circuit. Finally, the number and type of operations in each commuting composition gives a general understanding of the types or reductions possible. Sequences that contain more of a specific gate are more likely to result in a reduction in the overall count of that gate in the larger circuit. While not an exact indicator of what transformations will occur, understanding the composition of a circuit offers a rough gauge of the possible reductions. This could benefit work that is targeting specific gate types, such as that discussed in Section 5.3.5.

5.2.3 Proof of Non-Existence of Three-Gate NCT Commuting Compositions

The motivation for this research was based on the idea posed by Iten et al., that "it could happen that in a circuit $C = (C_1, C_2, C_3)$, no gates commute pairwise, but the unitary corresponding to (C_1, C_2) could commute with the unitary corresponding to C_3." [28]. This is the underlying idea that was generalized to form the commuting composition problem.

The third contribution of this research also advances the theory of the field by following on to that premise by proving it false for a circuit $C = (C_1, C_2, C_3)$, where C_1, C_2, C_3 are gates from the NCT set. This premise may be true for other gate sets, such as the XYZ circuit example in Section 5.3.1. However, the fact that it is not true for the NCT gate set contributes to the work done by Iten et al. by showing that modifying the algorithm to account for the premise stated above would yield no better results than the current version [28]. While the algorithm is capable of running with any gate set, the set of templates currently available and default to the pass creation in Qiskit is from the NCT set [4]. To modify the pattern matching algorithm to account for more commutations, the authors will either need to use a different set of templates or consider commutations of layers of gates.

5.3 Future Work

This section describes areas of future work within the commuting composition problem.

5.3.1 Search for Commuting Compositions of Alternative Gate Sets or Numbers of Elements

A straightforward area of future work is searching for commuting compositions of different types. This work looked specifically for commuting compositions composed of three layers of NCT gates, but that search could be extended to any gate set and number of elements. In particular, it would be interesting to see how commuting compositions of gates compare at a hardware level to that of the NCT gate set. Searching for sequences with this property within the Clifford+T gate set seems promising, as subcircuits composed of fewer than three qubits could be found. For example, using only one qubit, the sequence XYZ is a three layer commuting composition, as can be seen in Figures 73 through 76. Rearrangements of gates at a lower level of abstraction could potentially find more reductions than the higher level.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Figure 73: X, Y, and Z gate matrices

$$XY = \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix} \quad YX = \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix}$$

Figure 74: X and Y gates do not commute

$$YZ = \begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix} \quad ZY = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix}$$

Figure 75: Y and Z gates do not commute

$$XYZ = \begin{pmatrix} i & 0 \\ 0 & i \end{pmatrix} \quad YZX = \begin{pmatrix} i & 0 \\ 0 & i \end{pmatrix} \quad ZXY = \begin{pmatrix} i & 0 \\ 0 & i \end{pmatrix}$$

Figure 76: XYZ three-layer commuting composition

5.3.2 Develop an Efficient Method to Traverse a Quantum Circuit to Find Sub-Circuits and Replace with the Commuted Gate Ordering

Another extension of this work is to create an algorithm that searches for commuting composition sequences within a larger circuit and replaces the match with its commuted version. The most likely method for this is via the DAG format of the circuit. This problem boils down to searching a graph to see if it contains a specific subgraph. This is an instance of the subgraph isomorphism problem, which is NP-Complete [29]. However, it is possible that properties of the commuting composition search problem that are not shared by all instances of the subgraph isomorphism problem admit an efficient solution. Even if this is not the case, it is possible that there are heuristic solutions that could provide satisfactory results. Future work could look at existing heuristic algorithms for the subgraph isomorphism problem, and apply those to finding the commuting composition graph within the DAG representation for the entire circuit. Another possible solution would be treating the commuting composition circuits as templates, then using the pattern matching algorithm by Iten et al. to find and replace the commuting composition subcircuit [28].

5.3.3 Improve Implementation of CheckMatch Algorithm

To save time and resources, the CheckMatch algorithm could be modified to run more efficiently. Two possible improvements are parallel programming and storing computation results for future use. The algorithm is highly parallelizable, as checking each triplet for a match can be done independently. Furthermore, as the algorithm is currently implemented, it computes the same matrix multiplication many times: each time two layers are adjacent it recomputes the product of the two layers. This is a redundancy that could be eliminated by a technique such as memoization. Storing

previously computed results to be looked up rather than recomputed reduces the execution time associated with duplicating the work to compute the product of two matrices, at the expense of greater use of memory.

5.3.4 Analyze Trend between Sequence Count and Matches

The number of unique three layer NCT sequences in relation to three-layer NCT commuting compositions appears to have a log-linear relationship for three, four and five qubits. This relationship is shown in Figure 37. Future work could characterize this relationship and determine if it is expected to hold for six or more qubits. This would help predict the quantity of three-layer NCT commuting compositions that exist for higher qubit counts.

5.3.5 Apply Commuting Compositions to Reduce High-Cost Gates

One of the results of this work is the discovery that the existence of unnecessary adjacent X gates in a three-layer circuit could create a three-layer commuting composition, while the absence of such gates prevents the composition of two gates from commuting with the remaining one. At first glance, the inclusion of superfluous gates seems wasteful. However, when considering the ease of implementing an X gate compared to a CX gate or CCX gate, perhaps neighboring (and therefore unnecessary) X gates are useful in that they would allow elimination of the higher-cost gates.

The premise of this idea is to use commuting compositions to eliminate the maximum number of high-cost gates before eliminating lower cost gates. Given a circuit, it would be iteratively searched for cancellations of the highest cost gate (in the NCT set this would be the CCX gate). After the first iteration, the algorithm would rearrange the layers in the circuit according to the commuting compositions existent in it, then re-search the circuit for additional cancellations of the highest cost gate. This would

continue until no more commutations are available that result in elimination of the targeted gate. The algorithm would then proceed to do the same process for the next highest-cost gate, and so on until the lowest-cost gate. An optional first step for this method would be to insert extra X gate pairs prior to searching for cancellations of high-cost gates, as they would result in a greater number of commutations and thus potential rearrangements yielding a reduction.

The goal of this is to prioritize elimination of high-cost gates at the expense of more single qubit gates. Commuting compositions would be used to avoid preemptively canceling low-cost gates when their existence within the circuit could allow a commutation that would eliminate a higher cost gate.

5.4 Concluding Remarks

According to the *National Defense Strategy*, the primary national security concern of the United States is long term strategic competition with nation states that are advancing in military modernization and aggression [38]. These competitors understand the power technological superiority brings to the fight. One such technology with the potential to radically favor the nation that develops and operationalizes it first is the quantum computer. Leading the way in research, development, and application of quantum sciences is key to the United States maintaining a position of influence in the great power competition. One element of this is creating effiencient quantum circuits.

Quantum circuit simplification aims to reduce circuit cost so that circuits run more accurately and efficiently on quantum computers. This work contributes to that effort by presenting the generalized commuting composition problem, then proceeding to identify all three-layer commuting compositions for three-, four-, and five-qubit circuits composed of NOT, CNOT, and Toffoli gates.

These commuting compositions provide a way to reorder gates in a quantum circuit such that the circuit operator is unchanged. This provides alternative sequences of gates to be evaluated for circuit transpilation and optimization passes. A direct application of this is the potential to find more gate sequences that match templates for template optimization techniques. However, the benefits extend beyond this single case. Alternative orderings of gates in a circuit could result in better physical mapping solutions so that fewer SWAP gates are required; the ability to combine multiple gates into a single gate supported by the device; or the discovery of new sequences of gates to use as reducible templates. Each such improvement makes headway towards realizing a quantum computer that is capable of solving problems not feasible by classical computers.

Once this milestone is reached, quantum computers will significantly impact national security and economic prosperity. Some of the ways in which it will make a difference are known, such as defeating encryption algorithms used to secure sensitive data and communications [59]. Other applications remain to be developed: they could include solving multivariate optimization problems, developing new material technologies, or creating more effective pharmaceuticals [19, 45]. These technological advancements could lead to improving intelligence, surveillance, and reconnaissance efforts, planning in data-saturated conflict, and developing stronger weapon systems. Such applications of quantum computers will increase the United States military's competitive advantage, thereby strengthening its position of influence across the globe.

Appendix A. Asymptotic Time Complexity to Find Three-Layer NCT Commuting Compositions

This appendix presents the asymptotic time complexity of the problem of finding three-layer NCT commuting compositions as qubit count n increases to be $O(n!)$.

The dominating factor in the complexity of this problem is the number of three-layer sequences to be checked for the target property. The number of three-layer sequences is the cube of the number of unique layers. However, it will be shown that the number of sequences in a single layer is super-polynomial in n, so raising that number to a constant power does not change the asymptotic complexity. Therefore, this problem can be reduced to finding the complexity of the number of unique layers over n qubits using gates from the NCT set.

The number of layers can be calculated by defining cases based on gate composition that accounts for all possible gate combinations, then adding the quantities for each case.

The basic cases for this problem are X gates only, CX gates only, and CCX gates only. All other cases will be combinations of these three.

The X only case contributes an exponential complexity. This can be seen by the following: Given n qubits, the options for layers composed of only X gates are n, $(n-1)$, $(n-2)$,...,1 line(s) contain(s) an X gate(s). This corresponds to Equation (19).

$$\sum_{i=1}^{n} \binom{n}{i} = 2^n - 1 \tag{19}$$

The case of CX gates contributes a factorial complexity term to the expression for the overall complexity. The number of layers containing only CX gates for $n \geq 2$ qubits is given in Equation (20).

$$\sum_{i=1}^{m} \frac{n!}{i! \cdot (n-2i)!}, \text{ where } m = \left\lfloor \frac{n}{2} \right\rfloor \tag{20}$$

Each term in this summation corresponds to the number of layers possible with the application of i CX gates. The maximal number of CX gates that can be applied in a given layer is m, as two qubits are required for the gate operation and no two gates in a layer can operate on the same qubit. For n even, $m = \frac{n}{2}$. For n odd, $m = \frac{n-1}{2}$. The total number of possible layers is the sum of the number of layers with each possible number of CX gates from 1 to m.

The number of layers with one CX gate is the number of options for the control qubit multiplied by the number of options for the target qubit. With n qubits, this gives $n \cdot (n-1) = \frac{n!}{(n-2)!}$.

The number of layers with two CX gates is the number of options for choosing the first gate multiplied by the number of options for choosing the second gate. Choosing the first gate (as previously shown) is $n \cdot (n-1)$. There are $(n-2)$ lines remaining from which to choose the second gate, so there are $(n-2) \cdot (n-3)$ choices for the second gate. This gives the total number of options for two gates to be $n \cdot (n-1) \cdot (n-2) \cdot (n-3)$. This number accounts for all two-gate layers twice: for a layer with gates $g_1 g_2$, it accounts for g_1 chosen first and g_2 chosen second, as well as g_2 first and g_1 second. The order of gates does not matter, so the number of unique layers is $\frac{1}{2}(n \cdot (n-1) \cdot (n-2) \cdot (n-3)) = \frac{1}{2} \cdot \frac{n!}{(n-4)!}$.

Similarly, the number of layers with three CX gates is the number of options for the first gate multiplied by the number of options for the second gate multiplied by the number of options for the third gate. This is divided by 3! to avoid accounting for repeated layers. This gives $\frac{1}{3!}(n \cdot (n-1) \cdot (n-2) \cdot (n-3)) \cdot (n-4) \cdot (n-5)) = \frac{1}{3!} \cdot \frac{n!}{(n-6)!}$.

This continues up to m gates. The number of options for m gates is the number of options for one gate times two gates times three gates and so on until m gates. To eliminate repetition, this number is divided by $m!$.

For n even, this gives $\frac{1}{m!}(n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \cdots \cdot 2 \cdot 1) = \frac{1}{m!} \cdot \frac{n!}{(n-2m)!}$.

For n odd, this gives $\frac{1}{m!}(n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \cdots \cdot 3 \cdot 2) = \frac{1}{m!} \cdot \frac{n!}{(n-2m)!}$.

The total number of layers accounts for all the layer options for all cases from one to m gates. This summation gives Equation (20) and shows that the complexity for this case is $O(n!)$.

The final base case, CCX only, also contributes $O(n!)$ complexity. The generic formula for this case in given in Equation (21), with $n \geq 3$.

$$\sum_{i=0}^{m-1} \frac{1}{(i+1)!} \cdot (n-3i) \cdot \binom{(n-3i-1)}{2} = \sum_{i=0}^{m-1} \frac{1}{(i+1)!} \cdot \frac{(n-3i)!}{2! \cdot (n-3i-3)!}, \text{ where } m = \left\lfloor \frac{n}{3} \right\rfloor \tag{21}$$

This formula follows the same logic as the CX case. For each number of CCX gates k, the total layers is the number of options for the first gate times the number of options for the second gate, and so on until the k^{th} gate. This is divided by $k!$ to eliminate repetition. The total number of layers for all qubits is the summation of the number of options for k gates from $k = 1$ to $k = m$, where m is the maximal number of CCX gates that can fit in one layer over n qubits. For clarity of expression, Equation (21) begins with $i = 0$ to represent the first case of $k = 1$ gates, and finishes at $i = m - 1$ to compute the $k = m$ case.

This expression differs from the CX only case in that for each choice of target qubit, two control qubits must be chosen. The order of the control qubits for each target qubit does not matter, hence the use of a combination rather than permutation.

As can be seen from Equation (21), the complexity for the CCX only case is $O(n!)$. This means that of the basic cases, the highest complexity is $O(n!)$. All other cases for gate compositions per layer will be of the form

$$O(N_{all}) = O(N_X(j)N_{CX}(k)N_{CCX}(l)) \tag{22}$$

with $N_X(j)$ the number of options for j X gates, $N_{CX}(k)$ the number of options for k CX gates, $N_{CCX}(l)$ the number of options for l CCX gates, and $k+2j+3l \leq n$. Since the highest complexity of those three basic options is $O(n!)$, the highest complexity of any combination of those options will also be $O(n!)$.

Therefore, since the total number of layers using NCT gates on n qubits is the summation of the number of layers for each gate composition case, the asymptotic time complexity for total layers is that of the highest complexity existent within the summation, which is $O(n!)$.

Bibliography

1. N. Abdessaied, M. Amy, M. Soeken, and R. Drechsler, "Technology mapping of reversible circuits to Clifford+T quantum circuits," *Proceedings of The International Symposium on Multiple-Valued Logic*, vol. 2016-July, pp. 150–155, 2016.

2. N. Abdessaied and R. Drechsler, *Reversible and Quantum Circuits, Optimization and Complexity Analysis*, 1st ed. Springer International Publishing, 2016.

3. N. Abdessaied, M. Soeken, R. Wille, and R. Drechsler, "Exact template matching using boolean satisfiability," *Proceedings of The International Symposium on Multiple-Valued Logic*, pp. 328–333, 2013.

4. H. Abraham, AduOffei, R. Agarwal, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, M. Amy, E. Arbel, Arijit02, A. Asfaw, A. Avkhadiev, C. Azaustre, AzizNgoueya, A. Banerjee, A. Bansal, P. Barkoutsos, A. Barnawal, G. Barron, G. S. Barron, L. Bello, Y. Ben-Haim, D. Bevenius, A. Bhobe, L. S. Bishop, C. Blank, S. Bolos, S. Bosch, Brandon, S. Bravyi, Bryce-Fuller, D. Bucher, A. Burov, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Chen, C.-F. Chen, E. Chen, J. C. Chen, R. Chen, J. M. Chow, S. Churchill, C. Claus, C. Clauss, R. Cocking, F. Correa, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, C. Culver, A. D. Córcoles-Gonzales, S. Dague, T. E. Dandachi, M. Daniels, M. Dartiailh, DavideFrr, A. R. Davila, A. Dekusar, D. Ding, J. Doi, E. Drechsler, Drew, E. Dumitrescu, K. Dumon, I. Duran, K. EL-Safty, E. Eastman, G. Eberle, P. Eendebak, D. Egger, M. Everitt, P. M. Fernández, A. H. Ferrera, R. Fouilland, FranckChevallier, A. Frisch, A. Fuhrer, B. Fuller, M. GEORGE, J. Gacon, B. G. Gago, C. Gambella, J. M. Gambetta, A. Gammanpila, L. Garcia, T. Garg, S. Garion, A. Gilliam, A. Giridharan, J. Gomez-

Mosquera, Gonzalo, S. de la Puente González, J. Gorzinski, I. Gould, D. Greenberg, D. Grinko, W. Guan, J. A. Gunnels, M. Haglund, I. Haide, I. Hamamura, O. C. Hamido, F. Harkins, V. Havlicek, J. Hellmers, Ł. Herok, S. Hillmich, H. Horii, C. Howington, S. Hu, W. Hu, J. Huang, R. Huisman, H. Imai, T. Imamichi, K. Ishizaki, R. Iten, T. Itoko, JamesSeaward, A. Javadi, A. Javadi-Abhari, W. Javed, Jessica, M. Jivrajani, K. Johns, S. Johnstun, Jonathan-Shoemaker, V. K, T. Kachmann, A. Kale, N. Kanazawa, Kang-Bae, A. Karazeev, P. Kassebaum, J. Kelso, S. King, Knabberjoe, Y. Kobayashi, A. Kovyrshin, R. Krishnakumar, V. Krishnan, K. Krsulich, P. Kumkar, G. Kus, R. LaRose, E. Lacal, R. Lambert, J. Lapeyre, J. Latone, S. Lawrence, C. Lee, G. Li, D. Liu, P. Liu, Y. Maeng, K. Majmudar, A. Malyshev, J. Manela, J. Marecek, M. Marques, D. Maslov, D. Mathews, A. Matsuo, D. T. McClure, C. McGarry, D. McKay, D. McPherson, S. Meesala, T. Metcalfe, M. Mevissen, A. Meyer, A. Mezzacapo, R. Midha, Z. Minev, A. Mitchell, N. Moll, J. Montanez, G. Monteiro, M. D. Mooring, R. Morales, N. Moran, M. Motta, MrF, P. Murali, J. Müggenburg, D. Nadlinger, K. Nakanishi, G. Nannicini, P. Nation, E. Navarro, Y. Naveh, S. W. Neagle, P. Neuweiler, J. Nicander, P. Niroula, H. Norlen, NuoWenLei, L. J. O'Riordan, O. Ogunbayo, P. Ollitrault, R. Otaolea, S. Oud, D. Padilha, H. Paik, S. Pal, Y. Pang, V. R. Pascuzzi, S. Perriello, A. Phan, F. Piro, M. Pistoia, C. Piveteau, P. Pocreau, A. Pozas-Kerstjens, M. Prokop, V. Prutyanov, D. Puzzuoli, J. Pérez, Quintiii, R. I. Rahman, A. Raja, N. Ramagiri, A. Rao, R. Raymond, R. M.-C. Redondo, M. Reuter, J. Rice, M. Riedemann, M. L. Rocca, D. M. Rodríguez, RohithKarur, M. Rossmannek, M. Ryu, T. SAPV, SamFerracin, M. Sandberg, H. Sandesara, R. Sapra, H. Sargsyan, A. Sarkar, N. Sathaye, B. Schmitt, C. Schnabel, Z. Schoenfeld, T. L. Scholten, E. Schoute, J. Schwarm, I. F. Sertage, K. Setia, N. Shammah, Y. Shi, A. Silva, A. Simonetto, N. Singstock,

Y. Siraichi, I. Sitdikov, S. Sivarajah, M. B. Sletfjerding, J. A. Smolin, M. Soeken, I. O. Sokolov, I. Sokolov, SooluThomas, Starfish, D. Steenken, M. Stypulkoski, S. Sun, K. J. Sung, H. Takahashi, T. Takawale, I. Tavernelli, C. Taylor, P. Taylour, S. Thomas, M. Tillet, M. Tod, M. Tomasik, E. de la Torre, K. Trabing, M. Treinish, TrishaPe, D. Tulsi, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, A. C. Vazquez, V. Villar, D. Vogt-Lee, C. Vuillot, J. Weaver, J. Weidenfeller, R. Wieczorek, J. A. Wildstrom, E. Winston, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, S. Wood, J. Wootton, D. Yeralin, D. Yonge-Mallo, R. Young, J. Yu, C. Zachow, L. Zdanski, H. Zhang, C. Zoufal, Zoufalc, a kapila, a matsuo, bcamorrison, brandhsn, nick bronn, brosand, chlorophyll zz, csseifms, dekel.meirom, dekelmeirom, dekool, dime10, drholmie, dtrenev, ehchen, elfrocampeador, faisaldebouni, fanizzamarco, gabrieleagl, gadial, galeinston, georgios ts, gruu, hhorii, hykavitha, jagunther, jliu45, jscott2, kanejess, klinvill, krutik2966, kurarrr, lerongil, ma5x, merav aharoni, michelle4654, ordmoj, sagar pahwa, rmoyard, saswati qiskit, scottkelso, sethmerkel, shaashwat, sternparky, strickroman, sumitpuri, tigerjack, toural, tsura crisaldo, vvilpas, welien, willhbang, yang.luh, yotamvakninibm, and M. Čepulkovskis, "Qiskit: An open-source framework for quantum computing," 2019.

5. M. B. Ali, T. Hirayama, K. Yamanaka, and Y. Nishitani, "Quantum cost reduction of reversible circuits using new toffoli decomposition techniques," *Proceedings - 2015 International Conference on Computational Science and Computational Intelligence, CSCI 2015*, pp. 59–64, 2016.

6. C. G. Almudever, L. Lao, R. Wille, and G. G. Guerreschi, "Realizing quantum algorithms on real quantum computing devices," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 864–872.

7. P. M. Arpita, K. Datta, R. Vemula, and I. Sengupta, "Optimization of reversible circuits using triple-gate templates at quantum gate level," *2015 International Conference on Electronic Design, Computer Networks and Automated Verification, EDCAV 2015*, pp. 120–124, 2015.

8. A. Barenco, C. H. Bennett, R. Cleve, D. P. Divincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical Review A*, vol. 52, no. 5, pp. 3457–3467, 1995.

9. M. Bataille and J. G. Luque, "Quantum circuits of c-Z and SWAP gates: Optimization and entanglement," *Journal of Physics A: Mathematical and Theoretical*, vol. 52, no. 32, 2019.

10. L. Biswal, A. Bhattachaijee, S. Ghosh, and H. Rahaman, "Implementation of neareast neighbor quantum circuit with low quantum cost," *2018 International Symposium on Devices, Circuits and Systems, ISDCS 2018*, pp. 1–6, 2018.

11. L. Biswal, R. Das, C. Bandyopadhyay, A. Chattopadhyay, and H. Rahaman, "A template-based technique for efficient Clifford+T-based quantum circuit implementation," *Microelectronics Journal*, no. August, pp. 58–68.

12. S. Bravyi and A. Kitaev, "Universal quantum computation with ideal clifford gates and noisy ancillas," *Physical Review A*, vol. 71, no. 2, p. 022316, 2005.

13. L. Burgholzer, H. Bauer, S. Hillmich, and R. Willie, "JKQ QFR - A JKQ library for quantum functionality representation written in c++," 2020, Accessed on: November 20, 2020. [Online]. Available: https://github.com/iic-jku/qfr

14. A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," 2017.

15. A. A. de Almeida, G. W. Dueck, and A. C. R. da Silva, "Efficient realization of Toffoli and NCV circuits for IBM QX architectures," pp. 131–145, 2019.

16. A. A. De Almeida, G. W. Dueck, and A. C. Da Silva, "Finding optimal qubit permutations for IBM's quantum computer architectures," *Proceedings - 32nd Symposium on Integrated Circuits and Systems Design, SBCCI 2019*, pp. 1–6, 2019.

17. L. Debnath and P. Mikusinski, *Introduction to Hilbert Spaces with Applications*. Elsevier Academic Press, 2005.

18. D. P. DiVincenzo, "The physical implementation of quantum computation," *Fortschritte der Physik*, vol. 48, no. 9-11, pp. 771–783, 2000.

19. V. Elfving, B. W. Broer, M. Webber, J. Gavartin, M. D. Halls, K. Lorton, and A. Bochevarov, "How will quantum computers provide an industrially relevant computational advantage in quantum chemistry," *arXiv: Quantum Physics*, 2020.

20. A. Fedorov, A. Akimov, J. Biamonte, A. Kavokin, F. Y. Khalili, E. Kiktenko, N. Kolachevsky, Y. V. Kurochkin, A. Lvovsky, A. Rubtsov *et al.*, "Quantum technologies in Russia," *Quantum Science and Technology*, vol. 4, no. 4, p. 040501, 2019.

21. M. Giles, "The US and China are in a quantum arms race that will transform warfare," *MIT Technology Review*, 2019.

22. A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, "Quipper: a scalable quantum programming language," in *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, 2013, pp. 333–342.

23. IBM, "IBM quantum experience," Accessed on: 11 February, 2021. [Online]. Available: https://quantum-computing.ibm.com

24. IBM, "Open-source quantum development," Accessed on: 11 February, 2021. [Online]. Available: https://qiskit.org

25. IBM, "Qiskit 0.23.1 documentation," Accessed on: 10 February, 2021. [Online]. Available: https://qiskit.org/documentation/

26. IBM, "The qiskit elements," Accessed on: 11 February, 2021. [Online]. Available: https://qiskit.org/documentation/the_elements.html#aer

27. R. Iten, R. Colbeck, I. Kukuljan, J. Home, and M. Christandl, "Quantum circuits for isometries," *Physical Review A*, vol. 93, no. 3, pp. 1–21, 2016.

28. R. Iten, R. Moyard, T. Metger, D. Sutter, and S. Woerner, "Exact and practical pattern matching for quantum circuit optimization," 2020, Accessed on: 11 February, 2021. [Online]. Available: https://arxiv.org/abs/1909.05270v2

29. S. Kijima, Y. Otachi, T. Saitoh, and T. Uno, "Subgraph isomorphism in graph classes," *Discrete Mathematics*, vol. 312, no. 21, pp. 3164 – 3173, 2012, Accessed on: 15 October, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0012365X12003160

30. A. Kole and K. Datta, "Improved NCV Gate Realization of Arbitrary Size Toffoli Gates," *Proceedings - 2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems, VLSID 2017*, pp. 289–294, 2017.

31. C. Lalengmawia and A. Chakrabarty, "Compiling NCV quantum circuits for nearest neighbour realization," *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, pp. 1–5, 2020.

32. A. J. Landahl, D. S. Lobser, B. C. A. Morrison, K. M. Rudinger, A. E. Russo, J. W. V. D. Wall, and P. Maunz, "Jaqal, the quantum assembly language for QSCOUT," 2020.

33. N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, "Experimental comparison of two quantum computing architectures," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3305–3310, 2017.

34. M. Martonosi and M. Roetteler, "Next steps in quantum computing: Computer science's role," *arXiv preprint arXiv:1903.10541*, 2019.

35. D. Maslov, C. Young, D. Miller, and G. Dueck, "Quantum circuit simplification using templates." *Design, Automation and Test in Europe, Design, Automation and Test in Europe, 2005. Proceedings*, p. 1208, 2005.

36. D. Maslov, G. W. Dueck, and D. M. Miller, "Toffoli network synthesis with templates," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 807–817, 2005.

37. D. Maslov, G. W. Dueck, D. M. Miller, and C. Negrevergne, "Quantum circuit simplification and level compaction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 3, pp. 436–444, 2008.

38. J. Mattis, "Summary of the 2018 national defense strategy of the United States of America," Department of Defense Washington United States, Tech. Rep., 2018.

39. G. Meuli, M. Soeken, M. Roetteler, N. Wiebe, and G. De Micheli, "A best-fit mapping algorithm to facilitate ESOP-decomposition in Clifford+T quantum network synthesis," *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, vol. 2018-Janua, pp. 664–669, 2018.

40. D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Proceedings of the 40th annual Design Automation Conference*, 2003, pp. 318–323.

41. D. M. Miller, M. Soeken, and R. Drechsler, "Mapping NCV circuits to optimized Clifford+T circuits," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8507 LNCS, pp. 163–175, 2014.

42. P. Murali, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Formal constraint-based compilation for noisy intermediate-scale quantum systems," *Microprocessors and Microsystems*, vol. 66, pp. 102–112, 2019, Accessed on: 20 April, 2020. [Online]. Available: https://doi.org/10.1016/j.micpro.2019.02.005

43. P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, "Full-stack, real-system quantum computer studies: Architectural comparisons and design insights," pp. 527–540, 2019.

44. Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov, "Automated optimization of large quantum circuits with continuous parameters," *npj Quantum Information*, vol. 4, no. 1, pp. 1–12, 2018.

45. M. A. Nielsen and I. Chuang, *Quantum computation and quantum information*. Cambridge University Press, 2016.

46. P. Niemann, A. Gupta, and R. Drechsler, "T-depth optimization for fault-tolerant quantum circuits," *Proceedings of The International Symposium on Multiple-Valued Logic*, vol. 2019-May, pp. 108–113, 2019.

47. N. Q. C. Office. (2021) National quantum initiative: The federal source and gateway to quantum R&D across the U.S. government. Accessed on: 8 February, 2021. [Online]. Available: https://www.quantum.gov/

48. QuTech, "Knowledge base: Bloch sphere," Accessed on: January 5, 2021. [Online]. Available: https://www.quantum-inspire.com/kbase/bloch-sphere/

49. M. M. Rahman, A. Banerjee, G. W. Dueck, and A. Pathak, "Two-qubit quantum gates to reduce the quantum cost of reversible circuit," *Proceedings - 41st IEEE International Symposium on Multiple-Valued Logic, ISMVL 2011*, pp. 86–92, 2011.

50. M. M. Rahman and G. W. Dueck, "An algorithm to find quantum templates," in *2012 IEEE Congress on Evolutionary Computation*. IEEE, 2012, pp. 1–7.

51. ——, "Optimal quantum circuits of three qubits," IEEE, pp. 161–166, 2012.

52. M. M. Rahman, G. W. Dueck, A. Chattopadhyay, and R. Wille, "Integrated synthesis of linear nearest neighbor Ancilla-free MCT circuits," *Proceedings of The International Symposium on Multiple-Valued Logic*, vol. 2016-July, pp. 144–149, 2016.

53. M. M. Rahman, G. W. Dueck, and J. D. Horton, "An algorithm for quantum template matching," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 11, no. 3, pp. 1–20, 2014.

54. M. G. Raymer and C. Monroe, "The US national quantum initiative," *Quantum Science and Technology*, vol. 4, no. 2, p. 020504, 2019.

55. C. S. Review, *A Survey on quantum computing technology*, 2019.

56. S. M. Saeed, X. Cui, R. Wille, A. Zulehner, K. Wu, R. Drechsler, and R. Karri, "Towards reverse engineering reversible logic," 2017, Accessed on: 10 February 2020. [Online]. Available: http://arxiv.org/abs/1704.08397

57. Q. Schiermeier, "Russia joins race to make quantum dreams a reality," *Nature*, vol. 577, no. 7788, pp. 14–14, 2019.

58. N. O. Scott and G. W. Dueck, "Pairwise decomposition of Toffoli gates in a quantum circuit," *Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI*, pp. 231–235, 2008.

59. P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

60. R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: An online resource for reversible functions and reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, 2008, pp. 220–225, RevLib is available at http://www.revlib.org.

61. ——, "RevLib: An online resource for reversible functions and reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, 2008, pp. 220–225, Accessed on: August 7, 2020. [Online]. Available: http://www.revlib.org

62. R. Wille, A. Chattopadhyay, and R. Drechsler, "From reversible logic to quantum circuits: Logic design for an emerging technology," *Proceedings - 2016 16th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, SAMOS 2016*, pp. 268–274, 2017.

63. H. Wilson, "US Air Force science and technology strategy 2030 and beyond," Dept of the Air Force Washington United States, Tech. Rep., 2019.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
25–03–2021	Master's Thesis	Jun 2019 — Mar 2021

4. TITLE AND SUBTITLE

Commuting Compositions
for Quantum Circuit Reduction

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Cole, Brenna R, Capt

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-21-M-023

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RITQ
Ms. Laura Wessing
525 Brooks Road
Rome, NY 13441
COMM: 315-330-2937

10. SPONSOR/MONITOR'S ACRONYM(S)

AFR/RITQ

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Quantum circuit simplification improves program execution on quantum hardware by reducing error from prolonged environmental interaction and noisy gate operations. One simplification technique is template matching, which repeatedly conducts local optimization by replacing small sequences of gates within a circuit by optimized versions. Underlying this method is the problem of identifying sequences matching templates. This is challenging because some, but not all, gates can commute within a circuit. This means there may not be a subcircuit that matches a template in the original circuit specification, but a match may exist in an equivalent rearrangement of gates. In such cases, certain reductions are possible only after the consideration of alternative gate orderings. This research focuses on the identification of commuting gate sequences in support of circuit reduction. In particular, this work generalizes the notion of commuting gates and layers to n-layer commuting compositions and identifies all three-layer commuting compositions composed of Toffoli, CNOT, and NOT gates for circuits with three to five qubits.

15. SUBJECT TERMS

Quantum computing, Quantum circuits, Reversible circuits, Circuit optimization, Template matching

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Laurence D. Merkle, AFIT/ENG
U	U	U	UU	142	19b. TELEPHONE NUMBER *(include area code)* (937) 255-6565; laurence.merkle@afit.edu