Algorithms for the Optimization of Quantum Circuits

by

Matthew Amy

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science - Quantum Information

Waterloo, Ontario, Canada, 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis investigates techniques for the automated optimization of quantum circuits. In the first part we develop an exponential time algorithm for synthesizing minimal depth quantum circuits. We combine this with effective heuristics for reducing the search space, and show how it can be extended to different optimization problems. We then use the algorithm to compute circuits over the Clifford group and T gate for many of the commonly used quantum gates, improving upon the former best known circuits in many cases.

In the second part, we present a polynomial time algorithm for the re-synthesis of $CNOT$ and T gate circuits while reducing the number of phase gates and parallelizing them. We then describe different methods for expanding this algorithm to optimize circuits over Clifford and T gates.

Acknowledgements

I would first like to thank my supervisor, Michele Mosca, to whom I'm greatly indebted to for his teaching and insight. Were it not for him, I would never have gained an appreciation and love for quantum computing. I also wish to thank my reading committee, John Watrous and Richard Cleve, for their helpful comments and suggestions.

Throughout the course of my graduate studies, I had the pleasure of working with many excellent quantum computer scientists, and for that I'm extremely grateful. I wish to thank Dmitri Maslov for his guidance and mentoring, and for motivating me to pursue projects I would have been too short sighted to pursue otherwise. I would also like to thank Martin Roetteler, Austin Fowler, Richard Lazarus, and the rest of the TORQUE team for all their tireless efforts bringing such a far reaching project together; I am indeed indebted to them for the many stimulating discussions and the motivation for my own research that this project has provided.

My fellow students Vincent Russo, Adam Paetznick, and Vadym Kliuchnikov have also provided a wealth of helpful comments and discussions throughout the course of my research, to which I'm extremely grateful for.

For all the technical support of those listed above, this thesis would not have been possible without the encouragement of my friends and family. I'm deeply grateful to my parents, John and Ingrid Amy, whose love and support has helped me through many tough times, and for encouraging me to pursue a graduate education. I would also like to thank my good friends Alexandre Laplante, Vincent Launchbury, Parsiad Azimzadeh and Kyle Robinson, who provided some much needed distractions during times of high stress.

Finally, I wish to thank Rebecca Vasluianu for all of her love and for putting up with me when I was more than just a little distracted by my work. Her support has meant more to me than I can begin to describe here.

To Rebecca

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Circuit optimization, believed to be an intractable problem [3], is an important part of the design and construction of classical computational devices. The ability to produce smaller, energy efficient integrated circuits relies heavily on the ability to reduce the logical complexity of the circuit's functionality, especially with the gradual slowing of improvements to transistor technology. Accordingly, researchers have developed effective heuristic methods for minimizing logic in integrated circuits, notably the well-known Quine-McCluskey and ESPRESSO algorithms [4], the latter of which is used as a standard optimization procedure in modern logic synthesis tools and VLSI design.

Given the initially limited quantum computational resources available, in order to handle interesting problem sizes it will be even more important to reduce the resources required to implement a given quantum circuit through similar circuit optimization. In fact, as realistic quantum computers will likely require some fault tolerance scheme where the amount of error correction is proportional to the resources used, the effect of circuit optimizations becomes even more profound. While current experimental circuits can be optimized by hand, recent advances in quantum information processing devices (e.g. [5], [6], [7], [8]) and improvements to fault-tolerant thresholds (e.g. [9], [10], [11]) hint at the prospect of scalable quantum devices. As a result, there is a growing need for quantum circuit optimization tools that can be applied to large quantum circuits, in order to allow the best possible use of these new computational devices.

Unfortunately, few tools have to this point been developed for the direct purpose of optimizing quantum circuits. While techniques and recent breakthroughs in reversible circuit optimizations [12, 13, 14] are applicable to the classical subset of quantum computation, this leaves out a large class of quantum circuits from consideration – moreover,

reversible circuits themselves usually require higher level logic (e.g. Toffoli gates) that must be decomposed to a fault tolerant quantum gate set. Likewise, while there exist *synthesis* methods that can be used to optimize circuits, they are limited to single-qubit circuits, or are otherwise impractical for multi-qubit circuits. Recent developments in quantum circuit optimization [15, 16, 17] show promise, but the landscape still remains fairly barren.

This thesis makes progress on developing algorithms and techniques for optimizing quantum circuits themselves, both for large scale and small scale circuit optimization. The two approaches are complimentary in that small, common operations can be optimized exactly, while large compound circuits can be further optimized according to less effective but more efficient methods. In the first part of this thesis, we develop an algorithm and related techniques for finding exactly optimal circuits in exponential time, leading to efficient circuits for common quantum operations. In the second part, we build a family of heuristic polynomial time algorithms for optimization of large quantum circuits.

One pervading theme of this thesis is the optimization of *fault-tolerant* circuits, circuits composed of logical gates on encoded groups of qubits. Given the fragile nature of qubits, the prospects of scalable quantum computing without some systematic way of mitigating physical errors and noise are bleak – for this reason, we look to fault tolerance to guide optimization. In particular, the notions of T-count and T-depth, motivated by fault tolerance, inform many of our constructions.

1.1 Overview of Thesis

The thesis is organized as follows. Chapters 2 and 3 provide mathematical preliminaries and background. In Chapter 2 we briefly describe reversible and quantum computation, and define the notation and some basic lemmas we will use. Chapter 3 provides a short survey of the current state of quantum circuit optimization.

Chapters 4 and 5 present original work on the optimization of quantum circuits. Chapter 4 describes a general algorithm for speeding up brute-force optimization of quantum circuits, as well as search space reductions to make the algorithm more practical. The algorithm is motivated by the need to compile higher level gates into lower level gate sets in a depth-optimal way as circuit depth directly affects the run-time, and by extension the error rates. We then use this algorithm to compute depth-optimal decompositions of many common quantum gates into the standard fault-tolerant "Clifford + T" gate set. We also describe how to modify the algorithm to incorporate other cost metrics, ancillas, and synthesize approximate circuits.

In contrast to the algorithm described in Chapter 4, Chapter 5 presents a polynomial-time quantum circuit optimization algorithm that is shown to scale well to large, practical quantum circuits. The algorithm re-synthesizes quantum circuits consisting of Clifford + T gates while minimizing the number of T gates and placing them in parallel. Such properties have very recently become common optimization concerns [18, 19, 20, 21], as T gates require state distillation in the common fault tolerant quantum computing schemes – in fact, Fowler [22] describes how to perform fault tolerant quantum computation in one round of measurement per stage of parallel T gates. As a byproduct, we also reduce exact minimization of T-count to the minimization of polynomial equations in mixed arithmetic. This algorithm also represents progress towards the automated usage of ancillas in depth optimizations, as the usage of ancillas comes at effectively no performance cost.

Chapter 2

Reversible and Quantum Computation

In this chapter we detail the notions of reversible and quantum computation that will be relevant to the material presented in this thesis. We begin with an account of reversible computation, then extend this to cover quantum computations. As the optimizations we develop will be designed with fault tolerant circuits in mind, we end off with a brief discussion of quantum fault tolerance.

2.1 Reversible Computation

Classical computation is typically performed *irreversibly* – the inputs to a computation cannot generally be recovered from the outputs, and so the computation cannot be undone or reversed. It was Landauer who first noticed that this process, in effect destroying information, results in a dissipation of energy in the form of heat or noise [23]. Classical computers built from the irreversible NAND gate – defined as the two-input logic gate that returns 0 if and only if both inputs are 1 – thus waste huge quantities of energy through the process of constantly discarding information.

We could instead consider a model of classical computation in which computations are reversible. In particular, we will define a circuit model of classical computation that permits a straightforward restriction to reversible computations, which we see is equally powerful. While we are mostly interested in reversible computations as the classical subset of quantum

computations, the promise of low-power electronics has led researchers to seriously consider reversible computing models as an alternative to traditional digital logic.

While we save many details of circuit model for Section 2.3, which defines the more general quantum circuit model, we describe a few basic notions here. In the classical circuit model of computation, *wires* carry bits of information to logic gates, which produce new bits of information. The state of an individual bit can be represented as a binary value $b \in \mathbb{F}_2$, where $\mathbb{F}_2 = \{0, 1\}$ is the two element finite field with multiplication and addition corresponding to logical AND ($\wedge$) and exclusive-OR ($\oplus$), respectively – the representation of $\mathbb{F}_2$ as a field is not strictly necessary, but will help to classify types of classical functions. The state of a system of n bits is then represented by a binary string of length n or equivalently a vector in $\mathbb{F}_2^n$; likewise, classical functions map n-bit strings to m-bit strings, i.e. classical functions are operators $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$. We typically refer to classical functions as just functions, and if $m = 1$ we call f a *Boolean* function. An individual logic gate implements a particular function on its input bits, and a circuit threads bits through a sequence of gates.

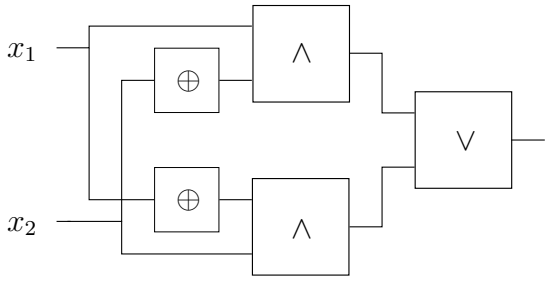


Figure 2.1: An example of a classical circuit computing $x_1 \oplus x_2$.

We define a set $\mathcal{G}$ of classical gates as *universal* for classical computations if and only if any classical function f can be computed by a circuit using only gates in $\mathcal{G}$. In particular, NAND (along with the ability to copy bits) is universal for classical computation. While we won't be particularly concerned with universality for classical computation, it will later play a more significant role in quantum computation.

In restricting our attention to the *reversible* circuit model, we require that logic gates implement invertible functions – in other words, classical functions $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ such that f^{-1} exists. As a result, the universal gate NAND no longer applies in the reversible model, and moreover no (reversible) gate set alone is universal simply by noting that a classical function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ may not be invertible and thus cannot be implemented directly.

We can recover universality by allowing the use of *ancillas*, bits that can be initialized to either 0 or 1, as a kind of temporary register. Under this assumption, the reversible

Toffoli gate,

$$TOF(x_1, x_2, x_3) = (x_1, x_2, x_3 \oplus (x_1 \wedge x_2)),$$

is universal, by noting that $TOF(x_1, x_2, 1) = (x_1, x_2, NAND(x_1, x_2))$ and $TOF(x_1, 1, 0) = (x_1, 1, x_1)$, i.e. the Toffoli gate can implement both NAND and copy bits. Of course, we need a way to reclaim the ancillas after a computation is finished, otherwise computations will continue to grow in space. One option is to erase the partial information contained in the ancillas – while this solution is suitable for classical computations, discarding such information can have a profound effect on the output in a quantum computation. Another option that avoids this problem in the quantum case is to copy the outputs to fresh ancillas, then *uncompute* f by applying $f^\dagger = f^{-1}$ to free up the used ancillas.

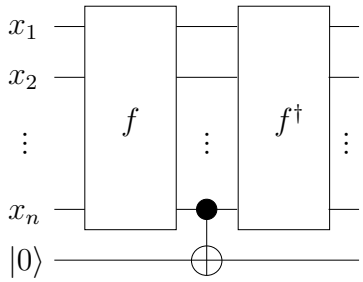


Figure 2.2: An example of a circuit reversibly computing f and cleaning up ancillas.

The addition of ancillary bits complicates our mathematical formalism to an extent. While a function f that accepts n primary inputs and $N - n$ ancillary bits (without loss of generality assume they are initialized to the 0 state) can be described as a classical function over $\mathbb{F}_2^N$, we only really care about its affect on some dimension n subspace V of $\mathbb{F}_2^N$. Though a seemingly inconsequential point, it will allow more precise analysis of computations using ancillas.

2.1.1 Linear functions

In this thesis, we will be particularly interested in classical functions that are linear over $\mathbb{F}_2$; recall that $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ is *linear* if $f(x \oplus y) = f(x) \oplus f(y)$ for every $x, y \in \mathbb{F}_2^n$. The *controlled-NOT* gate

$$CNOT(x_1, x_2) = (x_1, x_1 \oplus x_2),$$

is an example of a linear reversible gate. As an important result, the set of all linear reversible functions are those that can be computed by only using $CNOT$ gates [14].

Before ending off the discussion of reversible circuits, we turn our attention to classical functions which are both linear and Boolean, and how such functions can be implemented reversibly. As linear Boolean functions are linear transformations $\mathbb{F}_2^n \to \mathbb{F}_2$ and are thus represented by $1 \times n$ matrices over $\mathbb{F}_2$, we can see them as (column) vectors in the *dual vector space* $\mathbb{F}_2^{n*}$ of $\mathbb{F}_2^n$. In this view, any state x in $\mathbb{F}_2^n$ is also a linear Boolean function x^T, and vice versa, where x^T denotes the matrix transpose of x. This viewpoint has the advantage that we can view states as either vectors or functions, and apply definitions and theorems from linear algebra to linear Boolean functions, most notably those related to dimension.

Definition 1. Given a set of linear Boolean functions $S \subseteq \mathbb{F}_2^{n*}$, the *rank* of S, denoted $\text{rank}(S)$ is the dimension of the subspace spanned by the vectors in S.

As we will primarily be concerned with reversible computations, we prove a lemma relating linear Boolean functions to linear reversible functions.

Lemma 1. *Given a subspace V of $\mathbb{F}_2^N$ and a set of linear Boolean functions $S = \{f_1, f_2 \dots f_N\} \subseteq V^*$, the linear surjective function $f : V \to W$ defined as*

$$f(a_1, a_2, \dots, a_N) = (f_1(a_1, \dots, a_N), \dots, f_N(a_1, \dots, a_N))$$

is reversible if and only if $\text{rank}(S) = \dim(V)$.

Proof. Reversibility follows if and only if $\text{rank}(S) = \dim(V)$ due to a straightforward application of the rank-nullity theorem; $\dim(\text{im}(f)) + \dim(\ker(f)) = \dim(V)$. As the functions in S form the row vectors of f, $\text{rank}(S) = \dim(\text{im}(f)) = \dim(W)$ and so f is one-to-one if and only if $\text{rank}(S) = \dim(V)$. Thus f is invertible on its image W, i.e., there exists $f^{-1} : W \to V$ such that $ff^{-1} = f^{-1}f = I$. □

As a consequence of Lemma 1, we establish criteria on when a set S of linear Boolean functions can be computed simultaneously over an N-bit state space V – specifically, if there is a size N superset S' of S with dimension $\dim(V)$, there is a reversible circuit with outputs computing $S \subseteq S'$. We use this criteria to define the notion of (reversible) computability on sets of linear Boolean functions – as we will henceforth be concerned strictly with reversible computations, we omit the qualifier *reversible* in further discussions.

Definition 2. Given a dimension n subspace V of $\mathbb{F}_2^N$, a set $S \subseteq V^*$ is *(reversibly) computable* over V if there exists a size N superset S' of S such that $\text{rank}(S') = n$.

2.2 Quantum Computation

One of the most remarkable discoveries of the last century has been that quantum mechanics can directly be used to govern computations. By encoding information in physical systems that evolve according to the laws of quantum mechanics, computations, i.e. particular evolutions of the computer's state, can make use of quantum mechanical effects such as superposition and entanglement. In many cases, such effects can be used to solve problems more efficiently than the best known classical algorithms [25, 26]. For the purpose of this thesis, we provide an (incomplete) introduction to some of the relevant concepts in quantum mechanics – a full introduction can be found in any of [27, 28, 24].

Unlike classical computations, where the state of an n-bit system is represented by a vector in $\mathbb{F}_2^n$, the state of a quantum system is defined as an element of a finite-dimensional complex vector space[1] $\mathcal{H}$. Typically, we consider systems where the dimension of $\mathcal{H}$ is 2^n for some n – in this case, we say the system contains n *quantum bits* or *qubits*.

We use *Dirac notation* to refer to quantum states, where a vector in $\mathcal{H}$ is written as $|\psi\rangle$. For convenience, we distinguish a particular basis of $\mathcal{H}$ called the computational basis, and denote its elements $|x\rangle$ with binary strings x of length n, when the dimension of $\mathcal{H}$ is 2^n. As in reversible computation, we refer to the dual space of $\mathcal{H}$ as $\mathcal{H}^*$, and write vectors in $\mathcal{H}^*$ as $\langle\psi| : \mathcal{H} \to \mathbb{C}$. A vector $\langle\psi|$ is obtained by taking the adjoint of $|\psi\rangle$, denoted as $|\psi\rangle^\dagger$ – in the case of a vector (or matrix), the adjoint is obtained by taking its transpose and the complex conjugate of each entry. The inner product of $|\psi\rangle$ and $|\phi\rangle$ is thus given by $\langle\psi|\phi\rangle = \langle\psi| \cdot |\phi\rangle$ and their outer product is given as $|\psi\rangle\langle\phi|$.

Typically we will refer to systems composed of several subsystems, for instance when we want to separate data and ancillas. We use the *tensor product* to combine the state space of multiple systems, and write $\mathcal{H}_1 \otimes \mathcal{H}_2$ to denote the tensor product of the systems $\mathcal{H}_1$ and $\mathcal{H}_2$. The combined state of systems $\mathcal{H}_1$ and $\mathcal{H}_2$ in states $|\psi_1\rangle$ and $|\psi_2\rangle$ respectively is likewise written as $|\psi_1\rangle \otimes |\psi_2\rangle$ or just $|\psi_1\rangle|\psi_2\rangle$. Additionally we use $X^{\otimes n}$ to refer to the tensor product of n copies of X, where X can be anything for which tensor products make sense.

While we won't be concerned with measurement, according to the postulates of quantum mechanics we can measure a quantum state $|\psi\rangle$ according to some orthonormal basis $\{|b_i\rangle\}$. If $|\psi\rangle = \sum_i \alpha_i |b_i\rangle$, then the probability of obtaining outcome b_i is $|\alpha_i|^2$. This directly implies the postulate that the states of a quantum system are in fact unit vectors with respect to the Euclidean norm.

[1] $\mathcal{H}$ is used by convention – we could equivalently use $\mathbb{C}^d$

Quantum mechanics also postulates that the evolution of a closed system is unitary, meaning quantum states must evolve according to unitary operators $U : \mathcal{H} \to \mathcal{H}$. By unitary we mean that U is a linear operator on $\mathcal{H}$ such that $U^\dagger U = UU^\dagger = I$, i.e. U is invertible with $U^{-1} = U^\dagger$. Equivalently, U is a linear operator that preserves the Euclidean norm. We denote $U(d)$ to denote the set of unitary operators on a complex vector space of dimension d. For instance, the *Hadamard* gate

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

is an example of a unitary operator. We also distinguish the set $SU(d)$ of unitaries with determinant 1, i.e. the unitaries that are unique up to multiples of $e^{i\theta}$, as global phase does not affect measurement outcomes.

At this point we can view a correspondence between reversible and quantum computations. As the computational basis consists of all n bit strings, the basis states of a quantum system can be seen to be the set of classical states on the same number of bits. We might then consider the relation between reversible functions and unitary operators. In particular, any unitary $U \in U(2^n)$ that is a permutation operator in the computational basis implements some function $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$; since U is unitary, we also know that U is invertible and thus f is a reversible function. Likewise, any classical reversible function f can be computed on a quantum computer by a permutation $U : |x\rangle \mapsto |f(x)\rangle$ where $x \in \mathbb{F}_2^n$, since the (complex conjugate) transpose of a permutation matrix trivially gives its inverse.

2.3 The Quantum Circuit model

To provide a concrete model for quantum computation, we describe the *quantum circuit model*, one of the most prominent models of quantum computation (see [24] for a more detailed exposition).

The quantum circuit model is analogous to the classical or reversible circuit models, in that information – in this case, qubits – is carried on wires through gates which transform their state. For a system with n wires, the state space $\mathcal{H}$ has dimension 2^n, with computational basis states corresponding to the 2^n elements of $\mathbb{F}_2^n$. In accordance with the postulates of quantum mechanics, quantum gates are unitary operators on subsystems of $\mathcal{H}$, depending on which qubits the gate is applied to – more accurately, a gate is the tensor product of a unitary together with the identity operator (I) on the unaffected qubits. If

two adjacent gates are applied to non-overlapping sets of qubits, they can be written as a single tensor product and are said to be applied in *parallel*: $(g_1 \otimes I)(I \otimes g_2) = g_1 \otimes g_2$.

A circuit on n qubits is a finite sequence $i \mapsto U_i$ of gates applied in order from left to right to subsets of the n qubits, the effect of which is the functional composition $U_k \cdots U_2 U_1$ of the individual unitary operators corresponding to the gates. We use U_C to refer to the unitary computed by a circuit C so that we can refer to distinct circuits that compute the same unitary. Though we will mostly ignore measurements, a quantum circuit may also contain measurement operators over a given basis and wires carrying classical data emitted from such measurements.

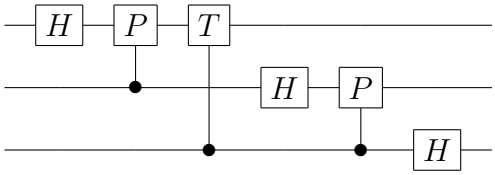


Figure 2.3: An example of a quantum circuit, implementing the quantum Fourier transform up to permutation of the outputs.

We define the *depth* of a quantum circuit as the maximum length of a path through the circuit. By path we mean a path in the directed acyclic graph representing the circuit, with nodes corresponding to the circuit's gates and edges corresponding to gate inputs/outputs. We will often refer to depths of specific gates in a circuit – e.g. T-depth – in which case we mean the maximum number of that gate in any path. As an example, the maximum path length in Figure 2.3 is 5.

A quantum circuit is typically expressed over a particular set of gates. We call a set of quantum gates $\mathcal{G}$ a *gate set* or *instruction set*, and say that C is a circuit over $\mathcal{G}$ if C is a quantum circuit containing only gates in $\mathcal{G}$; we denote the set of all such circuits $\langle \mathcal{G} \rangle$. We also define the closure of $\mathcal{G}$ over inversion as $\mathcal{G}^\dagger = \mathcal{G} \cup \{g^\dagger | g \in \mathcal{G}\}$ and all n-fold tensor products of the individual gates[2] as $\mathcal{G}_n$. We note that for any $\mathcal{G}$, $\mathcal{G}_n \subseteq U(2^n)$, and the notion of circuit depth corresponds to the number of gates in $\mathcal{G}_n$ used in the circuit. In fact, we see that the minimum depth of any circuit over $\mathcal{G}$ implementing $U \in U(2^n)$ is the minimum number of gates in $\mathcal{G}_n$ needed to implement U.

Lemma 2. *A unitary $U \in U(2^n)$ is implemented by a circuit of depth k over $\mathcal{G}$ if and only if $U = U_k \cdots U_2 U_1$ where $U_1, U_2, \ldots, U_k \in \mathcal{G}_n$.*

[2] n-fold tensor products of individual gates are commonly called *elementary transformations* [28]

2.3.1 Quantum gates

We now describe some of the common gates and groups of gates that we will use. For instance, the *Pauli gates*,

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

are commonly used to model error channels and are very important in the construction and analysis of quantum error correcting codes. We use $\mathcal{P}_n$ to refer to the set of n-fold tensor products of the Pauli gates.

Another important class of gates include the Hadamard (H), controlled-not ($CNOT$), and Phase (P) gates,

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad P = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}.$$

The Hadamard, CNOT, and Phase gates generate a group of unitaries called the *Clifford group*, denoted $\mathcal{C}$. The Clifford group on n qubits is equivalent to the normalizer of the Pauli group $\mathcal{P}_n$, $\mathcal{C}_n = \{U \in U(2^n) | U\mathcal{P}_n U^{-1} \subseteq \mathcal{P}_n\}$, and as a consequence contains the Pauli group as well. Remarkably, Gottesman and Knill [29] proved that any circuit composed of Clifford group gates can be efficiently simulated on a classical computer, and so such circuits clearly cannot perform all quantum computations.

We will also commonly refer to the previously seen Toffoli (TOF) gate, and the $\pi/8$ gate (T):

$$TOF = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \qquad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}.$$

Both the Toffoli and $\pi/8$ gates lie outside of the Clifford group, which make them useful for general quantum computing.

We note that each of the above gates are expressed as matrices over the ring $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$, or equivalently the ring of *dyadic fractions* $\mathbb{Z}\left[\frac{1}{2}\right]$ extended with $\omega = e^{i\pi/4}$. Recent results have shown that any unitary over $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$ can in fact be implemented over $\mathcal{C} \cup \{T\}$ [30, 31].

The notion of *controlled* quantum gates will also be important throughout this thesis. We define a *controlled-U* gate as $\Lambda(U) : |x\rangle|\psi\rangle \mapsto |x\rangle U^x|\psi\rangle$, where $x \in \{0, 1\}$; in a circuit, we typically use a solid dot to represent a *control* with a line to the controlled gate U. We can write a unitary matrix for a controlled-U gate as $\Lambda(U) = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U$:

$$(|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U)|x\rangle|\psi\rangle = \langle 0|x\rangle|0\rangle \otimes |\psi\rangle + \langle 1|x\rangle|1\rangle \otimes U|\psi\rangle = |x\rangle U^x|\psi\rangle.$$

We can further define Λ_m recursively as $\Lambda\left(\Lambda_{m-1}(U)\right)$, where m gives the number of controls on U. The previously shown gates $CNOT$ and TOF are in fact controlled X gates:

$$CNOT = \Lambda(X), TOF = \Lambda_2(X).$$

One important observation about the nature of controlled operations is that if there exists a circuit over $\mathcal{G}$ implementing $\Lambda(g)$ for each $g \in \mathcal{G}$, then for any unitary U implemented by a circuit over $\mathcal{G}$, $\Lambda_m(U)$ can also be implemented over $\mathcal{G}$. This is a result of the following lemma:

Lemma 3. *Suppose $U = U_k \cdots U_2 U_1$ for some unitaries $U, U_1, U_2, \ldots, U_k \in U(2^n)$. Then*

$$\Lambda(U) = \Lambda(U_k) \cdots \Lambda(U_2)\Lambda(U_1).$$

A simple method for constructing such a circuit then proceeds by replacing every gate g with a circuit for $\Lambda(g)$.

2.3.2 Universal gate sets

As in the classical case, we need to know which gate sets can be used to implement the set of quantum computations, $U(2^n)$. An immediate observation is that since there are uncountably many unitaries on any number of qubits, any gate set that can implement all quantum computations is necessarily uncountable in size. It turns out that $U(2) \cup \{CNOT\}$ can implement any quantum computation [1], though constructing such a gate set fault tolerantly would however be extremely unlikely. Instead we consider the possibility of *approximating* some quantum operations.

Definition 3. For unitaries $U, V \in U(2^n)$, U is an ϵ-approximation of V if $||U - V|| \leq \epsilon$ where

$$||U - V|| = \max_{|\psi\rangle} ||(U - V)|\psi\rangle|| = \sqrt{\langle\psi|(U - V)^\dagger(U - V)|\psi\rangle}$$

is the operator norm.

The operator norm is used as it corresponds closely to the maximum difference in the probability of obtaining a particular measurement outcome between $U|\psi\rangle$ and $V|\psi\rangle$. In general we may want to use different norms to define the error in approximations.

Definition 4. A gate set $\mathcal{G}$ is *universal* for quantum computation if for any unitary U and $\epsilon > 0$, there exists a circuit C over $\mathcal{G}$ such that U_C is an ϵ-approximation of U.

As a well-known result, $\mathcal{C}_n$ along with any unitary $U \notin \mathcal{C}_n$ is universal [32]. Since $\{H, P, CNOT\}$ generates the Clifford group up to global phase and $P = T^2$, the gate set consisting of $\{H, CNOT, T\}$ is universal for quantum computing. The set $\{H, TOF\}$ is also known to be universal, though it is not as common in fault-tolerant models.

A natural question to ask is how many gates from a universal set are needed to approximate a given unitary U to a desired accuracy – if the approximation is inefficient, any advantage of a quantum algorithm could be negated by the overhead for approximation. Fortunately, it turns out that only an overhead poly-logarithmic in $1/\epsilon$ is required to approximate a given unitary, a result known as the *Solovay-Kitaev theorem* [33].

Theorem 1. *(Solovay-Kitaev)*

Suppose $\mathcal{G}$ is a finite gate set and is universal for single qubit computations. Then for any single qubit unitary U and $\epsilon > 0$, there is a circuit over $\mathcal{G}^\dagger$ with length $O(\log^c(1/\epsilon))$ that is an ϵ-approximation of U, where c is a positive constant.

2.4 Fault Tolerance

The mathematical models previously described have assumed that computations can be performed exactly and without error. However, these models are idealized abstractions of physical and particularly imperfect processes. In any realistic device individual gates and operations may fail without warning or perform the wrong operation. At this point we have two choices: ignore the errors and hope they don't accumulate too much, or try to reduce the number of errors by shielding the information from the imperfect physical processes.

In the classical world, the first approach is widely used, as the error rates on most modern logic gates are insignificant for most computations of interest. For more fragile classical processes such as communication over noisy channels and data storage, the information is commonly protected with error correction. Similarly, quantum processes are very error-prone and individual qubits decohere even when they are not being manipulated, so it is hard to imagine large-scale quantum computations without some method for mitigating these errors.

A common approach is to use an *error-correcting code (ECC)* to encode the state of a group of logical qubits into many physical qubits, then perform logical gates directly on the encoded qubits by encoding the gates in a *fault-tolerant* manner. For the purposes of this thesis, we will primarily be concerned with the latter idea, as the construction of encoded gates informs our gate sets and optimization criteria – for this reason, the description will be cursory.

Error correcting codes, both classical and quantum, encode the state of a logical k-bit system in the state of a physical n-bit system by using *codewords*. The codewords rely on storing redundant information, so that if one bit fails the remaining bits can be used to determine the encoded information. In classical error correction, this can be accomplished by simply repeating the encoded information, though many more efficient codes exist. As an example, the three bit code encodes a logical 0 state as 000, and the logical 1 state as 111 – if at most one error occurs on a physical bit, the logical bit can be retrieved by taking the majority of all three bits.

For quantum error correcting codes, arbitrary quantum states cannot be copied and repeated to create the encoded state, a consequence of the *no-cloning* theorem [34]. Instead, quantum error correcting codes assign codewords to the computational basis states of the logical system. In this way, classical error correcting codes can double as quantum error correcting codes, such as the three bit code which, in a quantum system, protects against one *bit flip* error. Indeed, a large class of quantum error correcting codes, called *CSS* codes, are built by combining classical codes to protect against both bit flip and *phase flip* errors.

Another common class of quantum error correcting codes, known as *stabilizer* codes, define the code space as the set of states that remain unchanged under computations in some Abelian group $S \subseteq \mathcal{P}_n$. Given $S \subseteq \mathcal{P}_n$, the code space generated by S is the set $T = \{|\psi\rangle \in \mathcal{H} \mid M|\psi\rangle = |\psi\rangle \ \forall M \in S\}$, and S is called the stabilizer of T. As an example, the Steane code is a stabilizer code, with the following stabilizer generators:

$$\begin{Bmatrix} I \otimes I \otimes I \otimes X \otimes X \otimes X \otimes X, & I \otimes I \otimes I \otimes Z \otimes Z \otimes Z \otimes Z, \\ I \otimes X \otimes X \otimes I \otimes I \otimes X \otimes X, & I \otimes Z \otimes Z \otimes I \otimes I \otimes Z \otimes Z, \\ X \otimes I \otimes X \otimes I \otimes X \otimes I \otimes X, & Z \otimes I \otimes Z \otimes I \otimes Z \otimes I \otimes Z \end{Bmatrix}.$$

Additionally, the class of stabilizer codes contains all CSS codes.

As the process of encoding and decoding codewords every time a gate is applied would defeat the purpose of quantum error correction, logical gates must also be encoded, possibly by performing some quantum circuit. However, doing so could cause errors by virtue of the fact that an error on one physical qubit may propagate to errors on other physical qubits. Consider, for instance, the $CNOT$ gate: $CNOT \cdot (X \otimes I)|x_1\rangle|x_2\rangle = |\neg x_1\rangle|\neg x_1 \oplus x_2\rangle = (X \otimes X) \cdot CNOT|x_1\rangle|x_2\rangle$. The single X error *before* the $CNOT$ gate becomes two X errors *after*; if the error correcting code in use can only correct one Pauli error at a time, the originally correctable error can no longer be corrected after the $CNOT$ gate is applied. While errors are generally detected and corrected after every operation, the possibility of an error occurring between the last error correction and the application of a gate cannot be ruled out, and so logical gates must avoid the problem of error propagation.

The simplest way to avoid error propagation is to perform logical gates *transversally*, which means to apply an encoded U gate, a physical U gate is applied to each qubit in the physical system (or pairs of equivalent physical qubits in the case of two qubit unitaries). Errors thus cannot propagate within the same encoded qubit when applying logical gates, since no two encoding qubits interact with one another. Unfortunately, for an arbitrary stabilizer code a full universal gate set cannot be built transversally[3] [27], and so other techniques must be used.

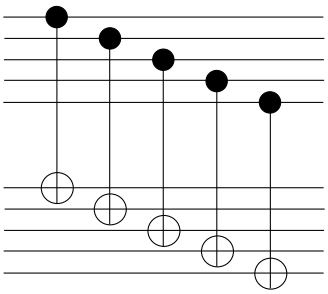


Figure 2.4: Transversal $CNOT$ between two qubits encoded in a 5-qubit code.

In the case when a logical U gate cannot be constructed by applying U transversally, a technique known as *gate teleportation* [36] is commonly used together with *state distillation* [37], a technique for distilling a resource state with high fidelity. As gate teleportation requires measurement, which is typically much slower than physical gates in quantum computer implementations, it is already a more costly procedure than transversal gates,

[3]Codes can be defined that admit a universal transversal gate set [35], but such codes necessarily go beyond basic stabilizer code theory.

both in terms of time and space. When state distillation is added the effect is compounded, as each round of state distillation typically requires concatenation of error correcting codes along with complicated logical circuits. These so called "ancilla factories" can require space-time volume that is orders of magnitude larger than the resources required for a transversal gate.

It turns out that most of the common CSS codes admit a transversal set of generators of the Clifford group [38]. While not strictly speaking transversal, the popular surface code also has an efficient encoded set of Clifford group generators [10]. Together with results from measurement-based quantum computing showing that any Clifford group circuit can be parallelized to constant depth [55], in most fault tolerant schemes it follows that Clifford group operations are extremely fast and space efficient. The non-Clifford operation in a universal gate set, typically the T gate as it admits reasonable state distillation protocols, is then the bottleneck in fault tolerant computations. Reducing the number of non-Clifford operations in a circuit would reduce the number of ancilla states that require preparation, at the same time reducing the fidelity required of each resource state. Likewise, placing T-gates in parallel reduces the amount of time spent waiting for gate teleportation, since multiple T gates can be teleported at the same time. Indeed, Fowler [22] shows how to perform fault-tolerant computations in time proportional to one round of measurement per layer of T-gates, and so the T-depth directly affects circuit run-times. For these reasons, we focus on the topic of T-count and depth optimization for much of this thesis.

Chapter 3

Quantum Circuit Optimization

In this section we formally introduce the problem of quantum circuit optimization and review the state of the art of the field.

The natural intuition is that to optimize a circuit we want to find the best circuit, according to some criteria, implementing the same functionality. We can formalize this idea, as below, but given that the intuition is clear we refrain from referring to quantum circuit optimization in such an abstruse way throughout the remainder of this thesis.

Definition 5. (Quantum Circuit Optimization)

Given a circuit $C \in \langle \mathcal{G} \rangle$ and cost function $c : \langle \mathcal{G} \rangle \to \mathbb{R}$, find some $C' \in \langle \mathcal{G} \rangle$ such that $U_{C'} = U_C$ and for any other circuit $C_0 \in \langle \mathcal{G} \rangle$ with $U_{C_0} = U_C$, $c(C') \leq c(C_0)$.

A closely related problem is that of quantum circuit *synthesis*, which is concerned with constructing a circuit implementing or approximating a given unitary. Again, we provide a formal definition merely for completeness.

Definition 6. (Quantum Circuit Synthesis)

Given a unitary $U \in U(2^n)$, gate set $\mathcal{G}$ and precision $\epsilon \geq 0$, find a circuit $C \in \langle \mathcal{G} \rangle$ such that $||U - U_C|| \leq \epsilon$. If $\epsilon = 0$ then the problem is referred to as *exact* synthesis.

The reason we are interested is circuit synthesis is that synthesis algorithms can be designed in such a way as to produce circuits optimal in some cost. Such algorithms can then be used to optimize a circuit C by synthesizing an optimal circuit for U_C, assuming U_C can be computed in reasonable time. Given that most synthesis algorithms work for a

few qubits at most, for circuits that can be optimized by such *re-synthesis* computing U_C should be fast.

Sometimes we want to relax the optimization and synthesis problems. In particular, we can consider situations where we only need to produce a circuit equivalent up to a global phase, i.e. we want C' such that $U_{C'} = e^{i\theta}U_C$ for some θ. We can also consider situations where ancillas may be used to optimize a given cost; in this case we require that C' implements $U_{C'}$ such that for any $|\psi\rangle \in \mathcal{H}$, $|0\rangle^{\otimes m} \otimes (U_C|\psi\rangle) = U_{C'}(|0\rangle^{\otimes m}|\psi\rangle)$. If the number of ancillas that are available is unbounded, we denote $m = \infty$.

The most common families of cost functions on quantum circuits involve gate counts and depth-related costs – in the most basic case, both types of costs have clear motivation in reducing the circuit time and complexity (also indirectly affecting error rates). A weighted gate count assigns a weight to each gate in $\mathcal{G}$, with the cost of a circuit equal to the sum of the cost of each gate. In the case where each gate has weight 1, we just call c a gate count. One particular weighted gate count that we will be concerned with, called the T-count and defined as $c(T) = 1$, $c(g) = 0\ \forall g \neq T$, arises from the consideration of fault tolerant gate constructions where T gates are expensive procedures. Circuit depth, defined earlier, is our other main optimization criteria, and we distinguish T-depth (recall: the maximum number of T gates in a path through the circuit) as an important depth quantity for optimization.

Another class of quantum circuit optimization problems arises when information about the physical architecture is included in the circuits. These problems are typically called *placement* problems, though in many cases (e.g. 2D nearest neighbour) they can be described as circuit optimization problems with additional constraints. In this thesis we do not consider such problems, though we direct the interested reader to some relevant results [40, 17, 41].

3.1 State of the Art

We review some of the previous results in quantum circuit optimization. As many quantum circuits have large sections that are strictly reversible circuits, for instance the reversible arithmetic in Shor's factoring algorithm [25], it is useful to lift reversible circuit optimizations to the quantum domain. Algorithms designed for *optimal* synthesis are also included in the discussion, though algorithms not covered here, such as [42], may also relevant from an optimization point of view in some restricted cases.

3.1.1 Exhaustive search

A large class of optimal synthesis algorithms work by exhaustively enumerating all circuits and returning the lowest cost circuit implementing the desired computation; this technique is commonly called exhaustive or brute-force searching. This method is particularly popular in the circuit synthesis community as a way of optimally compiling small commonly used gates or functions to the target gate set, and in these small instances it can be very effective.

Typical exhaustive search algorithms generate circuits in increasing size or depth, combined with heuristics to reduce the number of candidates to be generated. Such a search is called a *breadth-first* search, as the space of circuits over a gate set $\mathcal{G}$ can be viewed as a tree where each branch corresponds to the next gate in the circuit. Breadth-first searches are thus naturally tuned to optimize size and depth, though they can be used indirectly to optimize other costs. Such searches also typically involve the computation of some *database* of circuits, where the database is computed once and loaded whenever a new circuit is needed. If the database provides efficient lookups, circuits that are already in the database can be synthesized efficiently.

This technique has been used extensively in the optimal synthesis of reversible functions. Shende *et al.* [43] used a breadth-first search to synthesize all minimal gate count circuits for reversible functions on 3 bits, over the gate set $\{X, CNOT, TOF\}$. Their algorithm builds databases of optimal circuits up to a maximum gate count, and they devise a recursive scheme for increasing the search depth where they multiply the target function with some known permutation and synthesize the resulting function. Maslov and Miller [44] later expanded on those techniques and adapted them to synthesize all reversible functions on 3 bits over the quantum gate set $\{X, CNOT, \Lambda(V), \Lambda(V^\dagger)\}$. In particular, they reduced the search space by only storing functions the are unique up to relabeling of the inputs and outputs. Their approach, however, did not allow control qubits to carry *quantum* values, e.g. values resulting from a $\Lambda(V)$ or $\Lambda(V^\dagger)$ gate, and the resulting circuits are thus not provably optimal. Golubitsky and Maslov [12] further expanded this technique by removing inverse functions from the circuit databases, which along with other improvements allowed them to synthesize all optimal 4 bit reversible functions over $\{\Lambda_i(X) | 0 \leq i \leq 3\}$. An important consideration in these algorithms is the exact representation of functions and circuits; they typically use compact binary representations of the functions which allow easy searching over databases and low storage requirements [12].

While breadth-first searches are common in reversible circuit synthesis, the lack of efficient representations of unitaries make such approaches more difficult. Fowler [45] avoided this problem by performing the breadth-first search directly, i.e. without the use of a precomputed database implementing efficient lookup. His algorithm finds optimal gate

count ϵ-approximations of unitaries by performing a breadth first search and keeping track of the set of unique gate sequences up to a given depth. New circuits are then compared against the list of unique circuits and the circuit can be removed from consideration if some subcircuit is not unique. Furthermore, he shows how to skip to the next potentially unique circuit once such a circuit is found. Despite the expensive process of checking each subcircuit, the method proved effective for single qubit circuits over $C_1 \cup \{T, T^\dagger\}$, and until recently was the primary method for computing optimal gate count circuits.

More recently, Bocharov and Svore [19] developed a depth-optimal canonical form for single qubit circuits over the gate set $\{H, T\}$. They show that databases of canonical circuits can be built efficiently, compared to the costly procedure of generating each gate sequence and comparing it with the current database. Such databases are also claimed to use less memory than databases storing unitaries. While the resulting databases can be searched for ϵ-approximations efficiently, $O(\epsilon k 2^k)$ for databases consisting of canonical circuits with T-count at most k, it requires searching for each element of the double coset $C_1 U C_1$ where U is the target unitary. Furthermore, the canonical form applies only to single-qubit circuits over $\{H, T\}$, and thus cannot be used to compute minimal gate count or minimal depth multi-qubit quantum circuits.

3.1.2 Algorithmic synthesis

In some restricted gate sets and cost metrics, optimal circuits can be synthesized directly, rather than exhaustively searching for a circuit. We review some of the known techniques here.

In the reversible circuit domain, SAT solvers have been used to produce optimal gate count circuits. Hung *et al.* [13] use multiple-valued logic to synthesize reversible functions over the gate set $\{X, CNOT, \Lambda(V), \Lambda(V^\dagger)\}$ as in [44] – again, control bits are restricted to Boolean values. The multiple-valued logic synthesis is then performed by bounded model checking with a SAT solver, a technique where a transition system is encoded in propositional logic, then paths from an input state to an output state are found by solving SAT instances for increasing path lengths. Along similar lines, Große *et al.* [46] formulate the problem of reversible logic synthesis over $\{\Lambda_i(X) | i \geq 0\}$ as a sequence Boolean satisfiability problems, which are then solved via SAT solvers. As both algorithms require Boolean satisfiability to be solved, their complexity is effectively no better than exhaustive search – furthermore, exhaustive search has been shown to be more efficient in both cases [44, 12], albeit with greater memory usage.

Non-search based synthesis has occasionally been used in quantum computing. In

particular, Kliuchnikov *et al.* [30] describe an algorithm for decomposing an arbitrary single-qubit unitary over the ring $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$ into a circuit over $\{H, T\}$. Their method reduces single-qubit unitary synthesis to computing a circuit preparing a particular state, then they can recursively reduce the complexity of the state by applying HT^k for some particular choice of $k \in \{0, 1, 2, 3\}$. Furthermore, they prove that the number of T and H gates in the resulting circuit is optimal, effectively solving T-count optimization for single qubit circuits over $\{H, T\}$. Giles and Selinger [31] further extended this result to a synthesis method for multi-qubit unitaries over $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$, though the construction is non-optimal in terms of either gate counts or depth.

Other examples of synthesis in quantum computing focus on approximating unitaries outside of the $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$ ring. The classic example is the Solovay-Kitaev algorithm [47], a recursive algorithm that increases the accuracy of each successive approximation U_{n-1} by approximating unitaries V, W such that the residue $UU_{n-1}^{\dagger}$ is equal to the group commutator $VWV^{\dagger}W^{\dagger}$ – furthermore, the unitaries V and W must be *balanced*, in that they are both within a particular distance from the identity. While not optimal, the algorithm produces single-qubit circuits of size $O(\log^{3.97}(1/\epsilon))$, which will be useful for comparing to more recent methods. Specifically, Selinger [48] proved a worst-case lower bound of $4\log(1/\epsilon) - 9$ T gates to implement a single-qubit z-axis rotation over the Clifford group and T gates, and at the same time developed an algorithm that approximates a single-qubit z-axis rotation using $4\log(1/\epsilon) + 11$ T gates. Kliuchnikov *et al.* [49] describes another approximation algorithm that appears to achieve T-count of $3.21\log(1/\epsilon) - 6.93$ on average for ϵ-approximations of z rotations. Both algorithms proceed by constructing a unitary over $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$ approximating the target unitary, then decomposing the approximation optimally using [30]. However, these methods are limited to single-qubit unitaries, and their usefulness regarding circuit optimization is thus limited to circuits containing arbitrary single-qubit rotations.

3.1.3 Local rewriting

Very few methods exist for optimizing gate count in large quantum circuits, with the majority of the previous results being devoted to local optimization techniques. Such techniques can be loosely categorized as peephole optimizations or template-based optimization, both of which involve the replacement of subcircuits with smaller (in gate count or depth) equivalent circuits.

The former grew out of compiler terminology where optimizations are performed on

just a few lines of code, as if viewed through a "peephole". In relation to circuit optimization, peephole optimization commonly involves selecting subcircuits then synthesizing an optimal circuit computing the same function. Such a technique is useful in reversible circuit optimization, where every function on 4 bits can be synthesized quickly [12]. Prasad *et al.* [50] develop the technique of peephole optimization and apply it to large reversible circuits. Their results show on average 25% reduction in circuit size for random circuits, and circuits with up to a thousand gates could be optimized quickly.

Templates, while a theoretically similar technique, use circuit identities (e.g. $C = C'$) to transform the circuit, either in a directed way or to replace subcircuits with smaller equivalent circuits. Miller *et al* [51] introduced templates for the optimization of reversible circuits. In their algorithm, they search for a target template within the algorithm by choosing an initial gate, then use commutation rules to try and construct the remainder of the template; Maslov *et al* [52] later expanded on the theory of this technique and introduced a heuristic modification that produced better results by reducing the number of control bits.

Template optimization has also had success in being applied to general quantum circuits, particularly by Maslov *et al.* [15] to the $\{X, CNOT, \Lambda(V), \Lambda(V^\dagger)\}$ gate set – they show gate count reductions by over 30% on average for circuits with up to 21 qubits. Sedlák and Plesch [53] also used circuit identities to shift gates as far left as possible, then remove any gates that cancel. While their experimental results, having been applied to $CNOT$ and arbitrary single qubit rotations, are superseded by optimal unitary decompositions [42], they provide further indication for the effectiveness of this method.

3.1.4 Parallelization algorithms

The previously described algorithms have mostly focused on reducing gate counts, and if they reduced depth, it was mostly a side effect of reducing gate count. However, as in classical computing when there are many computational resources available it can often make sense to increase complexity in order to parallelize operations to make use of the extra resources. We review a few of the algorithms for the parallelization of quantum circuits in this way.

Moore and Nilsson [54] first defined the notion of **QNC**, a class of quantum computations that can be parallelized to poly-logarithmic depth using a polynomial number of ancillas. While they do not provide an actual algorithm for performing parallelization, they prove the parallalizeability of a number of gate sets, including $\{CNOT\}$, $\{H\} \cup \Lambda(\mathcal{P})$, and encoding/decoding circuits for stabilizer codes. They also provide a large number

of transformation rules that could be used to define a parallelization algorithm. Other similar results have shown that Clifford group [55] and diagonal unitaries [56] can be parallelized to constant depth using measurements, though we will primarily be concerned with measurement-free circuits.

More recently, transformations to and from the measurement-based quantum computing model have been used to parallelize quantum circuits. Broadbent and Kashefi [39] develop an algorithm for the translation of quantum circuits to a *pattern* (a computation in the measurement-based model) which adds a number of additional ancillas linear in the number of gates. The resulting pattern is then optimized by applying rewriting rules of the measurement calculus (known as standardization and signal shifting), then the final pattern is transformed back into a circuit. However, the construction requires that the circuits are written over the gate set $\left\{\Lambda(Z), J(\alpha) = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & e^{1\alpha} \\ 1 & e^{-i\alpha}\end{pmatrix}\right\}$ which does not appear to have fault-tolerant implementations in the common schemes, and requires the availability of ancillas. Dias da Silva *et al.* [16] later expanded this algorithm with new optimizations, and a set of rewrite rules for removing the qubits added during the transformation.

Chapter 4

Meet-in-the-Middle: a search-based synthesis algorithm

In this chapter we are primarily concerned with the problem of synthesizing a *depth-optimal* quantum circuit over an arbitrary gate set and with arbitrarily many qubits. The material presented here is largely based on [18].

Quantum algorithms are typically described at a high level of abstraction and feature many common high-level operations, such as the Toffoli gate or integer multiplication. On the other hand, fault-tolerant quantum computation requires specifying circuits in terms of a few encoded operations – the high-level operations then have to be *compiled* down to those low-level encoded gates. In the case of common operations like complex gates, circuits optimizing some particular cost, for instance depth, can be precomputed and substituted in during the compilation process. In particular, circuit depth determines both run-time and by extension error rates, hence circuit depth should be the primary optimization criteria. However, as evidenced in Chapter 3, algorithms performing this kind of synthesis are lacking in quantum computing; while many methods exist for reversible computation, depth-optimal synthesis of quantum circuits has only been performed on single-qubit circuits [45, 19, 30].

To fill this gap, we have developed a practical algorithm for computing depth-optimal fault tolerant circuits. In particular, we describe an algorithm and heuristics for speeding up the brute-force search for minimal depth quantum circuits, given a target unitary. Exhaustive search can in many circumstances be a viable optimization method, particularly when more efficient methods are either non-existent or fail to produce quality results. When combined with heuristics for reducing the size of the search space, the brute force search

can be an effective tool for solving such problems, especially in small instances when the best possible result is needed. Such techniques have been used very effectively for reversible circuit synthesis [43, 44, 12], and have shown better run-times than algorithmic synthesis.

The algorithm, called the *meet-in-the-middle* algorithm, reduces the time and space complexity of the brute-force search by roughly a square root. In allowing minimal depth circuits to be computed using only circuits of at most half the minimal depth, the algorithm generates just a square root of the circuits that would be generated in a naïve exhaustive search. Compared to less naïve approaches, our algorithm provides a space-time trade-off which mitigates the major bottleneck of space consumption in such approaches. Additionally, we show that the meet-in-the-middle algorithm, while designed with depth-optimal synthesis in mind, is very flexible and can be used to efficiently optimize different criteria as well. To further improve the performance, we also detail search space reductions that drastically reduce the number of generated circuits.

The methods developed here were largely designed for *exact* synthesis of quantum circuits – recall that exact synthesis produces circuits that implement the target unitary with no error. As a result, the algorithm is particularly suited for re-synthesis, since an initial decomposition ensures there exists an optimal circuit implementing the same unitary over the same basis, and individual subcircuits can be re-synthesized without affecting the overall accuracy of a circuit. The algorithm can, however, can be modified to synthesize circuits approximating a target unitary; we later develop such a modification and use it to achieve faster depth-optimal circuit approximations than previously known multi-qubit algorithms.

4.1 The Meet-in-the-middle algorithm

In this section we give a high level description of the meet-in-the-middle algorithm for computing depth-optimal quantum circuits over a gate set $\mathcal{G}$. The algorithm is motivated by the observation that any circuit of minimal depth can be divided into two circuits (or in general, any number of sub-circuits), each of which is a minimal depth circuit – this idea has been used before in reversible circuit synthesis [43, 12]. As a result, a circuit with minimal depth l can be found by finding its two halves of depth at most $\lceil l/2 \rceil$. We show below that this problem reduces to finding elements in the intersection of two sets of unitaries.

Lemma 4. *Let $S_i \subseteq U(2^n)$ be the set of all unitaries implementable in depth i over the gate set $\mathcal{G}$. Given a unitary U, there exists a circuit over $\mathcal{G}$ of depth l implementing U if and only if $S^\dagger_{\lfloor l/2 \rfloor} U \cap S_{\lceil l/2 \rceil} \neq \emptyset$ where $S_i^\dagger = \{U^\dagger | U \in S_i\}$.*

Proof. Suppose some depth l circuit C implements U. We divide C into two circuits of depth $\lfloor l/2 \rfloor$ and $\lceil l/2 \rceil$, implementing unitaries $V \in S_{\lfloor l/2 \rfloor}$ and $W \in S_{\lceil l/2 \rceil}$ respectively, where $VW = U$. Since we know $W = V^\dagger U \in S^\dagger_{\lfloor l/2 \rfloor} U$, we can observe that $W \in S^\dagger_{\lfloor l/2 \rfloor} U \cap S_{\lceil l/2 \rceil}$, as required.

Suppose instead $S^\dagger_{\lfloor l/2 \rfloor} U \cap S_{\lceil l/2 \rceil} \neq \emptyset$. We see that there exists some $W \in S^\dagger_{\lfloor l/2 \rfloor} U \cap S_{\lceil l/2 \rceil}$, and moreover by definition $W = V^\dagger U$ for some $V^\dagger \in S^\dagger_{\lfloor l/2 \rfloor}$. Since $W \in S_{\lceil l/2 \rceil}$, $VW = U$ is implementable by some circuit of depth $\lfloor l/2 \rfloor + \lceil l/2 \rceil = l/2$, thus completing the proof. □

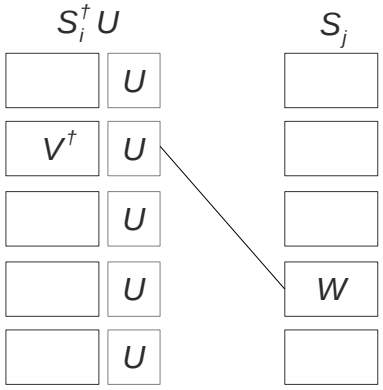


Figure 4.1: For each $V \in S_i$ we construct $W = V^\dagger U$ and search for W in S_j.

Using this lemma, we can describe a simple algorithm that determines whether there exists a circuit over $\mathcal{G}$ of depth at most l implementing unitary U, and if so return a minimum depth circuit implementing U. Given an instruction set $\mathcal{G}$ and unitary U, we repeatedly generate circuits of increasing depth, then use them to search for circuits implementing U with up to twice the depth. Specifically, at each step we generate all depth i circuits S_i by extending the depth $i-1$ circuits with one more level of depth, then we compute the sets $S^\dagger_{i-1} U$ and $S^\dagger_i U$ and see if there are any collisions with S_i (Figure 4.1). By Lemma 4, there exists a circuit of depth $2i-1$ or $2i$ implementing U if and only if $S^\dagger_{i-1} U \cap S_i \neq \emptyset$ or $S^\dagger_i U \cap S_i \neq \emptyset$, respectively, so the algorithm terminates at the smallest depth less than or equal to l for which there exists a circuit implementing U. In the case where U can be implemented in depth at most l, the algorithm returns one such circuit of minimal depth. The algorithm is given as pseudo-code in Algorithm 1.

As this algorithm requires the computation of $S^\dagger_i U$, it relies on the ability to quickly multiply unitaries. While this is not practical for large numbers of qubits, for the small

Algorithm 1 Meet-in-the-middle algorithm.

function MITM($\mathcal{G}$, U, l)
  $S_0 := \{I\}$
  $i := 1$
  for $i \leq \lceil l/2 \rceil$ **do**
    $S_i := \mathcal{G}_n S_{i-1}$
    if $S_{i-1}^\dagger U \cap S_i \neq \emptyset$ **then**
      return any circuit VW s.t.
        $V \in S_{i-1}, W \in S_i, V^\dagger U = W$
    else if $S_i^\dagger U \cap S_i \neq \emptyset$ **then**
      return any circuit VW s.t.
        $V, W \in S_i, V^\dagger U = W$
    end if
    $i := i + 1$
  end for
end function

numbers of qubits we're interested in matrix multiplication is sufficiently fast.

Furthermore, for this algorithm to be advantageous, we need a way of efficiently finding $S_i^\dagger U \cap S_j$. Fortunately, we can efficiently find the intersection by imposing a strict lexicographic ordering on unitaries – as a simple example two unitary matrices can be ordered according to the first element at which they differ. The set S_i can then be sorted with respect to this ordering in $O\left(|S_i| \log(|S_i|)\right)$ time, so that searching for an element of $S_{i-1}^\dagger U$ or $S_i^\dagger U$ takes time $O\left(\log(|S_i|)\right)$ – in practice, we would maintain S_i in a sorted data structure, rather than sort it at search time.

To achieve a (time) complexity bound we can note that $|S_i| \leq |\mathcal{G}_n|^i$, so that the ith iteration thus takes time bounded above by $3|\mathcal{G}_n|^i \log(|\mathcal{G}_n|^i)$. Since $\sum_{i=1}^{\lceil l/2 \rceil} |\mathcal{G}_n|^i \log(|\mathcal{G}_n|^i) \leq \sum_{i=1}^{\lceil l/2 \rceil} |\mathcal{G}_n|^i \log(|\mathcal{G}_n|^{\lceil l/2 \rceil})$ and $\sum_{i=1}^{\lceil l/2 \rceil} |\mathcal{G}_n|^i \leq |\mathcal{G}_n|^{\lceil l/2 \rceil}\left(1 + \frac{1}{|\mathcal{G}_n|^{\lceil l/2 \rceil} - 1}\right)$, we thus see that the algorithm runs in $O\left(|\mathcal{G}_n|^{\lceil l/2 \rceil} \log(|\mathcal{G}_n|^{\lceil l/2 \rceil})\right)$ time. It can also be noted that $|\mathcal{G}_n| \in O(|\mathcal{G}|^n)$, so the run-time is in $O\left(|\mathcal{G}|^{\lceil n \cdot l/2 \rceil} \log(|\mathcal{G}|^{\lceil n \cdot l/2 \rceil})\right)$, rather than $O(|\mathcal{G}|^{n \cdot l})$ as in the case of naïve searching.

We can achieve better on average by using a *hash table* instead. By hashing the entries of the circuit databases a given unitary can be found in time $O(1)$ on average, leading to an average time complexity of $O\left(|S_i|\right)$ to test the emptiness of $S_i^\dagger U \cap S_j$.

4.2 Search space reduction

To make the meet-in-the-middle algorithm practical for reasonable sized circuits, effort must be made to reduce the size of the search space. In this section we describe some reductions to the search space and how they can be combined with the meet-in-the-middle algorithm; the technique we use may be called "pruning" the search tree, as our goal is to remove redundant circuits from the databases S_i, in effect removing further branches from being explored.

We begin by noting that if a given unitary U has an implementation over $\mathcal{G}$ with depth i, then any simultaneous permutation of its input and output qubits can also be implemented in depth i simply by changing which qubits each individual gate acts on. Likewise, if $\mathcal{G}$ is closed under inversion, i.e. $\mathcal{G} = \mathcal{G}^\dagger$, then $U^\dagger$ also admits a depth i implementation, as if $U = U_i \cdots U_2 U_1$ with $U_1, U_2, \ldots, U_i \in \mathcal{G}_n$, then $U_1^\dagger, U_2^\dagger, \ldots, U_i^\dagger \in \mathcal{G}_n$ and so $U^\dagger = (U_i \cdots U_2 U_1)^\dagger = U_1^\dagger U_2^\dagger \ldots U_i^\dagger$ is a depth i implementation. These two observations imply that, given a gate set closed under inversion, we can restrict our attention to unitaries that are unique up to input/output relabeling and inversion. As we don't care about global phase in most cases, we can additionally restrict circuit databases to unitaries that are unique up to global phase factors.

More formally, we define an equivalence relation $\sim$ on unitaries where $U \sim V$ if and only if U is equal to V up to relabeling of the qubits, inversion, or global phase factors. This equivalence relation defines the equivalence class of a unitary $U \in U(2^n)$, denoted $[U]$, as $\{V \in U(2^n) | U \sim V\}$ – given a gate set closed under inversion, each unitary in a given equivalence class has the same minimal circuit depth, up to global phase. We then focus only on equivalence class *representatives*, rather than unitaries themselves, and in particular only store a circuit for each representative in a given set S_i. To do so, we define a canonical representative for each unitary equivalence class, then when a new circuit is generated we find the unitary representative and determine whether a circuit implementing it is already known.

We can define a canonical representative for each unitary equivalence class by lexicographically ordering unitaries and choosing the smallest unitary as the representative. Since relabeling of the qubits corresponds to simultaneous row and column permutations of the unitary matrix and the inverse of a unitary is given by its conjugate transpose, given an n qubit unitary U, all $2n!$ permutations and inversions of U can be generated and the minimum can be found in $O(n!)$ time. For the small instances of n that the meet-in-the-middle algorithm is designed for, the $O(n!)$ overhead to compute a canonical unitary is small, while the reducing the search space by a factor of $2n!$ reduces both space and time usage significantly.

Choosing a canonical unitary up to global phase is more difficult in general. In the case when the gate set contains only unitaries written over the ring $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$, we can generate each possible global phase factor in the equivalence class. In particular, $e^{i\theta} \in \mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$ if and only if $\theta = \frac{k\pi}{4}, k \in \mathbb{Z}$ [30], so there are only 8 possible global phase factors for any unitary over $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$. To find a representative, all $8 \cdot 2n!$ elements of $[U]$ are generated, and only the lexicographically earliest element is kept.

In practice, computing each phase factor causes a significant performance hit, so we sought a more efficient way of removing phase equivalences. We instead pick a reference element for each unitary and use it to define the canonical phase of the unitary – by convention we choose the first (scanning row by row) non-zero element of a unitary matrix. We immediately see that if $V = e^{i\phi}U$ for unitaries V, U, then the reference elements of V and U must be related by $e^{i\phi}$. For the reference element $re^{i\theta}$ of U, we can then define the canonical unitary as $e^{-i\theta}U$, so that the reference of V will be $re^{i(\theta+\phi)}$, and thus $e^{-i(\theta+\phi)}V = e^{-i\theta}U$. Conversely, if $e^{-i\phi}V = e^{-i\theta}U$, then $V = e^{i(\phi-\theta)}U$ and so V is equivalent to U up to phase.

As a matter of unitary representation, if $\theta \neq \frac{k\pi}{4}, k \in \mathbb{Z}$, this phase multiple will take U outside of the ring $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$. As a result, we actually remove the constraint that the canonical unitary is normalized instead; rather than taking $e^{-i\theta}U$ as the canonical unitary, we use $re^{-i\theta}U$ since $re^{-i\theta} \in \mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$. Again we can verify that two unitaries have the same canonical form if and only if they are equal up to phase: if $V = e^{i\phi}U$ then $re^{-i(\phi+\theta)}V = re^{-i\theta}U$, and if $r_1e^{-i\phi}V = r_2e^{-i\theta}U$ then

$$r_1 = |r_1e^{-i\phi}| \cdot ||V|| = ||r_1e^{-i\phi}V|| = ||r_2e^{-i\theta}U|| = |r_2e^{-i\theta}| \cdot ||U|| = r_2,$$

so $V = e^{i(\phi-\theta)}U$. While this is a minor detail, it allows comparisons to be performed symbolically over the ring, avoiding the need for costly, inaccurate floating point computations.

When generating a given S_i, for each circuit C in $\mathcal{G}_n S_{i-1}$ we compute the canonical representative of $[U_C]$, then the database of circuits is searched to determine if another circuit implementing the representative has already been found. Using suitable data structures, each previous set S_j, $1 \leq j \leq i$ can be searched in $O\big(\log(|S_j|)\big)$ time. If no such circuit is found, we store a circuit implementing the representative of $[U]$.

As a subtle point, if only representatives of equivalence classes in depth i and j are used for searching, not every equivalence class in depth $i + j$ will be found. Consider some unitary $U = VW$ where V is a circuit of depth i, and W is a circuit of depth j.

If V' is the representative of $[V]$ and W' is the representative of $[W]$, then in general $(V')^\dagger VW \notin [W]$, so just using class representatives to search will not suffice. However, $V^\dagger \in [V']$, so $W \in [V']VW$, and thus $[V']U = [W']$. Practically speaking, this means that any unitary $U = VW$ is found by computing the canonical representatives of $[[V]U]$, and so we can search all circuits in minimum depth by storing only equivalence class representatives.

In some cases an exact implementation with the same global phase is required, particularly when the circuit may be controlled on another qubit. While we could compute canonical representatives with respect to qubit relabeling and inversion only, any canonical phase implementation over $\mathcal{G} = \{H, CNOT, T, T^\dagger\}$ can be used to construct the correct global phase, if it is implementable over $\mathcal{G}$. It suffices to observe (as follows from [30]) that if U is implementable by a circuit over $\mathcal{G}$ and a circuit C over $\mathcal{G}$ implements $e^{i\theta}U$, then $\theta = \frac{k\pi}{4}$ for some $k \in \mathbb{Z}$. Since $(HT^\dagger T^\dagger)^3 = e^{-i\frac{\pi}{4}}I$, $e^{i\theta}(HT^\dagger T^\dagger)^{3k}U = U$, so a circuit implementing U exactly can be generated from C with an additive constant cost.

4.3 Extensions

While useful on its own for computing depth-optimal circuits, the meet-in-the-middle algorithm is also very flexible and can be extended in a number of ways. We describe a few such extensions in this section.

4.3.1 Alternative costs

While the meet-in-the-middle algorithm is most naturally suited to optimizing depth, it can also be used to find minimal circuits in other cost metrics. In particular, any cost that is strictly increasing, i.e. $c(C) < c(C')$ for any strict subcircuit C of a circuit C', can be optimized by the meet-in-the-middle algorithm. Since the cost is strictly increasing, given a solution of cost c, all circuits with cost up to c can be generated and searched using the meet-in-the-middle technique, finding a better circuit if one exists. In particular, searching follows Algorithm 1 with the addition of a minimum circuit cost for each database S_i, $c_{\min} = \min_{U \in S_i} c(U)$. Once a circuit C implementing U is found, if $c(C) < (c_{\min}(S_i))^2$, C must be a minimal cost circuit.

This method on its own is not particularly useful, as depending on the growth of the minimum circuit cost the depth required to find or verify a minimal cost circuit may be intractable to search, e.g. a cost function that assigns one gate a significantly higher cost

than the rest. Instead, the search tree should be pruned by removing from the databases all circuits with cost greater than that of the candidate solution – this method could potentially be very effective if an initial candidate solution is already known, as in cases when it is used as a re-synthesis optimization.

In other more specific cases, we can develop specially tuned modifications. Recall that non-Clifford group gates are typically much more costly in the common fault tolerant schemes. Given that the Clifford group is finite on any number of qubits, we can directly optimize the usage of non-Clifford group gates by modifying the initial gate set. The general procedure follows by generating the entire Clifford group using a set of generators, then the gate set can be defined as any single non-Clifford gate or any tensor product of non-Clifford group gates conjugated by the Clifford group, depending on whether the number of such gates is being optimized or the depth, respectively.

As a specific instance, we demonstrate the optimization of T-depth over the gate set $\{H, P, P^\dagger, CNOT, T, T^\dagger\}$. The algorithm first generates the Clifford group $\mathcal{C}_n$ by generating all minimal depth circuits over $\mathcal{G} = \{H, P, P^\dagger, CNOT\}$. To perform the meet-in-the-middle search, we denote $\mathcal{T} = \{T, T^\dagger\}$ and set $S_0 = \mathcal{C}_n$, then define $S_i = \mathcal{C}_n \left(\mathcal{T}_n \setminus \{I\}\right) S_{i-1}$. Each S_i thus contains every circuit with T-depth i in analogy to the depth-optimal meet-in-the-middle algorithm. Searching then proceeds by computing the intersections $S_{i-1}^\dagger U \cap S_i$ to search for T-depth $2i-1$ circuits, and $S_i^\dagger U \cap S_i$ for T-depth $2i$. The full algorithm is summed up in pseudocode below.

Algorithm 2 Meet-in-the-middle algorithm for T-depth.

function MITM-T($\mathcal{G}$, U, l)
  $S_0 := \{I\}$
  $i := 1$
  while $S_{i-1} \neq \emptyset$ **do**
    $S_i := \mathcal{G}_n S_{i-1} \setminus \cup_{j<i} S_j$
  end while
  $\mathcal{C}_n = \cup_{j<i} S_j$
  return MITM($\mathcal{C}_n(\mathcal{T}_n \setminus \{I\})$, U, l)
end function

Searching in this way becomes challenging for high dimensional state spaces, as the size of the Clifford group grows exponentially in the number of qubits. As an illustration, for 3 qubits the Clifford group has 92,897,280 elements up to global phase [57], which makes searching using modern computers impractical for more than a couple levels of depth; $\mathcal{C}_4$ would not even fit in a computer with a reasonable amount of memory using this method.

4.3.2 Ancillas

It is often necessary to add ancillas in order to optimally synthesize a given gate – for instance, the $\Lambda(T)$ gate cannot be synthesized over Clifford group and T gates without using ancillas [30]. The addition of ancillas may also allow lower depth circuits or even lower gate count circuits to be synthesized, allowing a possible tradeoff between space and time in quantum computations. As a result, it is important to build optimization tools that can make use of ancillary qubits. We develop such a method as an extension to the meet-in-the-middle algorithm – for our purposes, we require that the ancillas are initialized in the $|0\rangle$ state and returned to $|0\rangle$, though it may be possible to consider arbitrary ancilla states as in [1].

Suppose we have $U \in U(2^n)$ and we want to synthesize some $U' \in U(2^N)$ so that $U'(|0\rangle^{\otimes N-n}|\psi\rangle) = |0\rangle^{\otimes N-n}(U|\psi\rangle)$. We could naïvely search for $I \otimes U$ using the meet-in-the-middle algorithm; however, this over-constrains the problem as U' need not act as the identity operator on any other ancilla state. Furthermore, searching for circuits with N qubits is exponentially harder than search for circuits with n qubits, while the problem of searching for circuits with n qubits and $N - n$ ancillas seems comparable.

To address these issues, we note that any unitary U' that maps the input $|0\rangle^{\otimes N-n}|\psi\rangle$ to $|0\rangle^{\otimes N-n}(U|\psi\rangle)$ will agree with U on the first 2^n rows and columns. We could then exhaustively search for U in the circuits with unique top left $2^n \times 2^n$ submatrices to find a circuit implementing U using ancillas. However, in the case of meet-in-the-middle, many collisions would be lost, as it may be the case that $V'W'(|0\rangle^{\otimes N-n}|\psi\rangle) = |0\rangle^{\otimes N-n}(U|\psi\rangle)$ but $V'(|0\rangle^{\otimes N-n}|\psi\rangle) \neq |0\rangle^{\otimes N-n}(V|\psi\rangle)$, and likewise for W. In other words, the first 2^n rows and columns of V' and W' is in general not be enough to specify the action of $V'W'$ on $|0\rangle^{\otimes N-n}|\psi\rangle$.

Instead, we note that since $|0\rangle^{\otimes N-n}(U|\psi\rangle) = (I \otimes U)|0\rangle^{\otimes N-n}|\psi\rangle$, we want to find V, W such that $VW(|0\rangle^{\otimes N-n}|\psi\rangle) = (I \otimes U)|0\rangle^{\otimes N-n}|\psi\rangle$. Then, clearly $V^\dagger(I \otimes U)|0\rangle^{\otimes N-n}|\psi\rangle = W(|0\rangle^{\otimes N-n}|\psi\rangle)$, so we can restrict the sets $S^\dagger_{\{i-1,i\}}(I \otimes U)$ and S_i to their action on inputs of the form $|0\rangle^{\otimes N-n}|\psi\rangle$ and determine whether their intersection is non-zero. Practically speaking, this involves comparing the first 2^n columns of each possible collision, as $V^\dagger(I \otimes U)$ may not return the ancillas to the zero state.

One disadvantage of this algorithm is that qubit permutations can no longer be removed from the search space, since the entire unitary is require to construct each permutation. As a result, more circuits need to be generated and searched when using ancillas, though our experimental results show that it is still easier to search for N qubit circuits with ancillas than it is to search for circuits on the same number of qubits and no ancillas.

4.3.3 Approximate synthesis

As many interesting unitaries cannot be synthesized *exactly*, we would like to have methods for finding the best circuit *approximating* a given unitary. Typically, this circuit will only need to approximate the desired unitary to accuracy ϵ in order to achieve the desired error bound for the total computation. To deal with such situations, we develop an approximate circuit synthesis algorithm using the meet-in-the-middle algorithm. Unlike the previous extensions, approximate synthesis is reasonably more involved.

Recall that the approximate synthesis problem involves synthesizing some circuit C over $\mathcal{G}$ so that $||U - U_C|| \leq \epsilon$ for some $\epsilon > 0$. While we originally defined approximations using the operator norm, it will be useful to allow other norms to be used where more specific distance measures are required (e.g. the diamond norm), or where they are easier to compute (e.g. the Frobenius norm). The techniques we develop apply to all distance measures defined by the Schatten p-norms,

$$||U||_p = \left(\mathrm{Tr}\left(\left(U^\dagger U\right)^{p/2}\right)\right)^{1/p},$$

the key properties being the unitary invariance of Schatten norms, and that they, like all norms, obey the triangle inequality. We define a distance measure between two unitaries, $d(U, V)$, as the norm of their difference, $||U - V||_p$ for some p – such a distance measure is called a *metric*. The operator norm, as defined earlier, is equal to the Schatten norm in the limit as $p \to \infty$.

The exact synthesis algorithm relied on Lemma 4 to ensure that a minimal depth circuit implementing U could be found by looking only at minimal depth circuit halves. To use the same idea in the approximate synthesis setting, we first derive a similar result.

Lemma 5. *Let $S_i \subseteq U(2^n)$ be the set of all unitaries implementable in depth i over the gate set $\mathcal{G}$. Given a unitary U and $\epsilon \geq 0$, there exists a circuit over $\mathcal{G}$ of depth l implementing some U' with $d(U, U') \leq \epsilon$ if and only if $d(V^\dagger U, W) \leq \epsilon$ for some $V \in S_{\lfloor l/2 \rfloor}, W \in S_{\lceil l/2 \rceil}$.*

Proof. The lemma is a simple consequence of the unitary invariance of the Schatten p-norms, along with Lemma 4. In particular,

$$||U - VW||_p = ||V^\dagger U - V^\dagger V W||_p = ||V^\dagger U - W||_p.$$

□

Lemma 5 implies a meet-in-the-middle algorithm for approximate synthesis, except that instead of finding the intersection between $S^\dagger_{\lfloor l/2 \rfloor}U$ and $S_{\lceil l/s \rceil}$, we need to check for elements in the approximate intersection $\{(A, B) | A \in S^\dagger_{\lfloor l/2 \rfloor}U, B \in S_{\lceil l/2 \rceil}, d(A, B) \leq \epsilon\}$. Fortunately, this is an instance of the well studied *Nearest Neighbour* problem, which itself has applications to many other fields, e.g. image recognition, DNA sequencing and data compression.

In particular, given a set S with distance measure $d : S \times S \to \mathbb{R}$, a *nearest neighbour* of a query point $q \in S$ in database $S_D \subseteq S$ is an element $p \in S_D$ such that $\forall r \in S_D, d(q, p) \leq d(q, r)$. We denote the problem of finding such a point $NN(q, S_D)$. While it will not affect the solution in a substantial way, the problem we need to solve is a relaxation to what we call the *bounded near neighbour* problem, denoted $NN|_\epsilon(q, S_D)$: given q, S_D and $\epsilon > 0$, determine whether there exists $p \in S_D$ such that $d(q, p) \leq \epsilon$.

Given a procedure for computing $NN|_\epsilon(q, S_D)$ together with Lemma 5 we can modify the meet-in-the-middle algorithm to return approximating circuits, as shown in Algorithm 3. Using the analysis of Algorithm 1, the algorithm requires $O\left(|\mathcal{G}|^{\lceil n \cdot l/2 \rceil}\right)$ calls to the bounded near neighbour procedure.

Algorithm 3 Meet-in-the-middle algorithm

function MITM-APPROX($\mathcal{G}$, U, l, ϵ)

$S_0 := \{I\}$
$i := 1$
for $i \leq \lceil l/2 \rceil$ **do**
  $S_i := \mathcal{G}_n S_{i-1}$
  if $NN|_\epsilon(V^\dagger U, S_{i-1}) = W$ for any $V \in S_i$ **then**
    return VW
  else if $NN|_\epsilon(V^\dagger U, S_i) = W$ for any $V \in S_i$ **then**
    return VW
  end if
  $i := i + 1$
end for
end function

Computing $NN|_\epsilon(q, S_D)$

As noted above, the nearest neighbour problem is well studied due to its wide range of applications. The techniques can be (roughly) divided into those effective in low dimension spaces, and those effective in high dimension spaces [58]. In the former case *projection*-based techniques, which divide and search the space by projecting points onto each dimension (e.g. k-d trees and R-trees), can be very effective – in fact, the k-d tree has been previously used to implement the ϵ-net in implementations of the Solovay-Kitaev algorithm [47]. However, these methods scale poorly to situations where the dimension is high, such as multi-qubit approximate synthesis where unitaries on n qubits have dimension 2^{2n}.

For high dimensional data, fewer generic, efficient techniques exist. In some instances, transformations to reduce the high dimensional data to a lower dimension space are available (e.g. the discrete Fourier transform), however they are highly specialized for particular applications and typically don't provide exact solutions, as is the case for many common locality sensitive hash functions. However, a class of generic data structures has been developed for nearest neighbour searches in metric spaces, namely *distance*-based index structures, or in the case of distance-based trees, *metric trees*. The common metric trees have been shown to be very effective for nearest neighbour searching [58], and as a result we take this approach.

To efficiently compute $NN|_\epsilon(q, S_D)$ we use a *vantage-point (vp) tree*, introduced in [59], which has been shown to be very effective in high dimension similarity search, notably image recognition [60]. In analogy to simple binary search trees, which were used to store and search through circuit databases for exact synthesis, a vantage point tree recursively splits the database into two subtrees according to each point's distance to a distinguished *vantage* point. Each node in the tree has a unique vantage point p, with points within distance μ of p placed in one subtree, while those with distance greater than μ are placed in the other subtree. The radius μ of the partitioning ball can be chosen as the median distance between p and each point in the tree; if there are not too many points exactly distance μ from p, we can see that each subtree contains approximately half the points in the tree.

The advantage for near-neighbour searching can be seen by observing that for a given subtree S_D with vantage point p and radius μ, if our query point q satisfies $d(p, q) \leq \mu - \epsilon$, then for any r where $d(p, r) > \mu$,

$$d(q, r) \geq |d(p, r) - d(p, q)| > |\mu - (\mu - \epsilon)| = \epsilon$$

by the triangle inequality, and thus we only need to search for near neighbours in the subtree containing those elements less than distance μ from p. A similar identity holds if

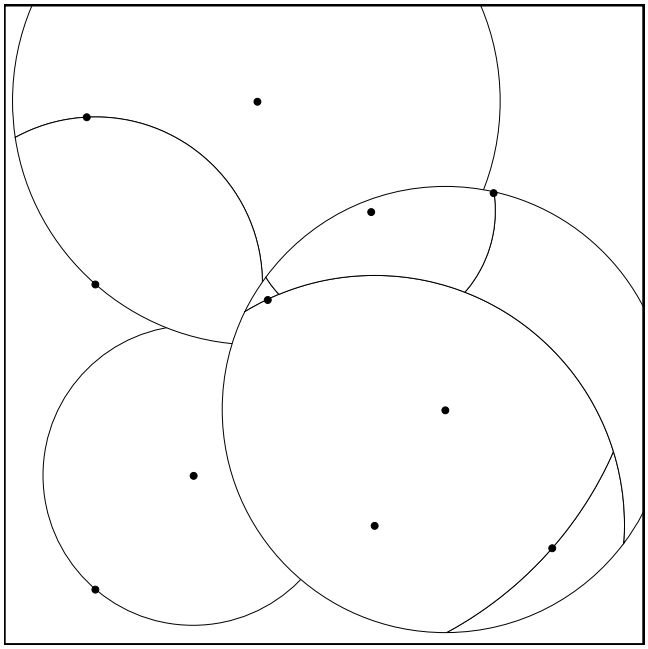


Figure 4.2: Visualization of a vp-tree partitioning a set of 2D points.

$d(p, q) \geq \mu + \epsilon$. The obvious drawback is that if $\mu - \epsilon < d(p, q) \leq \mu + \epsilon$, the search must recurse on both subtrees. The use of the triangle inequality to, in most cases, remove a large set of points from consideration forms the basis for many metric trees [58].

Algorithm 4 shows how to build a vp-tree on S in $O(n \log(n))$ time, given a distance oracle. A detailed analysis of the expected search time can be found in [59] – under specific assumptions regarding the probability distribution of the points, they show that searching can be performed in expected time $O(\log(n))$. Our experiments show comparable results in the case of unitary searching.

Remarks

In this section we close off the discussion with a few remarks regarding the choice of norms, and the combination of approximate meet-in-the-middle with the search space reductions.

One of the main bottlenecks in approximate meet-in-the-middle comes from having to deal with distance computations, requiring the computationally expensive process of computing Schatten norms. In the worst case the bounded near neighbour search becomes a

Algorithm 4 vp-tree construction and searching [59]

function VPTREE(S)
  if $S = \emptyset$ **then**
    return empty tree
  end if
  $T.p := p$ for some $p \in S$
  $T.\mu := \text{median}_{r \in S} d(p, r)$
  $T.L := \text{VPTREE}(\{r \in S | d(p, r) \leq T.\mu\})$
  $T.R := \text{VPTREE}(\{r \in S | d(p, r) > T.\mu\})$
  return T
end function

function $NN|_\epsilon(q, T)$
  $x := d(T.p, q)$
  if $x \leq \epsilon$ **then**
    return $T.p$
  else if $x \leq \mu + \epsilon$ **then**
    if $NN|_\epsilon(q, T.L) = p$ **then**
      return p
    end if
  else if $x > \mu - \epsilon$ **then**
    if $NN|_\epsilon(q, T.R) = p$ **then**
      return p
    end if
  end if
  return "not found"
end function

linear search, requiring the distance between every pair of elements in $S_i^\dagger U \times S_j$ to be computed. Even in the average case, there will still be $O(|S_i| \log(|S_j|))$ distance computations to find elements in the approximate intersection of $S_i^\dagger U$ and S_j. We thus aim to compute norms as efficiently as possible. Fortunately, in the case of the Frobenius norm ($p = 2$), we obtain a convenient formula that requires only one matrix multiplication, rather than computation of the singular values:

Lemma 6. *For* $U, V \in U(2^n), ||U - V||_2 = \sqrt{2 \cdot 2^{2n} - 2 \cdot \text{Re}(\text{Tr}(U^\dagger V))}$

Proof. We see that

$$\begin{aligned} ||U - V||_2 &= \sqrt{\text{Tr}\left((U^\dagger - V^\dagger)(U - V)\right)} \\ &= \sqrt{\text{Tr}\left(2I - U^\dagger V - (U^\dagger V)^\dagger\right)} \\ &= \sqrt{2 \cdot 2^{2n} - \text{Tr}(U^\dagger V) - \left(\text{Tr}(U^\dagger V)\right)^\dagger} \\ &= \sqrt{2 \cdot 2^{2n} - 2 \cdot \text{Re}(\text{Tr}(U^\dagger V))}. \end{aligned}$$

□

Approximate meet-in-the-middle also affects the search space reductions used in a non-trivial way. While circuit databases containing only circuit representatives can still be generated, our method of choosing representatives is no longer sufficient for searching; it may be the case that $||U - V||_p \leq \epsilon$ but $||U' - V'||_p > \epsilon$ for representatives U', V' of $[U], [V]$ respectively. To adapt the search space reductions to cover approximate synthesis, we must instead generate and search for every element in $[[V]U]$, rather than just its canonical representative as in the case of exact synthesis.

However, an issue remains with this approach when using the standard Clifford + T gate set: since U does not necessarily take its elements from the ring $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$, it may be an arbitrary phase away from any element in our database. We clearly cannot search for every possible phase multiple in $[[V]U]$, so to find approximations up to a global phase we need to use a norm invariant under global phase. A reasonable choice would be to define $d(U, V)$ as $||U \otimes U^\dagger - V \otimes V^\dagger||_p$ for some p. Clearly, $d(e^{i\theta}U, e^{i\phi}V) = d(U, V)$ for any θ, ϕ:

$$||e^{i\theta}U \otimes (e^{i\theta}U)^\dagger - e^{i\phi}V \otimes (e^{i\phi}V)^\dagger||_p = ||e^{i\theta}e^{-i\theta}U \otimes U^\dagger - e^{i\phi}e^{-i\phi}V \otimes V^\dagger||_p = ||U \otimes U^\dagger - V \otimes V^\dagger||_p.$$

The naïve method of computing this norm would require generating and computing the singular values of a $2^{2n} \times 2^{2n}$ matrix, which is a significant disadvantage performance wise. However, in the case of the Frobenius norm we see that the phase invariant distance can be computed at no extra cost.

Lemma 7. *For* $U, V \in U(2^n)$, $||U \otimes U^\dagger - V \otimes V^\dagger||_2 = \sqrt{2 \cdot 2^{4n} - 2 \cdot |\mathrm{Tr}(U^\dagger V)|^2}$

Proof. From Lemma 6 we have $||U \otimes U^\dagger - V \otimes V^\dagger||_2 = \sqrt{2 \cdot 2^{4n} - 2 \cdot \mathrm{Re}\left(\mathrm{Tr}\left((U^\dagger V) \otimes (UV^\dagger)\right)\right)}$. Since $\mathrm{Tr}\left((U^\dagger V) \otimes (UV^\dagger)\right) = \mathrm{Tr}(U^\dagger V)\,\mathrm{Tr}(V^\dagger U)$ by basic properties of the trace, we see that $2 \cdot \mathrm{Re}\left(\mathrm{Tr}\left((U^\dagger V) \otimes (UV^\dagger)\right)\right) = 2 \cdot \left|\mathrm{Tr}(U^\dagger V)\right|^2$, completing the proof. □

The phase-invariant Frobenius norm was previously used in [45] to synthesize approximate circuits. In both cases, we can see that the Frobenius norm gives an overestimate of the approximation error as given by the operator norm, since $||U||_p \geq ||U||_q$ for any $p \leq q$, and thus can be used to bound the approximation error in the operator norm.

4.4 Implementation details

The meet-in-the-middle algorithm, along with its extensions and search space reductions, were programmed in the C++ programming language; the implementation is available

online at http://code.google.com/p/mitms. In this section we discuss some of the relevant implementation details for the practical application of these synthesis methods.

It was alluded to earlier that the meet-in-the-middle algorithm offers no speed up over the naïve algorithm without suitably chosen data structures, as searching for collisions in unordered sets $S_i^\dagger U$ and S_j would use $O(|S_i| \cdot |S_j|)$ comparisons. However, by imposing a lexicographic ordering on the generated circuits, they can be searched in time logarithmic in the size of S_j. In our implementation, we use such an ordering to store each S_i as a red-black tree, a type of balanced binary tree. Balanced binary trees are a common choice for implementations of ordered sets and mappings in standard libraries, as well as industrial databases, due to predictable performance and scalability. Since deletions, typically the most computationally difficult task in a balanced binary tree [61], are never performed in our implementation, such trees are a natural choice of data structure.

Hash tables were also mentioned as a potential for achieving an average case complexity of $O(|\mathcal{G}_n|^{\lceil l/2 \rceil})$. To test the performance advantage of using hash tables, our code was adapted to use the hash table implementation in libstd++. While our hash function generated a very manageable number of key collisions on average – as an illustration, at most 52 out of 1,316,882 distinct 3-qubit unitaries were mapped to any one hash value – no significant performance gain was observed in our experiments.

As storage space is typically a bottleneck in breadth-first searches [43], even with the meet-in-the-middle algorithm, storing unitaries themselves proved infeasible for the circuits we were interested in. For instance, using the standard universal gate set $\mathcal{G} = \{H, P, P^\dagger, CNOT, T, T^\dagger\}$, there are 252 unique 3-qubit tensor products. if unitaries are stored as matrices with entries in $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$ each 3-qubit unitary requires $1,280$ bytes, and so all depth 5 circuits on 3 qubits would require more than 1 petabyte of storage space. Even with the search space reductions, which reduces the number of unitaries stored up to depth 5 to 37,402,324, over 47 gigabytes would be required to store just the circuit databases, not including space for maintaining data structures or the relevant circuits. While it is feasible that the storage space for unitaries over $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$ could be reduced by applying compression, it is still clear that for searches up to meaningful depth, the full unitary matrices cannot be stored and a space-time trade off must be made.

To make such a trade off, rather than store the generated unitaries, we store circuits as lists of gates in $\mathcal{G}_n$. Each such gate on n-qubits is represented as n bytes specifying which gate is acting on each qubit. While this method reduces the storage space required for unitaries by an order of magnitude, it significantly slows down searching, as each time a comparison is invoked the unitary implemented by the circuit needs to be computed by a sequence of matrix multiplications again.

As a compromise between space efficiency and time complexity, we developed a method of generating search keys for unitaries that works extremely well in practice. In particular, an $m \times m$ matrix M is stored as a key with each circuit, where for a circuit C $M(i,j) = v_i^\dagger U_C v_j$. The m vectors $\{v_i\}$ are chosen from $\mathbb{C}^{2^n}$ using a pseudorandom generator to generate the individual elements, and in practice $m = 1$ has been enough to search interesting depths for up to 4 qubits with extremely few key collisions. Furthermore, while this method of key generation uses floating point computations there is only one final round-off for each key, so equal unitaries will produce keys with equal numerical error. For gate sets that are expressed over the ring $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$, unitaries are computed symbolically, thus equivalent computed unitaries will always map to the same key. Experiments were also performed using pseudorandom vectors over $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$ to avoid floating point computations altogether, but they generated far too many key collisions with small key sizes to be of practical use.

It can also be noted that the problem is in a sense embarrassingly parallel, as long as each processor has access to the circuit databases. When testing the emptiness of $S_i^\dagger U \cap S_j$, each processor can search for a particular element of $S_i^\dagger U$ in parallel. To speed up our searches we parallelized our code using the pthreads C library to allow concurrent searching.

As a final point, it was noted earlier that breadth-first circuit synthesis typically involves the offline computation of circuit databases that are later searched – we include such functionality in our implementation, as the database generation takes significantly more time than searching.

4.5 Results

We tested our implementation in Debian Linux on a quad-core, 64-bit Intel Core i5 2.80GHZ processor with 16 GBs of RAM.

The main purpose of our experiments was to find optimal depth decompositions into the gate set $\{H, P, CNOT, T\}^\dagger$ for 2-5 qubit unitaries, with a secondary optimization criteria being the gate count. Table 4.1 lists some performance figures for our implementation – searching times for a given depth i describe the time the computation took to search through circuits of depth $2i-1$ and $2i$. As the variance between search times was extremely minor, one representative search was chosen for each set of data. We found that in cases when $S_j^\dagger U \cap S_i \neq \emptyset$, collisions are usually reported within a few minutes of searching.

Table 4.1: Performance figures for Algorithm 1.

# qubits \ depth		1	2	3	4	5	6
2	database size (#)	14	104	901	6,180	37,878	197,388
	RAM (KB)	2.092	16.686	146.701	1,013.358	6,249.708	32,766.246
	generation time (s)	0.001	0.015	0.155	1.354	10.761	75.301
	search time (s)	0.001	0.004	0.033	0.248	1.672	9.321
3	database size (#)	36	1,110	41,338	1,316,882	36,042,958	-
	RAM (KB)	5.633	179.657	6,737.931	215,968.485	7,738,582.749	-
	generation time (s)	0.012	1.059	40.619	1896.301	73,295.675	-
	search time (s)	0.015	0.350	12.619	414.722	11,759.390	-
4	database size (#)	84	9,984	1,755,677	-	-	-
	RAM (KB)	13.460	1,617.082	284,596.043	-	-	-
	generation time (s)	0.570	122.966	18,728.922	-	-	-
	search time (s)	0.603	71.420	12,853.887	-	-	-
5	database size (#)	172	79,586	-	-	-	-
	RAM (KB)	27.956	12,954.420	-	-	-	-
	generation time (s)	49.118	20,642.411	-	-	-	-
	search time (s)	66.517	32,508.756	-	-	-	-

It can be observed that search time for similar sized databases grows with the number of qubits – we conjecture that this is a result of the increasing complexity of matrix multiplication. For this reason, the meet-in-the-middle algorithm currently appears limited in usefulness beyond 4 qubits.

While tools performing multi-qubit quantum circuit synthesis techniques are not generally available, we compared our implementation of the meet-in-the-middle algorithm with an open-source Python implementation [62] of the Solovay-Kitaev algorithm [47]. This particular implementation was chosen over faster C versions or exact synthesis tools as it is the only existing tool to our knowledge decomposing multiple qubit operators over the Clifford + T gate set. Database generation times for two qubit circuits composed with H, T, $T^\dagger$, and $CNOT$ are shown in Figure 4.3; the meet-in-the-middle implementation shows a similar but compressed exponential curve.

A decomposition of the controlled-H gate was generated with both the Solovay-Kitaev and meet-in-the-middle algorithms. While our algorithm produced an exact decomposition with minimal depth (Figure 4.6a) in under 0.500s, the Solovay-Kitaev algorithm with 4 levels of recursion took over 2 minutes to generate a sequence consisting of over 1000 gates approximating the unitary to an error of 0.340 in the phase-invariant Frobenius norm. While we stress that the Solovay-Kitaev algorithm is not designed to factor unitaries

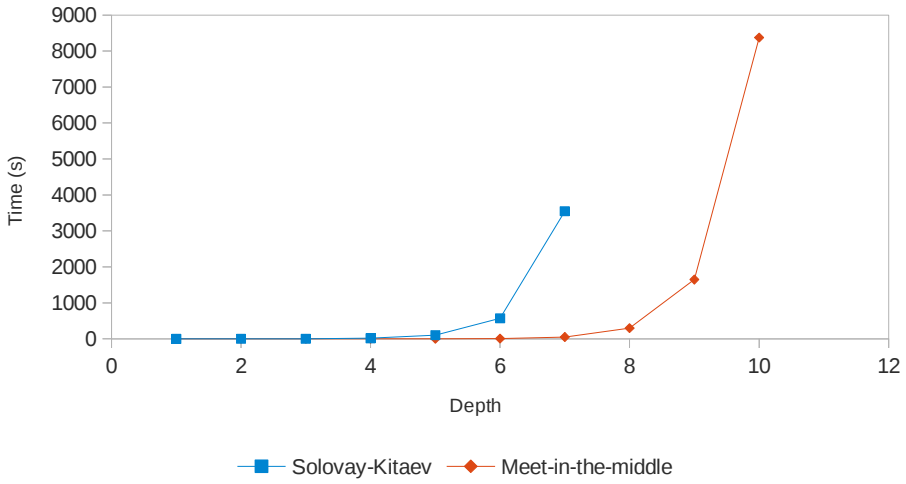
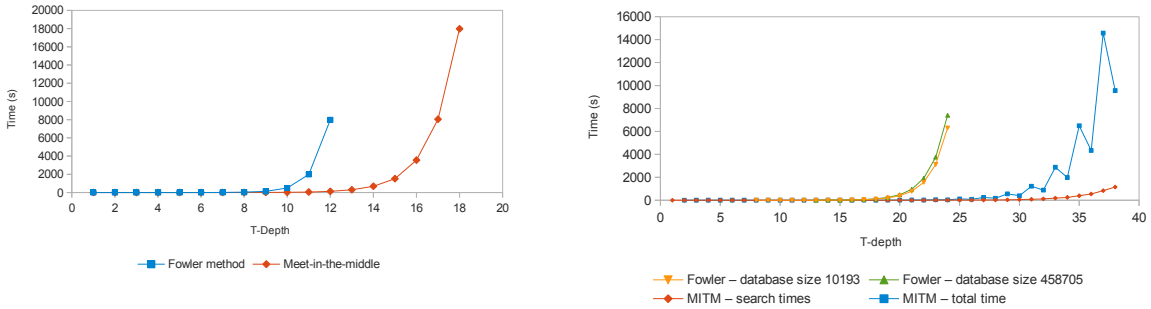


Figure 4.3: Database generation times for minimal depth two qubit circuits.

exactly over a gate set, this experiment serves to illustrate both the efficiency of our implementation, as well as the limitations of current quantum circuit synthesis tools.

We also compared our approximate synthesis algorithm (Algorithm 3) against Fowler's algorithm [45]. While Fowler's algorithm is no longer the best known method of optimal approximate single-qubit synthesis [19, 30], we chose it as a benchmark since it is the only algorithm to our knowledge that applies to multi-qubit depth-optimal approximate synthesis. Unfortunately, only a single-qubit implementation over the gate set $\mathcal{C}_1 \cup \{T, T^\dagger\}$ is available, so we could not compare the two algorithms on multi-qubit synthesis, though it seems reasonable to believe that the behaviour would be similar in the multi-qubit case. Again, this serves to illustrate that our algorithm fills a gap in the current set of quantum circuit optimization tools.

Figure 4.4a shows the comparative times both implementations take to build up a database of the unique circuits by brute force enumeration with increasing T-depth. Both curves show the same exponential growth, though the meet-in-the-middle technique is noticeably compressed, providing good evidence that storing only one circuit from each equivalence class permits databases to be built with greater depth. Figure 4.4b shows the subsequent time required to search each level of T-depth; for the meet-in-the-middle algorithm, the total time to generate the next database then search is also shown. The dips in total time are the result of using a modified version of the Algorithm 3 that searches for T-depth optimal circuits. Instead of increasing the T-depth by 1 with every iteration



(a) Database generation times.

(b) Search times.

Figure 4.4: Database generation and search times for minimal T-depth single qubit circuits.

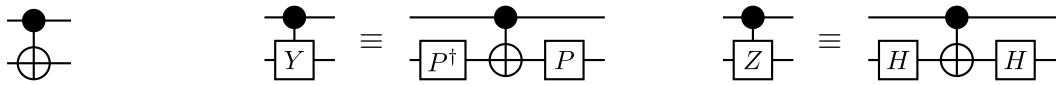
as in Algorithm 2, we either add a layer of Clifford group gates or T gates, with a layer of T gates taking significantly less time. At the ith iteration, we then search for circuits of T-depth i. Additionally, two different size databases were used to generate data for Fowler's method, one containing 10,193 unique circuits and one containing 458,705.

While better approximate synthesis methods exist for single-qubit circuits, Figure 4.4b gives strong evidence to its increased scalability over Fowler's algorithm for multiple qubits. With the meet-in-the-middle technique, significantly greater T-depths could be searched in less time even without precomputing circuit databases. If databases are precomputed, search times appear insignificant compared to Fowler's method. Furthermore, increasing the database size from 10,193 to 458,705 did not improve search times, while with Algorithm 3, increasing the size of circuit databases allows for greater T-depth to be searched.

4.5.1 Depth-optimal implementations

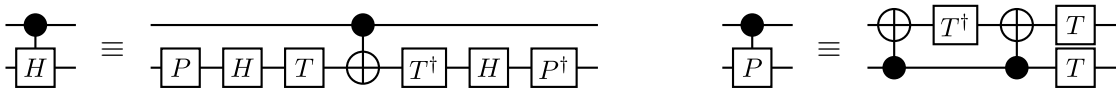
One of the main results of this work has been depth-optimized circuits over the gate set $\{H, P, CNOT, T\}^\dagger$ for many of the common logical quantum gates, with reductions of up to a third of the previous best known circuit depth. While the optimizations provide great advantages to the eventual implementation of algorithms using such gates, these circuits also give the first lower bounds for the depth of many quantum gates in the Clifford + T gate set. We also report the T-depth of optimized circuits, and later use Algorithm 2 to optimize T-depth directly.

Minimal depth implementations of the singly controlled versions of H, P, and $V = \sqrt{X} = \frac{1}{2}\begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix}$ were computed (Figure 4.6), along with the controlled-Z and -Y for completeness (Figure 4.5).

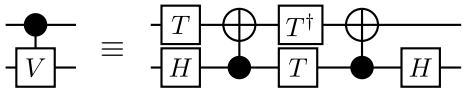


(a) Controlled-X (depth 1). (b) Controlled-Y (depth 3). (c) Controlled-Z (depth 3).

Figure 4.5: Controlled Paulis.



(a) Controlled-H (depth 7). (b) Controlled-P (depth 4).



(c) Controlled-V (depth 5).

Figure 4.6: Logical gate implementations of controlled unitaries without ancillas.

We also optimally decompose (Figure 4.7) the 2-qubit gate

$$W = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

which has found use in at least one interesting quantum algorithm [63].

Some 3-qubit unitaries with minimal depth implementations found include the well-known Toffoli gate (Figure 4.8a), Fredkin gate (Figure 4.8e, defined as the controlled-$SWAP$), quantum OR (Figure 4.8c, defined as the unitary mapping $|a\rangle|b\rangle|c\rangle \mapsto |a\rangle|b\rangle|c \oplus$

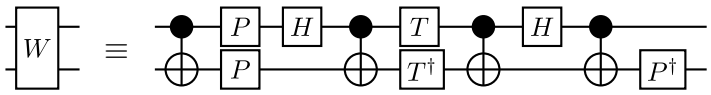


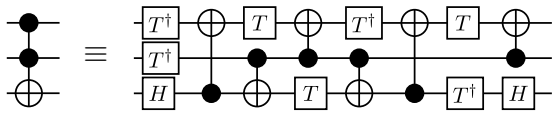
Figure 4.7: W gate (depth 9).

$a \vee b\rangle$), and Peres gate [64] (Figure 4.8). It should be noted our circuit reduces the total depth of the Toffoli gate from 12 [27] to 8.

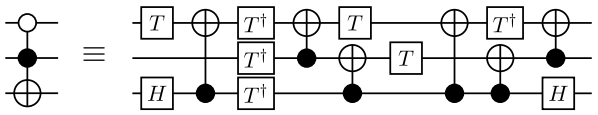
Searches were also performed for each of the above n-qubit gates using up to $4 - n$ ancillas – these searches were performed up to the maximum depth for the total number of qubits, as seen in Table 1. None of the logical gates tested were found to admit circuits with shorter depth or fewer T gates, though circuits for controlled-P and -V gates were found with smaller T-depth (Figure 4.9). Additionally, a circuit was found that did have reduced minimal depth when decomposed using an ancilla (Figure 4.10), together with the reduced T-depth circuits providing clear motivation for the use of ancillas to optimize circuit execution time.

Among other gates attempted were the 3-qubit quantum Fourier transform, which was proven to have no circuit in our instruction set with depth at most 10, and the 4-qubit Toffoli gate $\Lambda_3(X)$ and a 1-bit full adder, with no circuits of depth at most 6. Additionally, both the controlled-T and controlled-$\sqrt[4]{X}$ gates were proven to have no implementations of depths at most 10 or 6 using one or two ancillas, respectively.

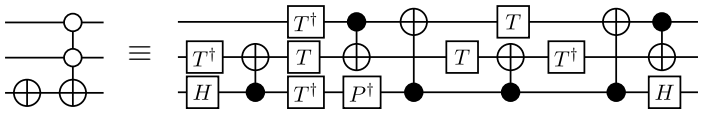
We did however optimize a known circuit implementing the controlled-T gate, as well as one implementing a 1-bit full adder using our algorithm. Specifically, we generated a circuit for $\Lambda(T)$ using the decomposition $FREDKIN \cdot (I \otimes I \otimes T) \cdot FREDKIN$, and a circuit for the 1-bit adder by using the implementation found in [65], substituting the circuit in Figure 4.8d for the Peres gate. Then we performed a peep-hole optimization by taking small subcircuits and replacing them with shorter, lower gate count circuits synthesized using our algorithm. The circuit for the controlled-T gate, shown in Figure 4.11, reduces the number of T gates from 15 to 9, $CNOT$ gates from 16 to 12, and T-depth from 9 to 5, while the 1-bit adder circuit (Figure 4.12) reduces the number of T gates from 14 to 8, $CNOT$ gates from 12 to 10, H gates from 4 to 2, and T-depth from 8 to 2. These results provide strong evidence for the effectiveness of peephole re-synthesis as a full-scale optimization tool.



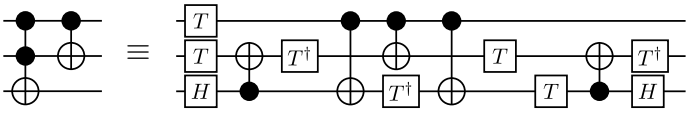
(a) Toffoli gate (depth 8).



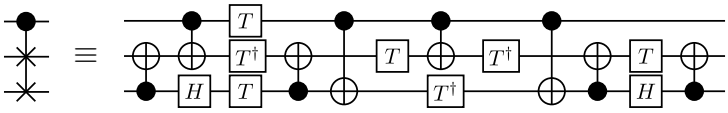
(b) Toffoli gate, one negative control (depth 8).



(c) Quantum OR gate (depth 8).



(d) Peres gate (depth 8).

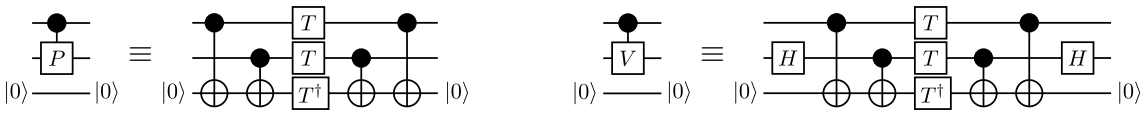


(e) Fredkin gate (depth 10).

Figure 4.8: 3-qubit logical gates with no ancillas.

4.5.2 T-depth-optimal implementations

Experiments were also performed to find circuits with minimum T-depth, using Algorithm 2. The bottleneck in this case is the sheer size of the Clifford group which is difficult to generate itself, much less use in exhaustive searching. For 2 qubits, generation of the 11,520 unique Clifford group elements (up to phase) required approximately 1 second of computing time and less than 2 seconds to search for a unitary up to 1 T-stage, or 2 T-stages and ending in a non-Clifford operation. In practice, this was enough to find minimum T-depth implementations of the 2-qubit gates in question. By contrast, generation of the 92,897,280 unique 3-qubit Clifford group elements required almost 4 days to compute.



(a) $\Lambda(P)$ (T-depth 1, total depth 5). (b) $\Lambda(V)$ (T-depth 1, total depth 5).

Figure 4.9: Reduced T-depth implementations utilizing ancillas.

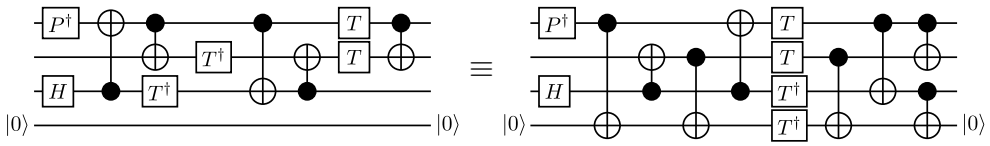


Figure 4.10: Addition of one ancilla reduces the minimum circuit depth from 7 to 6.

The minimal T-depth controlled-H gate (Figure 4.13) required less than one second to compute, after generating the Clifford group. Minimal T-depth circuits for other 2-qubit logical gates were not found to decrease the number of T-stages compared to the minimal depth circuits, and thus the circuits shown for controlled-P, controlled-V, and W are optimal both in circuit depth and T-depth. As a result, allowing the use of ancillas can strictly decrease the minimum T-depth required to implement a given unitary, since implementations of the controlled-P and -V gates with ancillas were found with lower T-depth.

While no Toffoli implementation has yet been found with provably minimal T-depth and zero ancilla, a circuit with T-depth 3 implementing the Toffoli gate (Figure 4.14) has been found using our main algorithm; as the Toffoli appears to require a minimum of 7 T gates to implement, we conjecture this is minimal. Furthermore, it reduces the number of T-stages from 5 [27] to 3, providing an approximate 40% speed-up in fault tolerant schemes where Clifford group gates have negligible cost compared to the T gate.

We return to the question of parallelizing T gates in more detail in Chapter 5.

4.5.3 Exact decomposition of controlled unitaries

Recall that the controlled version of any circuit can be generated by replacing each individual gate with a controlled version of that gate, with the control qubit of the entire circuit

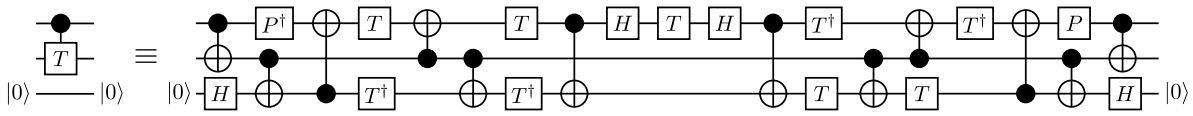


Figure 4.11: Controlled-T gate (depth 19).

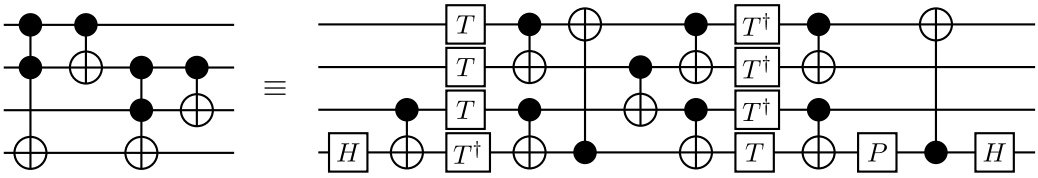


Figure 4.12: Circuit implementing a reversible 1-bit full adder.

functioning as the control qubit of each gate [24]. The minimal-depth circuits computed in Section 4.5 allow us to establish the following result, bounding the number of gates in the controlled version of any circuit over $\{H, P, CNOT, T\}^\dagger$.

Theorem 2. *Let the gate cost of a circuit be given by a vector* $\mathbf{x} = [x_H, x_P, x_C, x_T]^t$*, where* x_H *denotes the number of* H *gates,* x_P *denotes the number of* P*-gates or* $P^\dagger$*-gates,* x_C *denotes the number of* $CNOT$ *gates, and* x_T *denotes the number of* T*-gates or* $T^\dagger$*-gates. Suppose* U *can be implemented to error* $\epsilon \geq 0$ *by a circuit over* $\mathcal{G}^\dagger$ *where* $\mathcal{G} = \{H, P, CNOT, T\}$ *with gate cost of* $\mathbf{x}$*. Then* $\Lambda(U)$ *can be implemented to error at most* ϵ *over* $\mathcal{G}^\dagger$ *by a circuit of gate cost* $A\mathbf{x}$*, where*

$$A = \begin{bmatrix} 2 & 0 & 2 & 4 \\ 2 & 0 & 0 & 2 \\ 1 & 2 & 6 & 12 \\ 2 & 3 & 7 & 9 \end{bmatrix}.$$

The circuit for controlled-U *uses exactly one ancilla qubit if one or more* T*-gates are present in the decomposition of* U*, and no ancilla qubits otherwise. Furthermore, controlled-*U *can be implemented in a* T*-depth of at most* $x_H + 2x_P + 3x_C + 5x_T$*.*

Proof: Assume that U admits an ϵ-approximation over $\mathcal{G}$ with associated cost vector $\mathbf{x} = [x_H, x_P, x_C, x_T]^t$. As shown in Section 4.5, for each gate H, P, $CNOT$, T, the corresponding singly controlled gate can be implemented exactly over $\mathcal{G}$. Specifically,

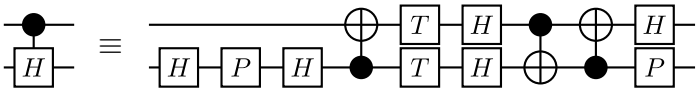


Figure 4.13: Circuit implementing a controlled-H gate (T-depth 1, total depth 9).

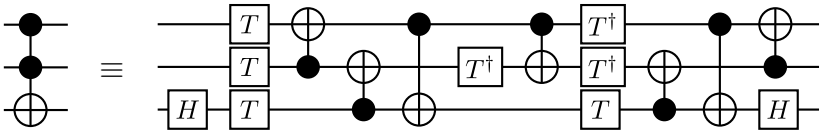


Figure 4.14: Circuit implementing a Toffoli gate (T-depth 3, total depth 9).

we obtain from Figure 4.6 for each controlled-H gate a cost of $[2, 2, 1, 2]$ and for each controlled-P gate a cost of $[0, 0, 2, 3]$. From Figures 4.8 and 4.11 we obtain costs for each controlled-$CNOT$ and each controlled-T gate of $[2, 0, 6, 7]$ and $[4, 2, 12, 9]$, respectively. Since the total cost is linear in the costs of the gates, we obtain that claimed total cost of $A\mathbf{x}$. The approximation error ϵ is unchanged as compared to U since no further errors are introduced in the factorization. Finally, the claimed bound for the overall T-depth holds since the T-depths of each H, P, $CNOT$, and T gates can be upper bounded by 1, 2, 3, and 5, respectively, where here we used the T-depth 3 circuit in Figure 4.14 to derive an upper bound for the Toffoli gate. □

4.6 Conclusions

In this chapter, we have developed a simple algorithm for finding a minimal depth quantum circuit implementing a given unitary U in a specific gate set. The primary focus was to find exact circuits with either minimal depth or minimal T-depth, though we also developed extensions to optimize over ancillas and to compute optimal approximating circuits. Our computations have found minimal depth Clifford + T circuits for many important logical gates, in some cases providing significant speed-up over previously known or algorithmically generated circuits.

The experimental results show that search-based circuit synthesis can very effective in the optimization of small multi-qubit quantum circuits, as arises when compiling high level quantum gates to a lower level fault tolerant gate set. For instance, our implementation

can find any optimal 3-qubit circuit up to depth 8 over $\{H, P, CNOT, T\}^\dagger$ in approximately 400 seconds, which no previously existing techniques could accomplish. Similarly, our approximate synthesis procedure performs much faster than the only previous depth-optimal multi-qubit approximate synthesis algorithm.

Compared to reversible synthesis where search-based techniques are prominent, our performance is slower, it appears that quantum circuit synthesis is a much more difficult problem. The best search-based reversible synthesis results [12] considered 4 bit Boolean functions, which admit a representation via 64 bits that also allows common operations such as permutation and inversion to be carried out by bitwise operations. While we use similar bit twiddling techniques for algebra in the ring $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$, a 4-qubit unitary over $\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$ requires specification of the entire unitary, a 16 by 16 matrix where each element can be represented by 5 integers. Without any kind of compression and using 32 bit integers, a single unitary would require $40,960$ bits of memory, a blowup by a factor of 640 compared to reversible functions. To compose circuits, expensive matrix multiplication needs to be performed, and even permutations or inversions need to examine each element of the matrix, adding significant complexity over the bitwise operations for reversible functions. As a result it would appear that unitary synthesis is a more complex problem.

As an additional point regarding this algorithm, we stress that space-time trade-offs were made to allow searches of reasonable depth to be performed. By storing only the circuits, not entire unitaries, we achieved a significant reduction in memory usage, reducing the minimum space to store an n-qubit unitary from $4^n \times 5 \times 4$ bytes to as few as n bytes. While permutation and inversion is cheap for these circuit descriptions, a circuit of depth d needs d matrix multiplications to compute the corresponding unitary, a major obstacle to performance. To alleviate this time penalty, circuit keys were introduced, storing only a small matrix of complex numbers so that unitaries are only generated when generating a new key, or if a key collision is found.

4.6.1 Future work

One possible direction of future work is to use the algorithms developed here to create better optimization or synthesis algorithms. While this method performs well for small unitaries, it remains exponential in the size of the instruction set and, by extension, depth, so it is unlikely to be useful for optimizing large circuits. Instead, databases generated using these techniques could be used to implement template-based algorithms, and moreover the algorithm could be used to implement effective peep-hole optimizations. Other more

scalable approaches to large-scale circuit optimization may also make use of some of the ideas presented here, if not necessarily the entire algorithm itself.

Along similar lines, our advantage over brute force searching could be used to speed up the Solovay-Kitaev algorithm. While methods exist for the optimal approximate synthesis of single qubit circuits, Solovay-Kitaev still remains the best method for the direct synthesis of approximate multi-qubit circuits. Unfortunately, the algorithm scales poorly to multiple qubits, due to the difficulty of computing and search through the initial ϵ-net. It's possible that by using our approximate synthesis procedure to implement the ϵ-net, multi-qubit Solovay-Kitaev could be a viable option for computing multi-qubit approximate circuits.

Other areas of future work could focus on improving these methods and implementations. In particular, there may be methods to store unitaries over particular gate sets efficiently, which would open up the option to store unitaries themselves, decreasing computation time. Better search space reductions may also be possible, particularly in the case of T-depth optimal searching, as the number of T-depth 1 circuits is significantly less than $|\mathcal{C}_n|^2|\mathcal{T}_n|$.

Chapter 5

Tpar: polynomial-time T-gate optimization

While the last chapter focused on quantum circuit optimization by the synthesis of minimal depth circuits, the technique was exponential in the number of qubits and circuit depth, making it impractical for large scale depth optimizations. Moreover, to optimize T-count and depth, the entire Clifford group – with size exponential the number of qubits squared [57] – was repeated at each iteration of the algorithm, making it difficult for more than 2 qubits. In this chapter, we instead consider the problem of scalable, approximate optimization of T-count and depth. This chapter is largely based on material presented in [66].

The advantages of optimizing for T-count and depth have been discussed earlier in this thesis, as well as numerous recent papers [18, 19, 22, 20]. In [18] and independently in [20] it was shown that different classes of circuits can be parallelized to T-depth 1 by adding ancilla qubits, which Figures 4.9a and 4.9b further demonstrate. However, these results focus on the optimization of small circuits and lower bounds on T-depth for restricted classes of circuits using unbounded ancillas. Moreover, they do not consider the question of optimizing T-count, which has a profound impact on circuit size.

To address these issues, we developed a re-synthesis circuit optimization algorithm that optimizes T-count and depth, which we call Tpar. The algorithm relies on the generation of a *sum over paths*-style representation [67, 68] of the target quantum circuit, expressed using Clifford and T gates. By calculating a circuit's sum over paths, we can cancel a large number of redundant T gates, then synthesize a new circuit applying the remaining T gates in parallel stages. Furthermore, the algorithm runs in time polynomial in the number of

T gates and qubits, and can efficiently utilize ancillas to further parallelize T gates.

A motivating example

To motivate the need for large-scale automated T-gate optimizations we begin with a simple example. Consider an implementation of the 4-bit Toffoli gate $\Lambda_3(X)$, shown in Figure 5.1.

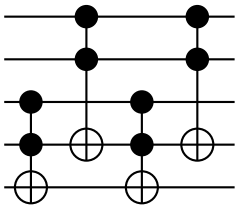


Figure 5.1: Implementation of a $\Lambda_3(X)$ gate [1].

If we replace two Toffolis with the implementation in Figure 4.14, and the other two with its inverse, we can remove 10 T gates by canceling identities and commuting T gates through the controls of $CNOT$ gates. Furthermore, we can trivially parallelize the remaining T gates to give a total T-depth of 10 (Figure 5.2).

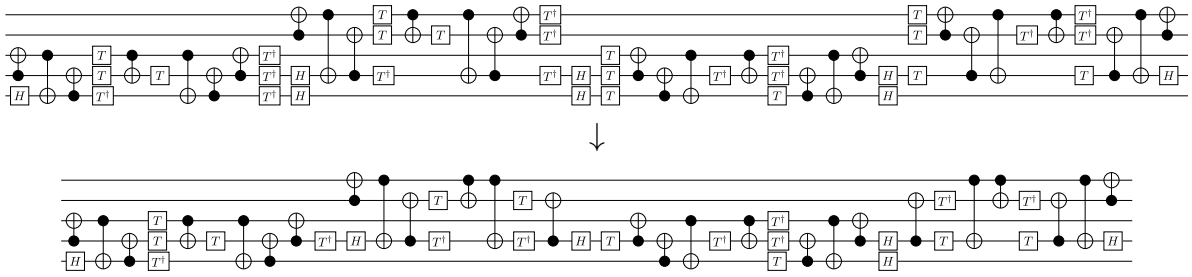


Figure 5.2: Optimized Clifford + T implementation of a $\Lambda_3(X)$ gate.

This small example clearly shows that remarkable reductions in T-count and T-depth can be made in full circuits, particularly those composed of Toffoli gates. However, the optimizations were performed by hand and are not practical for large circuits, so we would like an automated way to perform such optimizations. As described the optimizations are not particularly machine-amenable, given that they relied on the circuits used to implement

the Toffolis, followed by a sequence of applications of commutation rules and identities. While it's reasonable to think a computer could produce the same output with a simulated annealing type approach, this approach will have difficulty on larger circuits where phase gates may be far apart.

We can in fact do better by instead characterizing the circuit as a set of conditional phase factors and functions on the computational basis states. The theoretical advantage of such an approach is that redundant rotations would be immediately obvious from the fact that they have the same conditions. Furthermore, it would bypass dealing with circuit identities to move gates, instead allowing synthesis to schedule gates directly. Our algorithm takes this approach, allowing us to reproduce and go beyond the T-count and depth optimizations in Figure 5.2.

The construction of such an algorithm relies on the linearity of quantum mechanics to view circuits, gates or unitaries as functions mapping computational basis states (i.e. classical states $\mathbb{F}_2^n$) to some superposition of classical states with complex coefficients. Similar constructions [67, 68] have been connected with Feynman's sum over paths formulation of quantum mechanics [69] and used to deal with complexity-theoretic problems, though our motivation behind such a representation is based in the practical reconstruction of quantum circuits.

We begin by characterizing circuits composed of $CNOT$ and T gates, which as we show can be re-synthesized with optimal T-depth under the assumption that no "simpler" sum of paths implements the same unitary. Afterwards, we consider different methods for extending the re-synthesis to cover the universal Clifford + T gate set.

5.1 {CNOT, T} circuits

We first consider circuits over the gate set $\{CNOT, T, P = T^2, Z = T^4, T^\dagger = T^7, P^\dagger = T^6\}$, as they have a particular property that will be crucial to synthesizing low T-depth circuits. We usually omit the extraneous gates and refer to this gate set by its generating set $\{CNOT, T\}$. It can be observed that since $CNOT|xy\rangle = |x(y \oplus x)\rangle$ and $T|x\rangle = e^{i\pi/4}|x\rangle$, a $\{CNOT, T\}$ circuit can be described as a linear reversible function on the inputs, with an added phase that is some factor of $\omega = e^{i\pi/4}$. Furthermore, as the T gates are orthogonal to the $CNOT$ gates (in the sense that one applies to state-space, and the other applies to phase-space), each qubit is always in a state described by some linear Boolean function (i.e. an XOR) of the input variables; likewise, each T gate adds a factor of ω conditioned on a linear Boolean function. We can then tell exactly when T gates can be removed or replaced with a P or Z gate just by looking at the phase factors they add.

Stated more precisely,

Lemma 8. *A unitary $U \in U(2^n)$ is exactly implementable by an n-qubit circuit over $\{CNOT, T\}$ if and only if*

$$U|x_1 x_2 \dots x_n\rangle = \omega^{f(x_1,x_2,\dots,x_n)}|g(x_1, x_2, \dots, x_n)\rangle$$

where $f(x_1, x_2, \dots, x_n) = \sum_i c_i \cdot f_i(x_1, x_2, \dots, x_n)$ for some linear reversible function $g \in \mathbb{F}_2^n \to \mathbb{F}_2^n$ and linear Boolean functions $f_1, f_2, ..., f_k \in \mathbb{F}_2^{n*}$ with coefficients $c_1, c_2, ..., c_k \in \mathbb{Z}_8$.

Proof. The forward direction can be observed by writing the circuit implementing U as an alternating product of $CNOT$ and T circuits. Each $CNOT$ circuit computes a linear reversible function f on the inputs, while a following T gate on the ith qubit adds a global phase factor of $\omega^{f^i(x_1,x_2,\dots,x_n)}$ where f^i denotes the linear Boolean function corresponding to the ith output of f. If multiple T gates act on the same linear Boolean function, the multiplicity of the function in the phase is equal to the number of T gates acting on it. The overall effect of the circuit on the state is then a linear reversible function, completing the proof.

The reverse direction is equally simple by noting that for any linear Boolean function f_i on n inputs, f_i is an output of some linear reversible function on n inputs, and thus can be computed using only $CNOT$ gates [14]. By applying c_i T gates to the qubit with state $|f_i(a_1, a_2, ..., a_n)\rangle$ then uncomputing, the input state is recovered, with an added phase of $\omega^{c_i \cdot f_i(a_1,a_2,\dots,a_n)}$. This process can be repeated for each f_i. It then suffices to observe that g can be computed with $CNOT$ gates, thus U can be implemented with $CNOT$ and T gates. □

As an illustration, consider the circuit in Figure 5.3, implementing a doubly controlled Pauli-Z gate $\Lambda_2(Z)$ over $\{CNOT, T\}$ – recall that the doubly-controlled Z gate implements the transformation $\Lambda_2(Z) : |x_1x_2x_3\rangle \mapsto (-1)^{x_1 \wedge x_2 \wedge x_3}|x_1x_2x_3\rangle$. We can track the effect of each $CNOT$ gate on the state of the qubits (annotated in the circuit); when a T or $T^\dagger$ gate is applied, we record the state of the relevant qubit in a phase factor. As a result, the circuit implements the following transformation:

$$\Lambda_2(Z) : |x_1x_2x_3\rangle \mapsto \omega^{x_1+x_2+x_3-(x_1\oplus x_2)-(x_1\oplus x_3)-(x_2\oplus x_3)+(x_1\oplus x_2\oplus x_3)}|x_1x_2x_3\rangle.$$

In fact, since $2 \cdot (x \wedge y) = x + y - x \oplus y$ [20], we see that

$$\omega^{x_1+x_2+x_3-(x_1\oplus x_2)-(x_1\oplus x_3)-(x_2\oplus x_3)+(x_1\oplus x_2\oplus x_3)} = \omega^{4\cdot(x_1\wedge x_2\wedge x_3)}$$

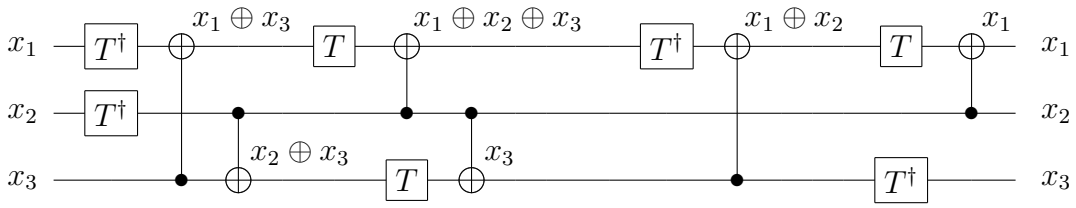


Figure 5.3: $\{CNOT, T\}$ circuit implementing the doubly controlled Z gate.

as expected.

As a result, we can fully characterize any unitary $U \in U(2^n)$ implementable by a $\{CNOT, T\}$ circuit with a set $S \subseteq \mathbb{Z}_8 \times \mathbb{F}_2^{n*}$ of linear Boolean functions together with coefficients[1] in $\mathbb{Z}_8$, and a (linear, reversible) output function $g : \mathbb{F}_2^n \to \mathbb{F}_2^n$, with the interpretation $U_{\langle S,g\rangle} : |x_1, x_2, ..., x_n\rangle \mapsto \omega^{\sum_{(c,f)\in S} c \cdot f(x_1,x_2,...,x_n)} |g(x_1, x_2, ..., x_n)\rangle$. In a sense all paths are simple in a $\{CNOT, T\}$ circuit since they do not introduce *path variables* [67], thus allowing the phase of the output state to only depend on the inputs of the circuit – we will later integrate it with operations that generate path variables, which will complicate re-synthesis.

Once such a characterization has been computed, it is clear from the proof of Lemma 8 that given S and g a circuit can be synthesized implementing $U_{\langle S,g\rangle}$ – however, the method described may end up with *worse* T-depth than the original circuit.

We can instead recall from Section 2.1 that if a set[2] $A \subseteq \mathbb{F}_2^{n*}$ is computable, we can construct a linear reversible function with $|A|$ of the outputs corresponding to the elements of A. The set S of linear Boolean functions can thus be partitioned into computable subsets, and a circuit can be synthesized by taking each partition $A \subseteq S$, computing a (reversible) superset of A with a stage of $CNOT$ gates, computing the relevant phase factors $\omega^{\sum_{(c,f)\in A} c \cdot f(x_1,x_2,...,x_n)}$, and finally uncomputing. As each $f_i \in A$ is one of the outputs of the $CNOT$ stage, the phase computations can be performed in parallel with at most $|A|$ T gates – under the assumption that $P, P^\dagger$ and Z are implemented directly as logical gates, this stage will have T-depth at most 1 since at most 1 T gate is required to apply a given phase factor.

As a trivial consequence, any unitary U implementable over $\{CNOT, T\}$ can be implemented in T-depth k where k is the minimum number of sets partitioning $S_T = \{(c, f) \in S | c \equiv 1 \mod 2\}$ into computable subsets – we don't include phases where $c \equiv 0 \mod 2$ as

[1] When referring to properties (and functions) pertaining to sets of linear Boolean functions, we will automatically lift these to properties defined on pairs of linear Boolean functions and coefficients.

[2] Recall the definition of $\mathbb{F}_2^{n*}$ as the *dual space* of $\mathbb{F}_2^n$, $\mathbb{F}_2^n \to \mathbb{F}_2$.

they can be applied without using T gates. While we haven't described how to find such partition yet, we digress for a moment and examine the effectiveness of such a method.

Given the set S we can observe that T-depth k is minimal, since any T-stage corresponds to a subset of S_T which must be reversibly computable. Likewise, the number of T gates, given by $|S_T|$, is also minimal with respect to the particular set of linear Boolean functions used. However, there may in general be some other equivalent expression of the phase using fewer T gates. Specifically, given S and g, there may be some set $S' \subseteq \mathbb{Z}_8 \times \mathbb{F}_2^{n*}$ such that $\sum_{(c,f)\in S} c \cdot f(x_1, x_2, \ldots, x_n) = \sum_{(c,f)\in S'} c \cdot f(x_1, x_2, \ldots, x_n)$ for all $x_1, x_2, \ldots, x_n \in \mathbb{F}_2$, implying that $U_{\langle S,g\rangle} = U_{\langle S',g\rangle}$. If $\sum_{(c,f)\in S'} c \mod 2 < \sum_{(c,f)\in S} c \mod 2$, synthesizing $U_{\langle S',g\rangle}$ will give a circuit with fewer T gates and possibly lower T-depth.

Consider, for example, the circuit in Figure 5.4 which has the effect

$$U : |x_1x_2x_3\rangle \mapsto \omega^{x_1+(x_1\oplus x_2)+x_2+(x_2\oplus x_3)}|x_1x_2x_3\rangle.$$

Since there are 4 functions and only 3 qubits, a minimal partitioning of this set has at least 2 partitions. However, we can note that $x_1 + (x_1 \oplus x_2) + x_2 + (x_2 \oplus x_3) = 2 \cdot x_1 + (x_1 \oplus x_2) + (x_1 \oplus x_3)$, which requires only 2 T gates to implement. We conjecture that finding an expression of the phase $\sum_{(c,f)\in S} c \cdot f(x_1, x_2, \ldots, x_n)$ minimizing $\sum_{(c,f)\in S} c \mod 2$ is NP-hard (see Appendix A).

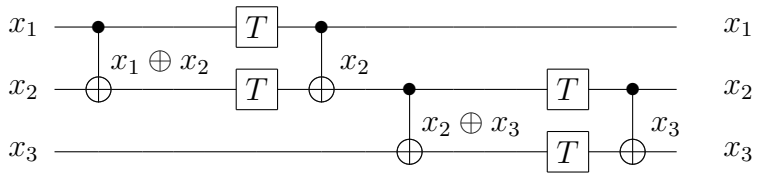


Figure 5.4: A circuit giving a non-optimal (in the T-count) phase expression.

For the remainder of this chapter we will not separate S into those phase factors that require a T gate and those that don't to avoid over complicating the presentation.

To return to the topic of synthesis, we first need an efficient method to determine whether a subset A of S is computable. In fact, by Definition 2, we know a set of linear Boolean functions A is computed by a linear reversible function on $\mathbb{F}_2^n$ if and only if there exists some superset A' of A such that $|A'| = n$ and $\text{rank}\,(A') = n$. In particular, we see that such a superset exists if and only if A is linearly independent, which can be efficiently tested. Notably, the T-depth is then given by a minimal partitioning of S into linearly independent sets.

The more interesting case is when the input state space is some subspace V of $\mathbb{F}_2^N$ with dimension n, which occurs when the circuit contains ancilla qubits, or if some qubits have

linearly dependent input states, e.g. $|x_1x_2(x_1 \oplus x_2)\rangle$. For any set $A = \{f_1, f_2, ..., f_k\}$, we can recall that A is computable over V if and only if we can add linear Boolean functions $f_{k+1}, \ldots, f_n$ to A so that rank $(\{f_1, f_2, \ldots, f_n\}) = n$. For A to be computable, the remaining $N-k$ functions must thus be able to make up the difference between rank(A) and n. This condition is formalized in the following lemma:

Lemma 9. *Given a subspace V of $\mathbb{F}_2^N$ with dimension n and set of linear Boolean functions $A \subseteq V^*$, there exists a superset A' of A with cardinality N such that* rank $(A') = n$ *if and only if*

$$n - \text{rank}\,(A) \leq N - |A|. \tag{5.1}$$

Proof. The proof of Lemma 9 follows from basic linear algebra. Suppose there exists a superset A' of A with cardinality N and rank$(A') = n$. Then $A' \setminus A$ contains at least $n - \text{rank}(A)$ linearly independent linear Boolean functions, and so $|A' \setminus A| = N - |A| \geq n - \text{rank}(A)$.

Suppose instead that $n - \text{rank}(A) \leq N - |A|$. Then any maximal linearly independent subset of A can be extended to a basis of V by adding $n - \text{rank}(A)$ linear Boolean functions (i.e., vectors in V^*). As a result there exists a rank n superset A' of A with cardinality $|A'| = |A| + n - \text{rank}(A) \leq N$, which suffices to show that a rank n superset exists with cardinality N. □

We can note that inequality (5.1) implies $|A| = \text{rank}(A)$, i.e., A is linearly independent, when $N = n$.

To illustrate these points, consider again the circuit in Figure 5.4. Recall that we can describe the computed unitary, with 3 inputs and no ancillas ($N = n = 3$), by

$$U : |x_1x_2x_3\rangle \mapsto \omega^{x_1+(x_1\oplus x_2)+x_2+(x_2\oplus x_3)}|x_1x_2x_3\rangle.$$

The T-stages in Figure 5.4 correspond to the partition $\{\{x_1, x_1 \oplus x_2\}, \{x_2, x_2 \oplus x_3\}\}$. While this partition is already minimal in size, we could instead synthesize a circuit using the partition $\{\{x_1, x_1 \oplus x_2, x_2 \oplus x_3\}, \{x_2\}\}$, since rank$(\{x_1, x_1 \oplus x_2, x_2 \oplus x_3\}) = 3$ and there exists a size 3 superset $\{x_1, x_2, x_3\} \supset \{x_2\}$ with rank 3. Conversely, $\{\{x_1, x_1 \oplus x_2, x_2\}, \{x_2 \oplus x_3\}\}$ cannot be synthesized, as rank$(\{x_1, x_1 \oplus x_2, x_2\}) = 2$, or more intuitively since the transformation $|x_1x_2x_3\rangle \mapsto |x_1(x_1 \oplus x_2)x_2\rangle$ is irreversible.

If instead we wanted to add an ancilla when re-synthesizing Figure 5.4, we could write the circuit in T-depth 1. We already know that $|x_1x_2x_3\rangle \mapsto |x_1(x_1 \oplus x_2)(x_2 \oplus x_3)\rangle$ is a reversible transformation, so $|x_1x_2x_3\rangle|0\rangle \mapsto |x_1(x_1 \oplus x_2)(x_2 \oplus x_3)\rangle|0\rangle$ is reversible as well.

We can then compute x_2 into the ancilla with two $CNOT$ gates, since $0 \oplus x_1 \oplus (x_1 \oplus x_2) = x_2$, and thus apply all 4 T gates at the same time. To connect this intuitive idea with the condition (5.1), we observe that $N = 4$, $n = 3$ since the input $|x_1x_2x_3\rangle|0\rangle$ spans a space of dimension 3, and $\text{rank}(\{x_1, x_1 \oplus x_2, x_2 \oplus x_3, x_2\}) = 3$, so

$$n - \text{rank}(\{x_1, x_1 \oplus x_2, x_2 \oplus x_3, x_2\}) = 0 \leq 0 = N - |\{x_1, x_1 \oplus x_2, x_2 \oplus x_3, x_2\}|.$$

Figure 5.5 shows a circuit computing U in T-depth 1.

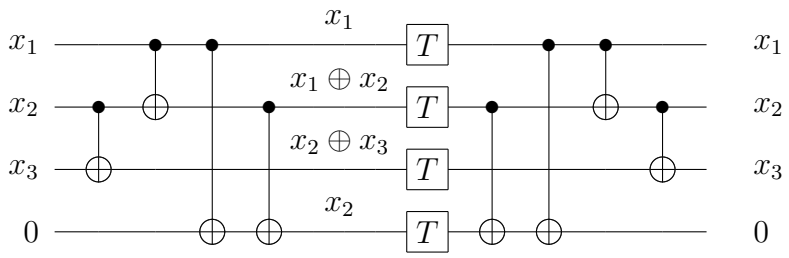


Figure 5.5: T-depth 1 implementation of Figure 5.4 with one ancilla.

5.2 Matroids

We now turn our attention to the problem of determining a minimal partition of a set of linear Boolean functions into computable sets.

While if we want to find a minimal partition with ancillas we can't easily phrase the problem as partitioning vectors into linearly independent sets, we are able to phrase the problem as an instance of the more general *matroid partitioning* problem. To do so, we first introduce the concept of a *matroid*, an algebraic structure that generalizes the idea of linear independence in vector spaces.

Definition 7. A finite matroid is a pair (S, I) where S is a finite set and I is a set of subsets of S such that

1. $\emptyset \in I$.
2. For all $A, B \subset S$, if $A \in I$ and $B \subset A$, then $B \in I$.
3. For all $A, B \in I$, if $|A| > |B|$, then there exists some $a \in A$ such that $B \cup \{a\} \in I$.

As we have a condition that behaves like linear independence in the case when the circuit has no ancillas, it is natural to think that the inequality (5.1) may define an independence relation. It turns out that a set of linear Boolean functions, together with an independence relation defined by the equality (5.1), forms a matroid:

Lemma 10. *For any subspace V of $\mathbb{F}_2^N$ with dimension n and set of linear Boolean functions $S = \{f_1, f_2, \ldots, f_k\} \subseteq V^*$, let I denote the set*

$$\{A \subseteq S | n - \operatorname{rank}(A) \leq N - |A|\}.$$

The pair (S, I) is a finite matroid.

Proof. We verify that (S, I) satisfies all three conditions of Definition 7.

1. $n - \operatorname{rank}(\emptyset) \leq N - |\emptyset|$ is trivially true since $n \leq N$. Thus $\emptyset \in I$.

2. Suppose $A, B \subset S$, where $A \in I$ and $B \subset A$. Clearly $\operatorname{rank}(B) \geq \operatorname{rank}(A) - (|A| - |B|)$, so $\operatorname{rank}(A) - \operatorname{rank}(B) \leq |A| - |B|$. Since $n - \operatorname{rank}(A) \leq N - |A|$ we see that

   $$n \leq N + \operatorname{rank}(A) - |A| \leq n + \operatorname{rank}(B) - |B|$$

   and thus $n - \operatorname{rank}(B) \leq N - |B|$.

3. Suppose $A, B \in I$ and $|A| > |B|$. If $\operatorname{rank}(A) \leq \operatorname{rank}(B)$, then

   $$n - \operatorname{rank}(B) \leq n - \operatorname{rank}(A) \leq N - |A| < N - |B|$$

   and thus $n - \operatorname{rank}(B \cup \{s\}) \leq N - |B \cup \{s\}|$ for any $s \in A$.

   Otherwise, $\operatorname{rank}(A) > \operatorname{rank}(B)$ and we can let A' and B' be maximal linearly independent subsets of A and B, respectively. Since $A' \not\subseteq \operatorname{span}(B')$, for any $s \in A \setminus \operatorname{span}(B')$, $B' \cup \{s\}$ is linearly independent. Then

   $$\begin{aligned} n - \operatorname{rank}(B' \cup \{s\}) &= n - \operatorname{rank}(B \cup \{s\}) \\ &= n - \operatorname{rank}(B) - 1 \\ &\leq N - |B| - 1 \\ &= N - |B \cup \{s\}|. \end{aligned}$$

□

5.2.1 Matroid partitioning

The *matroid partitioning* problem can be defined as follows

Definition 8. (Matroid partitioning)

Given a matroid (S, I), find a partition $\{A_1, A_2, \ldots, A_k\}$ of S such that $A_i \in I$ for each $1 \leq i \leq k$ and for any other partition $\{A'_1, A'_2, \ldots, A'_{k'}\}$ into independent subsets, $k' \geq k$.

Of particular interest is the fact that matroid partitioning can be solved in polynomial time, given an independence oracle for the matroid [70]. As a consequence of Lemmas 9 and 10, we see that the problem of finding a minimal partition of a set of linear Boolean functions into computable sets is reducible to the matroid partitioning problem, and thus can be solved in polynomial time given an independence oracle. Since the condition in Lemma 9 can be checked for a given subset A by using Gaussian elimination to compute the matrix rank, we thus see that a minimal partition of S into computable subsets can be computed in polynomial time. Specifically, the independence test requires at most[3,4] $O(N^3)$ time. The rest of this section describes an algorithm for computing a minimal partition.

The algorithm we use for solving the matroid partitioning problem is based on an algorithm due to Edmonds [70]. Given a matroid (S, I) and a minimal (matroid) partition P of $S' \subset S$, we take an element $s \in S \setminus S'$ not already partitioned and construct a minimal partition of $S' \cup \{s\}$. To create the new partition, we construct a directed graph G_s containing a vertex u for every $u \in S' \cup \{s\}$ as well as a vertex $\perp_p$ for every subset $p \in P$. The edges of G_s represent changes to the partition that are invariant under the property of each subset being independent. In particular, for any $u, v \in S' \cup \{s\}$ there is a directed edge $v \to u$ in G_s if and only if u is contained in some subset $p \in P$ and $(p \setminus \{u\}) \cup \{v\} \in I$, i.e. v can be added to p if we remove u. Additionally, given a subset $p \in P$ and element $u \in S' \cup \{s\}$, there exists an edge $u \to \perp_p$ if and only if $p \cup \{u\} \in I$. A path from s to $\perp_p$ for some subset p gives a set of updates to P that produce a valid partition P' of $S' \cup \{s\}$. Likewise, if there is no such path, there is no partition of size $|P|$ partitioning $S' \cup \{s\}$ [70], and so a new subset $\{s\}$ is added to P.

Rather than generating the graph G_s explicitly for each element s, we try to construct a path from s to some $\perp_p$ breadth-first (Algorithm 5). It is well known that the time

[3]We can reduce this to $O(n^2 N)$ in practice by storing vectors in V^* as length $\dim(V) = n$ vectors. The $O(N^3)$ bound is used for simplicity.

[4]Gaussian elimination cannot generally be bounded this way, as the representation of individual elements may grow exponentially as rows get added together. Since our arithmetic is performed over the finite field $\mathbb{F}_2$, we avoid this exponential blowup in the size of individual elements.

complexity of breadth-first search is $O(|E|+|V|)$ for a graph with edge set E and vertices V. We can note that there are $|S'|+|P|+1$ vertices and at most $|S'|^2+|P|\cdot(|S'|+1)+(|S'|+|P|)$ edges in G_s, as well as the fact that $|P| \leq |S'|$. Since each edge requires a single test for independence in $O(N^3)$ time, the breadth first search requires time in $O(|S'|^2 \cdot N^3 + |S'|)$.

Algorithm 5 Matroid partitioning algorithm

```
function PARTITION(s, P, (S, I))
    /* I denotes the independence oracle, P is a minimal partition */
    Create path queue Q, Q.enqueue(s → ∅)
    Unmark each element of S, mark s
    while Q non-empty do
        t := Q.dequeue()
        for each A ∈ P do
            Set A' := A ∪ {head(t)}
            if A' ∈ I then
                Set A := A'
                for each u → v in path t do
                    Replace u with v in its current partition
                end for
            else
                for each unmarked u ∈ A do
                    if A' \ {u} ∈ I then
                        Q.enqueue(u → t)
                        Mark u
                    end if
                end for
            end if
        end for
    end while
    If no path was found, set P := P ∪ {s}
end function
```

Algorithm 5 details the algorithm for adding an element s to a partition P of matroid (S, I) – the full algorithm follows by iteratively adding each element of the ground set to the initially empty partition, and correctness follows from the property that if P is minimal for (S, I), then the new partition P' is minimal for $(S \cup \{s\}, I)$.

Since adding a single element to a partition of i elements takes $O\left((2i)^2 \cdot N^3 + 2i\right)$ time,

and $\sum_i^{|S|} i^2 = \frac{|S|^3}{3} + \frac{|S|^2}{2} + \frac{|S|}{6}$, we see that Algorithm 5 can be used to partition the full set S of linear Boolean functions in $O(|S|^3 \cdot N^3)$ time.

5.3 Towards a universal gate set

In the previous sections we described (in pieces) a method for re-synthesizing $\{CNOT, T\}$ circuits while removing redundant T-gates and parallelizing the remaining ones. Specifically, given a circuit we can compute $\langle S, g \rangle$ and a new circuit computing $U_{\langle S,g \rangle}$ can be synthesized by first partitioning S into computable subsets using Algorithm 5, then completing each subset by adding elements to make them reversible. Many efficient algorithms exist [71, 14, 40] that can decompose the resulting linear reversible function into $CNOT$ gates – in our implementation we use a Gaussian elimination algorithm – so that each subset can be computed, followed by a layer of T gates to apply the phase factors, then uncomputed. To complete the procedure, we synthesize another linear reversible circuit computing g.

However, the usefulness of such an algorithm on its own is marred by the fact that $\{CNOT, T\}$ circuits are an extremely restricted class of quantum circuits – in particular, they do not create superpositions or interference between basis states. To apply the optimization procedure to interesting quantum circuits, we have to extend these ideas to deal with a universal gate set; this section details the different approaches we developed.

5.3.1 Embedded {CNOT, T} optimization

The most straightforward and obvious way of applying $\{CNOT, T\}$ re-synthesis to a universal gate set is to optimize only subcircuits. For instance, an optimization procedure could traverse a circuit, gather the largest volume $\{CNOT, T\}$ subcircuit that hasn't yet been optimized, then replace it with the re-synthesized circuit. However, finding the largest volume $\{CNOT, T\}$ subcircuit is a non-trivial problem on its own, and each iteration would generally require traversing the whole circuit again, which we would like to avoid.

A slight relaxation to make such an algorithm more practical is to take a greedy approach. In this approach, subcircuits are greedily constructed by traversing the circuit and repeatedly adding $\{CNOT, T\}$ gates to the current subcircuit until a non-$\{CNOT, T\}$ gate occurs. The most glaring drawback is that T gates cannot commute across subcircuit boundaries, removing a great deal of optimization potential. For instance, the optimizations in the motivating example (Figure 5.1) are no longer available, since Hadamard gates

separate the individual $\{CNOT, T\}$ circuits corresponding to doubly controlled-Z gates – as the $\Lambda_2(Z)$ gates already have minimal T-depth and count, this effectively removes any possibility of optimization.

Given this drawback, it is perhaps surprising that our experimental results in Section 5.5 show very good performance on many arithmetic circuits. In fact, this method works very well in circuits where nearby Toffolis do not mix control bits with target bits, since the Hadamards can be pushed outwards in these cases. Moreover, a great deal of T gate reductions are available when adjacent Toffolis share controls and/or targets. As a result, embedding $\{CNOT, T\}$ re-synthesis in general quantum circuits can be very effective under certain conditions.

5.3.2 Abstract Hadamard gates

Motivated by the drawbacks of the embedded re-synthesis approach, we can consider ways to optimize directly over a universal gate set, specifically by extending the previous techniques to include Hadamard gates.

As in $\{CNOT, T\}$ re-synthesis, the goal is to describe the phase factors in the output as (linear Boolean) functions of an arbitrary input basis state. A new circuit can then be synthesized applying those phase factors as parallel as possible. In the embedding of $\{CNOT, T\}$ re-synthesis into a universal gate set, each $\{CNOT, T\}$ subcircuit is interpreted as having a completely new input state, so the state space of any two subcircuits is disjoint. We can do better by more accurately tracking the state space of different $\{CNOT, T\}$ subcircuits, which would potentially allow phase factors to cancel and commute between them. To do so we need to know how the state space changes after application of a Hadamard gate.

We can observe that a Hadamard gate has the effect

$$H : |x\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{x' \in \mathbb{F}_2} (-1)^{x \cdot x'} |x'\rangle.$$

We call x' a *path variable*, which represents two possible paths in the final state. Since we only want the effect of the Hadamard gate on the basis state, we drop the extraneous information[5] and describe its action as

$$H' : |x\rangle \mapsto |x'\rangle, \; x' \in \mathbb{F}_2.$$

[5]The phase and magnitude are implicitly recovered in the synthesis procedure, so we do not need to keep track of it.

In effect, a Hadamard gate destroys the qubit's previous state and produces a new arbitrary state. All the non-Hadamard phase factors in a $\{H, CNOT, T\}$ circuit with N qubits and k Hadamard gates can then be described by $c \cdot f(x_1, x_2, ..., x_{N+k})$ for some linear Boolean function f and $c \in \mathbb{Z}_8$.

As an illustration, the Toffoli gate implementation in Figure 5.6 given the initial input $|x_1 x_2 x_3\rangle$ generates two free variables x_4, x_5, shown in the figure. The total phase given by the T gates can then be described as

$$\omega^{x_1+x_2+x_4+7\cdot x_1 \oplus x_2+7\cdot x_1 \oplus x_4+7\cdot x_2 \oplus x_4+x_1 \oplus x_2 \oplus x_4}.$$

The output state is then described as $|x_1 x_2 x_5\rangle$. Furthermore, the state space before the first H, for example, is spanned by $\{x_1, x_2, x_3\}$, so the phases conditioned on x_1, x_2, and $7 \cdot x_1 \oplus x_2$ could be applied *before* the first Hadamard when re-synthesizing, while all the other phases require x_4 and thus must be applied before the second Hadamard gate.

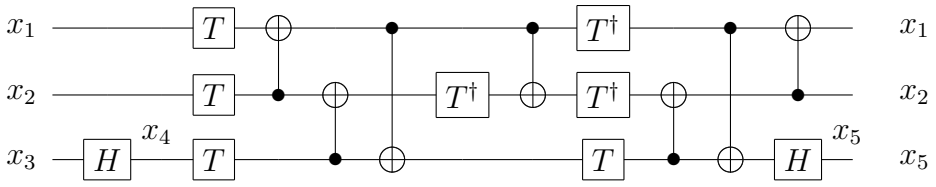


Figure 5.6: Clifford + T implementation of the Toffoli gate with the target on qubit 3.

As noted before, the abstraction H' is only used to more accurately describe the state space of different sub-circuits, which are then individually re-synthesized; the difference from embedded $\{CNOT, T\}$ optimization is that *all* phase factors are known at the same time, so T gates in separate subcircuits can be combined and/or assigned to different subcircuits to further minimize T-depth. Specifically, after each Hadamard gate the state space will be some subspace V of $\mathbb{F}_2^{N+k}$, which is distinct from the subspace after any other Hadamard gate. Each function $f(x_1, x_2, ..., x_{N+k})$ can then be computed in a subcircuit with state space V if and only if $f \in V^*$. The synthesis procedure can then decide in which subcircuit to apply each phase factor, in order to achieve efficient T-depth.

One approach to parsing out phase factors is to use the greedy nature of Algorithm 5 to maintain a partition of those phase factors in S that are currently in the state space of the circuit. We can iterate through the subcircuits, and for each one, partition any elements of S that are not already partitioned and are in the state space of the subcircuit – this step relies on the fact that Algorithm 5 is greedy to avoid having to partition everything again from scratch. As some of the partitioned functions may not be in the state space after the

next Hadamard gate, we then assign the phase factors in any partition block containing such a function to the current subcircuit and remove that block from the partition.

The process of removing partitions and partitioning new elements requires a closer look to ensure that the partition is actually a minimal matroid partition at every step. Fortunately, it is straightforward to observe that any subset P' of a minimal matroid partition P is also minimal for the remaining elements – if there existed a partition P_0 of the elements in P' such that $|P_0| < |P'|$, then $P_0 \cup P \setminus P'$ is a partition of the elements in P into fewer sets.

One problem arises in that the dimension of the state space may increase in the next subcircuit (e.g. if the Hadamard is applied to an ancilla qubit). In this case, the independence condition (5.1) of the matroid changes, and previous partitions may now be invalid under the new inequality. However, as a trivial consequence of the fact that the dimension increases by at most 1, a partition that is no longer independent can be modified to satisfy it by removing exactly one element. Furthermore, if all partitions are modified to satisfy the new independence condition in this way, the new partition is minimal.

Lemma 11. *Given a subspace V of $\mathbb{F}_2^N$ with dimension $n < N$ and a set of linear Boolean functions $S \subseteq V^*$, let $I_i = \{A \subseteq S | i - \text{rank}\,(A) \leq N - |A|\}$.*

If P is some minimal partition of (S, I_n), then the partition P' defined by removing one linearly dependent element from every $A \in P$ if $A \notin I_{n+1}$ is a minimal partition of (S', I_{n+1}), where $S' = \cup_{A \in P} A$ is the set of elements partitioned by P'.

Proof. Suppose there exists some partition P_0 of (S', I_{n+1}) with $|P_0| < |P'|$.

We first see that one element of $S \setminus S'$ can be added to any $A \in P_0$ to give a set in I_n. In particular, consider any $A \in P_0$. Since $n + 1 - \text{rank}(A) \leq N - |A|$ we see $n - \text{rank}(A) \leq N - |A| - 1 = N - |A \cup \{s\}|$[6] for any $s \in S \setminus S'$. Thus for any $A \in P_0, s \in S \setminus S'$ we have $A \cup \{s\} \in I_n$.

Next we note that for any $T \subseteq S$ there exists a partition of (T, I_n) with size at most $|T| - 1$. This is a simple result of the fact that $n < N$, as any subset $A \subseteq T$ of size 2 has rank at least 1, so $n - \text{rank}(A) \leq n - 1 \leq N - 2 = N - |A|$. Additionally, any size 1 subset of T is trivially independent under I_n.

Thus since we can add one element to every partition in P_0, and we can partition the remaining $|S \setminus S'| - |P_0|$ elements into at most $|S \setminus S'| - |P_0| - 1$ partitions, we see that

[6] We also note that $N - |A \cup \{s\}| \geq 0$ as required, since any subset of S has rank at most n, i.e. $\forall A \in I_{n+1}$, $n + 1 - \text{rank}(A) > 0$ and thus $|A| < N$.

there exists a partition of (S, I_m) with size at most $|P_0| + (|S \setminus S'| - |P_0| - 1) = |S \setminus S'| - 1$. Since $|S \setminus S'| - 1 < |P|$ we obtain a contradiction. □

5.3.3 Summing over paths

The last section implied a representation of $\{H, CNOT, T\}$ circuits using linear Boolean functions with k free variables over $\mathbb{F}_2^{N+k}$, but the Hadamard gates were left in place and only their effect on the state space was tracked. We can instead re-synthesize the entire circuit by fully describing the effect of a Hadamard gate,

$$H : |x\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{x' \in \mathbb{F}_2} \omega^{4 \cdot x \cdot x'} |x'\rangle,$$

leading to a complete algebraic representation of the circuit.

Lemma 12. *If unitary $U \in U(2^N)$ is exactly implementable by an N-qubit circuit over $\{H, CNOT, T\}$ with k H gates then for any computational basis state $x_1 x_2 ... x_N$,*

$$U : |x_1 x_2 ... x_N\rangle \mapsto \frac{1}{\sqrt{2^k}} \sum_{x_{N+1} ... x_{N+k} \in \mathbb{F}_2^k} \omega^{p(x_1, x_2, ..., x_{N+k})} |y_1 y_2 ... y_N\rangle$$

where $y_i = h_i(x_1, x_2, \ldots, x_{N+k})$ for some linear Boolean functions h_i, and

$$p(x_1, x_2, ..., x_{N+k}) = \sum_i c_i \cdot f_i(x_1, x_2, ..., x_{N+k}) + 4 \cdot \sum_{i=1}^{k} x_{N+i} \cdot g_i(x_1, x_2, ..., x_{N+k})$$

for some linear Boolean functions f_i, g_i and coefficients $c_i \in \mathbb{Z}_8$.

Proof. Follows from the effect of each gate on the computational basis states. □

Lemma 12 tells us that we can represent the unitary computed by a circuit over $\{H, CNOT, T\}$ just with a polynomial $p(x_1, x_2, ..., _{N+k})$ in mixed arithmetic over $\mathbb{Z}_2$ and $\mathbb{Z}_8$, and a set of linear Boolean functions for the outputs $y_1, y_2, ..., y_N$. A similar sum over paths representation of this gate set was briefly discussed in [67] as a reduction from evaluating quantum circuits to counting solutions to the polynomials. It's worth noting that unlike Lemma 8, the converse is not true – some polynomials will not correspond to a unitary transformation, as some are not even reversible. We leave testing of unitarity for a given expression of the type in Lemma 12 as an open question.

As in the last section, the synthesis procedure will have to make a "tour" through the various subspaces of $\mathbb{F}_2^{N+k}$ by applying Hadamard gates to instantiate the path variables and pick up the phase factor $\omega^{4 \cdot x_{N+i} \cdot g_i(x_1, x_2, ..., x_{N+k})}$. The difference is that the path is not fixed ahead of time, and so the synthesis procedure will need to reconstruct a sequence of Hadamard gates so that the circuit can transition through the state spaces and pick up each phase factor. In doing so, it's possible that an ordering that allows more parallelization may be chosen. We deal with the problem of choosing such a path in this section.

More precisely, we want to synthesize a $\{H, CNOT\}$ skeleton of a full circuit implementing U, specifically the unitary

$$U_0 : |x_1 x_2 ... x_N\rangle \mapsto \frac{1}{\sqrt{2^k}} \sum_{x_{N+1} ... x_{N+k} \in \mathbb{F}_2^k} (-1)^{\sum_{i=1}^{k} x_{N+i} \cdot g_i(x_1, x_2, ..., x_{N+k})} |y_1 y_2 ... y_N\rangle.$$

Once such a circuit is constructed we can add in the necessary phase factors between the Hadamard gates using the same method as in Section 5.3.2. Additionally, we assume that we are re-synthesizing a circuit, i.e. the polynomial $p(x_1, x_2, ..., x_{N+k})$ and outputs $y_1, y_2, ..., y_N$ were generated from a real circuit. We leave open the question of synthesis for arbitrary polynomial representations.

We begin by describing how to reconstruct the path from the original circuit, working backwards from the output state. Let $V_i \subseteq \mathbb{F}_2^{N+k}$ be the state space of the original circuit after the ith Hadamard gate. Since $x_{N+i} \in V_i$, we can choose a basis for V_i with respect to the standard basis of $\mathbb{F}_2^{N+k}$ as $B_i \cup \{x_{N+i}\}$, where no vector in B_i contains x_{N+i}. Furthermore, $B_i \cup \{g_i(x_1, x_2, ..., x_{N+k})\}$ also forms a basis of V_{i-1}. Given N qubits in states $\{b_1, b_2, ..., b_N\}$ forming a basis of V_i, we can then synthesize a circuit transforming the state space from V_{i-1} to V_i and picking up the correct phase: we first synthesize $V : |b_1 b_2 ... b_{N-1}\rangle \mapsto \left|x_{N+i} b'_1 ... b'_{N-1}\right\rangle$, where each of $b'_1, b'_2, ..., b'_N$ does not contain x_{N+i} in the standard basis, and then implement $V^\dagger (H \otimes I^{\otimes N-1})$, which has the effect

$$\left|g_i(x_1, x_2, ..., x_{N+k}) b'_1 ... b'_{N-1}\right\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{x_{N+i} \in \mathbb{F}_2} (-1)^{x_{N+i} \cdot g_i(x_1, x_2, ..., x_{N+k})} |b_1 b_2 ... b_N\rangle.$$

As the outputs $\{y_1, y_2, ..., y_N\}$ form a basis of V_k, we can thus synthesize a circuit implementing U' by iteratively synthesizing circuits transforming V_{i-1} to V_i.

At the moment the only advantage of this synthesis method is that the circuit can be more concisely represented using $p(x_1, x_2, .., x_{N+k})$ and set of output functions with no impact on the quality of the re-synthesized circuit. However, this method could also be used to parallelize or otherwise commute Hadamards as needed. Specifically, the method

works because the state space after every Hadamard contains at least one elementary vector x_{N+i}, and so we could create a basis diagonal in x_{N+i} allowing a circuit to be synthesized generating x_{N+i}. If the state space contains more than one such elementary vector, we could instead choose a basis that is diagonal in more than one path variable, and thus apply Hadamard gates in parallel. Care is still needed, as there may be phase factors that can only be computed between the Hadamard gates, but the flexibility to parallelize Hadamard gates could potentially lead to lower T-depths and lower total depths.

5.4 The Tpar algorithm

Up until now we've described the general ideas behind some T-gate optimization algorithms – in this section we present a concrete algorithm, Tpar, for optimizing T-count and depth in Clifford + T circuits, based on the representation described in Section 5.3.2. A version of the algorithm optimizing just $\{CNOT, T\}$ subcircuits, as described in Section 5.3.1, can be constructed by applying Tpar to subcircuits between Hadamard gates.

Recall that in the computational basis, the input space of a circuit with N qubits, $N-n$ ancillas and k Hadamard gates is a dimension n subspace V of $\mathbb{F}_2^{N+k}$. We required that all ancillas are initialized in state $|0\rangle$. We represent the state of a qubit as a linear Boolean function f defined on $\mathbb{F}_2^{N+k}$, or more succinctly as a vector in the dual space of $\mathbb{F}_2^{N+k}$, $\mathbb{F}_2^{(N+k)*}$. In contrast to previous discussions we also maintain a Boolean value b specifying the parity of the qubit, so that the full state is given by $b \oplus f(x_1, x_2, ..., x_{N+k})$ – as we will see, the parity corresponds to bit flips on the qubit.

Given a Clifford + T circuit C, written over $\mathcal{G} = \{X, Y, Z, P, P^\dagger, H, CNOT, T, T^\dagger\}$, we compute an abstraction $\langle S, Q, H\rangle$ of C where $S = \{(c, f) | c \in \mathbb{Z}_8, f \in \mathbb{F}_2 \times \mathbb{F}_2^{(N+k)*}\}$ represents the terms in the exponent of $\omega = e^{i\frac{\pi}{4}}$ as linear Boolean functions with a parity bit and multiplicity, $Q = (g_1, g_2, ..., g_N), g_i \in \mathbb{F}_2 \times \mathbb{F}_2^{(N+k)*}$ tracks the current state of each qubit, and $H = (h_1, h_2, ...h_k)$ gives a list of Hadamard gates, where each Hadamard gate h_i stores the input and output states, $h_i.Q_I$ and $h_i.Q_O$ respectively. We define the initial state for the circuit as $Q_0 = ((0, x_1), (0, x_2), ..., (0, x_n), (0, 0), ...)$, i.e. the first n qubits have initial states $|x_1\rangle, |x_2\rangle, ..., |x_n\rangle$ and the remainder are initialized as $|0\rangle$. To compute $\langle S, Q, H\rangle$ we begin with $\langle \emptyset, Q_0, \emptyset\rangle$ and apply the updates shown in Figure 5.7 sequentially for each gate in the circuit.

We define $S \uplus T$ as the union of S and T where any f such that $(c_1, f) \in S, (c_2, f) \in T$ is assigned coefficient $c_1 + c_2 \mod 8$. In this way phases applied to the same state are added, potentially reducing the total number of T gates. Furthermore, we can note that

$$
\begin{aligned}
update(X_i, \langle S, Q, H\rangle) &= \langle S, Q|_{g_i:=\neg g_i}, H\rangle \\
update(Z_i, \langle S, Q, H\rangle) &= \langle S \uplus \{(4, g_i)\}, Q, H\rangle \\
update(Y_i, \langle S, Q, H\rangle) &= \langle S \uplus \{(4, g_i), (2, \mathbf{0})\}, Q|_{g_i:=\neg g_i}, H\rangle \\
update(P_i, \langle S, Q, H\rangle) &= \langle S \uplus \{(2, g_i)\}, Q, H\rangle \\
update(P_i^\dagger, \langle S, Q, H\rangle) &= \langle S \uplus \{(6, g_i)\}, Q, H\rangle \\
update(H_i, \langle S, Q, (h_1, h_2, ..., h_i)\rangle) &= \langle S, Q', (h_1, h_2, ..., h_i, \{Q_I = Q, Q_O = Q'\})\rangle \\
&\qquad \text{where } Q' = Q|_{g_i:=a_{|H|}} \\
update(CNOT_{(i,j)}, \langle S, Q, H\rangle) &= \langle S, Q|_{g_j:=g_i \oplus g_j}, H\rangle \\
update(T_i, \langle S, Q, H\rangle) &= \langle S \uplus \{(1, g_i)\}, Q, H\rangle \\
update(T_i^\dagger, \langle S, Q, H\rangle) &= \langle S \uplus \{(7, g_i)\}, Q, H\rangle
\end{aligned}
$$

Figure 5.7: Gate update rules. A gate U_i denotes gate U applied to qubit i, $CNOT_{(i,j)}$ specifies i as the control qubit and j as the target.

bit flips on a state $g_i = (b, f)$ which naturally correspond to a negation of the state $b \oplus f(x_1, x_2, ..., x_{N+k})$ can be associated with the parity:

$$\neg(b \oplus f(x_1, x_2, ..., x_{N+k})) = 1 \oplus b \oplus f(x_1, x_2, ..., x_{N+k}) = \neg b \oplus f(x_1, x_2, ..., x_{N+k}).$$

So to apply a bit flip $\neg g_i$, we change the parity of the qubit rather than allowing negative occurrences of variables in linear Boolean functions.

The Tpar algorithm (Algorithm 6) proceeds as follows: after computing $\langle S, Q, H\rangle$, we iterate through the Hadamard gates in H, updating a partition P of the functions of S that are computable in the current subcircuit. In particular, we divide S into S_P and S_{-P}, where S_P are the already partitioned elements and S_{-P} are those not already partitioned. For a given $h_i = (Q_I, O_O)$, we check whether $f \in \text{span}(Q_I)$ for every $(c, f) \in S_{-P}$, and if so partition (c, f) using Algorithm 5 with the independence relation[7]

$$A \subseteq S \in I \text{ if and only if } \text{rank}(Q_I) - \text{rank}(A) \leq N - |A|.$$

After partitioning, we update S_P and S_{-P} accordingly.

[7] As before we implicitly lift the definition of rank for sets of linear Boolean functions to sets of linear Boolean functions with parity and coefficient.

Algorithm 6 T-parallelization algorithm

function TPAR(Clifford + T circuit C)
  Circuit:= $\emptyset$
  Compute $\langle S, Q, H\rangle$ for C
  Set $S_P := S;\ S_{-P} := \emptyset;\ P := \emptyset$
  for each $1 \leq i \leq k$ **do**
    $I := \{A \subseteq S | \operatorname{rank}(h_i.Q_I) - \operatorname{rank}(A) \leq N - |A|\}$
    for each $(c, f) \in S_{-P}$ **do**
      if $f \in \operatorname{span}(h_i.Q_I)$ **then**
        $P :=$Partition$((c, f), P, (S_P, I))$
        $S_P := S_P \cup \{(c, f)\};\ S_{-P} := S_{-P} \setminus \{(c, f)\}$
      end if
    end for
    for each $A \in P$ **do**
      if $i = k$ or $\exists (c, f) \in A$ such that $f \notin \operatorname{span}(h_i.Q_O)$ **then**
        Append(Circuit, Synthesize$(A, h_i.Q_I, h_i.Q_I)$)
        $P := P \setminus A$
      else if $\operatorname{rank}(h_i.Q_O) - \operatorname{rank}(A) > N - |A|$ **then**
        Choose $(c, f) \in A$ such that $\operatorname{rank}(A) = \operatorname{rank}(A \setminus \{(c, f)\})$
        $A := A \setminus \{(c, f)\};\ S_P := S_P \setminus \{(c, f)\};\ S_{-P} := S_{-P} \cup \{(c, f)\}$
      end if
    end for
    Append(Circuit, Synthesize$(\emptyset, h_i.Q_I, h_i.Q_O)$)
  end for
  return Circuit
end function

We note that as per Section 5.2, the complexity of this step can be loosely bounded by $O\left(|S_{-P}| \cdot (N + k)^3 + |S|^3 \cdot (N + k)^3\right)$. The actual run-time will be lower as the matrices on which we perform Gaussian elimination have dimension $n + k \times N$, though we omit a tighter analysis as the algorithm is heuristic in nature.

We next divide P into P_{frozen} and P_{float}, where

$$P_{frozen} = \{A \in P | f \notin \operatorname{span}(Q_O) \text{ for some } (c, f) \in A\}$$
$$P_{float} = P \setminus P_{frozen}.$$

This step requires time in $O\left(|S_P| \cdot (N + k)^3\right)$. The names $float$ and $frozen$ imply that

the partitions can either $float$ to the next subcircuit, or must be $frozen$ in the current one.

Once we have our partition P_{frozen}, we construct a circuit computing the relevant phase factors by taking each set $A \in P$, synthesizing a $CNOT$ circuit computing A, applying the necessary phase rotations, then uncomputing A. The general synthesis procedure is defined in Algorithm 7 and we defer discussion of the procedure for the moment, but for the purpose of analyzing complexity we claim that such a procedure takes $O(|S| \cdot (N+k)^3)$ time.

Finally, for every $A \in P_{float}$, we check whether A is an independent set under the new independence relation

$$A \subseteq S \in I \text{ if and only if } \operatorname{rank}(Q_O) - \operatorname{rank}(A) \leq N - |A|.$$

If $A \notin I$, we remove a linearly dependent element from A and update S_P and S_{-P} accordingly. As $|P_{float}| \leq |S|$ this requires an additional $O\left(|S| \cdot (N+k)^3\right)$ operations.

The entire algorithm, shown in Algorithm 6, thus runs in time

$$O\left(|C| \cdot N + k \cdot |S|^3 \cdot (N+k)^3\right).$$

As $|C| \cdot N$ is in most cases negligible compared to the $k \cdot |S|^3(N+k)^3$ factor, we can leave it out and describe the run-time as simply $O\left(k \cdot |S|^3 \cdot (N+k)^3\right)$. Moreover, it should be noted that if no repartitioning is done, the run-time is bounded by $O\left(|S|^3 \cdot (N+k)^3\right)$, as each element is partitioned exactly once, rather than the worst case of k times in general.

5.4.1 Extended {CNOT, T} synthesis

As mentioned above, the Tpar algorithm relies on a procedure to synthesize a circuit applying a (computable) set of phase factors $A \subseteq S$; we give pseudocode for a general procedure in Algorithm 7. Given input and output states for the qubits, Q_I and Q_O respectively, SYNTHESIZE(A, Q_I, Q_O) returns a circuit implementing

$$U : |Q_I\rangle \mapsto \omega^{\sum_{(c,(b,f)) \in A} c \cdot b \oplus f(x_1, x_2, \ldots, x_{N+k})} |Q_O\rangle.$$

The synthesis proceeds by first adding $N - |A|$ linear Boolean functions so that the resulting set A' has rank equal to $\operatorname{rank}(Q_I)$ – this can be accomplished by using row operations to reduce A to a subset of Q_I, then adding the missing vectors. Next we synthesize a circuit computing A' by reducing Q_I and A' to the same basis using Gaussian elimination in

Algorithm 7 Extended {CNOT, T} synthesis

function SYNTHESIZE(A, Q_I, Q_O)
/* Synthesize a $\{CNOT, X, T\}$ circuit with inputs Q_I, outputs Q_O, and phases A */
Compute $A' \supseteq A$ s.t. $\text{rank}(A') = \text{rank}(Q_I), |A'| = n + m$
Synthesize $\{CNOT, X\}$ circuit C_1 computing $|Q_I\rangle \mapsto |A'\rangle$
Synthesize $\{Z, P, T\}$ circuit C_2 computing $|A'\rangle \mapsto \omega^{\sum_{(c,b,f)\in A'} c \cdot b \oplus f(x_1,x_2,\ldots,x_{N+k})}|A'\rangle$
Synthesize $\{CNOT, X, H\}$ circuit C_3 computing $|A'\rangle \mapsto |Q_O\rangle$
Return $C_1C_2C_3$
end function

$O((N+k)^3)$ time, where addition of two rows corresponds to the application of a $CNOT$ gate, and parity changes correspond to X gates. The circuit reducing Q_I to the basis is applied forwards, while the circuit reducing A' is applied in reverse, giving a circuit mapping $|Q_I\rangle \mapsto |A'\rangle$. The phase factors are applied by constructing a combination of T, P and Z gates, corresponding to the relevant coefficients, then a similar process of Gaussian elimination is used to compute $|A'\rangle \mapsto |Q_O\rangle$. In the case when Q_O produces a new variable, the required Hadamard gate is also applied.

As $|A| \leq |S|$ we see that this step has time complexity $O(|S| \cdot (N+k)^3)$, and assuming the target fault tolerant architecture admits logical P and Pauli-Z gates, the T-depth of the resulting circuit is given by $|A|$.

This synthesis algorithm brings up an important practical issue. Our Gaussian elimination based implementation produces circuits that are non-optimal in terms of the number of gates or depth, resulting in a potential increase in the number of $CNOT$ gates. While our focus was on the optimization of T gates, there exist algorithms, [40, 14], that produce more efficient circuits for linear reversible functions. Specifically, [14] provides an algorithm to synthesize linear reversible circuits with $\Theta(n^2/\log(n))$ gates, and [40] reports an $O(n)$-depth algorithm. More recently, [72] described an optimization procedure for linear reversible circuits that could be used to produce better circuits. In practical implementations, one of these methods should be used instead of Gaussian elimination.

5.5 Results

We implemented[8] Algorithm 6 in C++ and applied it to optimize various quantum, specifically arithmetic, circuits from the literature. Individual circuits were written in the standard fault-tolerant universal gate set $\mathcal{G} = \{X, Y, Z, H, P, CNOT, T\}$, using the decompositions found in Chapter 4 where applicable. In particular, as most arithmetic circuits are dominated by Toffoli gates, we used a T-depth 3 implementation of the Toffoli gate (Figure 4.14).

Our first implementation included only $\{CNOT, T\}$ re-synthesis, which we applied to Clifford + T circuits by optimizing only $\{CNOT, T\}$ subcircuits as described in Section 5.3.1. For comparison between the two approaches, we ran experiments where Tpar was run only on subcircuits between Hadamard gates, as well as experiments using Tpar to optimize the whole circuit.

Results are reported in Tables 5.1 and 5.2. They were generated in Debian Linux running on a quad-core 64-bit Intel Core i5 2.80GHZ processor and 16 GB RAM. Table 5.1 gives gate counts for the circuits. Table 5.2 focuses on T-depth using either 0, N, or ∞ ancillas[9] where N denotes the original number of qubits. The T-depth for each circuit before optimization was computed by parallelizing the T-gates and Toffoli gates by hand, and writing each group of parallel Toffoli gates in T-depth 3. As an example of a non-trivial circuit, we show the initial (Figure 5.8) and Tpar optimized (Figure 5.9) circuits for the 6-bit Cuccaro adder [2].

The benchmarks (Tables 5.1 and 5.2) show that when applying Tpar to $\{CNOT, T\}$ subcircuits, the performance is strongly affected by the structure of the original circuit. In particular, the algorithm optimizes circuits where adjacent Toffoli gates share either controls or targets, in which case the $\{CNOT, T\}$ sub-circuits within the Toffolis can be combined. Each of the GF(2^m) multipliers shares this structure. In fact, the Tpar algorithm with or without Hadamard gates can parallelize any GF(2^m) multiplication circuit to T-depth 2 using sufficient ancillas, by noting that each such circuit can be written in two $\{CNOT, T\}$ stages. Those circuits that mix controls and targets between adjacent Toffoli gates are less affected by the optimization, e.g., CSUM-MUX$_9$, as the $\{CNOT, T\}$ sub-circuits are separated by H gates.

By contrast, when Tpar is used to optimize the whole circuit, all the tested benchmarks have significant reductions in terms of both T-count and depth. While the addition of

[8] The implementation is available at http://code.google.com/p/tpar/.

[9] N was chosen as an arbitrary non-constant number of ancilla qubits to illustrate the trade off between ancillas and T-depth.

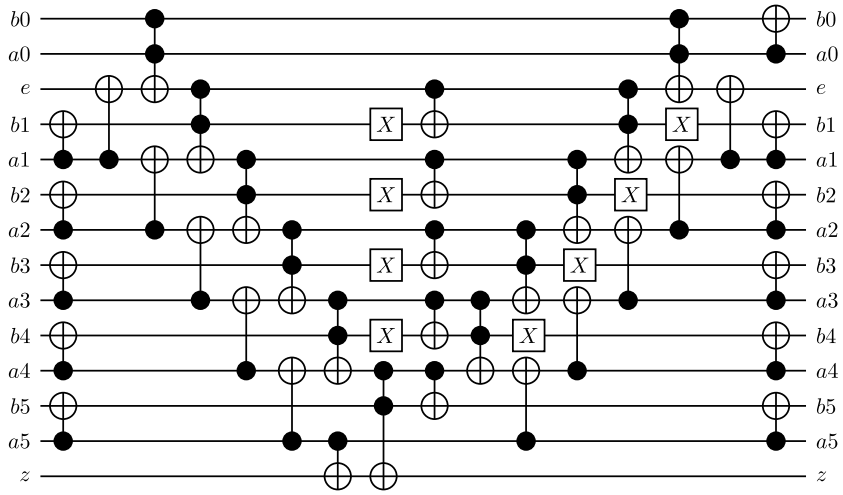


Figure 5.8: 6-bit Cuccaro adder without expanding Toffoli gates [2].

Hadamards makes the algorithm more complex, the running time is not significantly slower on any of the tested benchmarks, and T reduction and parallelization is better in all cases, providing strong evidence that abstracting Hadamard gates is a more effective way of dealing with universal gate sets – in fact, running time was reduced by more than 50% in the large benchmarks. The results themselves show that there are great opportunities for reducing T-count and depth in circuits, with average reductions of 40.7% and 57.8% respectively for the tested benchmarks. Combined with the fact that the algorithm scales favourably to large circuits, Tpar is an effective algorithm for the large-scale optimization of fault tolerant circuits.

We also tested our algorithm's ability to make use of ancillas to optimize T-depth (Table 5.2). For each of the benchmark circuits, we applied our algorithm with an added N ancillas, where the original circuit contained N qubits. We also report the minimum T-depth achievable for each circuit using our algorithm. It can be noted that our algorithm usually decreases in running time when ancillas are added, due to the reduced number of partitions and thus faster matroid partitioning. Specifically, when there are many ancillas, for the majority of the time when an item s is partitioned it can be directly added to one of the partitions in time $O(|P| \cdot (N + k)^3)$. As a result, the algorithm is very flexible and capable of exploiting ancillas to reduce T-depth. Furthermore, our experimental data illustrates a great potential for space-time trade-off in quantum circuits.

As a final remark, we note that our algorithm reproduces many of the previous re-

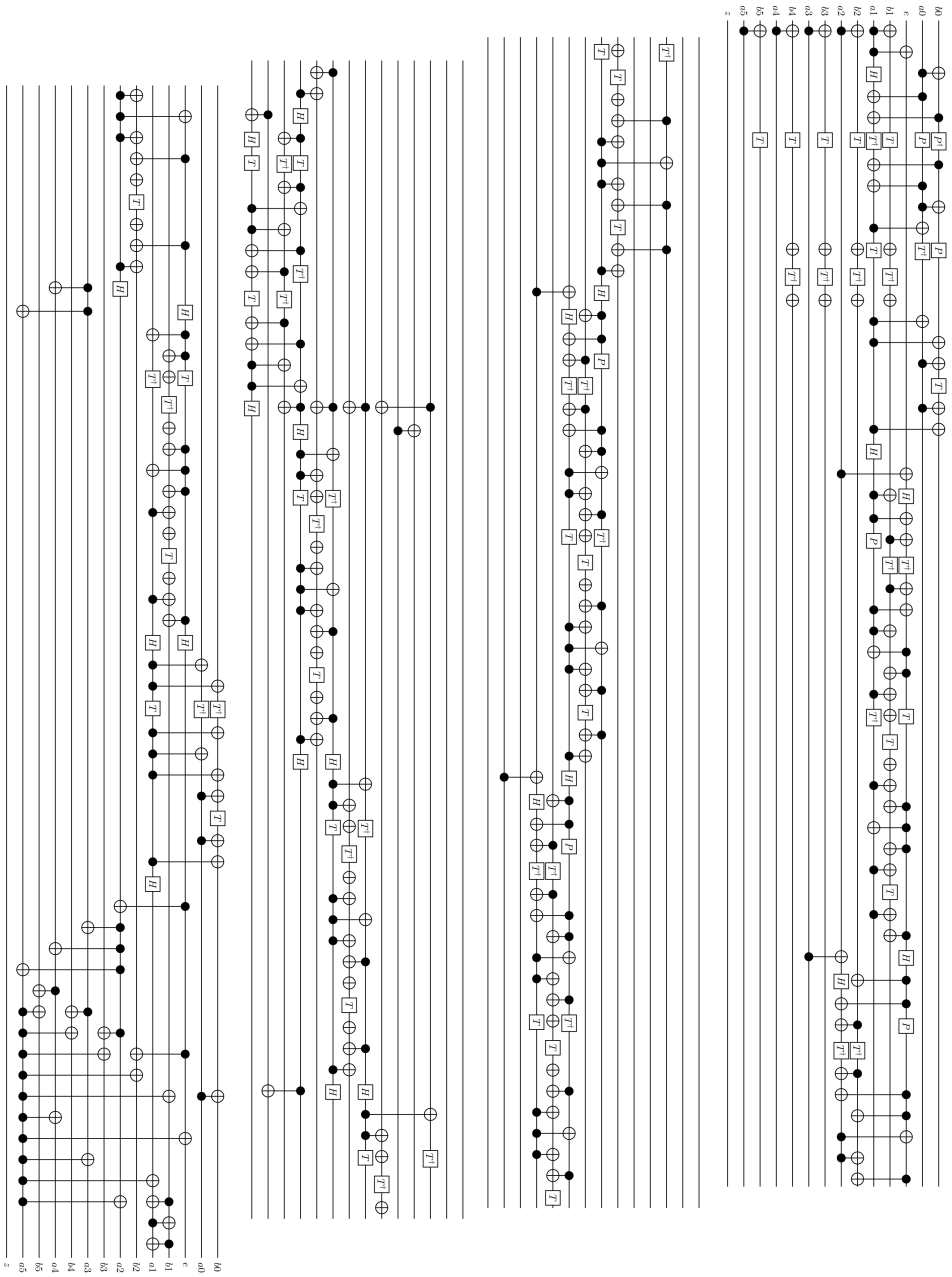


Figure 5.9: Optimized circuit from Figure 5.8 after expanding Toffolis. T-count was reduced from 77 to 63 and T-depth was reduced from 33 to 27.

sults regarding the optimization of T-depth. In particular, Figure 5.10 shows the circuit produced by running Tpar on an implementation of the Toffoli gate. The circuit mirrors the T-depth 1 Toffoli reported in [20]. Moreover, the full range of T-depths possible with different numbers of ancillas can be observed, as in Figure 5.11.

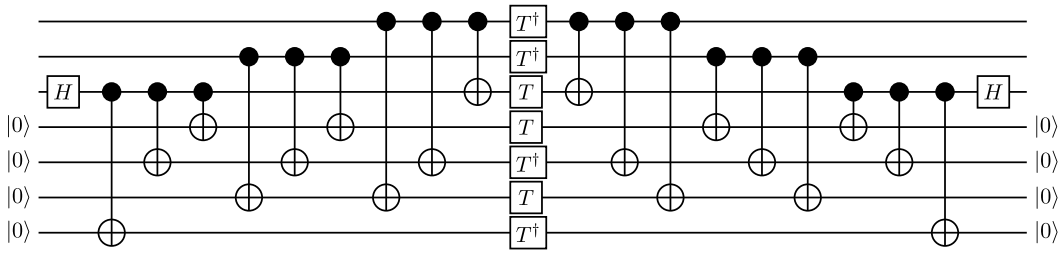


Figure 5.10: T-depth 1 implementation of the Toffoli gate.

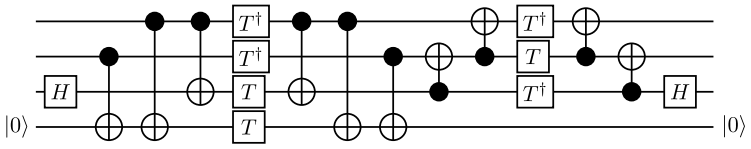


Figure 5.11: T-depth 2 implementation of the Toffoli gate.

We can further observe that other gates as computed in Chapter 4 can be parallelized by adding ancillas and applying Tpar – for instance, the T-depth 1 controlled-P and -V gates, as well as the controlled-T gate as shown in Figure 5.12. The $\Lambda_3(X)$ gate used in the motivating example (Figure 5.1) can also be reduced to T-count 16 and T-depth 8 (Figure 5.13), achieving lower T-count and -depth than the manual optimizations.

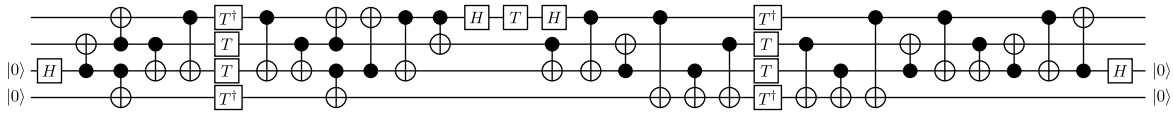


Figure 5.12: T-depth 3 implementation of the controlled-T gate.

5.6 Conclusions

We have described a closely related class of algorithms for re-synthesizing Clifford + T circuits with reduced T-count and depth. These algorithms use representations of circuits by

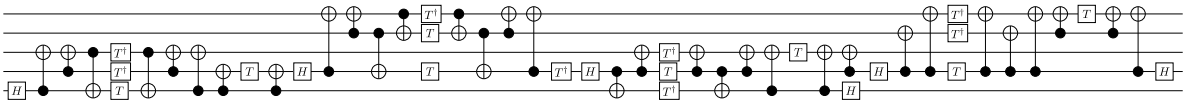


Figure 5.13: $\Lambda_3(X)$ gate.

sets of linear Boolean functions, which allows T gates to be combined and then parallelized by using Matroid theory to partition the functions into invertible subsets. The main algorithm, Tpar, has run-time cubic in the number of T gates, qubits, and Hadamard gates, though our experiments show that the algorithm is sufficiently fast for practical circuit sizes.

Our benchmarks (Tables 5.1 and 5.2) show that large gains can be made in reducing the T-count and depth of quantum circuits. In some cases, T-count was reduced by 65.7%, while T-depth could be reduced by up to 86.7%. Furthermore, the benchmarks illustrate that ancillas can be used to parallelize T gates further, and given the run-times reported the algorithm can be seen to provide substantial flexibility in exploring the trade-off between ancilla usage and T-depth. While the benchmarks demonstrated were all arithmetic or otherwise reversible operations, such operations typically require the majority of the resources in circuits for real quantum algorithms anyway.

5.6.1 Future work

As a major consequence of the Tpar algorithm, reducing the number of $\mathbb{Z}_8$ terms in the mixed arithmetic polynomials describing the phase corresponds directly to reducing the T-complexity of circuits. In fact, for circuits not containing Hadamard gates, Tpar returns a circuit of minimal T-count and depth if the phase function cannot be reduced. A natural avenue for future work is then to develop methods for optimizing such polynomials for T-count and depth – we conjectured that finding the exact optimal expression is intractable, though in practice there may be good heuristics or approximation algorithms.

Sum over path style representations may also prove useful in optimizing circuits over other gate sets, or optimizing circuits for different costs. They provide a very succinct and computationally efficient way of representing the action of a quantum circuit, as opposed to quantum circuits or unitary matrices, neither of which reveal as much of the algebraic structure of a computation. In particular, these representations appear particularly amenable to circuit synthesis, and so an interesting problem would be to consider the generation of the polynomial equations directly from unitaries, for instance those over the ring

$\mathbb{Z}\left[\frac{1}{\sqrt{2}}, i\right]$. Combined with Tpar, multi-qubit unitaries could then be directly synthesized with optimized T gates.

Finally, it may be useful to consider more advanced strategies for assigning phase gates to regions of the circuit. As an example, it can be shown that $\Lambda_m(X)$ for $m > 3$ can be implemented in T-depth $2(m-1)$ if partial results are not uncomputed – however, our algorithm produces circuits with T-depth $2(m-1)+1$ as a result of the heuristic used. An effective area of further research would be to develop heuristics for generating partitions that result in better distributions of phase gates between Hadamard separated regions.

Benchmark	N	x_C	x_T	x_g	Time (s)	x'_C	x'_T	x'_g	T-count reduction (%)
Mod 5_4 [73]	5	32	28	9	0.001	48	22	10	21.4
					0.001	40	16	13	42.9
VBE-Adder$_3$ [74]	10	80	70	20	0.001	94	40	15	42.9
					0.002	80	24	23	65.7
CSLA-MUX$_3$ [75]	15	90	70	20	0.002	189	64	23	8.6
					0.003	219	62	21	11.4
CSUM-MUX$_9$ [75]	30	196	196	84	0.011	242	112	172	42.9
					0.018	222	110	152	43.9
QCLA-Com$_7$ [76]	24	215	203	73	0.004	340	185	53	8.9
					0.012	282	95	168	53.2
QCLA-Mod$_7$ [76]	26	441	413	132	0.012	777	361	102	12.6
					0.025	717	249	230	39.7
QCLA-Adder$_{10}$ [76]	36	273	238	86	0.010	450	218	52	8.4
					0.038	482	162	73	31.9
Adder$_8$ [77]	24	466	399	126	0.010	792	315	134	21.1
					0.016	646	215	230	46.1
RC-Adder$_6$ [2]	14	104	77	30	0.002	161	77	30	0.0
					0.004	167	63	61	18.2
Mod-Red$_{21}$ [78]	11	121	119	58	0.004	230	119	53	0.0
					0.005	192	73	110	38.7
Mod-Mult$_{55}$ [78]	9	55	49	36	0.001	100	47	19	4.1
					0.002	104	37	53	24.5
$\Lambda_3(X)$ [1]	5	28	28	8	0.001	48	28	6	0.0
					0.001	38	16	12	42.9
$\Lambda_4(X)$ [1]	7	56	56	16	0.002	96	56	10	0.0
					0.002	72	28	23	50.0
$\Lambda_5(X)$ [1]	9	84	84	24	0.002	144	84	14	0.0
					0.002	106	40	34	52.4
$\Lambda_{10}(X)$ [1]	19	224	224	64	0.005	384	224	34	0.0
					0.007	276	100	89	55.4
GF(2^4)-Mult [79]	12	115	112	32	0.004	262	76	29	32.1
					0.005	245	68	27	39.3
GF(2^5)-Mult [79]	15	179	175	50	0.012	490	115	40	34.3
					0.009	452	111	36	36.6
GF(2^6)-Mult [79]	18	257	252	72	0.027	830	162	49	35.7
					0.020	759	150	43	40.5
GF(2^7)-Mult [79]	21	349	343	98	0.048	1143	217	56	36.7
					0.044	1060	217	36	36.7
GF(2^8)-Mult [79]	24	468	448	128	0.111	1773	280	60	37.5
					0.078	1591	264	40	41.1
GF(2^9)-Mult [79]	27	575	567	162	0.235	2184	351	64	38.1
					0.143	1886	351	44	38.1
GF(2^{10})-Mult [79]	30	709	700	200	0.420	3022	430	71	38.6
					0.279	2402	410	69	41.4
GF(2^{16})-Mult [79]	48	1856	1792	512	8.353	8994	1072	122	40.2
					4.257	8657	1040	82	42.0
GF(2^{32})-Mult [79]	96	7291	7168	2048	770.369	49230	4192	246	41.5
					370.400	41796	4128	166	42.4
GF(2^{64})-Mult [79]	192	28860	28672	8192	92236.729	273977	16576	494	42.2
					43979.598	214941	16448	334	42.6
Average									21.9
									40.7
Maximum									42.9
									65.7

Table 5.1: Gate count benchmarks. N specifies the number of qubits. x_C reports the number of $CNOT$ gates, x_T gives the number of T gates, and x_g gives the number of other gates. x' denotes the number of gates after optimization by Tpar on subcircuits without H gates (first row), and on the whole circuit (second row).

Benchmark	T-depth original	T-depth 0 ancilla	Red. (%)	Time (s)	T-depth N ancilla	Red. (%)	Time (s)	T-depth ∞ ancilla	Red. (%)
Mod 5_4 [73]	12	9	25.0	0.001	4	66.7	0.001	3	75.0
		7	41.7	0.001	4	66.7	0.001	3	75.0
VBE-Adder$_3$ [74]	24	15	37.5	0.001	5	79.2	0.001	5	79.2
		11	54.2	0.001	5	79.2	0.001	5	79.2
CSLA-MUX$_3$ [75]	21	10	52.4	0.001	4	81.0	0.001	4	81.0
		7	66.7	0.003	4	81.0	0.003	4	81.0
CSUM-MUX$_9$ [75]	18	12	33.3	0.004	5	72.2	0.004	3	83.3
		12	33.3	0.010	5	72.2	0.01	3	83.3
QCLA-Com$_7$ [76]	27	21	22.2	0.004	9	66.7	0.004	7	74.1
		14	48.1	0.009	7	74.1	0.008	7	74.1
QCLA-Mod$_7$ [76]	57	42	26.3	0.007	16	71.9	0.007	14	75.4
		30	47.4	0.020	15	73.7	0.019	14	75.4
QCLA-Adder$_{10}$ [76]	24	17	29.2	0.005	7	70.8	0.004	6	75.0
		12	50.0	0.017	7	70.8	0.016	6	75.0
Adder$_8$ [77]	69	44	36.2	0.007	16	76.8	0.007	15	78.3
		33	52.2	0.015	16	76.8	0.014	15	78.3
RC-Adder$_6$ [2]	33	33	0.0	0.002	11	66.7	0.002	11	66.7
		27	18.2	0.003	11	66.7	0.003	11	66.7
Mod-Red$_{21}$ [78]	48	40	16.7	0.003	15	68.8	0.006	15	68.8
		30	37.5	0.004	15	68.8	0.005	15	68.8
Mod-Mult$_{55}$ [78]	15	9	40.0	0.001	4	73.3	0.002	4	73.3
		9	40.0	0.002	4	73.3	0.002	4	73.3
$\Lambda_3(X)$ [1]	12	12	0.0	0.001	4	66.7	0.001	4	66.7
		9	25.0	0.001	4	66.7	0.001	4	66.7
$\Lambda_4(X)$ [1]	24	24	0.0	0.001	8	66.7	0.002	8	66.7
		17	29.2	0.001	8	66.7	0.001	8	66.7
$\Lambda_5(X)$ [1]	36	36	0.0	0.002	12	66.7	0.003	12	66.7
		25	30.6	0.002	12	66.7	0.002	12	66.7
$\Lambda_{10}(X)$ [1]	96	96	0.0	0.004	32	66.7	0.013	32	66.7
		65	32.3	0.006	32	66.7	0.01	32	66.7
GF(2^4)-Mult [79]	36	9	75.0	0.002	5	86.1	0.002	2	94.4
		7	80.6	0.003	4	88.9	0.003	2	94.4
GF(2^5)-Mult [79]	48	10	79.2	0.004	5	89.6	0.002	2	95.8
		9	81.3	0.006	5	89.6	0.005	2	95.8
GF(2^6)-Mult [79]	60	11	81.7	0.011	6	90.0	0.003	2	96.7
		11	81.7	0.011	5	91.7	0.007	2	96.7
GF(2^7)-Mult [79]	72	13	81.9	0.029	7	90.3	0.004	2	97.2
		12	83.3	0.028	7	90.3	0.011	2	97.2
GF(2^8)-Mult [79]	84	15	82.1	0.049	7	91.7	0.005	2	97.6
		13	84.5	0.041	7	91.7	0.014	2	97.6
GF(2^9)-Mult [79]	96	16	83.3	0.114	8	91.7	0.007	2	97.9
		15	84.4	0.085	7	92.7	0.028	3	96.9
GF(2^{10})-Mult [79]	108	18	83.3	0.311	9	91.7	0.008	2	98.1
		17	84.3	0.216	8	92.6	0.038	2	98.1
GF(2^{16})-Mult [79]	180	27	85.0	7.681	13	92.8	0.025	2	98.9
		24	86.7	2.715	12	93.3	0.18	2	98.9
GF(2^{32})-Mult [79]	372	51	86.3	1340.412	24	93.5	0.186	2	99.5
		51	86.3	266.300	24	93.5	2.678	2	99.5
GF(2^{64})-Mult [79]	756	101	86.6	268472.634	46	93.9	2.573	2	99.7
		104	86.2	46950.853	48	93.7	87.604	2	99.7
Average			45.7			78.9			82.9
			57.8			79.5			82.9
Maximum			86.6			93.9			99.7
			86.7			93.9			99.7

Table 5.2: T-depth benchmarks. We report the T-depth after no optimization (original), and after optimization with 0 (i.e. Table 5.1), N, or unbounded ancillas.

APPENDICES

Appendix A

Complexity of T-count minimization

This thesis has largely been concerned with the problem of minimizing T-count in quantum circuits. In this section we formalize a conjecture made earlier regarding the complexity of a T-count minimization problem.

In Chapter 5 it was mentioned that the $\{CNOT, T\}$ circuit re-synthesis algorithm doesn't necessarily find the minimal T-count (and by extension, T-depth) to implement a given unitary using $CNOT$ and $T^{(\dagger)}$, $P^{(\dagger)}$, or Z gates. Finding an expression of the circuit's phase as a polynomial over $\mathbb{Z}_2$ and $\mathbb{Z}_8$ minimizing the number of odd coefficients was shown to be equivalent to minimizing T-count. Here we formally state the conjecture that this problem is NP-hard.

Conjecture 1. For a sequence $(a_k)_{k\in\{0,1\}^n}, a_k \in \mathbb{Z}_8$, define the polynomial $p_{(a_k)}(x_1, x_2, ..., x_n)$ as

$$p_{(a_k)}(x_1, x_2, ..., x_n) = \sum_{k\in\{0,1\}^n} a_k \cdot (x_1^{k_1} \oplus x_2^{k_2} \oplus \cdots \oplus x_n^{k_n}).$$

Given a sequence $(a_k)_{k\in\{0,1\}^n}, a_k \in \mathbb{Z}_8$, the problem of finding some sequence $(b_k)_{k\in\{0,1\}^n}, b_k \in \mathbb{Z}_8$ such that $p_{(a_k)}(x_1, x_2, ..., x_n) = p_{(b_k)}(x_1, x_2, ..., x_n)$ for all $x_1, x_2, ..., x_n \in \{0, 1\}$ minimizing $\sum_{k\in\{0,1\}^n}(b_k \mod 2)$ is NP-hard.

We describe the phase polynomial in slightly different, more general terms here – particularly, each term $x_1^{k_1} \oplus x_2^{k_2} \oplus \cdots \oplus x_n^{k_n}$ is a linear Boolean function of the inputs $x_1, x_2, ..., x_n$. The connection to minimization of T-count, described in Chapter 5, is in the minimization $\sum_{k\in\{0,1\}^n}(b_k \mod 2)$, as each odd coefficient in the phase requires one T gate to implement. Since every polynomial of the form $p_{(a_k)}(x_1, x_2, ..., x_n)$ defines the global phase for some $\{CNOT, T\}$ circuit, the two minimization problems are informally equivalent.

References

[1] 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," *Phys. Rev. A*, vol. 52, pp. 3457–3467, Nov 1995, quant-ph/9503016.

[2] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. Petrie Moulton, "A new quantum ripple-carry addition circuit," *ArXiv e-prints*, Oct. 2004, quant-ph/0410184.

[3] V. Kabanets and J.-Y. Cai, "Circuit minimization problem," in *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, STOC '00, (New York, NY, USA), pp. 73–79, ACM, 2000.

[4] R. K. Brayton, A. L. Sangiovanni-Vincentelli, C. T. McMullen, and G. D. Hachtel, *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA, USA: Kluwer Academic Publishers, 1984.

[5] J. W. Britton, B. C. Sawyer, A. C. Keith, C.-C. J. Wang, J. K. Freericks, H. Uys, M. J. Biercuk, and J. J. Bollinger, "Engineered two-dimensional ising interactions in a trapped-ion quantum simulator with hundreds of spins," *Nature*, no. 7395, p. 489492, 2012.

[6] K. R. Brown, A. C. Wilson, Y. Colombe, C. Ospelkaus, A. M. Meier, E. Knill, D. Leibfried, and D. J. Wineland, "Single-qubit-gate error below 10^{-4} in a trapped ion," *Phys. Rev. A*, vol. 84, p. 030303, Sep 2011, arXiv:1104.2552.

[7] J. M. Chow, J. M. Gambetta, A. D. Córcoles, S. T. Merkel, J. A. Smolin, C. Rigetti, S. Poletto, G. A. Keefe, M. B. Rothwell, J. R. Rozen, M. B. Ketchen, and M. Steffen, "Universal quantum gate set approaching fault-tolerant thresholds with superconducting qubits," *Phys. Rev. Lett.*, vol. 109, p. 060501, Aug 2012, arXiv:1202.5344.

[8] C. Rigetti, J. M. Gambetta, S. Poletto, B. L. T. Plourde, J. M. Chow, A. D. Córcoles, J. A. Smolin, S. T. Merkel, J. R. Rozen, G. A. Keefe, M. B. Rothwell, M. B. Ketchen, and M. Steffen, "Superconducting qubit in a waveguide cavity with a coherence time approaching 0.1 ms," *Phys. Rev. B*, vol. 86, p. 100506, Sep 2012, arXiv:1202.5533.

[9] H. Bombin, R. S. Andrist, M. Ohzeki, H. G. Katzgraber, and M. A. Martin-Delgado, "Strong resilience of topological codes to depolarization," *Phys. Rev. X*, vol. 2, p. 021004, Apr 2012, arXiv:1202.1852.

[10] A. G. Fowler, A. M. Stephens, and P. Groszkowski, "High-threshold universal quantum computation on the surface code," *Phys. Rev. A*, vol. 80, p. 052312, Nov 2009, arXiv:0803.0272.

[11] A. G. Fowler, A. C. Whiteside, and L. C. L. Hollenberg, "Towards practical classical processing for the surface code," *Phys. Rev. Lett.*, vol. 108, p. 180501, May 2012, arXiv:1110.5133.

[12] O. Golubitsky and D. Maslov, "A study of optimal 4-bit reversible toffoli circuits and their synthesis," *Computers, IEEE Transactions on*, vol. 61, no. 9, pp. 1341–1353, 2012, arXiv:1103.2686.

[13] W. N. N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski, "Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 9, pp. 1652–1663, 2006.

[14] K. N. Patel, I. L. Markov, and J. P. Hayes, "Optimal synthesis of linear reversible circuits," *Quantum Info. Comput.*, vol. 8, pp. 282–294, Mar. 2008, quant-ph/0302002.

[15] D. Maslov, G. W. Dueck, D. M. Miller, and C. Negrevergne, "Quantum circuit simplification and level compaction," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 27, pp. 436–444, Mar. 2008, quant-ph/0604001.

[16] R. Dias da Silva, E. Pius, and E. Kashefi, "Global quantum circuit optimization," *ArXiv e-prints*, Jan. 2013, arXiv:1301.0351.

[17] A. Paetznick and A. G. Fowler, "Quantum circuit optimization by topological compaction in the surface code," *ArXiv e-prints*, Apr. 2013, arXiv:1304.2807.

[18] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 6, pp. 818–830, 2013, arXiv:1206.0758.

[19] A. Bocharov and K. M. Svore, "Resource-optimal single-qubit quantum circuits," *Phys. Rev. Lett.*, vol. 109, p. 190501, Nov 2012, arXiv:1206.3223.

[20] P. Selinger, "Quantum circuits of T-depth one," *Phys. Rev. A*, vol. 87, p. 042302, Apr 2013, arXiv:1210.0974.

[21] C. Jones, "Low-overhead constructions for the fault-tolerant toffoli gate," *Phys. Rev. A*, vol. 87, p. 022328, Feb 2013, arXiv:1212.5069.

[22] A. G. Fowler, "Time-optimal quantum computation," *ArXiv e-prints*, Oct. 2012, arXiv:1210.4626 [quant-ph].

[23] R. Landauer, "Information is physical," in *Physics and Computation, 1992. PhysComp '92., Workshop on*, pp. 1–4, 1992.

[24] P. Kaye, R. Laflamme, and M. Mosca, *An Introduction to Quantum Computing*. Oxford University Press, 2007.

[25] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pp. 124–134, 1994, quant-ph/9508027v2.

[26] S. Lloyd, "Universal quantum simulators," *Science*, vol. 273, no. 5278, pp. 1073–1078, 1996.

[27] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. Cambridge Series on Information and the Natural Sciences, Cambridge University Press, 2000.

[28] A. Y. Kitaev, A. H. Shen, and M. N. Vyalyi, *Classsical and Quantum Computation*. Graduate studies in mathematics, v. 47, American mathematical society, 2002.

[29] D. Gottesman, "The heisenberg representation of quantum computers," in *International Conference on Group Theoretic Methods in Physics*, p. 9807006, 1998, quant-ph/9807006v1.

[30] V. Kliuchnikov, D. Maslov, and M. Mosca, "Fast and efficient exact synthesis of single qubit unitaries generated by Clifford and T gates," *Quantum Info. Comput.*, vol. 13, pp. 607–630, July 2013, arXiv:1206.5236.

[31] B. Giles and P. Selinger, "Exact synthesis of multiqubit Clifford+T circuits," *Phys. Rev. A*, vol. 87, p. 032332, Mar 2013, arXiv:1212.0506.

[32] D. P. DiVincenzo, "Two-bit gates are universal for quantum computation," *Phys. Rev. A*, vol. 51, pp. 1015–1022, Feb 1995, cond-mat/9407022.

[33] A. Y. Kitaev, "Quantum computations: algorithms and error correction," *Russian Mathematical Surveys*, vol. 52, pp. 1191–1249, Dec. 1997.

[34] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, p. 802, Oct. 1982.

[35] A. Paetznick and B. W. Reichardt, "Universal fault-tolerant quantum computation with only transversal gates and error correction," *ArXiv e-prints*, Apr. 2013, arXiv:1304.3709.

[36] D. Gottesman and I. L. Chuang, "Quantum teleportation is a universal computational primitive," *Nature*, no. 6760, p. 390393, 1999, quant-ph/9908010.

[37] S. Bravyi and A. Kitaev, "Universal quantum computation with ideal clifford gates and noisy ancillas," *Phys. Rev. A*, vol. 71, p. 022316, Feb 2005, quant-ph/0403025.

[38] D. Gottesman, "Theory of fault-tolerant quantum computation," *Phys. Rev. A*, vol. 57, pp. 127–137, Jan 1998, quant-ph/9702029.

[39] A. Broadbent and E. Kashefi, "Parallelizing quantum circuits," *Theor. Comput. Sci.*, vol. 410, pp. 2489–2510, June 2009, arXiv:0704.1736.

[40] D. Maslov, "Linear depth stabilizer and quantum fourier transformation circuits with no auxiliary qubits in finite-neighbor quantum architectures," *Phys. Rev. A*, vol. 76, p. 052310, Nov 2007, quant-ph/0703211.

[41] M. Saeedi, R. Wille, and R. Drechsler, "Synthesis of quantum circuits for linear nearest neighbor architectures," *Quantum Information Processing*, vol. 10, no. 3, pp. 355–377, 2011, arXiv:1110.6412.

[42] M. Möttönen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, "Quantum circuits for general multiqubit gates," *Phys. Rev. Lett.*, vol. 93, p. 130502, Sep 2004, quant-ph/0404089.

[43] V. Shende, A. Prasad, I. Markov, and J. Hayes, "Synthesis of reversible logic circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 22, no. 6, pp. 710–722, 2003, quant-ph/0207001.

[44] D. Maslov and D. Miller, "Comparison of the cost metrics through investigation of the relation between optimal NCV and optimal NCT three-qubit reversible circuits," *Computers Digital Techniques, IET*, vol. 1, no. 2, pp. 98–104, 2007, quant-ph/0511008.

[45] A. G. Fowler, "Constructing arbitrary Steane code single logical qubit fault-tolerant gates," *Quantum Info. Comput.*, vol. 11, pp. 867–873, Sept. 2011, quant-ph/0411206.

[46] D. Grosse, R. Wille, G. Dueck, and R. Drechsler, "Exact multiple-control toffoli network synthesis with SAT techniques," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 5, pp. 703–715, 2009.

[47] C. M. Dawson and M. A. Nielsen, "The Solovay-Kitaev algorithm," *Quantum Info. Comput.*, vol. 6, pp. 81–95, Jan. 2006, quant-ph/0505030.

[48] P. Selinger, "Efficient Clifford+T approximation of single-qubit operators," *ArXiv e-prints*, Dec. 2012, arXiv:1212.6253.

[49] V. Kliuchnikov, D. Maslov, and M. Mosca, "Practical approximation of single-qubit unitaries by single-qubit quantum Clifford and T circuits," *ArXiv e-prints*, Dec. 2012, arXiv:1212.6964.

[50] A. K. Prasad, V. V. Shende, I. L. Markov, J. P. Hayes, and K. N. Patel, "Data structures and algorithms for simplifying reversible circuits," *J. Emerg. Technol. Comput. Syst.*, vol. 2, pp. 277–293, Oct. 2006.

[51] D. Miller, D. Maslov, and G. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Design Automation Conference, 2003. Proceedings*, pp. 318–323, 2003.

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

[53] M. Sedlk and M. Plesch, "Towards optimization of quantum circuits," *Central European Journal of Physics*, vol. 6, no. 1, pp. 128–134, 2008, quant-ph/0607123.

[54] C. Moore and M. Nilsson, "Parallel quantum computation and quantum codes," *SIAM J. Comput.*, vol. 31, pp. 799–815, Mar. 2002, quant-ph/9808027.

[55] R. Raussendorf and H. J. Briegel, "Computational model underlying the one-way quantum computer," *Quantum Info. Comput.*, vol. 2, pp. 443–486, Oct. 2002, quant-ph/0108067.

[56] D. E. Browne and H. J. Briegel, "One-way quantum computation - a tutorial introduction," *ArXiv e-prints*, Mar. 2006, quant-ph/0603226.

[57] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," *Phys. Rev. A*, vol. 70, p. 052328, Nov 2004, quant-ph/0406196.

[58] T. Bozkaya and M. Ozsoyoglu, "Indexing large metric spaces for similarity search queries," *ACM Trans. Database Syst.*, vol. 24, pp. 361–404, Sept. 1999.

[59] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA '93, (Philadelphia, PA, USA), pp. 311–321, Society for Industrial and Applied Mathematics, 1993.

[60] N. Kumar, L. Zhang, and S. Nayar, "What is a good nearest neighbors algorithm for finding similar patches in images?," in *Computer Vision ECCV 2008* (D. Forsyth, P. Torr, and A. Zisserman, eds.), vol. 5303 of *Lecture Notes in Computer Science*, pp. 364–378, Springer Berlin Heidelberg, 2008.

[61] S. Sen and R. E. Tarjan, "Deletion without rebalancing in balanced binary trees," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, (Philadelphia, PA, USA), pp. 1490–1499, Society for Industrial and Applied Mathematics, 2010.

[62] P. Pham, "Quantum compiler," 2011.

[63] A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, and D. A. Spielman, "Exponential algorithmic speedup by a quantum walk," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, (New York, NY, USA), pp. 59–68, ACM, 2003, quant-ph/0209131.

[64] A. Peres, "Reversible logic and quantum computers," *Phys. Rev. A*, vol. 32, pp. 3266–3276, Dec 1985.

[65] R. P. Feynman, "Quantum mechanical computers," *Foundations of Physics*, vol. 16, no. 6, pp. 507–531, 1986.

[66] M. Amy, D. Maslov, and M. Mosca, "Polynomial-time T-depth optimization of Clifford+T circuits via matroid partitioning," *ArXiv e-prints*, Mar. 2013, arXiv:1303.2042.

[67] C. M. Dawson, A. P. Hines, D. Mortimer, H. L. Haselgrove, M. A. Nielsen, and T. J. Osborne, "Quantum computing and polynomial equations over the finite field $\mathbb{Z}_2$," *Quantum Info. Comput.*, vol. 5, pp. 102–112, Mar. 2005, quant-ph/0408129.

[68] T. Rudolph, "Simple encoding of a quantum circuit amplitude as a matrix permanent," *Phys. Rev. A*, vol. 80, p. 054302, Nov 2009, arXiv:0909.3005.

[69] R. P. Feynman and A. R. Hibbs, *Quantum mechanics and path integrals*. International series in pure and applied physics, McGraw-Hill, 1965.

[70] J. Edmonds, "Minimum partition of a matroid into independent subsets," *JournaB1l of Research of the National Bureau of Standards*, vol. 69B, pp. 67–72, Jan. 1965.

[71] T. Beth and M. Roetteler, "Quantum algorithms: Applicable algebra and quantum physics," in *Quantum Information*, vol. 173 of *Springer Tracts in Modern Physics*, pp. 96–150, Springer Berlin Heidelberg, 2001.

[72] V. Kliuchnikov and D. Maslov, "Optimization of Clifford circuits," *ArXiv e-prints*, May 2013, arXiv:1305.0810.

[73] D. Maslov, "Reversible logic synthesis benchmarks page," 2011.

[74] V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *Phys. Rev. A*, vol. 54, pp. 147–153, Jul 1996, quant-ph/9511018.

[75] R. Van Meter and K. M. Itoh, "Fast quantum modular exponentiation," *Phys. Rev. A*, vol. 71, p. 052320, May 2005, quant-ph/0408006.

[76] T. G. Draper, S. A. Kutin, E. M. Rains, and K. M. Svore, "A logarithmic-depth quantum carry-lookahead adder," *Quantum Info. Comput.*, vol. 6, pp. 351–369, July 2006, quant-ph/0406142.

[77] Y. Takahashi, S. Tani, and N. Kunihiro, "Quantum addition circuits and unbounded fan-out," *Quantum Info. Comput.*, vol. 10, pp. 872–890, Sept. 2010, arXiv:0910.2530.

[78] I. L. Markov and M. Saeedi, "Constant-optimized quantum circuits for modular multiplication and exponentiation," *Quantum Info. Comput.*, vol. 12, pp. 361–394, May 2012, arXiv:1202.6614.

[79] D. Maslov, J. Mathew, D. Cheung, and D. K. Pradhan, "An $O(m^2)$-depth quantum algorithm for the elliptic curve discrete logarithm problem over GF(2^m)," *Quantum Info. Comput.*, vol. 9, pp. 610–621, July 2009, arXiv:0710.1093.

[80] P. Aliferis, D. Gottesman, and J. Preskill, "Quantum accuracy threshold for concatenated distance-3 codes," *Quantum Info. Comput.*, vol. 6, pp. 97–165, Mar. 2006, quant-ph/0504218.

[81] A. Parent, J. Parker, M. Burns, and D. Maslov, "QCViewer: a tool for displaying, editing, and simulating quantum circuits," 2013.

[82] X. Zhou, D. W. Leung, and I. L. Chuang, "Methodology for quantum logic gate construction," *Phys. Rev. A*, vol. 62, p. 052316, Oct 2000, quant-ph/0002039.