Quantum Compilers for Reducing Non-Clifford Gate Counts

Luke E. Heyfron

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

The University of Sheffield
Faculty of Science
Department of Physics and Astronomy

January 2021

Acknowledgements

The research presented in this PhD thesis was funded by the Engineering and Physical Sciences Research Council (EPSRC) through grant EP/M024261/1.

I would like to thank the following people for their support during this project. First, I would like to thank my family: Diane Lewis, Lawrence Heyfron, Amy Heyfron and Val Heyfron. I would like thank and celebrate the memory of "Nanny" Barbara Joan Lewis, Victor Heyfron, David Paul Lewis and Rhonda Lewis (née Alexander), who have passed away since I started the PhD. I would like to thank the following people whose support and friendship have been invaluable over the years: Olly Hall, John Harvey, James Collier, Wei-Yu Chen, Max Albrecht, Emily, Jeff, Eve and Anna van Zyl, Tamás Varga, Prea Singh, David Hurst, Jasminder Sidhu, Giuseppe Buonaiuto, Scott Vinay, Mark Pearce, Mark Howard, Emiliano Cancellieri, Pieter Kok, Joschka Roffe, Padraic Calpin and Mike Vasmer. Finally, I would like to thank my supervisor Dr. Earl T. Campbell, without whose guidance, professionalism and patience I would not have been able to complete this project.

Abstract

Quantum computers can solve certain problems much faster than classical computers. However, in order to benefit from the speed-up granted by quantum algorithms, they must first be rendered as hardware-level instructions (i.e. quantum circuits) in a process known as quantum compiling. Any choice of discrete gate set from which our compilation result is constructed should be both universal and fault-tolerant, such that any quantum algorithm can be compiled and successfully executed despite the presence of environmental noise. From these requirements, it follows that it is impossible to avoid including at least one gate that is disproportionately expensive relative to the others. Many leading proposals for the first generation of qubit-based quantum computers designate the T gate, otherwise known as the $\frac{\pi}{8}$ gate, to be this necessary yet costly gate. In this thesis, we present novel algorithms for compiling quantum circuits that reduce the T count of input circuits. In addition, we present analogous compilation algorithms for qudit-based quantum computers, for which the M gate is the designated expensive gate.

Contents

Declaration

I, Luke E. Heyfron, confirm that this thesis is my own work. I am aware of the University's Guidance on the Use of Unfair Means (www.sheffield.ac.uk/ssid/unfair-means). This work has not been previously been presented for an award at this, or any other, university.

Chapter 1

Introduction

Quantum computers are able to perform certain tasks much faster than classical computers. For example, Shor's quantum algorithm for prime factorisation finds the solution in polynomial time with respect to the size of the input. This problem is believed to be intractable for classical computers [2], a property which is exploited by internet security protocols such as RSA public key encryption [3].

The superior efficiency of quantum algorithms makes it a highly desirable prospect to build quantum computers that are able to perform any operation that the quantum programmers of the future could possibly wish to perform. Such a quantum computer would be *universal* in quantum computation, meaning the set of elementary quantum logic gates that the computer can perform (or *gate set*) is sufficient to implement any quantum algorithm up to an error that diminishes with the length of the gate sequence. The Gottesman-Knill theorem proves that a quantum computer is classical simulable if the gate set is restricted to members a particular group of quantum operators called the *Clifford group* [4]. This implies that the Clifford group alone is not sufficient for universal quantum computation, unless a universal quantum computer can be simulated by a classical computer. It follows that at least one non-Clifford operation must be included in the gate set of any universal quantum architecture. A conventional choice for the non-Clifford gate is the $\frac{\pi}{8}$ phase gate also known as the T gate.

Quantum computers are highly susceptible to decoherence due to mechanisms such as interactions with the environment, imperfect quantum gates and initial state preparation. The fragile nature of quantum systems gave rise to the theory of fault-tolerant quantum computation, a key result of which is the *threshold theorem*, which states that the probability of an error occurring in a quantum state encoded using a so called *quantum error correction* (QEC) code reduces to zero asymptotically with an increase in resource overhead (i.e. the number of redundant qubits) provided the per-gate error probability is below a certain threshold probability [5]. Otherwise, quantum memory decoheres to the maximally mixed state and the outcome of any quantum algorithm is meaningless. The value of the threshold depends on the quantum error correction used to encode the logical quantum state, as well as the error model.

While QEC codes overcome the problem of noise, they impose constraints upon the definitions of encoded quantum gates. In particular, they determine which encoded gates are *transversal*. By definition, an encoded gate that is transversal contains no entangling gates within the same code block of its physical implementation. Transversality is sufficient for fault-tolerance [6] and additionally ensures that the overhead associated with a transversal gate is 'low' as its cost is upper-bounded by a linear function of the code size. One of the most promising classes of quantum error correction codes are the topological codes due to their high thresholds (such as the toric code with a threshold of nearly 1% [7]) and transversal Clifford gates [8]. However, the T gate is not transversal for the toric code and its implementation requires many additional steps compared to that of the Clifford gates. A T gate may be implemented by first preparing special ancillary states known as T-type *magic states* and then applying a teleportation gadget to the magic state and the input state [9]. The preparation of magic states is expensive, requiring the consumption of many noisy raw magic states in recursive distillation processes such as [10] and [11] where additional layers of recursion suppress the error sufficiently but cause the number of elementary operations used to blow up considerably. This disparity between the required number of elementary gates for Clifford and T gates motivates the the *Clifford + T* model for quantum compilation, where Clifford gates are considered 'free' with a cost of 0 and T gates are considered 'expensive' with a cost of 1. In this case, the gate synthesis problem reduces to the problem of minimizing the number of T gates in the circuit.

It might seem highly unrealistic to assume that logical (i.e. fault-tolerant) Clifford gates are free since logical Clifford gates are far from "cheap" in an absolute sense (especially from an experimental perspective). However, we emphasise that each logical T gate synthesised using magic state distillation (MSD) requires an amount of logical Clifford gates (including CNOT's and Hadamards) that scales with the "overhead", or the number of raw T-type magic states required per logical T gate. The value of the overhead depends on architectural assumptions but tends to be between 50-1000 [12, 13, 14, 15] and typically increases with the desired accuracy of the logical T gate. Therefore, in the Clifford + T magic states model of quantum computation, a logical Clifford gate will invariably have a cost that is significantly lower than that of a logical T gate. One potential pitfall of this argument is if a T-count optimizer were to introduce an amount of Clifford gates that diverges as the T-count is decreased. In fact, for so-called "phase-polynomial" type T-optimizers for CNOT + T circuits such as [16, 17] and those introduced in this thesis (see chapters 4 and 5), we find that the opposite is the case. The number of CNOT gates required to implement an n-qubit phase-polynomial is $O(nm)$ in the worst case, where m is the number of linear boolean functions with non-zero coefficient (modulo 8). In the worst case, the value of m receives a contribution of $O(n)$ due to Clifford phases [18, 19] and of $O(n^2)$ due to non-Clifford phases [17, 19], so $m = O(n^2) + O(n) = O(n^2)$ and in practice we find that $m \approx \tau$, where τ is the T-count. It follows that the worst-case scaling of the Clifford gate-count actually *decreases* as the T-count decreases. Additionally, in leading models of fault-tolerant quantum computation such as Pauli-based computation through lattice surgery [20, 21, 22, 23], Clifford gates have exactly zero cost as they are only performed implicitly by adapting the basis in which Pauli measurements are made.

It is important that a gate-synthesis algorithm is efficient. Fast algorithms have already been found for compiling single qubit circuits in the Clifford + T basis [24, 25, 26, 27], so our focus will be on multi-qubit gate synthesis algorithms for Clifford + T circuits. There have been a number of recent developments in the field of multi-qubit T gate optimization. Amy, Maslov, Mosca and Roetteler devised an algorithm for optimizing the T-depth, which is a related metric to the T count and is equal to the minimal number of parallel columns of T gates found in any Clifford + T circuit implementing a particular unitary [28]. Their algorithm involves an exhaustive search over the space of circuits, S_d, implementable in depth d over a certain fixed gate set $\mathcal{G}$. The search is performed in lexicographic order over an alphabet $\mathcal{V}_{n,\mathcal{G}}$ composed of all possible depth-one n-qubit circuits on $\mathcal{G}$. By dividing the target unitary, $U = VW$, into two factors V and W of roughly equal depth and using a set relation (lemma 1 of [28]), an exhaustive search as described above can determine the existence of, and produce a depth-minimal circuit for unitaries of depth up to $2d$. This so called "meet-in-the-middle" approach is nearly quadratically faster than a brute force search over S_{2d}. However, the execution time remains exponential in both n and $|\mathcal{G}|$, so is impractical for large circuits.

Later, Amy, Maslov and Mosca developed a scheme for reducing the T depth and T count for Clifford + T circuits based on matroid partitioning [16]. It was this work that introduced the circuit-polynomial correspondence for multi-qubit CNOT + T circuits that would later be used in [17, 29, 19, 30], as well as chapters 4 and 5 of this thesis. This work contains the first instance of an efficient general purpose quantum compiler (known as *T-Par*) on the Clifford + T gate set that reduces the T count.

This was followed by Gosset, Kliuchnikov and Mosca's work [31], in which an algorithm is proposed that provides T count optimal solutions for circuits on the Clifford + T gate set. This was used to provide the first proof that the 7 T gate implementation of the Toffoli gate is optimal. Although the algorithm is optimal, it is inefficient with runtime that scales as $O\left(2^{nm}\text{poly}\left(2^n, m\right)\right)$ where n is the number of qubits and m is the T count, rendering it infeasible for scalable quantum computers.

More recently, Amy and Mosca discovered an equivalence between T gate optimization on n-qubit CNOT + T circuits and minimum distance decoding of the n-bit punctured Reed-Muller code of order $n-4$ [17]. This equivalence reveals new applications for previous work on fast near-optimal Reed-Muller decoders and strengthens the notion that T count optimization of general multi-qubit Clifford + T circuits is an intractable problem.

The remainder of this thesis is structured as follows. In chapter 2 we review the basics of quantum computation, introducing terminology and notation used throughout the thesis. Then in chapter 3, we motivate the Clifford + T cost model. In chapters, 4 and 5 we provide original work on T gate optimization using some of the observations from Amy and Mosca's work as a starting point. Finally, we conclude in chapter 6.

Chapter 2

Basics of Quantum Computation

In this chapter, we introduce theory and terminology necessary to understand the remainder of the thesis.

2.1 Quantum States

Qubits are elementary units of quantum information that are binary, much like bits, their classical cousins. A qubit lives in a 2-dimensional Hilbert space, $\mathcal{H}_2$, which, by convention, is spanned by the orthonormal *computational basis*, whose basis states are labelled with the classical bits "0" and "1". As these states are vectors, we place their labels inside 'kets' $|0\rangle$ and $|1\rangle$, according to the conventions of Dirac notation. The state of a qubit is a quantum superposition of the computational basis states, so a general state, $|\psi\rangle$, is given by the following:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{2.1}$$

where $\alpha, \beta \in \mathbb{C}$. The norm of $|\psi\rangle$ is defined to be:

$$|\,|\psi\rangle\,| = \sqrt{\langle\psi\mid\psi\rangle} = \sqrt{|\alpha|^2 + |\beta|^2} \tag{2.2}$$

If $|\psi\rangle$ is *normalised*, then α and β are *probability amplitudes*, whose modulus squared gives the probability of yielding a 0 or 1, respectively, upon measuring $|\psi\rangle$ in the "native" computational basis. The state $|\psi\rangle$ is said to be normalised iff the sum of the moduli squared of the probability amplitudes is equal to one:

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2.3}$$

In general, a quantum computer will require multiple qubits. An n-qubit quantum memory can be described in terms of a 2^n dimensional Hilbert space, $\mathcal{H}_2^{\otimes n}$, formed from the n-fold tensor product of the single qubit Hilbert space. It follows that the state of an n-qubit quantum computer is spanned by the n-qubit computational basis states, which take the following form:

$$|\mathbf{x}\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \cdots \otimes |x_n\rangle\,, \tag{2.4}$$

where the binary vector $\mathbf{x} \in \mathbb{Z}_2^n$. Therefore, a general state of an n-qubit quantum computer, $|\psi\rangle \in \mathcal{H}_2^{\otimes n}$, is written as

$$\begin{aligned} |\psi\rangle &= \sum_{\mathbf{x}\in\mathbb{Z}_2^n} c_{\mathbf{x}} |\mathbf{x}\rangle \\ &= c_{00\ldots0} |00\ldots0\rangle + c_{00\ldots1} |00\ldots1\rangle + \cdots + c_{11\ldots1} |11\ldots1\rangle , \end{aligned} \tag{2.5}$$

where the $c_{\mathbf{x}} \in \mathbb{C}$ are the complex probability amplitudes; the norm is defined as

$$|\, |\psi\rangle \,| = \sqrt{\sum_{\mathbf{x}\in\mathbb{Z}_2^n} |c_{\mathbf{x}}|^2} \tag{2.6}$$

and the normalization condition is

$$|\, |\psi\rangle \,|^2 = 1. \tag{2.7}$$

2.2 Quantum Operators

The dynamics of a quantum system are characterised by quantum operators, which are mathematical objects that map quantum states to other quantum states. Quantum operators are linear and norm preserving. Therefore, an n qubit quantum operator can be represented explicitly as n by n unitary matrix on $\mathbb{C}$. We refer to the set of all n-qubit unitary operators as $\mathcal{U}_n$ and will use the terms "quantum operator" or "operator" and "unitary" interchangeably.

2.2.1 Pauli Group

The single qubit Pauli operators are important for quantum computation and are given by the following matrices in the computational basis:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{2.8}$$

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \tag{2.9}$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{2.10}$$

It is intuitive to see that Pauli X behaves as the quantum version of the classical 'inverter' or NOT operator as an X applied to a $|0\rangle$ yields a $|1\rangle$ and an X applied to a $|1\rangle$ yields a $|0\rangle$. The Pauli Z operator also behaves as an inverter, albeit in a different basis. It acts as follows: $|0\rangle \to |0\rangle$ and $|1\rangle \to -|1\rangle$. So if we define $|+\rangle := \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|-\rangle := \frac{|0\rangle-|1\rangle}{\sqrt{2}}$, then the Pauli Z acts as an inverter between the orthonormal $|+\rangle$ and $|-\rangle$ states. The Pauli Y operator is acts as a Pauli X and Pauli Z applied in sequence (with an undetectable global phase) i.e. $Y = -iZX$.

The n-qubit *Pauli group*, $\mathcal{P}_n$, is the set of all n-fold tensor products of the single qubit Pauli operators, X, Y, Z and the operators $\pm iI$. Note that all Pauli operators either commute or anti-commute.

2.2.2 The Clifford Group

Another trio of important operators are the Hadamard, H, operator, the *phase operator*, S, and the CNOT operator, which are given by the following matrices,

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \tag{2.11}$$

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \tag{2.12}$$

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{2.13}$$

These operators all belong to the *Clifford group*, which is defined as the normalizer of the Pauli group:

$$\mathcal{C}_n := \{U \mid U^\dagger P U = P',\ P' \in \mathcal{P}_n \ \forall\ P \in \mathcal{P}_n\}. \tag{2.14}$$

For example, the Pauli X operator conjugates to Pauli Z under the Hadamard operator ($H^\dagger X H = Z$) and to negative Pauli Y under the phase operator ($S^\dagger X S = -Y$). Furthermore, the operators H, S and $CNOT$ form a minimum generating set of operators for the Clifford group.

2.2.3 The T Gate

The T gate, which is defined as

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \tag{2.15}$$

is central to the study of fault-tolerant quantum compilation and to this thesis. As we will see in section 3.1.1, the Clifford group must be extended by a non-Clifford gate in order to form a universal gate set. Due to results from fault-tolerant quantum computation, this gate is commonly chosen to be the T gate. Although T gates provide universality, fault-tolerant T gates are much more expensive than fault-tolerant Clifford gates because extra processes are required such as resource state preparation, magic state distillation and gate teleportation (see section 3.2.3). Hence, it is vital to consider T gate optimization in any Clifford + T quantum architecture.

The T gate belongs to the 3rd level of the *Clifford hierarchy* where the l^{th} level of the Clifford hierarchy on n qubits is defined recursively as follows:

$$\mathcal{C}_{n,l} = \{U \mid U^\dagger P U = Q,\ Q \in \mathcal{C}_{n,l-1} \ \forall\ P \in \mathcal{P}_n\}, \tag{2.16}$$

where recursion terminates with $\mathcal{C}_{n,1} = \mathcal{P}_n$. Note that $\mathcal{C}_{n,2} = \mathcal{C}_n$, the Clifford group. It follows from the definition of the Clifford hierarchy that the T gate conjugates to a Clifford gate under conjugation by a Pauli operator. This property is exploited in chapter 4 to reformulate input Clifford + T circuits into a T gate optimization friendly format (see fig. 2 of chapter 4).

2.3 Quantum Circuit Model

In the quantum circuit model, a quantum circuit is a time-ordered set of quantum operators taken from a discrete gate set $G = \{G_1, G_2, \ldots, G_m\} \subseteq \mathcal{U}_n$ where the qubit(s) to which each gate is applied has been specified. Quantum wires represent individual qubits and carry quantum information between inputs and outputs (either of the circuit or of individual gates). An example quantum circuit implementing the Toffoli gate is shown in figure 2.1.

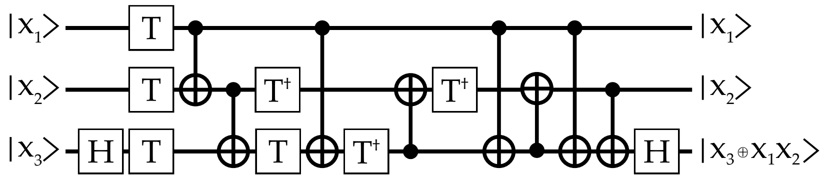


Figure 2.1: An example quantum circuit that implements the Toffoli gate, whose action on basis state $|x_1, x_2, x_3\rangle$ is defined as $U_{\text{Toff}}\,|x_1, x_2, x_3\rangle = |x_1, x_2, x_3 \oplus x_1 x_2\rangle$. Time flows from left to right. Each quantum wire represents a qubit and is labelled according to its input/output states. Single qubit quantum gates are boxes and act on the connected qubit with the contained unitary. The CNOT gate is a solid circle representing the control qubit, connected by vertical wire to an $\oplus$ representing the target. Multi-qubit gates can also be represented as labelled boxes but are absent in this example.

Let U be an n-qubit unitary for some quantum algorithm, and let

$$\mathcal{C} = \mathcal{U}_1 \circ \mathcal{U}_2 \circ \cdots \circ \mathcal{U}_N \tag{2.17}$$

be a circuit of length N where the gate $\mathcal{U}_t$ at time-step t implements some unitary $U_t \in \langle G \rangle$ and the operator $\circ$ performs circuit-concatenation with time ascending in left-to-right order. We say that the circuit $\mathcal{C}$ implements the unitary U if

$$U = U_N U_{N-1} \ldots U_1. \tag{2.18}$$

Here we emphasize the distinction between unitaries and quantum circuits that implement said unitaries. In other areas of study, it is common to use these terms interchangeably. However, in the study of quantum compilation where the cost of specific circuit decompositions is a key metric, it is worth clarifying this distinction.

2.4 Mixed States and the Density Operator

So far we have dealt exclusively with *pure* states, which are quantum superpositions over a set of orthonormal basis states. It is also possible to describe classical mixtures of an ensemble of pure states using *density operators*. Let $|\psi_1\rangle, |\psi_2\rangle, \ldots, |\psi_m\rangle$ be an ensemble of pure states and $p_1, p_2, \ldots, p_m$ be their respective classical probabilities. The density operator that describes this system is given by:

$$\rho = \sum_{i=1}^{m} p_i |\psi_i\rangle \langle\psi_i| , \tag{2.19}$$

and the normalisation condition is

$$\sum_{i=1}^{m} p_i = \mathrm{Tr}(\rho) = 1, \tag{2.20}$$

where $\mathrm{Tr}(\rho)$ denotes the trace of ρ. Density operators allow us to describe the dynamics of open quantum systems such as those affected by environmental noise or quantum measurement, which will become relevant in section 3.2.

Chapter 3

The Clifford + T Cost Model

As the original work presented in this thesis is concerned with T gate optimization, it is imperative to motivate the Clifford + T model for gate synthesis and in particular to motivate the T count. In this spirit, this chapter is divided into two parts. The first section demonstrates how the T gate - or any other non-Clifford gate - is necessary for universal quantum computation. The second section is dedicated to showing why T gates are unavoidably expensive in comparison to the other (Clifford) operations available in the quantum circuit model.

3.1 Universal Quantum Computation

The purpose of this section is to show how the Clifford + T gate set is universal for quantum computation. First, we will explain what universality means when describing a discrete quantum gate set and provide a definition for universality that allows for approximation. Then we show that Clifford gates by themselves are not universal, which implies that a non-Clifford gate is required for universality. Finally, we show that augmenting a generating set for the Clifford group with the T gate is sufficient for universal quantum computation. We do this in two stages: first, proving that $CNOT$ + single qubit gate is universal; and second, by proving that single qubit Cliffords + T is universal for single qubits. Universality of Clifford + T follows trivially as a corollary of these two parts.

3.1.1 What Does Universality Mean?

Universality is a desirable property of a quantum architecture that allows for any quantum operation to be performed. Specifically, in the quantum circuit model, universality means that any unitary can be constructed using only gates provided by the quantum architecture.

More concretely, let U be the target unitary and $G = \{G_1, G_2, \ldots, G_N\}$ be the discrete gate set provided by the quantum architecture. We say that G is universal for quantum computation if any U can be written as follows:

$$U = U_k U_{k-1} \ldots U_1, \tag{3.1}$$

where each of the k gates $U_i \in G$, for any choice of $U \in \mathcal{U}$.

In general, it is not possible to generate the full continuous space of quantum operators using a discrete gate set. That is, given a unitary U, one cannot necessarily expect any sequence of gates as in eq. (3.1) to implement U *exactly* (except in special cases). However, for most purposes, it suffices to find a gate sequence that implements U *approximately.*

We say that a unitary V approximates another unitary U up to an approximation error ϵ if it satisfies the following condition:

$$E(U, V) \equiv \max_{|\psi\rangle \in \mathcal{H}} \left[|U |\psi\rangle - V |\psi\rangle | \right] \leq \epsilon. \tag{3.2}$$

Now we can establish a more practical definition of universality that allows approximation.

Let G be the gate set provided by the quantum architecture and ϵ be the desired accuracy. We say that G is universal for quantum computation if, for any choice of target unitary $U \in \mathcal{U}$ and $\epsilon > 0$, there exists a unitary V such that $E(U, V) \leq \epsilon$, and V can be decomposed as a sequence of gates from G, as in:

$$V = V_k V_{k-1} \dots V_1, \tag{3.3}$$

where each of the k gates $V_i \in G$.

Now that we have a working definition of universality, in the following section we will see that a discrete gate set formed of only Clifford gates fails this criteria and so is not universal by itself. Moreover, the Gottesman-Knill theorem states that any quantum circuit composed of only Clifford operations can be simulated efficiently on a classical computer, which implies that at least one non-Clifford gate is required in any universal gate set (unless a quantum computer can be efficiently simulated on a classical computer).

As well as being universal, a gate set should be *efficient* in that the number of gates required to approximate a given unitary should not blow up exponentially as the approximation error tends to zero. Let G be a universal gate set such that $G^\dagger = G$. The *Solovay-Kitaev theorem* proves that any U can be approximated with error ϵ using $k = O\left(\ln\left(\frac{1}{\epsilon}\right)^c\right)$ gates from G where $c \approx 2$ is a constant [32, 33]. In other words, the Solovay-Kitaev theorem implies that universal gate sets are also efficient, universal gate sets.

3.1.2 Clifford Gates are Not Universal

Ignoring global phases, there are 6 states reachable by applying single-qubit Clifford gates to the state $|0\rangle$. These so called stabilizer states are: $|0\rangle$, $|1\rangle$, $|+\rangle$, $|-\rangle$, $\frac{|0\rangle + i|1\rangle}{\sqrt{2}}$ and $\frac{|0\rangle - i|1\rangle}{\sqrt{2}}$. Clearly, since there are only a finite number of reachable states, we can intuitively see that the single-qubit Clifford gates are not universal. Furthermore, we shall see from the following simple example that the universality criteria defined in section (3.1.1) fails.

Let our target unitary $U \neq I$ be some unitary that maps $|0\rangle$ to some state $|\psi\rangle \neq |0\rangle$, i.e.:

$$U |0\rangle = |\psi\rangle, \tag{3.4}$$

such that $|\,|0\rangle - |\psi\rangle\,| = \delta > 0$. Let us assume that $|0\rangle$ is the closest stabilizer state to $|\psi\rangle$ so that our best implementation using Cliffords is $V = I$. It follows that $E(U, V) \geq |\,|0\rangle - |\psi\rangle\,| = \delta$. Now if we choose an accuracy of $\epsilon < \delta$, we have $E(U, V) > \epsilon$ so the universality criteria from section 3.1.1 fails.

We have sketched a proof for single qubit Clifford operators, but a similar argument can be made for the general case of n-qubit Clifford operations. For any value of n, there are finitely many distinct *stabilizer states*, which are states accessible by applying U to $|\mathbf{0}\rangle$ for all $U \in \mathcal{C}$. Therefore, one can always find a point on the Bloch hypersphere with a distance greater than ϵ from any stabilizer state in the limit $\epsilon \to 0$. By construction, such a state cannot be prepared to arbitrary accuracy using only Clifford gates and so Clifford operations are non-universal.

3.1.3 CNOT + Single Qubit Rotation is Universal

In this section, we will show that the CNOT gate and the single qubit rotation gate $R_{\hat{n}}(\theta)$ form a universal gate set. This will be done in two parts: first, we will show that an arbitrary n-qubit unitary can be constructed from a sequence of 2-level unitaries. Secondly, we will show that any 2-level unitary can be formed from the single qubit rotation and the $CNOT$ gate.

Let U be the target n-qubit unitary. If we can find a sequence of unitaries $U_1, U_2, \ldots, U_m$ that transforms U into the identity, i.e:

$$U_m U_{m-1} \ldots U_1 U = I, \tag{3.5}$$

then we can recover a decomposition of U in terms of the U_i using the following:

$$U = U_1^\dagger U_2^\dagger \ldots U_m^\dagger. \tag{3.6}$$

Suppose each U_i were a 2-level unitary, by which we mean an n-qubit unitary that acts non-trivially on a 2-dimensional subspace. As $U_i^\dagger$ is also a 2-level unitary, it would follow that we can decompose U as a sequence of 2-level unitaries. In a process reminiscent of Gaussian elimination, our strategy is to find a formula for such U_i unitaries, each bringing U iteratively closer to I.

We proceed by constructing a template unitary, $V(U, k, l)$, which sets a single off-diagonal element of U to zero. More precisely, $V(U, k, l)$ performs the transformation $U \to U'$ such that $U'_{k,l} = 0$. We wish $V(U, k, l)$ to be a 2-level unitary, which enforces constraints. Assume $|k\rangle$ and $|l\rangle$ are the two computational basis states that span the space upon which $V(U, k, l)$ acts non-trivially. These constraints can be expressed as follows:

$$(V(U,k,l))_{i,j} = \begin{cases} a & i = j = l \\ b & i = k, j = l \\ c & i = l, j = k \\ d & i = j = k \\ \delta_{i,j} & \text{otherwise,} \end{cases} \tag{3.7}$$

where $a, b, c, d \in \mathcal{C}$ and $\delta_{i,j}$ is the Kronecker delta. It remains to find values for a, b, c and d. Let

$$\tilde{V}(U,k,l) = \begin{pmatrix} a & c \\ b & d \end{pmatrix} = \frac{1}{\sqrt{|U_{l,l}|^2 + |U_{k,l}|^2}} \begin{pmatrix} U^*_{l,l} & U^*_{k,l} \\ -U_{k,l} & U_{l,l} \end{pmatrix}, \tag{3.8}$$

be a matrix that is unitary and well-defined if $|U_{k,l}|^2 \neq 0$ or $|U_{l,l}|^2 \neq 0$. One can verify that the values of a, b, c and d implied by eq. (3.8) lead to a unitary $V(U,k,l)$ as in eq. (3.7) that performs the transformation $U' = V(U,k,l)U$ such that $U'_{k,l} = 0$. Note that in addition to setting $U'_{k,l} = 0$, the entire k^{th} and l^{th} rows of U' may differ from U. Otherwise, the elements of U' and U are identical.

Now that we have a unitary that sets off-diagonal elements to zero, let us construct a template unitary, $W(U,k)$ that sets the k^{th} diagonal element equal to one. If we assume that $U_{i,j} = 0$ for all $i \in [1, 2^n] \setminus j$, then

$$(W(U,k))_{i,j} = \begin{cases} U^*_{k,k} & i = j = k \\ \delta_{i,j} & \text{otherwise} \end{cases}. \tag{3.9}$$

We can now describe the full procedure to construct an arbitrary unitary from 2-level gates. Start by initializing an empty ordered list of 2-level unitaries that we denote as C. For each column of U, starting with the 1st column and proceeding in ascending order where j is the current column index, for each $i \in [(j+1), 2^n]$, if $\left((\prod_{t=1}^{|C|} C_t)\ U\right)_{i,j} = 0$, then continue to the next value of i. Otherwise, append $V(U,i,j)$ to C. Having iterated over all i values, if $\left((\prod_{t=1}^{|C|} C_t)\ U\right)_{j,j} = 1$, then continue to the next j value. Otherwise, append $W(U,j)$ to C. Having iterated over all j values, C now contains an ordered list of 2-level unitaries whose right-to-left product is $U^\dagger$. Finally, replace each element of C with its conjugate and reverse the order of C to recover U.

We have now shown that an arbitrary n-qubit unitary can be decomposed as a sequence of 2-level unitaries. We will now show that any 2-level unitary can be implemented as a circuit of $CNOT$ gates and single qubit gates.

Let V be the 2-level unitary and $|\mathbf{x}\rangle$ and $|\mathbf{y}\rangle$ be the two basis states that span the space upon which V acts non-trivially, where $\mathbf{x}$ and $\mathbf{y}$ are binary strings. Let $\tilde{V}$ be a single qubit unitary formed by removing the trivial rows and columns from V. If $\mathbf{x}$ and $\mathbf{y}$ are different in only one bit location, then V can be implemented by applying a controlled-$\tilde{V}$ gate whose target is the differing bit and is conditioned on the remaining bits being identical. Note that a controlled version of any unitary can be simulated using only $CNOT$ and single qubit gates [34].

In general, $\mathbf{x}$ and $\mathbf{y}$ are different in up to n locations. But we can form a circuit using only $CNOT$s and single qubit gates that permute the basis states such that $|\mathbf{x}\rangle$ is swapped with a basis state, $|\mathbf{z}\rangle$, where $\mathbf{z}$ is different to $\mathbf{y}$ in only one location. We do this by first forming a Gray code, C, between $\mathbf{x}$ and $\mathbf{y}$, which is sequence of binary strings terminating with $\mathbf{x}$ and $\mathbf{y}$ such that each pair of subsequent elements of C is different at exactly one bit location.

We can swap each subsequent pair of elements of C by performing a multiply-controlled Pauli-X gate conditioned on the remaining bits being identical. By concatenating such a gate for every subsequent pair of elements of $C \setminus \mathbf{y}$, we construct a circuit that implements the permutation, U_P, $|\mathbf{x}\rangle \leftrightarrow |\mathbf{z}\rangle$. The final circuit is thus U_P followed by the multiply-controlled $\tilde{V}$ gate followed by $U_P^\dagger$, which is U_P with the gates reversed and implements the 'uncompute' step.

3.1.4 $H + T$ approximates any Single Qubit Rotation

In this section we show how an arbitrary single qubit rotation can be constructed using only H and T gates. Let $\hat{\mathbf{n}}$ be a real unit vector and $\sigma = (X, Y, Z)$. The rotation operator $R_{\hat{\mathbf{n}}}(\theta)$ is a single qubit gate that performs a rotation on the Bloch sphere of θ radians around the $\hat{\mathbf{n}}$-axis and is given by:

$$R_{\hat{\mathbf{n}}}(\theta) = \exp(-i\frac{\theta}{2}\sigma \cdot \hat{\mathbf{n}}) = \cos\frac{\theta}{2} I - i\sin\frac{\theta}{2}\left(n_x X + n_y Y + n_z Z\right). \tag{3.10}$$

Up to an unimportant global phase, the T gate can be written in the form of equation (3.10) as:

$$T = \exp\left(-i\frac{\pi}{8}Z\right) = \cos\frac{\pi}{8} I - i\sin\frac{\pi}{8} Z \tag{3.11}$$

The operator HTH can be written as:

$$HTH = \cos\frac{\pi}{8} I - i\sin\frac{\pi}{8} X \tag{3.12}$$

Consider the composite operator $(T)(HTH)$, which we can construct by multiplying eq. (3.11) by eq. (3.12)

$$\begin{aligned}(T)(HTH) &= \left(\cos\frac{\pi}{8} I - i\sin\frac{\pi}{8} Z\right)\left(\cos\frac{\pi}{8} I - i\sin\frac{\pi}{8} X\right) \\ &= \cos^2\frac{\pi}{8} I - \sin^2\frac{\pi}{8} ZX - i\sin\frac{\pi}{8}\cos\frac{\pi}{8}\left(Z + X\right) \\ &= \cos^2\frac{\pi}{8} I - i\sin\frac{\pi}{8}\left[\cos\frac{\pi}{8} X + \sin\frac{\pi}{8} Y + \cos\frac{\pi}{8} Z\right]\end{aligned} \tag{3.13}$$

So the composite operator $(T)(HTH)$ is given by:

$$(T)(HTH) = \exp(-i\frac{\theta}{2}\sigma \cdot \hat{\mathbf{n}}) \tag{3.14}$$

where

$$\theta = 2\arccos\left(\cos^2\left(\frac{\pi}{8}\right)\right) \tag{3.15}$$

and

$$\hat{\mathbf{n}} = \sqrt{\frac{12 - 2\sqrt{2}}{17}} \begin{pmatrix} \cos\frac{\pi}{8} \\ \sin\frac{\pi}{8} \\ \cos\frac{\pi}{8} \end{pmatrix} \tag{3.16}$$

and $\sigma := (X, Y, Z)^T$. In summary, the rotation $(T)(HTH)$ is a rotation about the $\hat{\mathbf{n}}$ axis of an angle θ that is an irrational multiple of 2π.

Since θ is an irrational multiple of 2π, the set $\{n\theta \pmod{2\pi} \mid n \in \mathbb{Z}\}$ can be identified with the interval $[0, 2\pi]$. To see this, let j and k be distinct integers. The argument is that if $\frac{\theta}{2\pi} \notin \mathbb{Q}$, then $j\theta \pmod{2\pi} \neq k\theta \pmod{2\pi}$ for any choice of j and k such that $j \neq k$. If this were not true, then $j\theta = k\theta + 2\pi p$ for some integer p. Rearrange this and we have

$$\frac{\theta}{2\pi} = \frac{p}{j-k}, \tag{3.17}$$

and since $j \neq k$, this contradicts the previously stated non-membership of $\frac{\theta}{2\pi}$ in the rationals. This means that the choice of n is unique for a given value of $n\theta \pmod{2\pi}$ implying that the latter provides a dense cover over the interval $[0, 2\pi]$. Therefore, using only H and T, we can construct an arbitrary rotation around the $\hat{\mathbf{n}}$ axis: $R_{\hat{\mathbf{n}}}(\alpha) = R^a_{\hat{\mathbf{n}}}(\theta)$ where $\alpha \in [0, 2\pi]$ is any angle and $a \in \mathbb{Z}$ is any integer such that $|\alpha - \theta^a| \pmod{2\pi} \leq \delta$, our accuracy parameter.

It can be shown that a rotation of ϕ radians about an arbitrary axis, $\hat{\mathbf{m}}$, can be constructed from rotations around any two distinct axes. Since we can construct a rotation about an axis perpendicular to $\hat{\mathbf{n}}$ by conjugating $R_{\hat{\mathbf{n}}}(\alpha)$ with Hadamard gates, we have:

$$\begin{aligned} R_{\hat{\mathbf{m}}}(\phi) &= R_{\hat{\mathbf{n}}}(\alpha) H R_{\hat{\mathbf{n}}}(\beta) H R_{\hat{\mathbf{n}}}(\gamma) \\ &= R^a_{\hat{\mathbf{n}}}(\theta) H R^b_{\hat{\mathbf{n}}}(\theta) H R^c_{\hat{\mathbf{n}}}(\theta) \end{aligned}, \tag{3.18}$$

where the angles α, β and γ are fixed by $\hat{\mathbf{m}}$ and ϕ, and in turn, the integers a, b and c can be found for their corresponding angles. Note that any gate (not just T) that effects a rotation of an irrational multiple of π about some axis can be used in the above construction of universality.

3.2 Fault-Tolerant Quantum Computation

Quantum computers are highly sensitive to environmental noise. Although a closed quantum system evolves unitarily, it is impossible to decouple any system from its environment entirely. As far as we can tell, the universe is a closed system, and it is composed of the quantum computer and everything that is not the quantum computer, that latter of which we call the 'environment'. When we trace out the environment, it effects a noisy channel that makes the quantum state more 'mixed' or classical. This phenomenon is called decoherence, and in experimental settings is significant enough that unprotected qubits decohere too rapidly for any useful computation to be performed.

The situation is especially dire when compared to classical computers. Classical computers are inherently protected from noise because classical information is fundamentally *discrete*. The elementary values of '0' and '1' are represented by different voltage bands whose separation is typically much larger than the amplitude of the noise produced by background electromagnetic radiation sources.

Although classical computers suffer much less from noise, error correction codes have been developed to protect the transmission of classical information over noisy channels, over large distances, in RAM or to store information on hardware that may be more vulnerable to damage such as optical discs. These codes cannot be directly repurposed for quantum computers, as we will see, but ideas from the classical realm have driven the development of quantum error correction (QEC) codes that can successfully overcome the obstacle of quantum noise.

3.2.1 Quantum Error Correction

In this section, we will introduce quantum error correction codes by first looking at a simple example of a classical error correction code: the repetition code. Then we will discuss why quantum mechanics forbids the direct usage of such codes and describe these differences in terms of the closest quantum analogue of the three qubit repetition code. Then we introduce the stabilizer formalism, a powerful tool for concisely describing quantum error correction codes and the constraints it places on the physical implementation of encoded quantum operators. Finally, we introduce the notion of fault-tolerant quantum computation and the transversality property of quantum operators and see that the latter property is fixed by the definition of the QEC code.

Suppose we wish to transmit a single classical bit with value x across a noisy channel. For concreteness we will consider the binary symmetric channel, which flips the bit with probability p, otherwise it leaves the bit unchanged:

$$0 \to \begin{cases} 0 & (1-p) \\ 1 & p \end{cases}, \tag{3.19}$$

$$1 \to \begin{cases} 1 & (1-p) \\ 0 & p \end{cases}. \tag{3.20}$$

The 3-bit repetition code protects x by appending 2 redundant copies of x to the message to be transmitted:

$$0_L \to 000 \tag{3.21}$$

$$1_L \to 111 \tag{3.22}$$

where the subscript L denotes that the bit is the 'logical' or encoded version of that bit value. The strings of 'physical' bits, 000 and 111, used to represent each logical bit is called a *codeword*.

The noisy channel affects each bit independently. Suppose we want to transmit $x_L = 0_L$. We would encode our message giving 000. The received message after the noisy channel has affected our bit string could be any of the $2^3 = 8$ different 3-bit strings. Say 010 is received. We can recover the original message by taking the majority function $Maj(010) = 0$. This process is an example of a *decoder*, whose task it is to determine the most likely error that has occurred and apply an appropriate correction operator, in this case flipping the 2nd bit. The assumption here is that an error has occurred on a single bit. If instead the first two bits were flipped by the noisy channel, then the received message would be 110 so our decoder would determine the original message was $Maj(110) = 1$, which is incorrect. We say that this is a *logical* error. Fortunately, the probability of this occurring is $p^2 < p$, so the code has successfully suppressed the effect of noise on our system.

The above example of classical error correction relies on two key facts: that we can create copies of any binary value, and that we can evaluate the values of bits. Unfortunately, we cannot create copies of an arbitrary unknown quantum state due to the no-cloning theorem [35]. Furthermore, evaluating a qubit's 'value' requires measurement, which destroys entanglement between qubits and should only be done at the end of the qubit's usefulness in a given quantum algorithm. In addition, only classical information can be extracted from quantum measurement and the quantum information stored in the probability amplitudes would be destroyed.

Now we will construct a QEC code that can correct a single quantum bit flip, or Pauli-X error. Say we want to send a single qubit, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, down a binary symmetric channel that experiences a Pauli-X error with probability p. We can protect $|\psi\rangle$ by encoding the *basis* states using the quantum repetition code:

$$|0_L\rangle \rightarrow |000\rangle \tag{3.23}$$
$$|1_L\rangle \rightarrow |111\rangle \tag{3.24}$$

so that $|\psi_L\rangle = \alpha|000\rangle + \beta|111\rangle$. This can be done using $CNOT$ gates in a circuit such as in fig. 3.1.

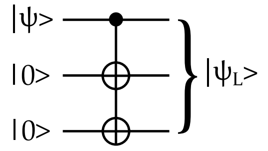
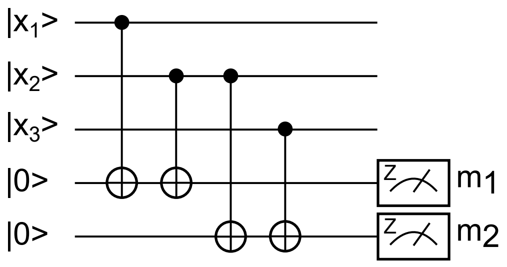


Figure 3.1: Circuit that encodes the 3-qubit repetition code.

Now suppose a Pauli-X error occurs on the 2nd qubit. The state is now

$$|\psi'_L\rangle = \alpha|010\rangle + \beta|101\rangle . \tag{3.25}$$

We cannot simply measure each qubit as we might naively expect from the classical code as this will collapse the state into $|010\rangle$ or $|101\rangle$, destroying the quantum information contained within the probability amplitudes α and β. However, we can perform non-destructive projective measurement of the operators $Z \otimes Z \otimes I$ and $I \otimes Z \otimes Z$ using a circuit such as in fig. (3.2), which yield eigenvalues $m_1, m_2 \in \{+1, -1\}$ that allow us to determine upon which qubit the single qubit Pauli-X error occurred. The pair (m_1, m_2) is an example of a *syndrome* and the process of using syndromes to determine the required recovery operator is called *decoding.* In our example, measuring $|\psi'_L\rangle = \alpha|010\rangle + \beta|101\rangle$ yields the syndrome $(-1, -1)$, which corresponds to a single qubit Pauli-X error acting on the 2nd qubit with probability p or two errors (on the 1st and 3rd qubits) with probability p^2. Assuming the most likely error occurred, we can recover the original state by applying the recovery, or correction operator, $I \otimes X \otimes I$. This protocol fails with probability $p^2 < p$, which means the noise has been effectively suppressed.



m_1	m_2	E
+1	+1	I
-1	+1	X_1
-1	-1	X_2
+1	-1	X_3

Figure 3.2: A circuit that measures the parity operators $Z\otimes Z\otimes I$ and $I\otimes Z\otimes Z$, whose eigenvalues m_1 and m_2, respectively, allows one to determine the most likely single qubit Pauli-X error, E, that occurred on the 3-qubit repetition code.

We have demonstrated a simple quantum error correction code that corrects a single Pauli-X error. A similar code can be constructed that corrects Pauli-Z errors by using $|0_L\rangle = |+++\rangle$ and $|1_L\rangle = |---\rangle$. We can then *concatenate* the two types of repetition code to form a code that can correct both Pauli-X and Z errors:

$$\begin{aligned} |0_L\rangle &= |+_{L'} +_{L'} +_{L'}\rangle \\ &= \frac{1}{2\sqrt{2}} (|0_{L'}\rangle + |1_{L'}\rangle) \otimes (|0_{L'}\rangle + |1_{L'}\rangle) \otimes (|0_{L'}\rangle + |1_{L'}\rangle) \\ &= \frac{1}{2\sqrt{2}} (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \end{aligned} \tag{3.26}$$

$$\begin{aligned}|1_L\rangle &= |-_{L'}-_{L'}-_{L'}\rangle \\ &= \frac{1}{2\sqrt{2}}\left(|0_{L'}\rangle - |1_{L'}\rangle\right)\otimes\left(|0_{L'}\rangle - |1_{L'}\rangle\right)\otimes\left(|0_{L'}\rangle - |1_{L'}\rangle\right) \\ &= \frac{1}{2\sqrt{2}}\left(|000\rangle - |111\rangle\right)\otimes\left(|000\rangle - |111\rangle\right)\otimes\left(|000\rangle - |111\rangle\right)\end{aligned} \tag{3.27}$$

The resulting code is Shor's 9-qubit code and can be decoded by measuring the operators

$$\begin{array}{ccccccccc} Z & Z & I & I & I & I & I & I & I \\ I & Z & Z & I & I & I & I & I & I \\ I & I & I & Z & Z & I & I & I & I \\ I & I & I & I & Z & Z & I & I & I \\ I & I & I & I & I & I & Z & Z & I \\ I & I & I & I & I & I & I & Z & Z \\ X & X & X & X & X & X & I & I & I \\ I & I & I & X & X & X & X & X & X \end{array} \tag{3.28}$$

We have given an example QEC code that we have defined in terms of encoded quantum *states*. One can describe QEC codes more concisely in terms of quantum *operators* using the *stabilizer formalism*. In the stabilizer formalism, a quantum code is defined in terms of a special set of operators called *the stabilizer*. The stabilizer is an abelian subgroup of the Pauli group on n qubits, excluding $-I$. Elements of the stabilizer are called stabilizer operators and are defined to be the set of operators that act trivially on encoded states:

$$S = \{s | s\,|\psi_L\rangle = |\psi_L\rangle\} \subset \mathcal{P}_n. \tag{3.29}$$

For example, the rows in equation (3.28) are elements of the stabilizer for Shor's 9-qubit code. In fact, they form a minimum generating set for the "full" stabilizer.

The above definition can be used in reverse: given a stabilizer, the protected space is spanned by the intersection of the $+1$ eigenstates of each member of the stabilizer. An $[[n,k,d]]$ stabilizer code is defined by a set of independent stabilizer generators $\langle s_1, s_2, \ldots, s_l\rangle \subset \mathcal{P}_n$ where $l = n-k$ and a set encoded (or *logical*) Pauli-X and Z operators $\{X_{L,1}, Z_{L,1}, X_{L,2}, Z_{L,2}, \ldots, X_{L,k}, Z_{L,k}\}$. Note that n is the number of physical qubits, k is the number of encoded qubits and d is the *distance* of the code, which is the minimum weight over the set of all operators that transform one encoded basis state into another, where the weight of an operator is the number of qubits with non-trivial support. The logical operators must commute with the stabilizer. Otherwise, measuring the stabilizer would interfere with the encoded state. Each pair of logical operators must obey the commutation relations of the operators that they encode. For instance, X_L must anticommute with Z_L. In fact, there are 2^l physical implementations of each logical Pauli as $SZ_L\,|\psi\rangle = Z_L S\,|\psi_L\rangle = Z_L\,|\psi_L\rangle$, so SZ_L is also a logical Pauli-Z. Having defined a stabilizer code, including the logical Paulis, the encoded versions of other operators are also fixed by their conjugation relations E.g. H_L must be implemented so that $H_L X_L H_L^\dagger = Z_L$ and so on.

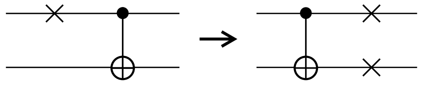


Figure 3.3: An example error propagation process is shown. Errors represented by crosses can multiply by conjugating through entangling gates, such as the $CNOT$ gate.

We have seen how a well defined QEC code fixes the definition of encoded quantum operators by their conjugation relations. However, the existence of such a definition is not our sole concern; it is also essential that each encoded gate can be performed *fault-tolerantly.* In the fault-tolerant quantum computation (FTQC) model, we have a register of *logical* qubits, each of which is represented by a number of *physical* qubits that contribute toward the state space of a QEC code. Each bundle of physical qubits that is used to implement an independent instance of a QEC code is called a *codeblock.* The primary concern of fault-tolerant quantum computation is to ensure that errors do not propagate (i.e. multiply within the same codeblock) when the dynamics of a quantum computer is effected by encoded operators. If this were allowed, then the act of executing a quantum algorithm would cause errors to accumulate and eventually overwhelm the QEC code's capabilities. We say that an encoded gate is fault-tolerant iff, failure of a single component of the encoded gate leads to failure on at most a single physical qubit in the same codeblock after the encoded gate has been applied. For example, H_L for the Steane code is $H^{\otimes 7}$ [36, 37]. This operator does not propagate errors due to the bit-wise nature of its physical implementation. If an encoded operator (as above) has a physical implementation involving only physical components that act independently (i.e. in a bitwise fashion) on each physical qubit within the same code block, then we say that the encoded operator is *transversal.* From construction, transversality is sufficient for fault-tolerance. In the next section, we will see that for 'good' codes, Clifford gates have transversal implementations, whereas the T gate does not.

3.2.2 Transversal Gates

In the previous section, we saw that QEC codes are necessary to protect quantum computers from noise and that the specific code used fixes the encoded gates, and in particular, their transversality properties, which determine how (and how cheaply) an encoded gate can be implemented fault-tolerantly. As well as how costly it is to effect fault-tolerant gates, it is necessary to consider how effective a QEC code is at combating noise. To this end, we will introduce the notion of the 'threshold' as a metric for determining a QEC code's efficacy, culminating in a statement of the 'threshold theorem'. We will then introduce the 'toric code', provide its definition including its stabilizer generators and logical Pauli-X and Z gates. Then we will show how the toric code has a high threshold and a transversal $CNOT$ gate and inexpensive (albeit not transversal) fault-tolerant S and H gates. Finally, we show how, despite these desirable properties, the toric code lacks a transversal implementation of the T gate, implying another method must be used that must be fault-tolerant in order to obtain the full universal gate set of H, S, $CNOT$ and T. We will conclude the section with the more general result of Eastin-Knill, which states that there exists no non-trivial QEC code that permits a gate set which is both universal and transversal.

Consider an $[[n, k, d]]$ stabilizer code, S. One can imagine taking each of the n physical qubits used as input for S and replacing it with a logical qubit, itself encoded with another instance of S that we denote as S'. Each of the n physical qubits of the S's can in turn be replaced by a further instance of S, called S'' and so forth ad infinitum. This process of recursive nesting of QEC codes is called *concatenation* and allows for the construction of very large codes from relatively simple descriptions.

But how do additional concatenation layers affect the probability of failure of the resulting QEC code when compared to the original? One might naively think that a larger code increases protection against noise. However, if each component introduces noise to the system, then increasing code size also increases noise. One can imagine that below a certain per-gate physical noise level (or *threshold*), increasing the code size by a single level will reduce the logical failure probability more than the additional noise increases it. It would follow that increasing the code size asymptotically reduces the logical error rate to zero. Conversely, above the threshold, increasing the code size introduces more noise than the additional protection can cope with. So asymptotically, the logical failure rate tends to unity. The *threshold theorem* states that this is indeed the case and every family of codes has a threshold for a given noise model.

Threshold theorem for quantum computation.
The following statement of the threshold theorem is taken verbatim from [38].

> A quantum circuit containing $p(n)$ gates may be simulated with probability of error at most ϵ using
>
> $$\mathcal{O}\left(\text{poly}\left(\log\left(p(n)/\epsilon\right)\right)p(n)\right) \tag{3.30}$$
>
> gates on hardware whose components fail with probability at most p, provided p is below some constant *threshold*, $p < p_{\text{th}}$ and given reasonable assumptions about the noise in the underlying hardware.

One family of codes is the *toric code*, which is defined on an $N \times N$ square lattice, where qubits live on the edges. The lattice has periodic boundary (hence toric), so the $(N+1)^{\text{th}}$ horizontal (vertical) grid line is identified with the 1^{st} horizontal (vertical) grid line etc.

The stabilizer generators come in two forms: stars and plaquettes. The star operators are centred on each vertex of the lattice and are composed of Pauli-X

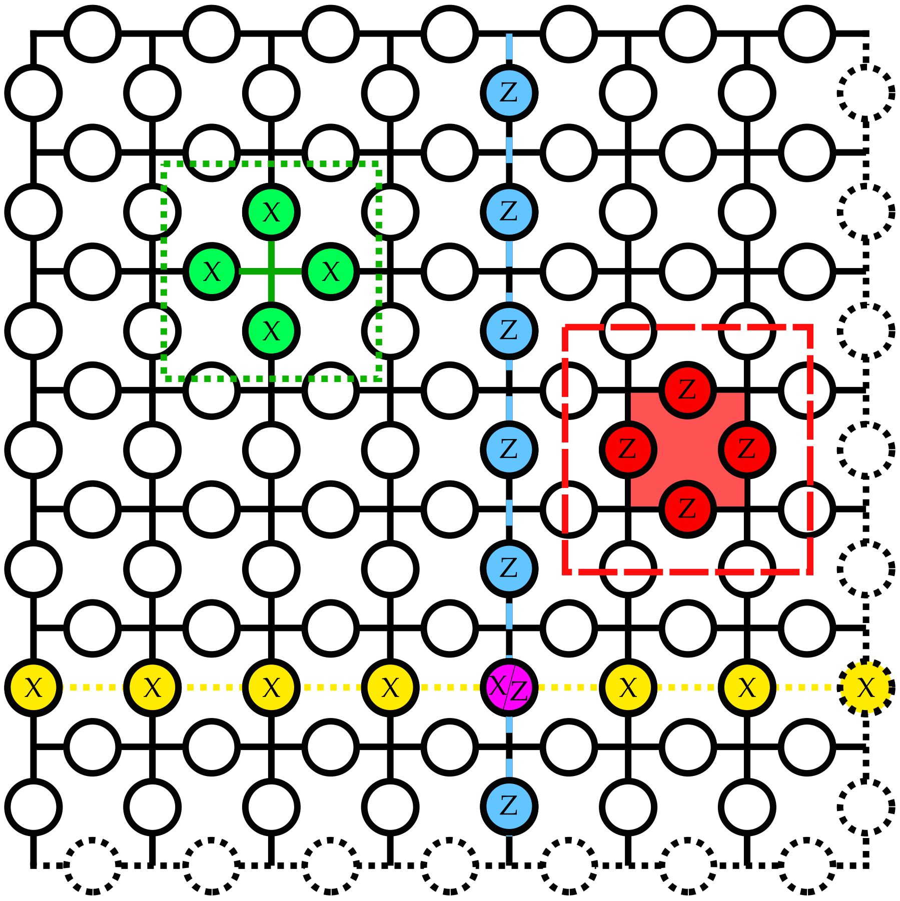


Figure 3.4: The toric code for $N = 7$ along with a representative for each species of stabilizer generator, star and plaquette, bounded by dotted and dashed boxes, respectively. The Logical Pauli X and Z operators are shown as dotted and dashed lines, respectively, which span the lattice.

gates applied to each of the 4 adjoining edges. Plaquettes are similarly defined but are centred on the body of each square in the lattice. It is straightforward to see that the stabilizer generators all mutually commute: either they are same type (vertex or plaquette) or they are not. If they are, they commute trivially as they are each composed of either Pauli-X or Pauli-Z's, and all operators self-commute. If they are not the same type, then they either share non-trivial support one or more qubit, or they do not. If they share no qubits then they trivially commute. Otherwise, consider a plaquette with a star on the top-right. They share support on 2 qubits, each contributing a factor of -1 to the phase, so they commute. By rotational symmetry, all other overlapping

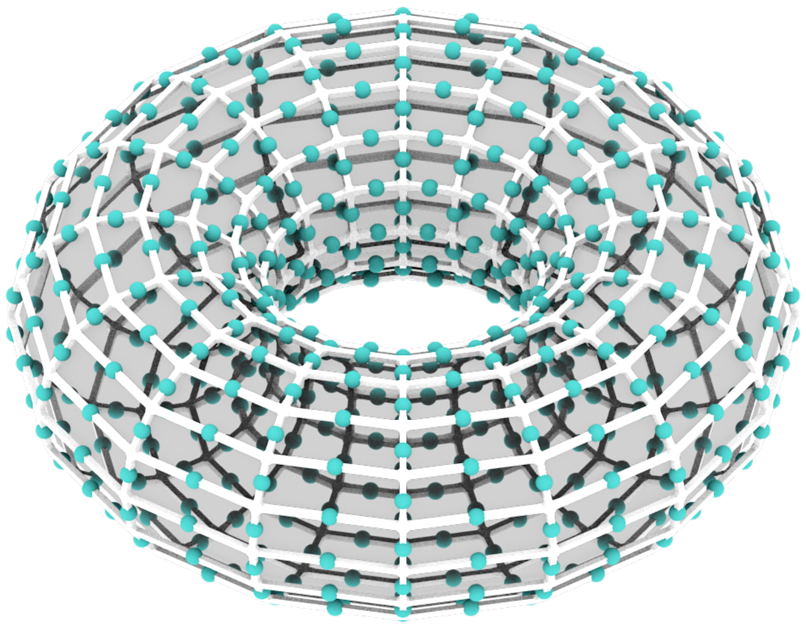


Figure 3.5: Visualisation of the $N = 16$ toric code.

pairs of plaquettes and stars commute. It remains to define the logical Pauli operators. These are strings that circumnavigate the periodic lattice along a particular axis: the Pauli-Z's along lines and X's through plaquettes. There are 2 equivalence classes of logical Pauli: one for each orientation, horizontal and vertical, meaning the toric code encodes two qubits. Note that a pair of logical Pauli-Z and Pauli-X gates with *odd* overlap (i.e. strings of physical Pauli-Z's and Pauli-X's that traverse *opposite* axes in the toric lattice) belong to the *same* equivalence class and those with even overlap do not. One can verify that the logical Pauli-X's and Z's have even overlap with the plaquettes and stars, respectively, so they commute with the stabilizer generators and therefore the full stabilizer. Conversely, the logical Pauli's of the same equivalence class share an odd overlap so anticommute, as required, and Pauli's of opposite equivalence class commute.

Now we have established that the toric code is a valid stabilizer code, we refer the reader to evidence from previous work that the toric code has a high threshold of $\approx 10^{-2}$ [13]. This is much higher than the concatenated Steane code, for instance, which has a threshold of between 10^{-5} and 10^{-6}. Given that the toric code has a high threshold, it is desirable to find fault-tolerant implementations of gates for the toric code that form a universal gate set. The $CNOT$ gate has a transversal implementation. A logical $CNOT$ is implemented by physical $CNOT$'s applied where the targets and controls are applied in strings circumnavigating the target and control code blocks, respectively. Targets are strings through plaquettes and controls are strings along grid lines. The S gate does not have a transversal implementation for the toric code. However, it can be implemented using magic state distillation (see section 3.2.3) using slightly cheaper resource states than with the T gate or by code deformation techniques [39]. A logical H gate is implemented by a Hadamard on each qubit followed by a reflection along the lattice diagonal. Although the first part of the procedure is transversal, the second part is not. However, this implementation is still fault-tolerant as the reflection only requires swap gates between physical qubits in the same code block, so does not propagate errors. In addition, this implementation is 'cheap' in that it requires a quantity of gates that is linear in n.

In summary, we have fault-tolerant and cheap (albeit not necessarily transversal) definitions of the $CNOT$, H and S gate, which generate the Clifford group on n qubits, each gate using no more than $\mathcal{O}(n^2)$ physical gates. It remains to find a fault-tolerant implementation of the T gate. According to Bravyi-Koenig [40], no 2D topological code has a transversal T gate. In fact, there is a more general result of Eastin-Knill [41], which proves that no non-trivial code has a gate set that is both universal and transversal. Fortunately, we can implement a fault-tolerant T gate using a gate teleportation circuit such as in figure 3.6, provided a supply of high fidelity T states. As we will see in the next section, the latter can be provided in a fault-tolerant manner using magic state distillation.

3.2.3 Magic State Distillation

In the previous section, we saw how QEC codes with otherwise desirable properties lack a transversal implementation of any non-Clifford gate, such as the T gate, which is necessary for universal quantum computation. This implies some other method must be used to obtain a fault tolerant T gate. In this section, we describe such a method by first demonstrating the existence of a circuit that simulates an ideal T gate using only Clifford gates and a single copy of a pure ancilla $|T\rangle := T|+\rangle$ state. Second, we account for noise by assuming the resource state is a 'noisy' mixed state ρ with a certain fidelity with $|T\rangle$. We show for which values of fidelity of ρ we can achieve a given desired total accuracy for a circuit with N_T T gates. Next, we show that multiple copies of ρ_{raw} can be consumed to create fewer ρ with arbitrarily high fidelity, provided the fidelity of ρ_{raw} is greater than a certain threshold, and that this can be achieved in so called magic state distillation (MSD) protocols. We go on to describe the high level overview of a typical MSD subroutine, such as the $15 \to 1$ subroutine from [10] or triorthogonal codes from [11]. We conclude this section by discussing the high cost the T gate relative to Clifford gates and introduce the T count as a key metric for approximating the cost of quantum circuits.

The T gate can be simulated by a circuit such as in figure 3.6, which uses only Clifford gates and a single copy of the pure state

$$|T\rangle := T\,|+\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{i\frac{\pi}{4}}\,|1\rangle\right). \tag{3.31}$$

One can verify that this circuit, which we will call the 'T gadget', effects the T gate.

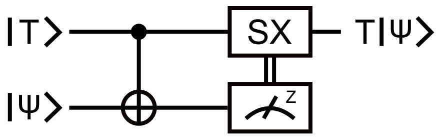


Figure 3.6: Circuit for implementing the T gate using Clifford gates and an ancillary $|T\rangle$ state.

The T gadget requires a supply of pure T-type magic states to implement an ideal T gate. These assumptions are unrealistic. So to account for noise, we update our model such that the ancilla is actually a mixed state ρ with a certain fidelity with respect to $|T\rangle$

$$F(\rho) = \max_{U \in \mathcal{C}_1} \sqrt{\langle T|\, U^\dagger \rho U\, |T\rangle}, \tag{3.32}$$

where $\mathcal{C}_1$ is the single qubit Clifford group.

To verify that the T gadget with noisy ancilla is fault-tolerant, consider uncorrelated noise occurring on the top qubit line at the beginning of the circuit. With probability p_z, the pure $|T\rangle$ ancilla is affected by a Pauli-Z error and with probability p_x, it is affected by a Pauli X error. If a Z error occurs, it commutes through the control of the $CNOT$ gate and, up to a global phase, to the end of the circuit. If an X error occurs, commuting it through the $CNOT$ gate propagates an additional error to the bottom line. However, the bottom qubit is subsequently measured, so the propagated X error only affects whether the SX correction is applied to the top line. Therefore, failure on a single component only introduces an error on a single qubit and the T gadget satisfies the criteria for fault-tolerance.

The mixed state ρ can be expressed as

$$\rho = (1-\epsilon)\,|T\rangle\,\langle T| + \epsilon\,|T_-\rangle\,\langle T_-|\,, \tag{3.33}$$

where $|T_-\rangle = T\,|-\rangle$ and $\epsilon \in [0,1]$ is the infidelity of ρ with respect to $|T\rangle$. Suppose we wish to implement a circuit with accuracy ϵ_{total} composed of some number of Clifford gates and N_T T gates. As noise combines linearly, each resource T state, ρ, should have a fidelity of

$$F \geq \sqrt{\frac{1+\epsilon}{2}} = \sqrt{\frac{1+\frac{\epsilon_{\text{total}}}{N_T}}{2}}. \tag{3.34}$$

The problem of implementing a fault-tolerant T gate has been reduced to the problem of preparing sufficiently high-fidelity T states for a given quantum algorithm. Magic state distillation allows us to convert large numbers of low-fidelity T states to a small number of high-fidelity T states. The fact that the T state possesses both the ability to efficiently implement a universal gate set and the ability to be distilled to arbitrarily high fidelity is why they are known as *magic states.*

A typical magic state distillation subroutine consists of the following five steps. First, a register of k qubits is prepared in the stabilizer state $|+\rangle^{\otimes d}$. Secondly, the register is then encoded using Clifford gates into an $[[n, k, d]]$ stabilizer code for which exists a transversal logical T gate, which is applied to the encoded register in the third step. Each unencoded T gate in step 3 is implemented using the circuit from fig. (3.6), consuming a T-type magic state and introduces noise in situ. Hence in step 4, the Pauli X stabilizer generators are measured non-destructively and a correction operation is applied to the encoded register, conditioned on the measured syndrome. In the final step, the register is decoded and if the last $n-k$ qubits are in any state other than $|0\rangle^{\otimes(n-k)}$, then an error has been detected and the result is discarded and the process must be restarted from the beginning. Otherwise, the subroutine has succeeded and the first k qubits are each in the mixed state

$$\rho' = (1-\epsilon')\,|T\rangle\,\langle T| + \epsilon'\,|T_{-}\rangle\,\langle T_{-}|\,, \tag{3.35}$$

where $\epsilon' \approx \epsilon^d < \epsilon$ for sufficiently small values of ϵ, so increases the fidelity. In general, the fidelity is increased provided that $\epsilon < \epsilon_{\text{th}}$ for some threshold value that depends on the code used and the noise model. For example, the 15-to-1 code has a threshold of $\epsilon_{\text{th}} \approx 0.141$ and has a failure probability of $p_{\text{out}} \approx 35p_{\text{in}}^3$. If we assume an input error probability of $p_{\text{in}} \approx 10^-2$, then a single round of the 15-to-1 code yields an output error probability of $p_{\text{pout}} \approx 35p_{\text{in}}^3 = 3.5 \times 10^{-5}$, which remains too noisy for most applications. However, we can *concatenate* MSD subroutines together by using the output ρ states of one instance as the input resource state of another. By nesting MSD protocols together in this way, we can reduce noise to arbitrarily low levels. Again using the 15-to-1 code as an example, it takes two rounds and a total of 225 resource T states to reduce the noise for a single output T state from $p_{\text{in}} = 10^-2$ to $p_{\text{out}} \approx 1.5 \times 10^{-12}$, which begins to approach the per-bit error rates of modern classical computers of $\approx 10^{-15}$. The number of resource states required per output magic state for a given physical error rate and target error rate is called the *overhead.* We refer the reader to references [11, 14] for insight into how overhead scales with these parameters for various configurations of concatenated MSD subroutines.

While the overhead is sensitive to the precise configuration of MSD protocol used, it remains high (> 100) for practical output error probabilities. We remind the reader that the overhead reflects only a portion of the total cost of implementing a fault-tolerant T gate, as many Clifford gates are required at all stages of each recursion: to perform the encoding, syndrome measurement and decoding stages etc. In order to fairly compare the relative costs of the T gate and Clifford gates one must perform a full space-time cost analysis. However, this is a highly involved calculation and depends on architectural assumptions. As such, it falls beyond the scope of this thesis. But in practice, the overhead typically exceeds ≈ 1000 [12]. This disparity in cost between the fault-tolerant T gate and Clifford operations motivates the Clifford + T cost model for quantum circuits, in which it is assumed that Clifford gates are 'free' and T gates have unit cost. Following naturally from the definition of the Clifford + T model is the problem of T gate optimization, which is the primary concern of this thesis. In the following chapters we provide novel methods for solving this problem, including the TODD algorithm in chapter 4 and a similar algorithm for qudits in chapter 5.

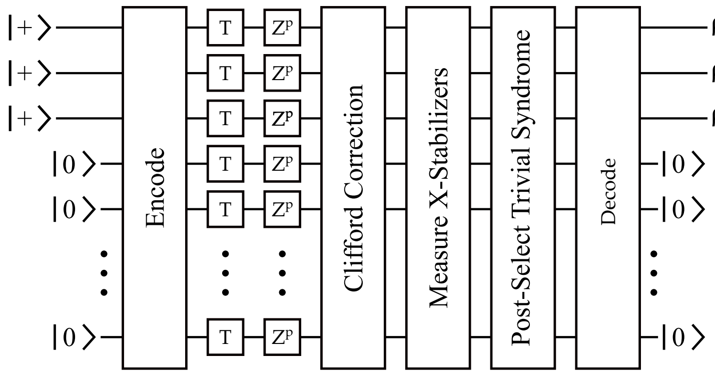


Figure 3.7: High-level overview of a circuit that implements a typical the magic state distillation protocol, such as the 15-to-1 code from reference [10] or the protocol from reference [11] based on triorthogonal codes.

Chapter 4

An Efficient Quantum Compiler that Reduces T Count

The following journal article is an original work by the present author and Dr. Earl T. Campbell that was published in Quantum Science and Technology on 12th September 2018 [30]. The full text is provided below in partial fulfilment of the requirements for the degree of Doctor of Philosophy. The source code for the quantum compiler "TOpt" described in this chapter is available here: [42].

Quantum Science and Technology

PAPER

An efficient quantum compiler that reduces *T* count

To cite this article: Luke E Heyfron and Earl T Campbell 2019 *Quantum Sci. Technol.* **4** 015004

View the article online for updates and enhancements.

Recent citations

- Methods for classically simulating noisy networked quantum architectures
  Iskren Vankov *et al*

IOP ebooks™

Bringing you innovative digital publishing with leading voices to create your essential collection of books in STEM research.

Start exploring the collection - download the first chapter of every title for free.

This content was downloaded from IP address 90.213.19.64 on 20/01/2020 at 16:23

Quantum Science and Technology

RECEIVED
15 December 2017

REVISED
4 July 2018

ACCEPTED FOR PUBLICATION
26 July 2018

PUBLISHED
12 September 2018

PAPER

An efficient quantum compiler that reduces T count

Luke E Heyfron[1] and Earl T Campbell

Department of Physics and Astronomy, University of Sheffield, Sheffield, United Kingdom
[1] Author to whom any correspondence should be addressed.

E-mail: leheyfron1@sheffield.ac.uk and earltcampbell@gmail.com

Keywords: quantum computation, gate synthesis, quantum compiler, quantum circuits, fault-tolerance, qubits, symmetric tensors

Abstract

Before executing a quantum algorithm, one must first decompose the algorithm into machine-level instructions compatible with the architecture of the quantum computer, a process known as quantum compiling. There are many different quantum circuit decompositions for the same algorithm but it is desirable to compile leaner circuits. A fundamentally important cost metric is the T count—the number of T gates in a circuit. For the single qubit case, optimal compiling is essentially a solved problem. However, multi-qubit compiling is a harder problem with optimal algorithms requiring classical runtime exponential in the number of qubits. Here, we present and compare several efficient quantum compilers for multi-qubit Clifford + T circuits. We implemented our compilers in C++ and benchmarked them on random circuits, from which we determine that our TODD compiler yields the lowest T counts on average. We also benchmarked TODD on a library of reversible logic circuits that appear in quantum algorithms and found that it reduced the T count for 97% of the circuits with an average T-count saving of 20% when compared against the best of all previous circuit decompositions.

Compiling is the conversion of an algorithm into a series of hardware level commands or elementary gates. Better compilers can implement the same algorithm using fewer hardware level instructions, reducing runtime and other resources. Quantum compiling or gate-synthesis is the analogous task for a quantum computer and is especially important given the current expense of quantum hardware. Early in the field, Solovay and Kitaev proposed a general purpose compiler for any universal set of elementary gates [1–3]. Newer compilers exploit the specific structure of the Clifford + T gate set and have reduced quantum circuit depths by several orders of magnitude [4–7], often improving the classical compile time. The Clifford + T gate set is natural since it is the fault-tolerant logical gate set in almost every computing architecture [8]. Moreover, fault-tolerance protocols have been proposed such as magic state distillation [9] that lead to a cost per T gate which is several hundred times larger than that of Clifford gates [10–12], which suggests T count as the key metric of compiler performance. Furthermore, the T count is an important metric beyond the standard compiling problem because it relates to the classical overhead of simulating quantum circuits [13–15] as well as the distillation cost of synthillation [16]. For these reasons, it is clear that developing methods for minimizing the T count is crucial for a variety of applications in quantum computation.

Significant progress has been made on synthesis of single-qubit unitaries from Clifford + T gates. For purely unitary synthesis, the problem is essentially solved since we have a compiler that is asymptotically optimal and efficient [4, 7]. Although further improvements are possible beyond unitary circuits, by making use of ancilla qubits and measurements [13, 17–19] or adding an element of randomness to compiling [20, 21]. On the other hand, the multi-qubit problem is much more challenging. An algorithm for multi-qubit unitary synthesis over the Clifford + T gate set is known that is provably optimal in terms of the T count but the compile runtime is exponential in the number of qubits [6, 22]. Compilers with efficient runtimes have been proposed but with no promise of T count optimality [23, 24]. We seek a compiler that runs efficiently and yields circuits with T counts that are as low as practically achievable.

A useful strategy is to take an initial Clifford + T circuit and split it into subcircuits containing Hadamards and subcircuits containing CNOT, S and T gates. One can then attempt to reduce the number of T gates within

© 2018 IOP Publishing Ltd

just the latter type of subcircuit. Amy and Mosca recently showed that this restricted problem is formally equivalent to error decoding on a class of Reed–Muller codes [25], which is in turn equivalent to finding the symmetric tensor rank of a 3-tensor [26]. Unfortunately, even this easier sub-problem is difficult to solve optimally. Nevertheless, it is more amenable to efficient solvers that offer reductions in T count. Amy and Mosca proved that an n-qubit subcircuit (containing CNOT, S and T gates) has an optimal decomposition into $n^2/2 + O(n)$ T gates. At the time, known efficient compilers could only promise an output circuit with no more than $O(n^3)$ T gates. Later, Campbell and Howard [16] sketched a compiler that is efficient and promises an output circuit with at most $n^2/2 + O(n)$ T gates. This shows efficient compilers can in this sense be 'near-optimal' with respect to worst case scaling. On the mathematical level, Campbell and Howard exploited a previously known efficient and optimal solver for a related 2-tensor problem [27] but suitably modified so that it nearly-optimally solves the required 3-tensor problem.

This paper develops several different compilers that have polynomial runtime in n and are near-optimal in the above sense when restricted to CNOT + T circuits. We modify the compiler to also accommodate Hadamard gates using a gadgetisation trick that requires additional resources (measurements, feed-forward and ancillas) and find that it performs well in practice. We provide the first implementations of such compilers (the source code is available here [28]) and compare performance against: a family of random circuits; and a library of benchmark circuits that implement actual quantum algorithms. For random circuits, we observe $O(n^2)$ scaling in T count for all variants of our compiling approach compared with $O(n^3)$ scaling for compilers based on earlier work. Quantum algorithms are highly structured and far from random, so the number of T gates can not be meaningfully compared with the worst case scaling. Instead, we benchmark against the best previously known results and found on average a 20% T count reduction. In one instance, our compiler gave a 51% T count reduction and it performed better than previous results for all but one of the benchmarked circuits. Of course, the T count is not the only metric relevant to gate synthesis. We discuss the limitations of the T count, as well as other metrics in section 4.3.

All of the near-optimal compilers described in this paper look for inspiration in algorithms for the related 2-tensor problem, which we call Lempel's algorithm. We give specific details for a compiler here called TOOL (target optimal by order lowering) that comes in two different flavours (with and without feedback). The TOOL compilers can be considered concrete versions of the approach outlined by Campbell and Howard [16]. Also described in this paper is the TODD (third order duplicate and destroy) compiler, which is again inspired by Lempel but in a more direct and elegant way than TOOL. In benchmarking, we find that TODD often achieved even lower T count than TOOL.

1. Preliminaries

The Pauli group on n qubits $\mathcal{P}^n$ is the set of all n-fold tensor products of the single qubit Pauli operators $\{X, Y, Z, \mathbb{I}\}$ with allowed coefficients $\in \{\pm 1, \pm i\}$. The kth level of the *Clifford hierarchy* $\mathcal{C}_k^n$ is defined as follows,

$$\mathcal{C}_k^n = \{U | U\mathcal{P}^n U^\dagger \subseteq \mathcal{C}_{k-1}^n\}, \tag{1}$$

with recursion terminated by $\mathcal{C}_1^n = \mathcal{P}^n$. The Clifford group on n qubits $\mathcal{C}^n$ is the normalizer of $\mathcal{P}^n$. We define $\mathcal{D}_k^n$ to be the diagonal elements of $\langle \mathrm{CNOT}, T\rangle$. We will omit the superscript n when the number of qubits is obvious from context. We define Clifford to be any generating set for the Clifford group on n qubits such as $\{\mathrm{CNOT}, H, S\}$. We define the CNOT + T gate set to be $\{\mathrm{CNOT}, S, T\}$, where we include the phase gate $S = T^2$ as a separate gate due to the magic states cost model for gate synthesis [9]. A quantum circuit decomposition for a unitary U is denoted $\mathcal{U}$; conversely we say that $\mathcal{U}$ implements U. Similarly, a circuit $\mathcal{E}$ implements non-unitary channel $\rho \to \varepsilon(\rho)$. We refer to a circuit $\mathcal{U}$ that implements a $U \in \mathcal{D}_3$ as a *diagonal* CNOT + T *circuit.*

2. Work-flow overview

In this section, we give a high level work-flow of our approach to compiling as sketched in figure 1. In stages 1–3, some simple circuit preprocessing is performed so that a Clifford + T circuit is converted into a form where the only non-Clifford part is a diagonal CNOT + T gate (an element of $\mathcal{D}_3$). Section 2.1 describes this preprocessing. In stages 4–6, the technically difficult aspect of compiling is addressed using a series of different algebraic representations of the circuit and these stages are described in section 2.2.

2.1. Circuit preprocessing

The input circuit $\mathcal{U}_{\mathrm{in}} \in \langle \mathrm{Clifford}, T\rangle$ implements some unitary U. It acts on a register we denote x, which is composed of n qubits and spans the Hilbert space $\mathcal{H}_{\mathrm{x}}$. The output of our compiler is a circuit $\mathcal{E}_{\mathrm{out}}$ composed of

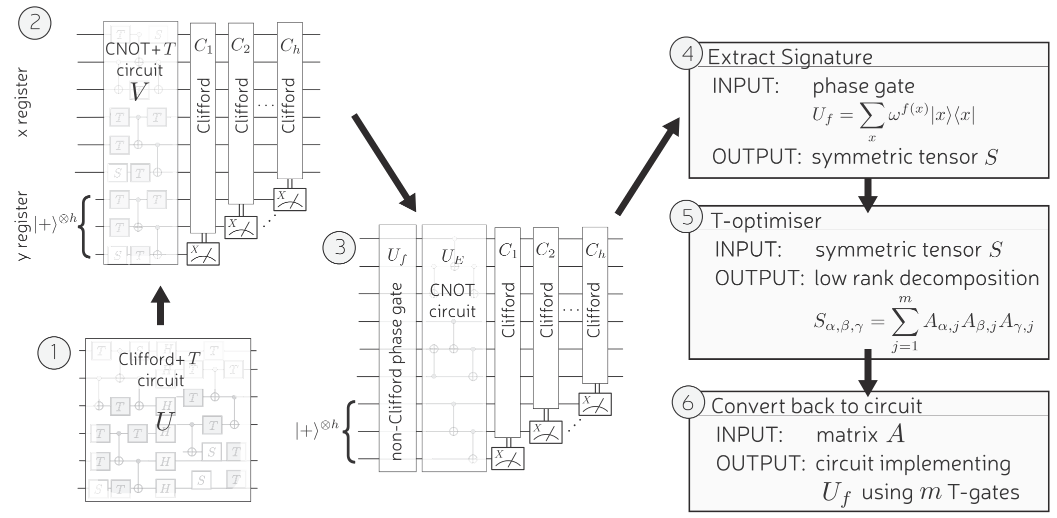

2
CNOT+T circuit V
C_1
C_2
C_h
Clifford
x register
y register
$|+\rangle^{\otimes h}$
1
Clifford+T circuit U
3
U_f
U_E
CNOT circuit
non-Clifford phase gate
4 Extract Signature
INPUT: phase gate $U_f = \sum_x \omega^{f(x)}|x\rangle\langle x|$
OUTPUT: symmetric tensor S
5 T-optimiser
INPUT: symmetric tensor S
OUTPUT: low rank decomposition $S_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} A_{\alpha,j}A_{\beta,j}A_{\gamma,j}$
6 Convert back to circuit
INPUT: matrix A
OUTPUT: circuit implementing U_f using m T-gates

Figure 1. The high level work-flow of the T gate optimization protocol is shown. A Clifford + T circuit is converted to the CNOT + T gate set by introducing ancillas and performing classically controlled Clifford gates. A non-Clifford phase gate is extracted, which maps to a signature tensor upon which the core optimization algorithm is performed. The optimized symmetric tensor decomposition is then converted back into a circuit of the form in panel (2) yielding an implementation of the original Clifford + T circuit with reduced T count.

Clifford and T gates but additionally allows: the preparation of $|+\rangle$ states; measurement in the Pauli-X basis, and classical feedforward. To account for ancilla $|+\rangle$ qubits, we include a register labelled y that is composed of h qubits and spans the Hilbert space $\mathcal{H}_y$. The circuit $\mathcal{E}_{\text{out}}$ will realize the input unitary after the y register is traced out

$$\text{Tr}_y[\varepsilon_{\text{out}}(\rho_x)] = \text{Tr}_y[\varepsilon_{\text{post}}(V(\rho_x \otimes |+\rangle\langle+|^{\otimes h}))V^\dagger)], \tag{2}$$

$$= U\rho_x U^\dagger, \tag{3}$$

where ρ_x is the density matrix for an arbitrary input pure state on $\mathcal{H}_x$. Furthermore, $V \in \mathcal{C}_3$ is the unitary portion of $\mathcal{E}_{\text{out}}$, and $\varepsilon_{\text{post}}$ is a quantum channel that is associated with the sequence of Pauli-X measurements and subsequent classically controlled Clifford gates, $C_1, C_2, \ldots, C_h$, seen in figure 1.

We emphasize that later stages of compiling will make use of a framework valid only for CNOT + T circuits, which makes Hadamard gates an obstacle. There are two commonly used methods for dealing with Hadamard gates: first, we can partition the quantum circuit into alternating $\langle$CNOT, $T\rangle$ and $\langle H\rangle$ subcircuits and optimize each CNOT + T subcircuit independently [23]. The second way is to replace each Hadamard gate with a gadget (see for example [29, 30]) that makes use of extra resources (ancillas, measurements and feedforward). The central portion of the gadget contains all of the non-Clifford behaviour and is in the CNOT + T gate set, so is directly compatible with our T-optimizers. The remainder of this section focusses on the second method (Hadamard gadgetization), but we discuss the Hadamard-bounded partitioning method in more detail in appendix A.

Each[2] of the h Hadamard gates is replaced by a *Hadamard-gadget* (as shown in panel 1) of figure 2. A Hadamard-gadget consists of a CNOT + T block followed by a Pauli-X gate conditioned on the outcome of measuring a *Hadamard-ancilla* (a qubit in the y register initialized in the $|+\rangle$ state) in the Pauli-X basis, so the size of the y register is h. After Hadamard-gadgetisation, we commute the classically controlled Pauli-X gates to the end of the circuit, starting with the right-most and iteratively working our way left (see panel 3 of figure 2). The end result is a circuit composed of a single CNOT + T block on $n + h$ qubits, followed by a sequence of classically controlled Clifford operators conditioned on Pauli-X measurements. The latter sequence of non-unitary gates constitutes the circuit $\mathcal{E}_{\text{post}}$. This method of circumventing Hadamards is preferred over forming Hadamard-bounded partitions as in previous works [23] because it allows us to convert most of the input circuit into the optimization-compatible gate set, which we find leads to better performance of the T-*Optimizer* subroutine (see appendix A for numerical evidence of this).

Once the internal Hadamards are removed, we are left with a CNOT + T circuit that implements unitary V, whose action on the computational basis is fully described [16, 23, 25, 31] by two mathematical objects: a *phase*

[2] To be precise, gadgets are only need for internal Hadamards. The external Hadamards that appear at the beginning and end of the circuit do not need to be replaced with Hadamard gadgets.

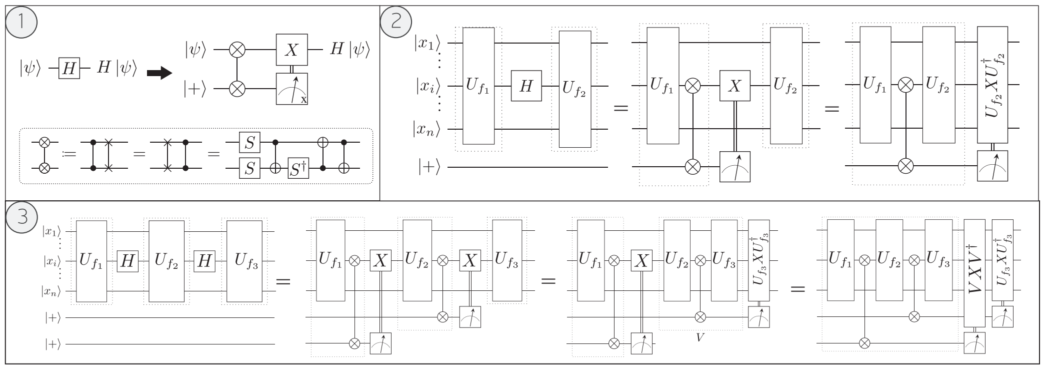


Figure 2. Hadamard gates are replaced by Hadamard-gadgets according to the rewrite in the upper part of panel (1). In the lower part, we define notation for the phase-swap gate and provide an example decomposition into the CNOT + T gate set. Panel (2) shows an example of a Hadamard gate swapped for a Hadamard-gadget where the classically controlled Pauli-X gate is commuted through U_{f_2} to the end. The CNOT + T—only region increases as shown by the dotted lines. As $U_{f_2} \in \mathcal{C}_3$, it follows that $U_{f_2} X U_{f_2}^{\dagger} \in \mathcal{C}_2$ as per equation (1), so has a T-count of 0. The example in panel (3) shows the same process as (2) but for 2 internal Hadamards. As $\mathcal{D}_3$ is a group, the operator $V \in \mathcal{D}_3$ and the second Pauli-X gate can also commute to the end to form a Clifford. This leads to a decomposition of the form in panel (2) of figure 1.

function, $f: \mathbb{Z}_2^n \mapsto \mathbb{Z}_8$, and an invertible matrix $E \in \mathbb{Z}_2^{(n,n)}$, such that

$$V|\mathbf{x}\rangle = \omega^{f(\mathbf{x})}|E\mathbf{x}\rangle, \tag{4}$$

where $\omega = e^{i\frac{\pi}{4}}$. It has been shown [16, 25] that $V = U_E U_f$ where $U_f \in \mathcal{D}_3$ can be implemented with a diagonal CNOT + T circuit and gives the phase

$$U_f|\mathbf{x}\rangle = \omega^{f(\mathbf{x})}|\mathbf{x}\rangle, \tag{5}$$

and U_E can be implemented with CNOTs.

2.2. Diagonal CNOT + T framework

In section 2.1, we isolated all the non-Clifford behaviour of a Clifford + T circuit within a diagonal CNOT + T circuit defined on a larger qubit register. This method allows us to map the T gate optimization problem for any Clifford + T circuit to the following.

Problem 2.1. (**T-OPT**) Given a unitary $U_f \in \mathcal{D}_3$, find a circuit decomposition $\mathcal{U}_f \in \langle CNOT, T, S\rangle$ that implements U_f with minimal uses of the T gate.

This section describes how we map the T-OPT problem from the quantum circuit picture to an algebraic problem following stages 4–6 of figure 1. Throughout this section we use the framework for diagonal CNOT + T circuits (also called *linear phase operators* [25]) introduced in [31] and built upon in [16, 23, 25]. We proceed by recalling from equation (5) that the action of any $U_f \in \mathcal{D}_3$ on the computational basis is given by $U_f|\mathbf{x}\rangle = \omega^{f(\mathbf{x})}|\mathbf{x}\rangle$ and that U_f is completely characterized by the phase function, f. A phase function can be decomposed into a sum of linear, quadratic and cubic monomials on the Boolean variables x_i. Each monomial of order r has a coefficient in $\mathbb{Z}_8$ and is weighted by a factor 2^{r-1}, as in the following:

$$f(\mathbf{x}) = \sum_{\alpha=1}^{n} l_\alpha x_\alpha + 2\sum_{\alpha<\beta}^{n} q_{\alpha,\beta} x_\alpha x_\beta + 4\sum_{\alpha<\beta<\gamma}^{n} c_{\alpha,\beta,\gamma} x_\alpha x_\beta x_\gamma \pmod 8, \tag{6}$$

where l_α, $q_{\alpha,\beta}$, $c_{\alpha,\beta,\gamma} \in \mathbb{Z}_8$. We refer to decompositions of f that take the form of equation (6) as *weighted polynomials* as in [16], in which it was shown that $U_{2f} = U_f^2 \in \mathcal{C}_2$ for any weighted polynomial, f. This implies that any two unitaries with weighted polynomials whose coefficients all have the same parity are Clifford equivalent. Note that the weighted polynomial can be lifted directly from the circuit definition of U_f if we work in the $\{T, CS, CCZ\}$ basis, as each kind of gate corresponds to the linear, quadratic and cubic terms, respectively.

In stage 4 of figure 1, we define the *signature tensor*, $S^{(U_f)} \in \mathbb{Z}_2^{(n,n,n)}$, to be a symmetric tensor of order 3 whose elements are equal to the parity of the weighted polynomial coefficients of U_f according to the following relations:

$$S_{\sigma(\alpha,\alpha,\alpha)} = S_{a,a,a} = l_\alpha \pmod 2, \tag{7a}$$

$$S_{\sigma(\alpha,\beta,\beta)} = S_{\sigma(\alpha,\alpha,\beta)} = q_{\alpha,\beta} \,(\mathrm{mod}\ 2), \tag{7b}$$

$$S_{\sigma(\alpha,\beta,\gamma)} = c_{\alpha,\beta,\gamma} \,(\mathrm{mod}\ 2) \tag{7c}$$

for all permutations of the indices, denoted σ. It follows that any two unitaries with the same signature tensor are Clifford equivalent.

We recall the definition of gate synthesis matrices from [16], where a matrix, A in $\mathbb{Z}_2^{(n,m)}$, is a gate synthesis matrix for a unitary U_f if it satisfies,

$$f(\mathbf{x}) = |A^T\mathbf{x}| (\mathrm{mod}\ 8) = \sum_j \left[\bigoplus_i A_{i,j} x_i\right] (\mathrm{mod}\ 8), \tag{8}$$

where $|.|$ is the Hamming weight of a binary vector. Notice that inside the square brackets is evaluated modulo 2 and outside is evaluated modulo 8.

Obtaining a gate synthesis matrix from a quantum circuit is best understood via the phase polynomial representation. A phase polynomial of a phase function, f, is a set, $P_f = \{\{\lambda_1, a_1\}, \{\lambda_2, a_2\}, \ldots, \{\lambda_p, a_p\}\}$, of linear boolean functions $\lambda_k(\mathbf{x})$, together with coefficients $a_k \in \mathbb{Z}_8$ such that

$$f(\mathbf{x}) = \sum_{k=1}^{P_f} a_k \lambda_k(\mathbf{x}) (\mathrm{mod}\ 8). \tag{9}$$

A phase polynomial can be extracted from a diagonal CNOT + T circuit by tracking the action of each gate on the computational basis states through the circuit [23, 31]. We then map P_f to an A matrix with a procedure such as the following. Start with an empty A matrix. Then for each $\{\lambda_k, a_k\} \in P_f$,

1. Define column vector, $\mathbf{v} \in \mathbb{Z}_2^n$, such that $\lambda_k(\mathbf{x}) = v_1 x_1 \oplus v_2 x_2 \oplus \ldots \oplus v_n x_n$.
2. Append a_k copies of $\mathbf{v}$ to the right-hand end of A.

We define a *proper* gate synthesis matrix to be an A matrix with no all-zero or repeated columns, and we define the function proper such that $A' = \mathrm{proper}(A)$ is the proper gate synthesis matrix formed by removing all all-zero columns and pairs of repeated columns from A. The purpose of this function is to strip away the Clifford behaviour from the gate synthesis matrix.

We will exploit the key property of A matrices described in the following lemma, which is a corollary of lemma 2 of [31].

Lemma 2.1. *Let $U_f \in \mathcal{D}_3$ be a unitary with phase function $f(\mathbf{x}) = |A^T\mathbf{x}|$ and $A' = \mathrm{proper}(A) \in \mathbb{Z}_2^{(n,m)}$. It follows that one can generate a circuit that implements U_f with $m = \mathrm{col}(A')$ uses of the T gate.*

Proof. First, we note from the definition of A in equation (8) that the jth column of A leads to a factor of $\omega^{\lambda_j(\mathbf{x})}$ appearing in the diagonal elements of U_f as written in equation (5), where λ_j is a reversible linear Boolean function given by,

$$\lambda_j(\mathbf{x}) = A_{1,j} x_1 \oplus A_{2,j} x_2 \ \oplus \ \ldots \oplus A_{n,j} x_n. \tag{10}$$

The action of a circuit generated by CNOT gates on computational basis state $|\mathbf{x}\rangle$ is to replace the value of each qubit with a reversible linear Boolean function on $x_1, x_2, \ldots, x_n$. Next, we show how to add the phase $\omega^{\lambda_j(\mathbf{x})}$. We define B_j to be a CNOT unitary such that after applying B_j the first qubit is mapped $|x_1\rangle \to |\lambda_j(\mathbf{x})\rangle$. A T gate subsequently applied to this qubit will now produce the desired phase. We then uncompute B_j by reversing the order of the CNOT gates. This procedure is repeated for every j until all columns of A have been implemented in this way. Only the columns of A that also appear in A' require the use of a T gate as all other columns have duplicates, where any pair of duplicates can be implemented by replacing the T gate with an S gate in the above procedure. Therefore the T count is equal to $m = \mathrm{col}(A')$. □

The signature tensor of U_f can be determined from an A matrix of U_f using the following relation,

$$S_{\alpha,\beta,\gamma}^{(A)} = \sum_{j=1}^{m} A_{\alpha,j} A_{\beta,j} A_{\gamma,j} (\mathrm{mod}\ 2). \tag{11}$$

Therefore, the gate synthesis problem T-OPT reduces to the following tensor rank problem.

Problem 2.2. (**3-STR**) Given a symmetric tensor of order 3, $S \in \mathbb{Z}_2^{(n,n,n)}$, find a matrix $A \in \mathbb{Z}_2^{(n,m)}$ that satisfies equation (11) with minimal m.

Any algorithm attempting to solve 3-STR can be used in stage 5 of figure 1. The observation that T-OPT reduces to 3-STR is not new as it follows directly from earlier work. Amy and Mosca [25] proved that T-OPT is equivalent to minimum distance decoding of the punctured Reed–Muller code of order $n - 4$ and length n (often written as RM*$(n - 4, n)$). Furthermore, in 1980 Seroussi and Lempel [26] recognized that this Reed–Muller decoding problem is equivalent to 3-STR and conjectured that this is a hard computational task. A non-symmetric generalization of 3-STR has been proved to be NP-complete [32], giving further weight to the conjecture. This imposes a practical upper bound on the number of qubits, n_{RM}, over which circuits can be optimally synthesized.

The problem 3-STR is closely related to

Problem 2.3. (**2-STR**) Given a symmetric tensor of order 2, $S \in \mathbb{Z}_2^{(n,n)}$, find a matrix $A \in \mathbb{Z}_2^{(n,m)}$ that satisfies

$$S_{\alpha,\beta}^{(A)} = \sum_{j=1}^{m} A_{\alpha,j}A_{\beta,j} (\mathrm{mod}\ \ 2). \tag{12}$$

with minimal m.

This could also be stated as a matrix factorization $S = AA^T$ problem. As such, we say any A satisfying $S = AA^T$ is a factor of S and a minimal factor is one with the minimum possible number of columns. As is often the case in complexity theory, the matrix variant of the problem is considerably simpler than the higher order tensor variant. Lempel gave an algorithm that finds an optimal solution to 2-STR in polynomial time [27]. We call this Lempel's factoring algorithm and for completeness describe it in appendix B. Our main strategy to T count optimization is to take insights from Lempel's algorithm for 2-STR and apply them to 3-STR. In doing so, our compilers will be efficient but lose the promise of optimality, instead providing approximate solutions to 3-STR and T-OPT.

In the final stage (see 6 of figure 1), we map the output matrix of stage 5 back to a diagonal CNOT + T circuit, $\mathcal{U}_{f'}$, that comprises m instances of the T gate using lemma 2.1. The circuit $\mathcal{U}_{f'}$ implements a unitary $U_{f'} = U_f\, U_{\mathrm{Clifford}}$, where U_{Clifford} is a diagonal Clifford factor. The input weighted polynomial stored since step 4 contains sufficient information to generate a circuit for $U_{\mathrm{Clifford}}^{\dagger}$ (see appendix D), hence we recover the original unitary, $U_f = U_{f'}\, U_{\mathrm{Clifford}}^{\dagger}$. The final part of step 6 constitutes replacing $\mathcal{U}_f$ with $(\mathcal{U}_{\mathrm{Clifford}}^{\dagger} \circ \mathcal{U}_{f'})$. At this stage, the protocol terminates returning the final output, $\mathcal{E}_{\mathrm{out}} = (\mathcal{U}_{\mathrm{Clifford}}^{\dagger} \circ \mathcal{U}_{f'} \circ \mathcal{U}_E \circ \mathcal{E}_{\mathrm{post}})$.

3. *T-optimizer*

Until now the *T-optimizer* subroutine of our protocol has been treated as a black box whose input is a signature tensor S and the output is a gate synthesis matrix A with few columns. In this section, we describe the inner workings of the various T-optimizers we have implemented in this work.

3.1. Reed–Muller decoder (RM)

Although Reed–Muller decoding is believed to be hard, a brute force solver can be implemented for a small number of qubits. We implement such a brute force decoder and found its limit to be $n_{\mathrm{RM}} = 6$. To gain some intuition for the complexity of the problem, consider the following. The number of codespace generators for RM*$(n - 4, n)$ is equal to $N_G = \sum_{r=1}^{n-4} \binom{n}{r}$. Therefore, the size of the search space is $N_{\mathrm{search}} = 2^{N_G}$. On a processor with a clock speed of 3.20 GHz, generously assuming we can check one codeword per clock cycle, it would take over 91 years to exhaustively search this space for $n = 7$. Performing the same back-of-the-envelope calculation for $n = 6$, it would take $\approx 7 \times 10^{-4}$ seconds. In practice, we find the brute force decoder executes in around 10 minutes for $n = 6$, so the time for $n = 7$ would be significantly worse. Clearly, we need to develop heuristics for this problem.

3.2. Recursive expansion (RE)

The simplest means of efficiently obtaining an A matrix for a given signature tensor S is to make use of the modulo identity $2ab = a + b - a \oplus b$. More concretely, for each non-zero coefficient in the weighted polynomial l_α, $q_{\alpha,\beta}$, $c_{\alpha,\beta,\gamma}$, make the following substitutions to the corresponding monomials:

$$x_\alpha \rightarrow x_\alpha, \tag{13}$$

$$2x_\alpha x_\beta \rightarrow x_\alpha + x_\beta - (x_\alpha \oplus x_\beta), \tag{14}$$

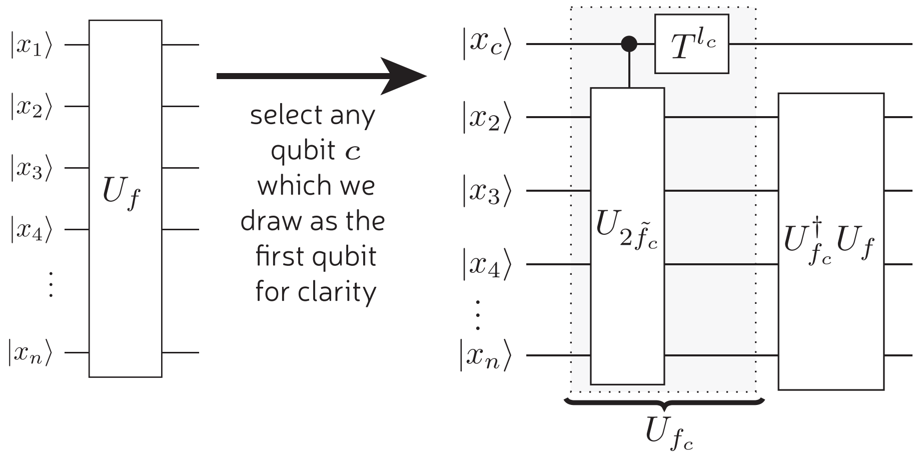


Figure 3. A sketch of one round of TOOL (without feedback). We identify a sub-circuit U_{f_c} with a single control qubit and then use that such a subcircuit can be efficiently and optimally compiled using Lempel's algorithm. The remaining circuit $U_{f_c}^\dagger U_f$ contains one fewer qubit and so the process can be iterated until the circuit is down to 6 qubits when it can be optimally compiled by brute force.

$$4x_\alpha x_\beta x_\gamma \to x_\alpha + x_\beta + x_\gamma - (x_\alpha \oplus x_\beta) - (x_\alpha \oplus x_\gamma) - (x_\beta \oplus x_\gamma) + (x_\alpha \oplus x_\beta \oplus x_\gamma), \tag{15}$$

from which the corresponding A matrix can be easily extracted. We call this the RE algorithm, which has been shown to yield worst-case T counts of $O(n^3)$. It is straightforward to understand this cubic scaling because any proper gate synthesis matrix resulting from the RE algorithm may include any column of Hamming weight 3 or less. There are $\sum_{k=1}^{3}\binom{n}{k} = O(n^3)$ such columns so from lemma 2.1 there can be at most $O(n^3)$ T gates in the corresponding circuit decomposition.

3.3. Target optimal by order lowering (TOOL)

Campbell and Howard [16] proposed an efficient heuristic for T-OPT that requires at most $O(n^2)$ T gates compared to $O(n^3)$ of the best previous (RE) optimizer. In the quantum circuit picture, the algorithm involves decomposing the input CNOT + T circuit into a cascade of control-$U_{2\tilde{f}}$ operators where $\tilde{f}$ is quadratic rather than cubic. Lowering the order in this way means that each control-$U_{2\tilde{f}}$ can be synthesized both efficiently and optimally using Lempel's factoring algorithm. For this reason we call it the TOOL algorithm. Figure 3 shows a single step of how TOOL pulls out a single control-$U_{2\tilde{f}}$ operator, reducing the number of qubits non-trivially affected by the remaining unitary. The process is repeated until the circuit is small enough to be solved using the RM algorithm. The core of the algorithm was already outlined in previous work [16] but for completeness appendix C describes both plain TOOL and a variant called TOOL (with feedback). This paper presents the first numerical results obtained from an implementation of TOOL.

3.4. Third order duplicate and destroy (TODD)

In this section, we present an algorithm based on Lempel's factoring algorithm [27] that is extended to work for order 3 tensors. Since this algorithm does not appear in any previous work, we will provide an extended explanation here. This algorithm requires some initial A matrix to be generated by another algorithm such as RE or TOOL, then it reduces the number of columns of the initial gate synthesis matrix iteratively until exit. In section 4, we present numerical evidence that it is the best efficient solver of the T-OPT problem developed so far. We call this the TODD algorithm because, much like the villainous Victorian barber, it shaves away at the columns of the input A matrix iteratively until the algorithm finishes execution. Pseudo-code is provided in appendix E.

We begin by introducing the key mechanism through which TODD reduces the T count of quantum circuits: by *destroying* pairs of duplicate columns of a gate synthesis matrix, a process through which the signature tensor is unchanged, as shown in the following lemma.

Lemma 3.1. *Let $A \in \mathbb{Z}^{(n,m)}$ be a gate synthesis matrix whose* a*th and* b*th columns are duplicates. Let $A_{\text{des}} \in \mathbb{Z}^{(n,m-2)}$ be a gate synthesis matrix formed by removing the* a*th and* b*th columns of A. It follows that $S^{(A)} = S^{(A_{\text{des}})}$ for any such A and A_{des}.*

Proof. We start by writing the signature tensor in terms of the elements of A according to equation (11),

$$S^{(A)}_{\alpha,\beta,\gamma} = \sum_{k=1}^{m} A_{\alpha,k}A_{\beta,k}A_{\gamma,k} (\mathrm{mod}\ 2), \tag{16}$$

and separating the terms associated with $a,\ b$ from the rest of the summation,

$$S^{(A)}_{\alpha,\beta,\gamma} = \left(\sum_{j\in\mathcal{J}} A_{\alpha,j}A_{\beta,j}A_{\gamma,j}\right) + A_{\alpha,a}A_{\beta,a}A_{\gamma,a} + A_{\alpha,b}A_{\beta,b}A_{\gamma,b} (\mathrm{mod}\ 2), \tag{17}$$

where $\mathcal{J} = [1,\ m]\backslash\{a,\ b\}$, so that

$$S^{(A)}_{\alpha,\beta,\gamma} = S^{(A_{\mathrm{des}})}_{\alpha,\beta,\gamma} + A_{\alpha,a}A_{\beta,a}A_{\gamma,a} + A_{\alpha,b}A_{\beta,b}A_{\gamma,b} (\mathrm{mod}\ 2), \tag{18}$$

As stated in the lemma, the ath and bth columns of A are duplicates and so

$$A_{i,a} = A_{i,b} \ \ \forall\, i \in [1,\ n]. \tag{19}$$

Now substitute equation (19) into (18),

$$S^{(A)}_{\alpha,\beta,\gamma} = S^{(A_{\mathrm{des}})}_{\alpha,\beta,\gamma} + 2A_{\alpha,a}A_{\beta,a}A_{\gamma,a} (\mathrm{mod}\ 2) \tag{20}$$

$$= S^{(A_{\mathrm{des}})}_{\alpha,\beta,\gamma} (\mathrm{mod}\ 2), \tag{21}$$

where the last step follows from modulo 2 addition. □

Lemma 3.1 gives us a simple means to remove columns from a gate synthesis matrix by destroying pairs of duplicates columns and thereby reducing the T count of a CNOT + T circuit by 2. However, it is often the case that the A matrix does not already contain any duplicate columns. Therefore, we wish to perform some transformation: $A \to A'$ such that

(a) A' has duplicate columns;

(b) the transformation preserves the signature tensor of A.

In the following lemma we introduce a class of transformations that *duplicate* a particular column of an A matrix such that property (a) is met. We then use lemma 3.3 to establish what conditions must be satisfied for the duplication transformation to have property (b).

Lemma 3.2. *Let $A \in \mathbb{Z}_2^{(n,m)}$ be a proper gate synthesis matrix. For some choice of a and b, let $\mathbf{c}_a(A)$ and $\mathbf{c}_b(A)$ denote the ath and* b*th columns of A and define $\mathbf{z} = \mathbf{c}_a(A) \oplus \mathbf{c}_b(A)$. Let $\mathbf{y} \in \mathbb{Z}_2^m$ be any vector such that $y_a \oplus y_b = 1$. We consider duplication transformations of the form $A \to A' = A \oplus \mathbf{z}\mathbf{y}^T$. It follows that the* a*th and* b*th columns of A' are duplicates and so property (a) holds.*

Proof. We begin by finding expressions for the matrix elments of A' in terms of A, $\mathbf{z}$ and $\mathbf{y}$,

$$A'_{i,j} = A_{i,j} \oplus z_i y_j, \tag{22}$$

and substitute the definition of $\mathbf{z}$,

$$A'_{i,j} = A_{i,j} \oplus (A_{i,a} \oplus A_{i,b})y_j. \tag{23}$$

Now we can find the elements of the columns a and b of A',

$$A'_{i,a} = A_{i,a} \oplus (A_{i,a} \oplus A_{i,b})y_a, \tag{24}$$

$$A'_{i,b} = A_{i,b} \oplus (A_{i,a} \oplus A_{i,b})y_b. \tag{25}$$

We substitute in the condition $y_b = y_a \oplus 1$ into equation (25),

$$\begin{aligned} A'_{i,b} &= A_{i,b} \oplus (A_{i,a} \oplus A_{i,b})(y_a \oplus 1) \\ &= A_{i,b} \oplus (A_{i,a} \oplus A_{i,b})y_a \oplus A_{i,a} \oplus A_{i,b} \\ &= A_{i,a} \oplus (A_{i,a} \oplus A_{i,b})y_a \\ &= A'_{i,a}, \end{aligned} \tag{26}$$

where the two $A_{i,b}$ terms cancel in the second step of equation (26). □

Lemma 3.3. *Consider a duplication transformation of the form $A \rightarrow A' = A \oplus \mathbf{z}\mathbf{y}^T$ where $\mathbf{z}$, $\mathbf{y}$ are vectors of appropriate length. It follows that $S^{(A)} = S^{(A')}$ (satisfying property (b)) if the following conditions hold true:*

C1: $|\mathbf{y}| = 0 \pmod 2$

C2: $A\mathbf{y} = \mathbf{0}$

C3: $\chi(A, \mathbf{z})\mathbf{y} = \mathbf{0}$,

where we define $\chi(A, \mathbf{z})$ as follows. Given some gate synthesis matrix, A, and a column vector $\mathbf{z} \in \mathbb{Z}_2^n$ let χ be a matrix with rows labelled by (α, β, γ) and of the form

$$\mathbf{R}_{\alpha,\beta,\gamma} = (z_\alpha \mathbf{r}_\beta \wedge \mathbf{r}_\gamma) \oplus (z_\beta \mathbf{r}_\gamma \wedge \mathbf{r}_\alpha) \oplus (z_\gamma \mathbf{r}_\alpha \wedge \mathbf{r}_\beta), \tag{27}$$

where $\mathbf{r}_\alpha$ is the αth row of A, and $\mathbf{x} \wedge \mathbf{y}$ is the element-wise product of vectors $\mathbf{x}$ and $\mathbf{y}$. The order of the rows in χ is unimportant, but must include every choice of $\alpha, \beta, \gamma \in \mathbb{Z}_n$ with no pair of indices being equal.

Proof. We begin by finding an expression for $S(A')$ using equation (11),

$$S^{(A')}_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} (A_{\alpha,j} \oplus z_\alpha y_j)(A_{\beta,j} \oplus z_\beta y_j)(A_{\gamma,j} \oplus z_\gamma y_j) \pmod 2, \tag{28}$$

and expanding the brackets,

$$\begin{aligned} S^{(A')}_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} (&A_{\alpha,j}A_{\beta,j}A_{\gamma,j} \oplus z_\alpha z_\beta z_\gamma y_j \\ &\oplus z_\alpha z_\beta A_{\gamma,j} y_j \oplus z_\beta z_\gamma A_{\alpha,j} y_j \oplus z_\gamma z_\alpha A_{\beta,j} y_j \\ &\oplus z_\alpha A_{\beta,j} A_{\gamma,j} y_j \oplus z_\beta A_{\gamma,j} A_{\alpha,j} y_j \oplus z_\gamma A_{\alpha,j} A_{\beta,j} y_j) \pmod 2. \end{aligned} \tag{29}$$

We can see that the first term of equation (29) summed over all j is equal to $S^{(A)}$, by definition. The task is to show that the remaining terms sum to zero under the specified conditions. Next, we sum over all j and substitute in the definitions of $|\mathbf{y}|$, $A\mathbf{y}$ and $\chi(A, \mathbf{z})\,\mathbf{y}$,

$$S^{(A')}_{\alpha,\beta,\gamma} = S^{(A)}_{\alpha,\beta,\gamma} \oplus z_\alpha z_\beta z_\gamma |\mathbf{y}| \oplus z_\alpha z_\beta [A\mathbf{y}]_\gamma \oplus z_\beta z_\gamma [A\mathbf{y}]_\alpha \oplus z_\gamma z_\alpha [A\mathbf{y}]_\beta \oplus (\mathbf{R}_{\alpha,\beta,\gamma} \cdot \mathbf{y}). \tag{30}$$

By applying condition *C1*, the second term is eliminated; by applying condition C2, the next three terms are eliminated, and by applying condition C3, the final term is eliminated. □

Having shown how to duplicate and destroy columns of a gate synthesis matrix, we are ready to describe the TODD algorithm, presented as pseudo-code in algorithm 1. Given an input gate synthesis matrix A with signature tensor S, we begin by iterating through all column pairs of A given by indices a, b. We construct the vector $\mathbf{z} = \mathbf{c}_a \oplus \mathbf{c}_b$ where $\mathbf{c}_j$ is the jth column of A, as in lemma 3.2. We check to see if the conditions in lemma 3.3 are satisfied for $\mathbf{z}$ by forming the matrix,

$$\tilde{A} = \begin{pmatrix} A \\ \chi(A, \mathbf{z}) \end{pmatrix}. \tag{31}$$

Any vector, $\mathbf{y}$, in the null space of $\tilde{A}$ simultaneously satisfies C2 and C3 of lemma 3.3. We scan through the null space basis until we find a $\mathbf{y}$ such that $y_a \oplus y_b = 1$. At this stage we know that we can remove at least one column from A, depending on the following cases

i:If $|\mathbf{y}| = 0 \pmod 2$ then condition *C1* is satisfied and we can perform the duplication transformation from lemma 3.3;

ii:If $|\mathbf{y}| = 1 \pmod 2$ then we force *C1* to be satisfied by appending a 1 to $\mathbf{y}$ and an all-zero column to A before applying the duplication transformation.

Finally, we use the function proper as in appendix E to destroy all duplicate pairs to maximize efficiency. In case i, at least two columns have been removed and in case ii at least one column has been removed[3]. This reduces the number of columns of A and therefore the T count of U_f. We now start again from the beginning, iterating over columns of the new A matrix. The algorithm terminates if every column pair has been exhausted without success.

[3] Other column pairs may be destroyed after the duplication transformation in addition to the ath and bth columns but only for the latter pair is destruction guaranteed.

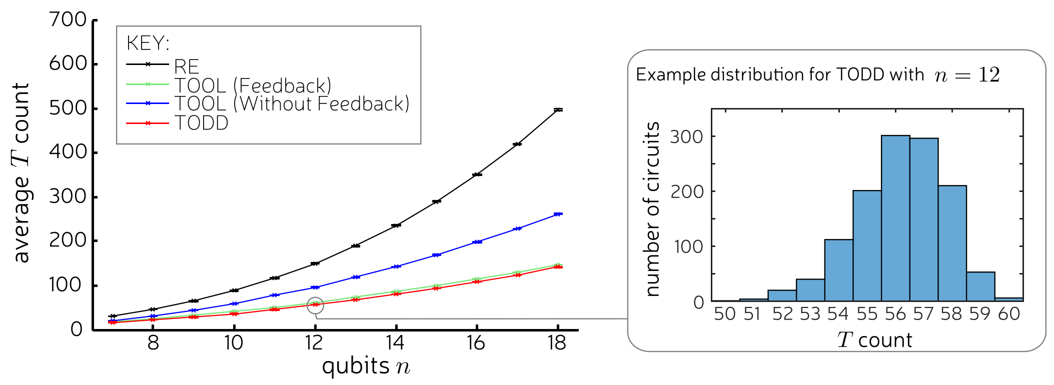


Figure 4. Circuits generated by the CNOT and T gate were randomly generated for varying number of qubits n then optimized by our implementations of RE, TOOL and TODD. The average T-count for each n over many random circuits are shown on the vertical axis. TODD produces circuit decompositions with the smallest T-counts on average but scales the same as the next best algorithm, TOOL (Feedback). Both of these algorithms are better than RE by a factor n. The difference between the T-counts for TODD and TOOL (Feedback) converge to a constant 5.5 ± 0.7 for large n.

4. Results and discussion

We implemented our compiler, which we call *TOpt*, in C++ including each variant of *T-Optimizer* described in section 3, and tested it on two types of benchmark. First, we performed a random benchmark, in which we randomly sampled signature tensors from a uniform probability distribution for a range of n and used them as input for the four versions of *T-optimizer*: RE, TOOL (feedback), TOOL (without feedback) and TODD. The results for the random benchmark are shown in figure 4. Second, we tested the compiler on a library of benchmark circuits taken from Dmitri Maslov's Reversible Logic Synthesis Benchmarks Page [33], Matthew Amy's GitHub repository for T-par [34] and Nam *et al*'s GitHub repository [35] for [24]. These circuits implement useful quantum algorithms including Galois Field multipliers, integer addition, nth prime, Hamming coding functions and the hidden weighted bit functions. The results for the quantum algorithm benchmark are listed in table 1. For all benchmarks, the results were obtained on the University of Sheffield's Iceberg HPC cluster[36].

4.1. Random circuit benchmark

We performed the random benchmark in order to determine the average case scaling of the T-count with respect to n for each computationally efficient version of *T-optimizer* with results shown in figure 4. For both versions of TOOL, we find that the numerical results for the T count follow the expected analytical scaling of $O(n^2)$ and correspondingly the results for RE scales as $O(n^3)$. We see that TODD slightly outperforms the next best algorithm, TOOL (without feedback) and is therefore the preferred algorithm in settings where classical runtime is not an issue. Furthermore, for all compilers the distribution of T-counts (for fixed n) concentrates around the mean value. Figure 4 includes error bars showing the distribution but they are too small to be clearly visible, so for one data point we highlight this with an inset histogram. Therefore, TODD performs better, not just on average, but on the vast majority of random circuits so far tested. While both have a polynomial runtime, we found TOOL runs faster than TODD. Therefore, TOOL may have some advantage for larger circuits that are impractically large for TODD. However, TODD can always partition a very large circuit into several smaller circuits at the cost of being slightly less effective at reducing T count. Consequently, for very large circuits, it is unclear which compiler will work best and running both is recommended.

The random benchmark effectively uses diagonal CNOT + T circuits. This gate set is not universal and therefore is computationally limited. However, these circuits are generated by $\{T, CS, CCZ\}$, which all commute. This means such circuits lie in the computational complexity class IQP (which stands for *instantaneous quantum polynomial-time*) that feature in proposals for quantum supremacy experiments [29, 37, 38]. Low cost designs of IQP circuits provided by our compiler would therefore be an asset for achieving quantum supremacy.

4.2. Quantum algorithms benchmark

The results in table 1 show that the TODD algorithm reduced or preserved the T count for every input quantum circuit upon which it was tested, as expected. Additionally, TODD yields a positive saving over the best previous

Table 1. T-counts of Clifford + T benchmark circuits for the TODD, TOOL(F) (with feedback) and TOOL(NF) (without feedback) variants of the TOpt compiler are shown. Results for other variants can be seen in table A1 of appendix A. Columns n and n_h show the number of qubits for the input circuit and the number of Hadamard ancillas, respectively. The T-count for the circuit is given: before optimization (Pre-Opt.); after optimization using the best previous algorithm (Best prev.); and post-optimization using our implementation of TODD, TOOL(F) and TOOL(NF). The best previous algorithm is given in the Alg. column where: T-par is from [23]; $\mathrm{RM_m}$ and $\mathrm{RM_r}$ are the majority and recursive Reed–Muller optimizers, respectively, both from [25]; and Auto_H is the heavy version of the algorithm from [24]. We show the T-count saving for each TOpt variant over the best previous algorithm in the s columns and the execution time as run on the Iceberg HPC cluster in the t columns. Results where the execution time is marked with† were obtained using an alternative implementation of TODD that is faster but less stable. The row *Positive saving* shows the proportion of the benchmark circuits, as a percentage, for which the corresponding compiler yields a positive saving over the best previous result.

	Pre-Opt.		Best prev.		TOpt	TODD			TOOL(F)			TOOL(NF)		
Circuit	n	T	T	Alg.	n_h	T	t (s)	s(%)	T	t (s)	s (%)	T	t (s)	s (%)
Mod 5_4 [34]	5	28	16	T-par	6	16	0.04	0	19	0.38	−18.75	19	0.37	−18.75
8-bit adder [34]	24	399	213	$\mathrm{RM_m}$	71	129	40914.1	39.44	279	71886.1	−30.99	284	55574.6	−33.33
CSLA-MUX_3 [35]	16	70	58	$\mathrm{RM_r}$	17	52	30.41	10.34	84	122.95	−44.83	73	84.54	−25.86
CSUM-MUX_9 [35]	30	196	76	$\mathrm{RM_r}$	12	72	587.21	5.26	83	2081.13	−9.21	104	340.19	−36.84
GF(2^4)-mult [34]	12	112	68	T-par	7	54	8.88	20.59	75	5.96	−10.29	75	2.38	−10.29
GF(2^5)-mult [34]	15	175	101	$\mathrm{RM_r}$	9	87	66.83	13.86	109	17.6	−7.92	107	28.27	−5.94
GF(2^6)-mult [34]	18	252	144	$\mathrm{RM_r}$	11	126	521.86	12.50	165	82.52	−14.58	157	60.16	−9.03
GF(2^7)-mult [34]	21	343	208	$\mathrm{RM_r}$	13	189	2541.4	9.13	277	226.4	−33.17	209	122.17	−0.48
GF(2^8)-mult [34]	24	448	237	$\mathrm{RM_r}$	15	230	36335.7	2.95	370	379.97	−56.12	281	322.83	−18.57
GF(2^9)-mult [34]	27	567	301	$\mathrm{RM_r}$	17	295	50671.1	1.99	454	1463.02	−50.83	351	816.04	−16.61
GF(2^{10})-mult [34]	30	700	410	T-par	19	350	15860.3†	14.63	550	7074.29	−34.15	434	988.04	−5.85
GF(2^{16})-mult [34]	48	1792	1040	T-par	31		—		1723	75204.8	−65.67	1089	30061.1	−4.71
Grover_5 [34]	9	52	52	T-par	23	44	17.07	15.38	106	110.29	−103.85	83	117.39	−59.62
$\mathrm{Hamming}_{15}$ (low) [34]	17	161	97	T-par	34	75	902.69	22.68	161	2787	−65.98	132	1041.22	−36.08
$\mathrm{Hamming}_{15}$ (med) [34]	17	574	230	T-par	85	162	12410.8†	29.57	727	176275	−216.09	277	59112.2	−20.43
HWB_6 [33]	7	105	71	T-par	24	51	55.66	28.17	189	140.79	−166.20	149	59.24	−109.86
Mod-Mult_{55} [34]	9	49	35	$\mathrm{RM_{m\&r}}$	10	17	0.26	51.43	35	5.45	0	19	0.92	45.71
Mod-Red_{21} [34]	11	119	73	T-par	17	55	25.78	24.66	68	40.82	6.85	71	19.76	2.74
*n*th-prime_6 [33]	9	567	400	$\mathrm{RM_{m\&r}}$	97	208	37348†	48	830	205869	−107.50	344	135165	14
QCLA-Adder_{10} [34]	36	238	162	T-par	28	116	5496.66	28.40	167	7544.58	−3.09	180	4560.78	−11.11
QCLA-Com_7 [34]	24	203	94	$\mathrm{RM_m}$	19	59	198.55	37.23	79	420.95	15.96	125	465.41	−32.98
QCLA-Mod_7 [34]	26	413	235	$\mathrm{Auto_H}$	58	165	46574.3	29.79	295	35249.2	−25.53	310	22355.4	−31.91
QFT_4 [34]	5	69	67	T-par	39	55	93.65	17.91	67	1602.91	0	59	2756.34	11.94
RC-Adder_6 [34]	14	77	47	$\mathrm{RM_{m\&r}}$	21	37	18.72	21.28	48	1238.12	−2.13	44	81.77	6.38
NC Toff_3 [34]	5	21	15	T-par	2	13	$<10^{-2}$	13.33	14	0.02	6.67	14	0.01	6.67
NC Toff_4 [34]	7	35	23	T-par	4	19	0.06	17.39	22	0.24	4.35	22	0.12	4.35
NC Toff_5 [34]	9	49	31	T-par	6	25	0.4	19.35	31	1146.04	0	29	0.67	6.45
NC Toff_{10} [34]	19	119	71	T-par	16	55	44.78	22.54	65	1357.98	8.45	67	110.44	5.63
Barenco Toff_3 [34]	5	28	16	T-par	3	14	$<10^{-2}$	12.50	16	0.02	0	16	0.03	0
Barenco Toff_4 [34]	7	56	28	T-par	7	24	0.45	14.29	26	0.88	7.14	27	0.56	3.57

Table 1. (Continued.)

	Pre-Opt.		Best prev.		TOpt	TODD			TOOL(F)			TOOL(NF)		
Circuit	n	T	T	Alg.	n_h	T	t(s)	s(%)	T	t(s)	s(%)	T	t(s)	s(%)
Barenco Toff$_5$ [34]	9	84	40	T-par	11	34	1.94	15	42	12.6	−5	42	2.59	−5
Barenco Toff$_{10}$ [34]	19	224	100	T-par	31	84	460.33	16	120	1938.01	−20	122	1269.03	−22
VBE-Adder$_3$ [34]	10	70	24	T-par	4	20	0.15	16.67	24	1639.76	0	38	1.93	−58.33
Mean								**19.76**			−31.59			−14.13
Standard error								2.12			8.87			4.69
Min								0			−216.09			−109.86
Max								**51.43**			15.96			45.71
Positive saving (%)							**96.88**			**18.18**			**30.30**	

algorithm for all benchmarks except Mod 5_4 with an average and maximum saving of 20% and 51%, respectively. This is immediately useful due to the lower cost associated with solving these problems.

Crucially, the output circuits of our protocol often require a considerable number of ancilla qubits due to our use of Hadamard gadgets. This space-time trade-off is justifiable when the cost of introducing an additional qubit is small in comparison to that of performing an additional T gate [39]. Furthermore, our compilers can be executed with a cap, h_{cap}, on the size of the ancilla register by dividing the circuit into subcircuits containing no more than h_{cap} Hadamard gates. A larger number of Hadamard gates generally leads to an increased classical compilation time for TODD as well as an increased T count for TODD-part (see appendix A), which naturally motivates future investigation into Hadamard gate optimization as a pre-processing step of TOpt-like compilers. Finally, further reductions in the space (and other) resource requirements may be possible by back-substituting the Hadamard gadget identity from figure 2 post-optimization.

The TOOL algorithms (with and without feedback) reduced T counts below those of the best previous result for 18% and 30% of the benchmark circuits, respectively. But for the majority, we find that TOOL actually results in negative savings. This seems to contradict the result for the random benchmark (see figure 4) in which TOOL (feedback) nearly performs as well as TODD. We offer the following explanation for this apparent contradiction. The circuits generated as input for the random benchmark typically have optimal T counts close to the worst-case bound of $O(n^2)$. TODD yields T counts very close to optimal because it only terminates when nearly all avenues for T count reduction have been exhausted. The TOOL algorithm outputs T counts below $O(n^2)$, so closely competes with TODD for random circuits. However, for the Clifford + T benchmark, the optimal T count is typically much less than the worst-case $O(n^2)$ bound. It is important to recall at this stage that TOOL is optimal for the special case where the circuit implements a control-Clifford. But even for this special case, TOOL needs to know which qubit is the control qubit in order to take advantage of this special case behaviour. Consequently, a general-purpose automated compiler without prior knowledge about the input quantum circuit must have access to an additional subroutine which determines the control qubit. For general quantum circuits, the task is especially challenging because the circuit must also be optimally partitioned into a sequence of control-Cliffords. As such, we have left this task as an avenue of future work. Our implementation of TOOL uses a naive random control-qubit selection subroutine, so regardless of the low optimal T count, TOOL will often output T counts that remain close to the worst-case of $O(n^2)$. We suggest that this is the principle cause for the relatively poor performance seen in table 1, which has lead to negative savings not only over the best previous result and TODD, but sometimes also over the input circuit, and conclude that a better control-qubit selector would unlock more of TOOL's T-optimizing potential.

4.3. The *T* count and other metrics

We acknowledge that the T count does not account for the full space-time cost of quantum computation. Recall that we justified neglecting the cost of Clifford gates due to the high ratio between the cost of the T gate and that of Clifford gates. The full space-time cost is highly sensitive to the architecture of the quantum computer, but for the surface code, this ratio is estimated to be between 50 and 1000 [12, 40–42], depending on architectural assumptions.

Note that while our protocol leads to circuits with low T count, the final output often has an *increased* CNOT count. This is largely due to step 6 of our protocol where we map the phase polynomial back to a quantum circuit using a naive approach. Although T gates cost significantly more than CNOTs individually, the lower bound on number of CNOT gates required to implement high complexity reversible functions exceeds the upper bound on the number of T gates required by an amount that grows exponentially in n [39]. So for large n, our focus should turn instead to CNOT optimization. In this paper, we focus exclusively on T count optimization, which is relevant not just to circuit optimization but also to classical simulation runtime [13–15] and distillation of magic states [16]. For this reason, we omit the CNOT count from our benchmark tables and leave the problem of optimizing CNOT count as an avenue for future work.

5. Conclusions

In this work, we have developed a framework for compiling and optimizing Clifford + T quantum circuits that reduces the T count. This scheme maps the quantum circuit problem to an algebraic problem involving order 3 symmetric tensors, for which we have presented an efficient near-optimal solver, and we have reviewed previous methods. We implemented our protocol in C++ and used it to obtain T count data for quantum circuit benchmarks. Each variant of the compiler has managed to produce quantum circuits for quantum algorithms with lower T-counts than any previous attempts known to us. However, we find that the TODD compiler with Hadamard gadgets performs the best in practice. This lowers the cost of quantum computation and takes us closer to achieving practical universal fault-tolerant quantum computation.

Acknowledgments

We acknowledge support by the Engineering and Physical Sciences Research Council (EPSRC) through grant EP/M024261/1. We thank Mark Howard and Matthew Amy for valuable discussions, and Dmitri Maslov for comments on the manuscript. We thank Quanlong Wang for spotting an error in an earlier draft of the manuscript.

Appendix A. Clifford + T benchmarks for TODD-part and TODD-h_{cap}

In order to investigate the relative effectiveness of the Hadamard gadget and Hadamard-bounded partition methods for dealing with Hadamard gates, we repeated the benchmarks from table 1 but for the latter method. The results are shown in the TODD-part column group of table A1. For the Hadamard partition method, we found that the compiler runtime is significantly decreased, making the optimization of larger quantum circuits feasible. However, the performance is worse in terms of raw T count reductions, often leading to higher T counts than the best previous result. It is important to note that for a given input circuit, the T count is highly sensitive on the choice of Hadamard partitioning, of which, in general, there are many. Our implementation does not optimize over Hadamard partitioning choices, so there is potential for developing a more powerful version of TODD-part that makes use of an advanced Hadamard partitioning algorithm, which may lead to greater T count reductions.

The TODD compiler completely gadgetizes each Hadamard gate, whereas the TODD-part compiler completely partitions the circuit into Hadamard-bounded partitions. It is possible to interpolate between these two approaches using a parameter h_{cap} that enforces a cap on the number of available Hadamard ancillas. Upon reaching this cap, the compiler synthesizes the circuit encountered so far, freeing up the Hadamard ancillas for the subsequent Hadamard partition. We have implemented this feature, and in order to quantify the overhead required to see a T count reduction, we ran each benchmark repeatedly, incrementing the value of h_{cap} until we saw a reduction over the best previous result. The results for this experiment are presented in table A1. We found that the relationship between h_{cap} and T count savings is favourable: relatively few Hadamard gadgets are required to see a reduction over the best previous result. Over all the benchmark circuits, where the number of qubits and the T count ranges up to $n = 36$ and $T = 6671$, respectively, we found that on average 9 Hadamard ancillas are required to see positive saving and at most 23 ancillas are needed for all but one exceptional result (Cycle 17_3), which requires 43. This suggests that, while TODD combined with full Hadamard gadgetization is clearly the forerunner amongst our compilers for reducing the T count, a modest improvement in the Hadamard partitioning scheme, or adding a pre-processing step that looks for Hadamard gate reductions may lead to a better version of TODD that requires no non-unitary gadgets, has feasible compiler runtimes for large circuits, and yields positive T count savings.

Appendix B. Lempel's factoring algorithm

We describe Lempel's factoring algorithm (originally from [27]) using conventions consistent with our description of the TODD algorithm to more easily see how TODD generalizes Lempel's algorithm for order 3 tensors. Lempel's factoring algorithm takes as input a symmetric tensor of order 2 (a matrix), which we denote $S \in \mathbb{Z}_2^{(n,n)}$ and outputs a matrix $A \in \mathbb{Z}_2^{(n,m)}$ where the elements of A and S are related as follows:

$$S_{\alpha,\beta} = \sum_{k=1}^{m} A_{\alpha,k} A_{\beta,k} (\mathrm{mod}\ 2). \quad (32)$$

Lempel proved that the minimal value of m is equal to

$$\mu(S) = \rho(S) + \delta(S), \quad (33)$$

where $\rho(S)$ is the rank of matrix S and

$$\delta(S) = \begin{cases} 1 & \text{if } S_{\alpha,\alpha} = 0\ \forall\ \alpha \in [1, n] \\ 0 & \text{otherwise.} \end{cases} \quad (34)$$

Lempel's algorithm solves the problem of finding an A matrix that obeys equation (32) for a given S matrix such that $m = \mu(S)$. Such an A matrix is referred to as a minimal factor of S.

In the following, we denote the number of columns of A as $c(A)$ and the jth column of A as $\mathbf{c}_j(A)$. Lempel's algorithm is the following:

1. Generate an initial (necessarily suboptimal) A matrix for S.

Table A1. T-counts of Clifford + T benchmark circuits for the TODD-part and TODD-h_{cap} variants of TOpt are shown. TODD-part uses Hadamard-bounded partitions rather than Hadamard gadgets and ancillas and TODD-h_{cap} sets a fixed cap, h_{cap}, on the number of Hadamard ancillas available to the compiler.Starting at $h_{\text{cap}} = 1$, we iteratively incremented the value of h_{cap} by 1 until obtaining the first result with a positive T-count saving over the best previous algorithm.The value of h_{cap} for which this occured is reported in the h_{cap} column, and the number of partitions, T-count, execution time and percentage saving for this result are detailed by column group TODD-h_{cap}. TODD-h_{cap} results that yield a positive saving for $h_{\text{cap}} = 0$ correspond to results for TODD-part and results that require $h_{\text{cap}} = n_h$ Hadamard ancillas correspond to results for TODD. As we are strictly interested in intermediate values of h_{cap}, we omit these data and refer the reader to the appropriate result.The number of Hadamard partitions is given by the N_p columns. As in table 1, n is the number of qubits for the input circuit; T are T-counts: for the circuit before optimization (Pre-Opt.); due to the best previous algorithm (Best prev.); and post-optimization using variants of our compiler.The best previous algorithm is given in the Alg. column where:T-par is from [23]; RM$_{\text{m}}$ and RM$_{\text{r}}$ are the majority and recursive Reed–Muller optimizers, respectively, both from [25]; and Auto$_{\text{H}}$ is the heavy version of the algorithm from [24].We show the T-count saving for each TOpt variant over the best previous algorithm in the s columns and the execution time as run on the Iceberg HPC cluster in the t columns.Results where the execution time is marked with† were obtained using an alternative implementation of TODD that is faster but less stable. *Positive saving* shows the proportion of the benchmark circuits, as a percentage, for which the corresponding compiler yields a positive saving over the best previous result.

	Pre-Opt.		Best prev.		TODD-part				TODD-h_{cap}				
Circuit	n	T	T	Alg.	N_p	T	t(s)	s(%)	h_{cap}	N_p	T	t(s)	s(%)
Mod 5$_4$ [34]	5	28	16	T-par	7	18	$<10^{-2}$	−12.50	1	4	16	$<10^{-2}$	0
8-bit adder [34]	24	399	213	RM$_{\text{m}}$	20	283	12.63	−32.86	13	5	212	227.81	0.47
CSLA-MUX$_3$ [35]	16	70	58	RM$_{\text{r}}$	7	62	0.38	−6.90	5	3	54	3.73	6.90
CSUM-MUX$_9$ [35]	30	196	76	RM$_{\text{r}}$	3	76	20.31	0	4	2	74	36.57	2.63
Cycle 17$_3$ [34]	35	4739	1944	RM$_{\text{m}}$	573	2625	1001.11	−35.03	43	15	1939	25507.5†	0.26
GF(2^4)-mult [34]	12	112	68	T-par	3	56	0.55	17.65			0 *See result for TODD-part*		
GF(2^5)-mult [34]	15	175	101	RM$_{\text{r}}$	3	90	6.96	10.89			0 *See result for TODD-part*		
GF(2^6)-mult [34]	18	252	144	RM$_{\text{r}}$	3	132	121.16	8.33			0 *See result for TODD-part*		
GF(2^7)-mult [34]	21	343	208	RM$_{\text{r}}$	3	185	153.75	11.06			0 *See result for TODD-part*		
GF(2^8)-mult [34]	24	448	237	RM$_{\text{r}}$	3	216	517.63	8.86			0 *See result for TODD-part*		
GF(2^9)-mult [34]	27	567	301	RM$_{\text{r}}$	3	301	2840.56	0	8	2	295	3212.53	1.99
GF(2^{10})-mult [34]	30	700	410	T-par	3	351	23969.1	14.39			0 *See result for TODD-part*		
GF(2^{16})-mult [34]	48	1792	1040	T-par	3	922	76312.5†	11.35			—		
Grover$_5$ [34]	9	52	52	T-par	18	52	0.02	0	5	4	50	0.3	3.85
Hamming$_{15}$ (low) [34]	17	161	97	T-par	22	113	0.53	−16.49	5	6	93	2.93	4.12
Hamming$_{15}$ (med) [34]	17	574	230	T-par	59	322	1.57	−40	11	7	226	58.08	1.74
Hamming$_{15}$ (high) [34]	20	2457	1019	T-par	256	1505	16.84	−47.69	13	24	1010	595.8	0.88
HWB$_6$ [33]	7	105	71	T-par	15	82	0.01	−15.49	3	6	68	0.13	4.23
HWB$_8$ [33]	12	5887	3531	RM$_{\text{m\&r}}$	709	4187	6.53	−18.58	9	110	3517	259.14	0.40
Mod-Adder$_{1024}$ [34]	28	1995	1011	T-par	234	1165	98.8	−15.23	10	27	978	665.5	3.26
Mod-Adder$_{1048576}$ [34]	0	0	7298	T-par	2030	9480	89486.5†	−29.90			—		
Mod-Mult$_{55}$ [34]	9	49	35	RM$_{\text{m\&r}}$	6	28	0.02	20			0 *See result for TODD-part*		
Mod-Red$_{21}$ [34]	11	119	73	T-par	15	85	0.06	−16.44	4	5	69	0.59	5.48
nth-prime$_6$ [33]	6	567	400	RM$_{\text{m\&r}}$	63	402	0.17	−0.50	2	29	384	0.98	4
nth-prime$_8$ [33]	12	6671	4045	RM$_{\text{m\&r}}$	774	5034	8.4	−24.45	12	105	4043	898.98	0.05
QCLA-Adder$_{10}$ [34]	36	238	162	T-par	6	184	223.25	−13.58	5	3	157	366.1	3.09
QCLA-Com$_7$ [34]	24	203	94	RM$_{\text{m}}$	7	135	11.62	−43.62	16	2	81	170.77	13.83

Table A1. (Continued.)

	Pre-Opt.		Best prev.		TODD-part				TODD-h_{cap}				
Circuit	n	T	T	Alg.	N_p	T	t (s)	s (%)	h_{cap}	N_p	T	t (s)	s (%)
QCLA-Mod$_7$ [34]	26	413	235	Auto$_H$	15	305	34.76	-29.79	23	3	221	289.77†	5.96
QFT$_4$ [34]	5	69	67	T-par	38	67	$<10^{-2}$	0	2	13	63	0.02	5.97
RC-Adder$_6$ [34]	14	77	47	RM$_{\text{m\&r}}$	13	59	0.11	-25.53	6	3	45	0.97	4.26
NC Toff$_3$ [34]	5	21	15	T-par	3	15	$<10^{-2}$	0			$2 = n_h$ *See result for TODD*		
NC Toff$_4$ [34]	7	35	23	T-par	5	23	$<10^{-2}$	0			$4 = n_h$ *See result for TODD*		
NC Toff$_5$ [34]	9	49	31	T-par	7	31	0.01	0	5	2	29	0.2	6.45
NC Toff$_{10}$ [34]	19	119	71	T-par	17	71	0.74	0	10	3	69	12.48	2.82
Barenco Toff$_3$ [34]	5	28	16	T-par	4	22	$<10^{-2}$	-37.50	2	2	14	$<10^{-2}$	12.50
Barenco Toff$_4$ [34]	7	56	28	T-par	8	38	0.01	-35.71	4	2	26	0.06	7.14
Barenco Toff$_5$ [34]	9	84	40	T-par	12	54	0.03	-35	6	2	38	0.35	5
Barenco Toff$_{10}$ [34]	19	224	100	T-par	32	134	2.27	-34	16	2	98	54.75	2
VBE- Adder$_3$ [34]	10	70	24	T-par	5	36	0.04	-50			$4 = n_h$ *See result for TODD*		
Mean								-13.19	**9**				4.05
Standard error								3.15	1.65				0.64
Min								-50	1				0
Max								20	43				13.83
Positive saving (%)							**20.51**				**96.30**		

2. Check if $c(A) = \mu(S)$. If true, exit and output A. Otherwise, perform steps 3 to 7.
3. Find a $\mathbf{y} \in \mathbb{Z}_2^m$ such that $A\mathbf{y} = \mathbf{0}$ and $0 < |y| < c(A)$.
4. If $|y| = 1 \pmod 2$ then update $\mathbf{y} \rightarrow (\mathbf{y}^T, 1)^T$ and $A = (A \quad \mathbf{0})$.
5. Find a pair of indices $a, b \in [1, m]$, $a \neq b$ such that $y_a \oplus y_b = 1$.
6. Apply transformation $A \rightarrow A \oplus \mathbf{z}\mathbf{y}^T$, where $\mathbf{z} = \mathbf{c}_a(A) \oplus \mathbf{c}_b(A)$.
7. Remove the ath and bth columns from A, then go to step 2.

Note that the key difference between the Lempel and TODD algorithm is that TODD additionally requires condition C3 from lemma 3.3 to be satisfied.

Appendix C. TOOL algorithm

Here we give a detailed description of TOOL, with the main idea illustrated by figure 3. TOOL is best explained in terms of weighted polynomials (recall equation (6)). The algorithm is iterative, where each round consists of the five steps detailed below. Before the first round, we initialize an 'empty' output gate synthesis matrix, $A_{\text{out}} \in \mathbb{Z}_2^{(n,0)}$.

1. Choose an integer $c \in [1, n]$ such that there is at least one term in f with x_c as a factor. If no such c exists, the algorithm terminates and outputs A_{out}.
2. Find $\tilde{f}_c$, the *target polynomial* of f with respect to x_c (see equation (35) below).
3. Determine the order 2 signature tensor, $\tilde{S}$, of $\tilde{f}_c$.
4. Find $\tilde{A}$, a minimal factor of $\tilde{S}$, using Lempel's factoring algorithm.
5. Recover an order 3 gate synthesis matrix, A, for $\tilde{A}$, and append it to A_{out}. Replace f with $f - |A^T\mathbf{x}|$.

Each round of TOOL gives a new f that depends on fewer x variables. When f depends on only n_{RM} or fewer variables, we switch to the optimal brute force optimizer, RM.

We will now explain each step of the above description in detail, unpacking the contained definitions. In step 1, we select an index c, which corresponds to the control qubit of the control-$U_{2\tilde{f}_c}$ operator shown in figure 3. The order that we choose c for each round can affect the output and therefore is a parameter of TOOL. For all results, we randomly selected c with uniform probability from the set of all indices $\{c\}$ for which x_c is a factor of at least one term in f.

Next, we observe that any f can be decomposed into $f = f_c + f_c'$, where we define f_c as a weighted polynomial containing all terms of f with x_c as a factor. The former part, f_c, can be further decomposed as follows,

$$f_c = 2x_c\tilde{f}_c + l_c x_c, \tag{35}$$

where $\tilde{f}_c$ is quadratic and so can be optimally synthesized efficiently. In step 2, we extract $\tilde{f}_c$, which is implicitly fixed by the above equations. We refer to $\tilde{f}_c$ as a *target polynomial* because it corresponds to the target of a control-U_{2f} operator, where $f = \tilde{f}_c$ and $|x_c\rangle$ is the control qubit.

As an aside, we remark that the target polynomial is related to Shannon cofactors that appear in Boole's expansion theorem. Specifically, we have

$$\tilde{f}_c = \frac{f_c^+ - f_c^- - l_c}{2}, \tag{36}$$

where f_c^+ and f_c^- are the positive and negative Shannon cofactors, respectively, of f with respect to x_c, and l_c is the linear coefficient of f associated with x_c.

In step 3, we map $\tilde{f}_c$ to a signature tensor of order 2 (a matrix) for use with Lempel's factoring algorithm. Let $\tilde{l}_\alpha$, $\tilde{q}_{\alpha,\beta}$ be the linear and quadratic coefficients of $\tilde{f}_c$, respectively. For each $\alpha, \beta \neq c$, the elements of $\tilde{S}$ are obtained as follows.

$$\tilde{S}_{\alpha,\beta} = \begin{cases} \tilde{l}_\alpha \pmod 2 & \text{if } \alpha = \beta \\ \tilde{q}_{\alpha,\beta} \pmod 2 & \text{if } \alpha \neq \beta. \end{cases} \tag{37}$$

Finding a minimal factor of $\tilde{S}_{\alpha,\beta}$ is the problem 2-STR. Therefore, we can use Lempel's algorithm (see appendix B) to find a matrix $\tilde{A} \in \mathbb{Z}_2^{(n,\tilde{m})}$, which is a minimal factor of $\tilde{S}$ such that

$$\tilde{f}_c = |\tilde{A}^T \mathbf{x}| = \sum_{j=1}^{\tilde{m}} \left[\bigoplus_{i=1}^{n} \tilde{A}_{i,j} x_i \right] (\text{mod } 8). \tag{38}$$

By substituting equation (38) into (35) we obtain

$$f_c = 2x_c |\tilde{A}^T \mathbf{x}| + l_c x_c, \tag{39}$$

$$= \sum_{j=1}^{\tilde{m}} 2x_c \left[\bigoplus_{i=1}^{n} \tilde{A}_{i,j} x_i \right] + l_c x_c (\text{mod } 8), \tag{40}$$

where we have taken the factor $2x_c$ within the Hamming weight summation. Next, we use the modular identity $2ab = a + b - a \oplus b$ with $a = x_c$ and b as the contents of the square brackets. This gives

$$f_c = \sum_{j=1}^{\tilde{m}} \left(x_c + \left[\bigoplus_{i=1}^{n} \tilde{A}_{i,j} x_i \right] - x_c \oplus \left[\bigoplus_{i=1}^{n} \tilde{A}_{i,j} x_i \right] \right) + l_c x_c (\text{mod } 8), \tag{41}$$

$$= x_c(\tilde{m} + l_c) + \sum_{j=1}^{\tilde{m}} \left[\bigoplus_{i=1}^{n} \tilde{A}_{i,j} x_i \right] - \sum_{j=1}^{\tilde{m}} x_c \oplus \left[\bigoplus_{i=1}^{n} \tilde{A}_{i,j} x_i \right] (\text{mod } 8), \tag{42}$$

$$= x_c(\tilde{m} + l_c) + |\tilde{A}^T \mathbf{x}| - |(\tilde{A} \oplus B_c)^T \mathbf{x}| (\text{mod } 8), \tag{43}$$

where $B_c \in \mathbb{Z}_2^{(n,m)}$ is a matrix with elements

$$[B_c]_{i,j} = \begin{cases} 1 & \text{if } i = c \\ 0 & \text{otherwise.} \end{cases} \tag{44}$$

This is now in the form of a phase polynomial (e.g. see equation (8)) with no more than $1 + 2\tilde{m}$ terms, where $\tilde{m}$ was the optimal size of the factorization found using Lempel's algorithm.

There are two versions of TOOL: with and without feedback. The difference between these versions determines whether all of equation (43) is put into A_{out} or whether parts are 'fed back' into f for subsequent rounds. This leads to two distinct definitions of the A matrix referred to in step 5 of TOOL:

$$|A^T \mathbf{x}| = \begin{cases} (\tilde{m} + l_c)x_c - |(\tilde{A} \oplus B_c)^T \mathbf{x}| & \text{feedback} \\ (\tilde{m} + l_c)x_c - |(\tilde{A} \oplus B_c)^T \mathbf{x}| + |\tilde{A}^T \mathbf{x}| & \text{without feedback.} \end{cases} \tag{45}$$

Notice that both $(\tilde{m} + l_c)x_c$ and $|(\tilde{A} \oplus B_c)^T \mathbf{x}|$ depend on x_c, so must be sent to output. furthermore, they comprise *all* the terms that depend on x_c, which is why sending $|\tilde{A}^T \mathbf{x}|$ to output is optional, and why the number of dependent variables is reduced by at least 1 each round. For the *feedback* version, $|\tilde{A}^T \mathbf{x}|$ is kept within f during step 5, whereas it is sent to output A_{out} in the *without feedback* version.

Appendix D. Calculating Clifford correction

We will now describe how to determine the Clifford correction required to restore the output of *T-Optimizer* to the input unitary. Let the input of *T-Optimizer* be a weighted polynomial f that implements unitary $U_f \in \mathcal{D}_3$, and let the output be a weighted polynomial g. Any f can be split into the sum

$$f = f_1 + f_2, \tag{46}$$

where the coefficients of f_1 are in $\mathbb{Z}_2$ and those of f_2 are even. From the definition of *T-Optimiser*, we know the coefficients of f and g have the same parity i.e.

$$g = g_1 + g_2 = f_1 + g_2, \tag{47}$$

where g_1, g_2 are similarly defined for g. Using equations (46) and (47) we find,

$$g = f + (g_2 - f_2). \tag{48}$$

equation (48) implies that $U_{\text{Clifford}} = U_{(g_2 - f_2)} \in \mathcal{D}_2$. Therefore, the Clifford correction is $U_{\text{Clifford}}^{\dagger} = U_{(g_2 - f_2)}^{\dagger} = U_{(f_2 - g_2)}$. We can map $(f_2 - g_2)$ to a phase polynomial and subsequently to a quantum circuit, $\mathcal{U}_{\text{Clifford}}^{\dagger}$.

Appendix E. TODD pseudocode

Algorithm 1. Third order duplicate-then-destroy (TODD) algorithm

```
Input: Gate synthesis matrix $A \in \mathbb{Z}_2^{(n,m)}$.
Output: Gate synthesis matrix $A' \in \mathbb{Z}_2^{(n,m')}$ such that $m' \leqslant m$ and $S^{(A')} = S^{(A)}$.
    • Let $\mathrm{col}_j(A)$ be a function that returns the $j$th column of $A$.
    • Let $\mathrm{cols}(A)$ be a function that returns the number of columns of $A$.
    • Let $\mathrm{nullspace}(A)$ be a function that returns a matrix whose columns generate the right null space of A.
    • Let proper(A) be a function that returns matrix $A$ with every pair of identical columns and every all-zero column removed.
  procedure TODD
    Initialize $A' \leftarrow A$
  start:
    for all $1 \leqslant a < b \leqslant \mathrm{cols}(A')$ do
      $\mathbf{z} \leftarrow \mathrm{col}_a(A') + \mathrm{col}_b(A')$
      $\tilde{A} \leftarrow \begin{pmatrix} A' \\ \chi(A', \mathbf{z}) \end{pmatrix}$
      $N \leftarrow \mathrm{nullspace}(\tilde{A})$
      for all $1 \leqslant k \leqslant \mathrm{cols}(N)$ do
        $\mathbf{y} \leftarrow \mathrm{col}_k(N)$
        if $y_a \oplus y_b = 1$ then
          if $|\mathbf{y}| = 1 (\mathrm{mod}\ 2)$ then
            $A' \leftarrow (A' \ \ \mathbf{0})$
            $\mathbf{y} \leftarrow \begin{pmatrix} \mathbf{y} \\ 1 \end{pmatrix}$
          $A' \leftarrow A' + \mathbf{z}\mathbf{y}^T$
          $A' \leftarrow \mathrm{proper}(A')$
          goto start
```

Appendix F. Computational efficiency of TODD

In this appendix, we calculate an upper-bound on the worst-case computational efficiency of the TODD algorithm as described in appendix E, in terms of the number of arithmetic operations on $GF(2)$ required.

Let A be a gate synthesis matrix with n rows and m columns that is used as input for the TODD algorithm. The loop, L_1, over each column pair (a, b) requires at most $\binom{m}{2} = O(m^2)$ iterations to complete. Inside L_1, there are four lines of pseudocode: a column addition, requiring no more than n operations; a matrix concatenation and calculation of $\chi(A, \mathbf{z})$, requiring E_1 operations; a nullspace calculation, requiring $O(n^3) + O(m^2 n)$ operations using Gaussian elimination; and finally a nested loop L_2, requiring E_2 operations.

From equation (27), we see that each row of $\chi(A, \mathbf{z})$ can be calculated with $O(m)$ operations. There are a maximum of $\binom{n}{3}$ rows in $\chi(A, \mathbf{z})$ so the total number of operations required to calculate χ is $O(n^3 m)$. Combining this with the matrix concatenation, we find that $E_1 = O(n^3 m) + nm = O(n^3 m)$.

The loop L_2 executes in at most

$$\mathrm{cols}(\mathrm{nullspace}(\tilde{A})) := \mathrm{colrank}(\mathrm{nullspace}(\tilde{A})) = m - \mathrm{rank}(\tilde{A}) \leqslant m - \mathrm{rank}(A) \leqslant m - n \tag{49}$$

iterations. The identity between the column rank and the number of columns follows from the assertion that the nullspace function outputs a matrix whose columns are a linearly independent basis for the nullspace of A.

The loop L_2 is composed of a conditional that requires 1 addition (by merging the first line of L_2 and the conditional). The content of the conditional is only evaluated once, so can be considered as part of L_1 for this calculation. Therefore, the number of operations performed in L_2 is $E_2 = m - n$.

The nested conditional requires at most $m + n + 1$ operations, where the terms are due to the Hamming weight of $|\mathbf{y}|$, concatenating an all-zero column to A' and concatenating a one to $\mathbf{y}$, respectively. The line $A' \leftarrow A' + \mathbf{z}\mathbf{y}^T$ requires at most $n(m + 1)$ operations and the proper function can be computed using at most m operations by keeping track of all-zero columns with a Boolean array, for a small physical overhead of m.

The outermost loop (between *start* and **goto** *start*) by definition executes in no more than $m - m'$ iterations where m' is the number of columns of the output. In this worst-case calculation, we assume $m' = 0$.

So the TODD algorithm can be executed using

$$O(m[n + O(n^3 m) + O(n^3) + O(m^2 n) + (m - n) + (m + n + 1) + n(m + 1) + m]) \tag{50}$$

$$= O(m[O(n^3 m) + O(n^3) + O(m^2 n)]) \tag{51}$$

$$= O(n^3 m^2) + O(n m^3) \tag{52}$$

operations.

Therefore, given a family of Clifford + T circuits with n qubits, h Hadamard gates and t T gates, we would expect our compiler to execute in time asymptotically upper-bounded by a function of the following form

$$O((n+h)^3t^2) + O((n+h)t^3) \tag{53}$$

$$= O(n^3t^2) + O(h^3t^2) + O(nt^3) + O(ht^3), \tag{54}$$

where we have made the reasonable assumption that the computational bottleneck is due to the TODD algorithm, rather than the circuit preprocessing stages or mapping between different circuit representations, for instance.

In practice, the actual runtimes for the benchmark quantum circuits seen in table 1 are much lower than this worst-case upper-bound. Furthermore, the compiler runtime is dependent on the structure of the input quantum circuit, rather than simply the number of qubits and gates from which it is composed. Consequently, we do not see a simple relation between circuit parameters n, t, h and the runtime for the benchmarks in table 1.

Note that in our calculation of the complexity, we assumed that we must calculate *every* row of $\chi(A, \mathbf{z})$. In practice, we find that many of the rows are identical. An algorithm that calculates only the unique rows may lead to improved computational efficiency.

ORCID iDs

Luke E Heyfron https://orcid.org/0000-0003-1379-923X

References

[1] Kitaev A Y, Shen A and Vyalyi M N 2002 *Classical and Quantum Computation* vol 47 (Providence, RI: American Mathematical Society)
[2] Dawson C M and Nielsen M A 2006 *Quantum Info. Comput.* **6** 81–95
[3] Fowler A G 2011 *Quantum Inf. Comput.* **11** 867–73
[4] Kliuchnikov V, Maslov D and Mosca M 2013 *Phys. Rev. Lett.* **110** 190502
[5] Selinger P 2013 *Phys. Rev.* A **87** 042302
[6] Gosset D, Kliuchnikov V, Mosca M and Russo V 2014 *Quantum Inf. Comput.* **14** 1261
[7] Ross N J and Selinger P 2016 *Quantum Inf. Comput.* **16** 901
[8] Campbell E T, Terhal B M and Vuillot C 2016 *Nature* **549** 172
[9] Bravyi S and Kitaev A 2005 *Phys. Rev.* A **71** 022316
[10] Raussendorf R, Harrington J and Goyal K 2007 *New J. Phys.* **9** 199
[11] Fowler A G, Mariantoni M, Martinis J M and Cleland A N 2012 *Phys. Rev.* A **86** 032324
[12] O'Gorman J and Campbell E T 2017 *Phys. Rev.* A **95** 032338
[13] Howard M and Campbell E 2017 *Phys. Rev. Lett.* **118** 090501
[14] Bravyi S, Smith G and Smolin J A 2016 *Phys. Rev.* X **6** 021043
[15] Bravyi S and Gosset D 2016 *Phys. Rev. Lett.* **116** 250501
[16] Campbell E T and Howard M 2017 *Phys. Rev.* A **95**
[17] Paetznick A and Svore K M 2014 *Quantum Inf. Comput.* **14** 1277
[18] Bocharov A, Roetteler M and Svore K M 2015 *Phys. Rev.* A **91** 052317
[19] Bocharov A, Roetteler M and Svore K M 2015 *Phys. Rev. Lett.* **114** 080502
[20] Campbell E 2017 *Phys. Rev.* A **95** 042306
[21] Hastings M B 2017 *Quantum Info. Comput.* **17** 5–6
[22] Amy M, Maslov D, Mosca M and Roetteler M 2013 *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **32** 818
[23] Amy M, Maslov D and Mosca M 2014 *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **33** 1476
[24] Nam Y, Ross N J, Su Y, Childs A M and Maslov D 2018 *npj Quantum Information* **4** 23
[25] Amy M and Mosca M 2016 arXiv:1601.07363
[26] Seroussi G and Lempel A 1980 *SIAM J. Comput.* **9** 758
[27] Lempel A 1975 *SIAM J. Comput.* **4** 175–86
[28] Source code 10.1109/JPHOT.2014.2352635
[29] Bremner M J, Jozsa R and Shepherd D J 2011 *Proc. R. Soc.* A **467** 459–73
[30] Montanaro A 2017 *J. Phys. A: Math. Theor.* **50** 084002
[31] Amy M, Maslov D, Mosca M and Roetteler M 2013 *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **32** 818
[32] Håstad J 1990 *J. Algorithms* **11** 644
[33] Maslov D 2011 Reversible logic synthesis benchmarks http://webhome.cs.uvic.ca/~dmaslov/
[34] Amy M T-par GitHub https://github.com/meamy/t-par
[35] Nam Y, Ross N J, Su Y, Childs A M and Maslov D GitHub for [24], https://github.com/njross/optimizer
[36] Iceberg HPC Cluster https://sheffield.ac.uk/wrgrid/iceberg
[37] Harrow A W and Montanaro A 2017 *Nature* **549** 203
[38] Shepherd D and Bremner M J 2009 *Proc. R. Soc.* A **465** 1413
[39] Maslov D 2016 *Quant. Inf. Comp.* **16** 1096–112
[40] Raussendorf R, Harrington J and Goyal K 2007 *New J. Phys.* **9** 199
[41] Fowler A G, Devitt S J and Jones C 2013 *Sci. Rep.* **3** 1939
[42] Fowler A G and Devitt S J 2012 arXiv:1209.0510

Chapter 5

A Quantum Compiler for Qudits of Prime Dimension Greater than 3

The following manuscript is an original work by the present author and Dr. Earl T. Campbell that was submitted to the online pre-print archive, arXiv.org, on 14th February 2019 [43]. The full text is provided below in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

arXiv:1902.05634v1 [quant-ph] 14 Feb 2019

A quantum compiler for qudits of prime dimension greater than 3

Luke E. Heyfron and Earl Campbell

Department of Physics and Astronomy, University of Sheffield, Sheffield, UK

Prevailing proposals for the first generation of quantum computers make use of 2-level systems, or *qubits*, as the fundamental unit of quantum information. However, recent innovations in quantum error correction and magic state distillation protocols demonstrate that there are advantages of using d-level quantum systems, known as *qudits*, over the qubit analogues. When designing a quantum architecture, it is crucial to consider protocols for compilation, the optimal conversion of high-level instructions used by programmers into low-level instructions interpreted by the machine. In this work, we present a general purpose automated compiler for multiqudit exact synthesis based on previous work on qubits that uses an algebraic representation of quantum circuits called phase polynomials. We assume Clifford gates are low-cost and aim to minimise the number of M gates in a Clifford+M circuit, where M is the qudit analog for the qubit T or $\pi/8$ phase gate. A surprising result that showcases our compiler's capabilities is that we found a unitary implementation of the CCZ or Toffoli gate that uses 4 M gates, which compares to 7 T gates for the qubit analogue.

I. INTRODUCTION

Despite its ubiquity in computing, the choice to use binary instead of ternary or some other numeral system is almost arbitrary. From a purely information theoretic perspective, there is no reason to prefer bits over d-value anologues, known as *dits*. In fact, successful experiments into 3-value logic were realised in the form of the *Setun*, a ternary computer built in 1958 by Sergei Sobolev and Nikolay Brustentsov at Moscow State University [1]. The near universal adoption of binary can be explained from an engineering perspective in that it is much simpler to manufacture binary components. However, since as early as the 1940's with the biquinary Collossus computer, it has been widely understood that there are intrinsic efficiency benefits of using higher dimensional logic components in that fewer are required.

In the standard paradigm, there are three components required for a fault tolerant quantum computing architecture: quantum error correction (QEC) codes; magic state distillation (MSD) protocols; and finally, quantum compilers. For qudits, there has been progress showing that both qudit QEC [2–6] and qudit MSD [7–11] offer a resource advantage in shifting from qubits to qudits. However, surprisingly little work has been done on qudit compiling, except for the special case of qutrits where $d = 3$ [12, 13]. Therefore, compiling is the crucial missing piece in understanding quantum computing with qudit logic beyond $d = 3$.

A standard metric for quantum compilers to minimize is the number of expensive gates that require magic state distillation. In the qubit case, the T gate is typically the designated magic gate in the low-level instruction set and much progress has been made on gate synthesis in this context. For single qubits, the Matsumoto-Amano normal form [14–16] leads to decompositions of single qubit unitaries as sequences of gates from the Clifford + T gate set that is optimal with regards to T count for a given approximation error. So for single qubits, the problem is essentially “solved”. For multi-qubit operators, methods for T-optimal exact compilation have been developed but suffer exponential runtime [17]. More recently, efficient optimizers have been developed that successfully reduce T count, some of which are

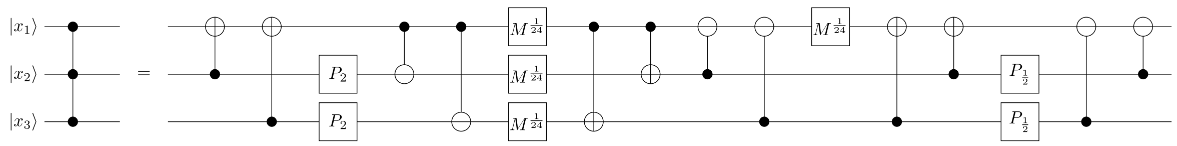


FIG. 1. A unitary implementation of the CCZ gate that uses 4 non-Clifford M^k gates. Note that the $\frac{1}{24}$ in the exponent of the M^k gates is the multiplicative inverse of 24 in the field $\mathbb{Z}_d$. Gates used here are defined in Sec. II.

based on a correspondence between unitaries on a restricted gate set and so called phase-polynomials [18–21], and others that are based on local rewrite rules [22]. For qudits, there has been some work on single qutrit (three level systems) synthesis [23] that can be considered a qutrit generalisation of the Matsumoto-Amano normal form.

In this work, we borrow ideas from the phase-polynomial style T count optimization protocols and apply these insights to qudits. We provide a general purpose compiler for exact synthesis of multiqudit unitaries generated by M, P_l and SUM gates where the M gate is the canonical "expensive" magic gate (i.e. the qudit analogue of the T gate). We present an example of a M count reduction only possible for odd prime $d > 3$. This is the CCZ gate, which is known to have optimal T count of 7 when synthesised unitarily using qubit based quantum computers, whereas our decomposition has M count of 4. Until now, this reduced cost has only been achieved in the qubit setting using non-unitary gadgets that exploit ancillas [24].

II. PRELIMINARIES

Let $d > 3$ be a prime integer. We define a Hilbert space on n qudits spanned by the computational basis vectors $\{|\mathbf{x}\rangle \mid \forall \mathbf{x} \in \mathbb{Z}_d^n\}$. We define the single qudit Pauli operators $X := \sum_{x\in\mathbb{Z}_d} |x+1\rangle\langle x|$, where addition is performed modulo d; $Z := \sum_{x\in\mathbb{Z}_d} \omega^x |x\rangle\langle x|$, where $\omega := \exp^{i\frac{2\pi}{d}}$ is a primitive d^{th}root of unity. The set of all n qudit unitaries generated by X and Z form the Pauli group, $\mathcal{P}$.

The Clifford group, $\mathcal{C}$, is the normalizer of the Pauli group and is generated by:

$$H := \frac{1}{\sqrt{d}} \sum_{x,y\in\mathbb{Z}_d} \omega^{xy} |y\rangle\langle x| \tag{1}$$

$$SUM := \sum_{c,t\in\mathbb{Z}_d} |c, t+c\rangle\langle c,t| \tag{2}$$

$$S := \sum_{x\in\mathbb{Z}_d} \omega^{x^2} |x\rangle\langle x| \tag{3}$$

We refer to H as the Hadamard; the Hadamard gate, SUM is the two-qudit SUM gate; and S is the *phase gate.* Note that the Z and S gates are both diagonal and correspond to linear and quadratic terms, respectively, appearing in the exponent of the phase.

We further define the Clifford unitaries

$$P_l := \sum_{x\in\mathbb{Z}_d} |lx\rangle\langle x| \tag{4}$$

for all integer $l \neq 0$, which we call *product operators* as they perform field multiplication between the input basis states and a non-zero field element, l. It can be shown that all product operators are in the Clifford group.

As in previous works [8, 25, 26], we define the canonical non-Clifford gate to be

$$M := \sum_{x \in \mathbb{Z}_d} \omega^{x^3} |\mathbf{x}\rangle \langle \mathbf{x}| , \tag{5}$$

which lies in the third level of the Clifford hierarchy and in standard fault tolerant architectures are much more costly than Clifford gates due to the need for MSD.

III. THE COMPILING PROBLEM

A compiler converts high-level instructions into low-levels ones. In this paper, we concern ourselves with high-level instructions that take the form of n-qudit unitaries which can be exactly synthesised by a discrete gate set, $\mathcal{G}$. By low-level instructions, we specifically refer to quantum circuits, which are represented as *netlists*, or time-ordered lists of gates taken from $\mathcal{G}$ where the qudits to which they apply (as well as any other gate parameters) are specified for each gate. The unitary that a particular quantum circuit implements is simply the right-to-left matrix product of each gate in the netlist extracted in time-order.

Problem 1. *(Compiling Problem). Given a unitary $U \in \mathcal{G}$, find a quantum circuit that implements U with the lowest cost.*

Note that the compiling problem is ill-defined and depends on the definition of cost. The most accurate metric of quantum circuit cost is the full space-time volume, which is the number of machine level operations multiplied by the number of physical qubits. The calculation required to determine the full space-time volume is lengthy and is highly sensitive to the choice of architecture [27–29]. The M *count*, or the number of M^k gates in a quantum circuit, is an alternative cost metric that gives a good approximation to the full cost and can be easily read off compiler-level quantum circuits. Using the M count in problem 1, we obtain a well defined compiling problem.

Problem 2. *(M-Minimization). Given a unitary $U \in \mathcal{G}$, find a quantum circuit that implements U with the fewest M^k gates.*

We choose our gate set to be $\mathcal{G} = \{Z, S, M^k, P_l, SUM\}$ for all available choices of k and l. While we would ideally work with a universal gate set such as Clifford + M, the compiling problem is known to be intractable in the universal case so we focus on this simpler sub-problem. For the selection of gates in $\mathcal{G}$, we have taken inspiration from previous work [18–21], where it was demonstrated that such a restriction leads to an algebraic reformulation of the compiler problem that is more amenable to computational methods, including efficient heuristics.

IV. PHASE POLYNOMIAL FORMALISM

The formalism described in this section allows us to reframe the M-minimization problem as a computationally-friendly problem on integer matrices. It applies strictly to unitaries

$U \in \langle Z, S, M^k, P_l, SUM \rangle$ and is a straightforward generalisation of previous work [18–21]. We proceed with a lemma that establishes a correspondence between unitaries generated by $\mathcal{G}$ and cubic polynomials that we call *phase polynomials.*

Lemma 1. *Any n qudit unitary $U_f \in \langle \mathcal{G} \rangle$ can be expressed as follows:*

$$U_f = \sum_{\mathbf{x} \in \mathbb{Z}_d^n} \omega^{f(\mathbf{x})} \left| E\mathbf{x} \right\rangle \left\langle \mathbf{x} \right|, \tag{6}$$

where E is an invertible matrix implementable with SUM gates, and $f : \mathbb{Z}_d^n \mapsto \mathbb{Z}_d$ is a polynomial of order less than or equal to 3.

Proof. To prove the first part, we first show that each gate in the generating set can be written in the above form, then show that the set generated by these operators form a group. From the definitions provided in section II, we have that Z, S and M gate applied to the t^{th} qudit can be written in the form of equation (6) with $f(\mathbf{x}) = x_t, x_t^2, x_t^3$, respectively, and with $E = \mathbb{I}$. P_l applied to the t^{th} qudit has $f(\mathbf{x}) = 0$ (as does the SUM gate) and $E = \mathbb{I} + ((l-1)\delta_{i,t}\delta_{j,t})$ with inverse $E^{-1} = \mathbb{I} + ((\frac{1}{l} - 1)\delta_{i,t}\delta_{j,t})$. Finally, the SUM gate whose control and target are the c^{th} and t^{th} qudits, respectively, has $E = \mathbb{I} + (\delta_{i,t}\delta_{j,c})$, which has inverse $E^{-1} = \mathbb{I} - (\delta_{i,t}\delta_{j,c})$. By definition, the set generated by $\mathcal{G}$ is closed under multiplication and as each generator is a unitary matrix, the associative property holds. Finally, $\mathbb{I}, Z^\dagger, S^\dagger, M^\dagger \in \langle \mathcal{G} \rangle$ so the identity and inverse group axioms are satisfied.

To prove the second part, that $f(\mathbf{x})$ is cubic, we note that the only gates which contribute to $f(\mathbf{x})$ are Z, S and M, which add a term equal to the state of the acted-upon qudit raised to the first, second and third power, respectively. Because the Z, S and M gates are diagonal, the state of any qudit at any point in the circuit can only change due to the P_l and SUM gates, which together map the state of each qudit to linear functions of the input states with coefficients in $\mathbb{Z}_d$. The linear functions can, at most, be raised to the 3rd power (due to the M gate), before contributing a term to $f(\mathbf{x})$. Therefore, the order of $f(\mathbf{x})$ is at most cubic. □

The linear and quadratic terms of any $f(\mathbf{x})$ can be implemented using just Clifford operations, which cost considerably less than the cubic terms that require M gates. Therefore, we assume that $f(\mathbf{x})$ is a homogeneous cubic polynomial. It follows that $f(\mathbf{x})$ can be decomposed in the monomial basis as follows:

$$f(\mathbf{x}) = \sum_{\alpha,\beta,\gamma=1}^{n} S_{\alpha,\beta,\gamma} x_\alpha x_\beta x_\gamma, \tag{7}$$

where $S \in \mathbb{Z}_d^{(n,n,n)}$. Since every choice of (α, β, γ) for $\alpha \leq \beta \leq \gamma$ corresponds to a different linearly independent monomial, if we enforce that S is symmetric, it follows that the elements of S uniquely determine the function $f(\mathbf{x})$. For this reason, we call it the *signature tensor.*

The phase polynomial $f(\mathbf{x})$ can also be decomposed as a sum over linear forms raised to the third power, as in the following:

$$f(\mathbf{x}) = \sum_{j=1}^{m} \lambda_j \left(\sum_{i=1}^{n} A_{i,j} x_i \right)^3, \tag{8}$$

where $\lambda \in (\mathbb{Z}_d \setminus \{0\})^m$ and $A \in \mathbb{Z}_d^{(n,m)}$ such that for each column in A, there is at least one non-zero element. It is straightforward to calculate the signature tensor from the elements of A and λ using the following relation,

$$S_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} \lambda_j A_{\alpha,j} A_{\beta,j} A_{\gamma,j}. \tag{9}$$

Definition 1. ***Implementation.*** *Let U_f be a unitary with signature tensor $S \in \mathbb{Z}_d^{(n,n,n)}$. Let $A \in \mathbb{Z}_d^{(n,m)}$ and $\lambda \in (\mathbb{Z}_d \setminus \{0\})^m$. We say that the tuple (A, λ) is an* implementation *of S if it satisfies equation* (9)*.*

We refer to the tuple (A, λ) as an implementation because it reveals information sufficient to construct a quantum circuit that implements U_f with known M count, as stated in the following lemma.

Lemma 2. *Let U_f be a unitary with an implementation (A, λ) that has m columns. It follows that a quantum circuit can be efficiently generated which implements U_f using no more than m M gates.*

As proof of lemma 2, we provide in appendix A an explicit algorithm for efficiently converting an implementation with m columns into a quantum circuit with m M^k gates.

The connection between column count of implementations and M count of quantum circuits is central to the understanding of this work and leads to a restatement of the compiler problem that is more amenable to computational solvers.

Problem 3. *(Column-minimization). Let S be a signature tensor. Find an implementation (A, λ) that implements S with minimal columns.*

V. EXAMPLE: CCZ GATE

Take the CCZ gate as an example, which acts upon the computational basis as follows.

$$CCZ \left| x_1, x_2, x_3 \right\rangle = \omega^{x_1 x_2 x_3} \left| x_1, x_2, x_3 \right\rangle . \tag{10}$$

In the monomial basis, the phase polynomial can be read off directly as $f(\mathbf{x}) = x_1 x_2 x_3$, which corresponds to a signature tensor with $S_{\sigma(1,2,3)} = \frac{1}{6}$ for all permutations σ and $S_{\alpha,\beta,\gamma} = 0$ for all other elements. However, to generate a quantum circuit for $U_f = CCZ$, we first need to find an implementation for S. By applying knowledge of the qubit version of the CCZ gate to qudits [18–21], we arrive at the following implementation[30] that has an M count of 7:

$$\begin{pmatrix} A \\ \hline \lambda \end{pmatrix} = \left(\begin{array}{ccccccc} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & -\frac{1}{6} & -\frac{1}{6} & -\frac{1}{6} & \frac{1}{6} \end{array}\right), \tag{11}$$

which corresponds to the phase polynomial,

$$\begin{aligned} f(\mathbf{x}) = \frac{1}{6}x_1^3 + \frac{1}{6}x_2^3 + \frac{1}{6}x_3^3 - \frac{1}{6}(x_1 + x_2)^3 \\ - \frac{1}{6}(x_1 + x_3)^3 - \frac{1}{6}(x_2 + x_3)^3 + \frac{1}{6}(x_1 + x_2 + x_3)^3. \end{aligned} \tag{12}$$

We remind the reader that all elements of an implementation are in $\mathbb{Z}_d$, so the fraction $\frac{1}{6} = x \in \mathbb{Z}_d$ where x solves $6x = 1 \pmod{d}$. One can easily verify that the above implementation, (A, λ), satisfies equation (9) for every element of the signature tensor, $S_{a,b,c}$, confirming that it implements the CCZ gate.

Using a computer aided discovery method described in section VI, we have found an implementation with M count 4 that works for all choices of d. This is a key result of the present work and is provided below.

$$\left(\frac{A}{\lambda}\right) = \left(\begin{array}{cccc} 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ \hline \frac{1}{24} & \frac{1}{24} & \frac{1}{24} & \frac{1}{24} \end{array}\right). \tag{13}$$

This corresponds to the phase polynomial

$$\begin{aligned} f(\mathbf{x}) = \frac{1}{24}(x_1 + x_2 + x_3)^3 + \frac{1}{24}(x_1 - x_2 - x_3)^3 \\ + \frac{1}{24}(x_2 - x_1 - x_3)^3 + \frac{1}{24}(x_3 - x_1 - x_2)^3. \end{aligned} \tag{14}$$

An explicit quantum circuit for the above implementation of the CCZ gate is provided in figure 1.

VI. COMPILERS

A. Brute-Force

In order to construct an M-optimal implementation for a given phase polynomial, one can perform a brute-force search over all possible implementations, checking in polynomial-time in each case that it corresponds to the correct signature tensor using equation (9). However, the size of the search space scales as $O(d^{(n+1)m})$, which makes execution times impractical, even for modest sized inputs. However, by searching in m-order where m is the candidate number of M gates, we can optimally compile unitaries on $n = 3$ ququints ($d = 5$) with M count of up to 4 (and lower bound unitaries with an implicitly higher optimal M count). It was through this brute-force method that we were able to discover the implementation of CCZ with M count of 4 presented in equation (13).

B. Monomial Substitution

It is critically important that a general-purpose compiler is efficient. Fortunately, there is a simple method to map a phase polynomial in the monomial basis to an implementation. There are three kinds of monomial that may appear in a phase polynomial, which are distinguished by the number of variables they take. These are x_a^3, $x_a x_b^2$ and $x_a x_b x_c$. As each monomial is linearly independent, if we can find a prototypical implementation for each kind of monomial, then it follows that we can compile an implementation for a general phase polynomial by substituting instances of the prototypes for each monomial. Again using [21] as inspiration, we provide prototype implementations for the three kinds of monomial below.

$$x_a^3 \to x_a^3 \tag{15}$$

$$x_a x_b^2 \to \frac{1}{6}(x_a + x_b)^3 + \frac{1}{6}(x_a - x_b)^3 - \frac{1}{3}x_a^3 \tag{16}$$

$$x_a x_b x_c \to \frac{1}{24}(x_1 + x_2 + x_3)^3 + \frac{1}{24}(x_1 - x_2 - x_3)^3 + \frac{1}{24}(x_2 - x_1 - x_3)^3 + \frac{1}{24}(x_3 - x_1 - x_2)^3, \tag{17}$$

where we have used the implementation from equation (14) for the $x_a x_b x_c$ prototype. Of course, we can also use the "legacy" M-count 7 implementation from equation (12), which for certain input unitaries (e.g. ones that contain many gates on the same qudit lines) lead to lower M count implementations due to column merging (see section VI C).

We call the above method *monomial substitution*, which executes in time that scales as $O(n^3)$ in the worst case, making it efficient. However, the output M count should be considered a crude initial guess at the optimal M count and can be significantly improved by the optimization methods described in remainder of this section.

C. M-Optimization

One approach to solving problem 3 is to try and 'merge' columns of an existing implementation. A pair of columns can be merged if they are duplicates of one another. This is because we can collect like terms in the phase polynomial where the coefficients combine linearly. An illustrative example is the following. Let f be a phase polynomial with two terms, hence has implementation matrix A with two columns,

$$f(\mathbf{x}) = \lambda_1(A_{1,1}x_1 + A_{2,1}x_2 + \cdots + A_{n,1}x_n)^3 \tag{18}$$

$$+ \lambda_2(A_{1,2}x_1 + A_{2,2}x_2 + \cdots + A_{n,2}x_n)^3 \tag{19}$$

if the two columns of A are duplicates, then we have $A_{i,1} = A_{i,2}\ \forall\ i \in [1, n]$. And so

$$f(\mathbf{x}) = (\lambda_1 + \lambda_2)(A_{1,1}x_1 + A_{2,1}x_2 + \cdots + A_{n,1}x_n)^3, \tag{20}$$

which needs only a single column to represent it, and therefore only a single magic state to implement it.

Of course, it is often the case that an A matrix does not contain any duplicates. In this case, we wish to transform A in some way in order to make it contain duplicates, and in such a way that it does not alter the unitary it implements. In appendix B, we describe an M-optimizer that systemically searches for and performs such "duplication transformations", and subsequently merges the duplicated columns. For this reason, we call it the Duplicate And Merge (DAM) algorithm.

The algorithm runs in time that scales as $O(mn^3d^m)$ so is inefficient. However, in practice, it executes much faster than the brute-force compiler and often outputs M-optimal implementations, albeit non-deterministically, and is useful for raising the practical limit on input circuits.

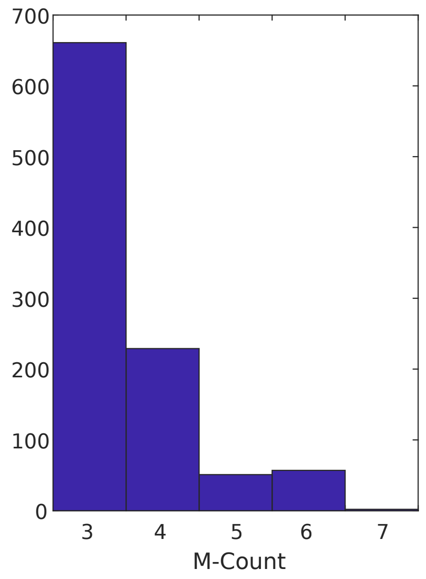


FIG. 2. The distribution of M-counts for DAM run on a single random implementation with $d = 5$, $n = 3$ and known optimal M-count of 3 performed 1000 times.

VII. BENCHMARKS

In order to determine the speed benefits of using DAM over a brute force search (BFS) and to assess the inevitable drop in M-optimality, we performed a benchmark on randomly generated implementations with an M-count of 3 for $d = 5$ and $n = 3$. These parameters were chosen as they are the largest parameters that are feasible for BFS where many repetitions are required. Each of the 100 random implementations were first compiled by BFS, then the legacy monomial substitution compiler was run using the signature tensor as input, which was subsequently optimized using DAM 1000 times. The distribution of M-counts after optimization with DAM was recorded and an example of a single random instance is shown in figure 2.

We have performed a number of benchmarks on larger circuits in order to compare the monomial substitution, legacy monomial substitution and DAM compilers, which are shown in table I. From the data we see that MS is the preferred compiler for low depth circuits such as the $CCZ^{\otimes k}$ family, which is to be expected as it is likely that the optimal M-count in this case is $4k$. In contrast, the DAM compiler consistently outperforms MS and Legacy for random circuits. Unfortunately, DAM is too inefficient to be practically useful for scalable quantum computers as we see from the large execution times. The computational bottleneck for DAM is due to the hardness of solving the multivariate cubic system of equations in equation (35). Our implementation uses a search over the space of all vectors on $\mathbb{Z}_d$ that are of length m in the worst case (for details see appendix B). Therefore, the search consists of $O(d^m)$ iterations and as all other parts of the DAM algorithm executes in polynomial time, this is the sole source of inefficiency. It follows that if one had access to an efficient heuristic that solves the system of equations in (35), then DAM immediately becomes efficient. Although, it would probably be the case that as a consequence of it being a heuristic, not all column mergings would be discoverable.

As DAM is a non-deterministic heuristic, it is important to quantify the stability of its output. We used the results to estimate the probability that DAM produces an M-optimal implementation of an unknown quantum circuit for the given circuit parameters and found it to be $p_{\text{opt}} := p(\text{optimal} \mid d = 5, n = 3, m = 3) = 0.47 \pm 0.02$. This probability should in no way be interpreted as representative of DAM's performance in general (esp. not for larger circuits) but instead suggests that a best-of-N approach (i.e. where DAM is run N times and the implementation with the minimum M-count is returned) would be effective and may return M-optimal solutions.

The probability of optimality can be used to estimate the minimum number of DAM repetitions, N, that one should perform in order to reach a desired confidence threshold, p_{conf}, using the following:

$$N = \left\lceil \frac{\ln(1 - p_{\text{conf}})}{\ln(1 - p_{\text{opt}})} \right\rceil. \tag{21}$$

Using our results for p_{opt} and a confidence threshold of $p_{\text{conf}} = 0.95$, we arrive at $N = 5$. The mean execution time[31] for a single run of DAM was $0.119s \pm 0.006s$ compared to $91s \pm 6s$ for BFS. Therefore, the best-of-5 DAM compiler remains faster than BFS by two orders of magnitude.

TABLE I. Benchmark table for the Monomial Substitution (MS), Legacy (Leg.) and Duplicate And Merge (DAM) compilers. For each benchmark, a best-of-10 method was used for DAM. The M-counts are shown in the M_{compiler} columns and the executions times in the t_{compiler} columns. The executions times were obtained on the Iceberg HPC Cluster at the University of Sheffield. The "Random" circuits are randomly generated signature tensors where each element is a non-zero value with 50% probability the subsequent value is selected uniformly. All M-counts and execution times reported in these rows are mean values taken over 100 random circuits, or as many circuits as could be synthesised within a 24 hour window. The notation $\text{CCZ}_{\#m}$ refers to a circuit with m CCZ gates where they all share exactly 1 control qudit in common.

Circuit	**d**	**n**	$\mathbf{M}_{\text{Leg.}}$	$\mathbf{M}_{\text{MS}}$	$\mathbf{M}_{\text{DAM}}$	$\mathbf{t}_{\text{Leg.}}$ **(s)**	$\mathbf{t}_{\text{MS}}$ **(s)**	$\mathbf{t}_{\text{DAM}}$ **(s)**
CCZ	5	3	7	4	5	0.10	0.02	0.98
$\text{CCZ}^{\otimes 2}$	5	6	14	8	10	0.03	0.01	20.87
$\text{CCZ}^{\otimes 3}$	5	9	21	12	16	0.02	<0.01	813.02
CCZ	7	3	7	4	7	0.08	0.03	6.99
$\text{CCZ}^{\otimes 2}$	7	6	14	8	10	0.04	0.01	182.69
CCZ	11	3	7	4	7	0.09	0.04	38.81
$\text{CCZ}^{\otimes 2}$	11	6	14	8	12	0.04	0.01	11489.15
$\text{CCZ}_{\#2}$	5	5	13	8	8	0.09	0.02	34.9
$\text{CCZ}_{\#3}$	5	7	19	12	12	0.04	0.01	2219.68
$\text{CCZ}_{\#2}$	7	5	13	8	8	0.08	0.02	214.09
$\text{CCZ}_{\#3}$	7	7	19	12	12	0.04	0.01	23950.64
$\text{CCZ}_{\#2}$	11	5	13	8	8	0.08	0.02	7976.43
Random	5	3	8.26	11.38	4.52	<0.01	<0.01	1.40
Random	5	4	16.93	28.86	7.21	<0.01	<0.01	825.4959
Random	7	3	8.68	11.59	4.38	<0.01	<0.01	11.76
Random	7	4	16.88	28.75	7.13	<0.01	<0.01	10416.00
Random	11	3	9.08	12.05	4.38	<0.01	<0.01	185.46

VIII. CONCLUSIONS AND ACKNOWLEDGEMENTS

In this work we have generalised the phase polynomial type optimizers to qudit based quantum computers and have used it to demonstrate cost savings only possible in the qudit picture. This motivates serious discussion into fundamental questions regarding the nature of first generation fault-tolerant architectures, namely whether they use qubits or qudits.

We acknowledge support by the Engineering and Physical Sciences Research Council (EPSRC) through grant EP/M024261/1. We thank Mark Howard for discussions throughout the project.

Appendices

A. PROOF OF LEMMA 2

Let $U_f \in \langle \mathcal{G} \rangle$ be a unitary and (A, λ) be an implementation for f with m columns. We can efficiently generate a circuit, C, on $\mathcal{G}$ that implements U_f from (A, λ) using m M gates with the following algorithm:

1. Initialize an empty circuit, C.
2. For each $j \in [1, m]$:
   (a) Initialize an empty circuit, D.
   (b) Let $H := \{i \mid A_{i,j} \neq 0\}$.
   (c) Arbitrarily choose a $t \in H$.
   (d) Append $P_{A_{t,j}}$ on qudit line t to D.
   (e) For each $c \in H \setminus \{t\}$:
      i. Append $P_{A_{c,j}}$ on qudit line c to D.
      ii. Append $SUM_{c,t}$ to D.
      iii. Append $P_{\frac{1}{A_{c,j}}}$ on qudit line c to D.
   (f) Append $P_{\frac{1}{A_{t,j}}}$ on qudit line t to D.
   (g) Append D to C.
   (h) Append M^{λ_j} on qudit line t to C.
   (i) Append $D^\dagger$ to C.

First, observe steps 2d to 2f, which creates a subcircuit D using only P_l and SUM gates that maps the state of the t^{th} qudit to a linear function of the n input qudits $x_1, x_2, \ldots, x_n$ that has coefficients given by the j^{th} column of A. After D is appended to the output circuit, C, step 2h applies an M^k gate with $k = \lambda_j$, which adds a term to the phase polynomial $f(\mathbf{x})$ equal to the aforementioned linear function cubed multiplied by λ_j, as required. Finally, in step 2i, the linear function is uncomputed by $D^\dagger$. The whole process is repeated for each of the m columns of A. Each iteration requires only one M^k gate so the total number of M gates required is m. The algorithm executes in $O(mn)$ steps and so is efficient. We end this appendix with the disclamation that the above algorithm is not intended to be optimal with respect to the number of Clifford gates (i.e. P_l and SUM) used.

B. THE DUPLICATE AND MERGE OPTIMIZER

The following is a direct generalisation of the TODD compiler from reference [21] to qudits. The key difference is that the the null space step from TODD is replaced with a multivariate cubic system, for which a common root must be found. We refer the reader to Section 3.4 and Algorithm 1 of [21] for an overview of the TODD compiler, which may aid in understanding DAM.

Definition 2. ***Duplication Transformation.*** *Let $A \in \mathbb{Z}_d^{(n,m)}$ be an implementation and $\mathbf{y} \in \mathbb{Z}_d^m$ be a vector. We define the* duplication transformation *as follows:*

$$A \to A + (\mathbf{c}_b - \mathbf{c}_a)\mathbf{y}^T, \tag{22}$$

where $\mathbf{c}_j$ is the j^{th} column of A.

We can use this transformation to 'create' duplicates as the following lemma shows.

Lemma 3. *Let $A' = A + (\mathbf{c}_b - \mathbf{c}_a)\mathbf{y}^T$ and assume that $y_a - y_b = 1$. It follows that $\mathbf{c}'_a = \mathbf{c}'_b$.*

Proof. From the definition of A',

$$A'_{i,j} = A_{i,j} + z_i y_j, \tag{23}$$

now substitute in $z_i \equiv A_{i,b} - A_{i,a}$,

$$A'_{i,j} = A_{i,j} + (A_{i,b} - A_{i,a})y_j. \tag{24}$$

Apply equation (24) to both $(\mathbf{c}_a)_i \equiv A'_{i,a}$ and $(\mathbf{c}_b)_i \equiv A'_{i,b}$,

$$A'_{i,b} = A_{i,b} + (A_{i,b} - A_{i,a})y_b, \tag{25}$$

$$A'_{i,a} = A_{i,a} + (A_{i,b} - A_{i,a})y_a. \tag{26}$$

Substitute $y_a = y_b + 1$ into equation (26) and rearrange,

$$\begin{aligned} A'_{i,a} &= A_{i,a} + (A_{i,b} - A_{i,a})(y_b + 1) && (27)\\ &= A_{i,a} + (A_{i,b} - A_{i,a})y_b + A_{i,b} - A_{i,a} && (28)\\ &= A_{i,b} + (A_{i,b} - A_{i,a})y_b = A'_{i,b}. && (29) \end{aligned}$$

$\forall\ i \in [1, n]$ so $\mathbf{c}'_a = \mathbf{c}'_b$. $\square$

The duplication transformation must not alter f. This leads to the condition $S'_{\alpha,\beta,\gamma} = S_{\alpha,\beta,\gamma}\ \forall\ \alpha, \beta, \gamma \in [1, n]$, where S' and S are the signature tensors for A' and A, respectively. So,

$$\begin{aligned} S'_{\alpha,\beta,\gamma} &= \sum_{j=1}^{m} \lambda_j A'_{\alpha,j} A'_{\beta,j} A'_{\gamma,j}, && (30)\\ &= \sum_{j=1}^{m} \lambda_j (A_{\beta,j} + z_\beta y_j)(A_{\beta,j} + z_\beta y_j)(A_{\gamma,j} + z_\gamma y_j), && (31)\\ &= \sum_{j=1}^{m} \lambda_j A_{\alpha,j} A_{\beta,j} A_{\gamma,j} + \Delta_{\alpha,\beta,\gamma} = S_{\alpha,\beta,\gamma} + \Delta_{\alpha,\beta,\gamma}, && (32) \end{aligned}$$

where we define,

$$\begin{aligned} \Delta_{\alpha,\beta,\gamma} := \sum_{j=1}^{m} \lambda_j (&A_{\alpha,j} A_{\beta,j} z_\gamma y_j + A_{\beta,j} A_{\gamma,j} z_\alpha y_j + A_{\gamma,j} A_{\alpha,j} z_\beta y_j \\ &+ A_{\alpha,j} z_\beta z_\gamma y_j^2 + A_{\beta,j} z_\gamma z_\alpha y_j^2 + A_{\gamma,j} z_\alpha z_\beta y_j^2 + z_\alpha z_\beta z_\gamma y_j^3), \end{aligned} \qquad (33)$$

and $\mathbf{z} := \mathbf{c}_b - \mathbf{c}_a$. In order for $S' = S$, we require that

$$\Delta_{\alpha,\beta,\gamma} = 0 \quad \forall \quad \alpha, \beta, \gamma \in [1, n]. \qquad (34)$$

This leads to a system of $\sum_{i=1}^{3} \binom{n}{i}$ cubic polynomials on r variables $(y_1, y_2, \ldots, y_m)$ that can be rewritten as follows:

$$\sum_{j=1}^{m} \left(l_{\alpha,\beta,\gamma,j} y_j + q_{\alpha,\beta,\gamma,j} y_j^2 + c_{\alpha,\beta,\gamma,j} y_j^3 \right) = 0, \qquad (35)$$

$$y_a - y_b - 1 = 0 \qquad (36)$$

where the linear, quadratic and cubic coefficients for variable t are given by,

$$\begin{aligned} l_{\alpha,\beta,\gamma,j} &= \lambda_j (A_{\alpha,j} A_{\beta,j} z_\gamma + A_{\beta,j} A_{\gamma,j} z_\alpha + A_{\gamma,j} A_{\alpha,j} z_\beta) && (37)\\ q_{\alpha,\beta,\gamma,j} &= \lambda_j (A_{\alpha,j} z_\beta z_\gamma + A_{\beta,j} z_\gamma z_\alpha + A_{\gamma,j} z_\alpha z_\beta) && (38)\\ c_{\alpha,\beta,\gamma,j} &= \lambda_j z_\alpha z_\beta z_\gamma, && (39) \end{aligned}$$

respectively.

Any $\mathbf{y}$ that is a simultaneous solution to equations (35) and equation (36) allows us to reduce the number of columns of A using the duplication transformation from definition (2). Unfortunately, the problem of solving a general multivariate cubic system such as this is known to be NP-complete. A brute-force solver that searches through every possible $\mathbf{y}$ runs in $O(d^m)$ time. However, we can significantly speed up the search using the following relinearisation technique. First, we introduce new variables, $y_{m+1}, y_{m+2}, \ldots, y_{3m}$, such that

$$\begin{aligned} y_{m+j} &= y_j^2, && (40)\\ y_{2m+j} &= y_j^3, && (41)\\ & && (42) \end{aligned}$$

for all $j \in [1, m]$. The system of equations from (35) becomes:

$$\sum_{j=1}^{m} l_{\alpha,\beta,\gamma,j} y_j + \sum_{k=m+1}^{2m} q_{\alpha,\beta,\gamma,k} y_k + \sum_{l=2m+1}^{3m} c_{\alpha,\beta,\gamma,l} y_l = 0, \tag{43}$$

which is linear in the $\{y_j\}$. Let D be the coefficient matrix defined as follows: For each triple $\{(\alpha, \beta, \gamma) \mid \alpha \leq \beta \leq \gamma \in [1, n]$, there exists a row in D of the following form:

$$\textsc{Row}_{\alpha,\beta,\gamma}(D) = \left((l_{\alpha,\beta,\gamma,j}) \;\; (q_{\alpha,\beta,\gamma,j}) \;\; (c_{\alpha,\beta,\gamma,j}) \right). \tag{44}$$

Now we can calculate a complete basis for the solutions to equation (43) by calculating the right null space of D, which we denote N_D. We can think of the columns of N_D as a basis for the 'partial' solutions of the system of equations (35). In order to promote them to 'full' solutions, we need to enforce conditions from equations (40) and (36), which we do using the following algorithm.

1. Form N'_D by erasing all but the first m rows of N_D.
2. Form N''_D by column-reducing N'_D and subsequently removing every all-zero column.
3. Let $\mu := \textsc{Cols}(N''_D)$.
4. For each $\mathbf{x} \in \mathbb{Z}_d^{\mu}$:
   (a) Construct $\mathbf{y_x} = N''_D \mathbf{x}$
   (b) Construct $\mathbf{y}'_\mathbf{x} = \begin{pmatrix} \mathbf{y_x} \\ \mathbf{y}_\mathbf{x}^2 \\ \mathbf{y}_\mathbf{x}^3 \end{pmatrix}$
   (c) If $D\mathbf{y}'_\mathbf{x} = \mathbf{0}$ and (36) holds, then return $\mathbf{y_x}$.
5. Return "No Solution".

In effect, the relinearlisation method replaces a search over every $\mathbf{y} \in \mathbb{Z}_d^m$ with a search over every $\mathbf{x} \in \mathbb{Z}_d^{\mu}$, so it only runs faster if $\mu < m$. It is certainly the case that $\mu \leq m$ as μ is the column rank of N'_D, which has m rows. Whether or not the strict inequality holds depends on the input circuit but in practice, we often find that it does hold and leads to significant speed-up over the naive brute force approach.

[1] N. P. Brusentsov and J. Ramil Alvarez, IFIP Advances in Information and Communication Technology , 74 (2011).
[2] G. Duclos-Cianci and D. Poulin, Phys. Rev. A **87**, 062338 (2013).
[3] H. Anwar, B. J. Brown, E. T. Campbell, and D. E. Browne, New Journal of Physics **16**, 063038 (2014).
[4] A. Hutter, D. Loss, and J. R. Wootton, New Journal of Physics **17**, 035017 (2015).
[5] F. H. E. Watson, E. T. Campbell, H. Anwar, and D. E. Browne, Phys. Rev. A **92**, 022312 (2015).
[6] F. H. E. Watson, H. Anwar, and D. E. Browne, Phys. Rev. A **92**, 032309 (2015).

[7] H. Anwar, E. T. Campbell, and D. E. Browne, New Journal of Physics **14**, 063006 (2012).
[8] E. T. Campbell, H. Anwar, and D. E. Browne, Phys. Rev. X **2**, 041021 (2012).
[9] E. T. Campbell, Phys. Rev. Lett. **113**, 230501 (2014).
[10] J. Haah, M. B. Hastings, D. Poulin, and D. Wecker, Quantum **1**, 31 (2017).
[11] A. Krishna and J.-P. Tillich, arXiv preprint arXiv:1811.08461 (2018).
[12] F. S. Khan and M. Perkowski, arXiv preprint quant-ph/0511041 (2005).
[13] A. Bocharov, M. Roetteler, and K. M. Svore, Physical Review A **96**, 012306 (2017).
[14] K. Matsumoto and K. Amano, Pre-print arXiv:0806.3834 (2008).
[15] B. Giles and P. Selinger, Pre-print arXiv:1312.6584 (2013).
[16] V. Kliuchnikov, D. Maslov, and M. Mosca, Quantum Info. Comput. **13**, 607 (2013).
[17] D. Gosset, V. Kliuchnikov, M. Mosca, and V. Russo, Quantum Info. Comput. **14**, 1261 (2014).
[18] M. Amy, D. Maslov, and M. Mosca, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **33**, 1476 (2014).
[19] M. Amy and M. Mosca, Pre-print arXiv:1601.07363 (2016).
[20] E. T. Campbell and M. Howard, Phys. Rev. A **95**, 022316 (2017).
[21] L. E. Heyfron and E. T. Campbell, Quantum Science and Technology **4**, 015004 (2019).
[22] Y. Nam, N. J. Ross, Y. Su, A. Childs, and D. Maslov, npj Quantum Information **4** (2017), 10.1038/s41534-018-0072-4.
[23] A. N. Glaudell, N. J. Ross, and J. M. Taylor, Pre-print arXiv:1803.05047 (2018).
[24] C. Jones, Physical Review A **87**, 022328 (2013).
[25] M. Howard and J. Vala, Physical Review A **86**, 022316 (2012).
[26] S. X. Cui, D. Gottesman, and A. Krishna, Phys. Rev. A **95**, 012329 (2017).
[27] A. G. Fowler, A. C. Whiteside, and L. C. Hollenberg, Physical review letters **108**, 180501 (2012).
[28] J. O'Gorman and E. T. Campbell, Physical Review A **95**, 032338 (2017).
[29] R. Babbush, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, A. Paler, A. Fowler, and H. Neven, arXiv preprint arXiv:1805.03662 (2018).
[30] Note that for notational convenience, we often write an implementation as a single matrix where λ is the final row and the rest is the A matrix with a separating horizontal line between them.
[31] Execution times were obtained on a laptop with an Intel Core i7 2.40GHz processor, 12GB of RAM running Microsoft Windows 10 Home edition.

Chapter 6

Conclusion

In this thesis we have investigated quantum compilation methods that reduce the cost of executing quantum algorithms in the context of fault-tolerant quantum computing. In particular, we have developed methods for minimizing the number of expensive non-Clifford gates in quantum circuits. We have done this in two distinct cases: for qubit-based and qudit-based architectures. In the case of qubit-based architectures, the non-Clifford gate under investigation was the T-gate and in the qudits case, it was the M gate. In each case, we developed general purpose multi-qubit/qudit compilers that take as input quantum circuits that are exactly synthesisable on the native gate set and return specific circuit decompositions with non-Clifford gate counts that are less than or equal to that of the input.

It is known that the problem of decomposing a general multi-qubit/qudit unitary into a circuit that is optimal with respect to the non-Clifford gate count is intractable. As such, an object of this thesis has been to discover heuristics that are favourable in both the non-Clifford gate count reductions and the asymptotic behaviour of classical compiler execution times. In the qubit case, we were able to devise a compiler which successfully yields reduced T counts when tested on standard benchmark circuits and executes in time that is polynomial in the number of qubits and the provisional T count of the input circuit. This has led to implementations of useful quantum algorithms with lower T counts than any previous effort. The compiler has been implemented as a command-line utility called *TOpt* that was written in C++ whose source code is freely available under an open source GNU license. Several subsequent works have already been built following on from our compiling approach that have used *TOpt* as a subroutine, including references [44, 45]. Both of these works make use of the ZX-calculus to overcome or bypass the obstacle that Hadamard gates pose to phase-polynomial style T count optimizers, while reducing the T count by eliminating or fusing "phase-gadgets" that correspond to the columns of a gate synthesis matrix such as A in equation (8) of chapter 4. Both works use the TODD algorithm as a post-processing stage after having isolated CNOT + T subcircuits from the input Clifford + T circuit and in many cases, TODD successfully managed to reduce the T count below that of the reported results without TODD. The "PyZX" compiler in reference [44] focussed specifically on ancilla-free optimization and in this special case, many circuits produced by PyZX combined with TODD have lower T counts than those produced by any previous compiler, including TOpt. Since then, the compiler from reference [45] has managed to reduce the T count for some circuits (notably the GF(2^n)-mult family) even further, while TODD with fully gadgetized Hadamard gates still holds the record for lowest T count for many other benchmark circuits. Hopefully, the availability of the code will lead to further usage of the quantum compiler in research projects and to continued development of the compiler itself.

Although the T count optimization component of the compiler runs in time polynomial in the size of the circuit, we found that in practice, the order of the polynomial is prohibitively large for very large circuits. This meant we were unable to obtain results for the largest of the standard benchmark circuits and the highest execution times for completed runs were in the order of days. We have demonstrated that partitions can be used to mitigate this problem at the cost of T count optimization performance. However, we emphasise the pertinence of a more in-depth exploration into the trade-off between classical compile time and magic state resource savings. In this vein, we suggest that an area of future work could be to devise algorithms for automating the partitioning of quantum circuits into pareto optimal partitions with respect to both classical compile time and T count. The T optimization subroutine (e.g. TOOL, TODD or even some as-yet undiscovered heuristic) would be a parameter of such an investigation.

In the case of qudits, we were able to obtain a unitary quantum circuit for the Toffoli gate that uses only 4 M gates, which is an improvement over the best previous unitary implementation that uses 7 M gates. As well as offering immediate cost savings for many quantum algorithms, this discovery reveals a distinct advantage of using a qudit based quantum computing paradigm. This is because the 4 M gate circuit is optimal for d-level qudits where d is a prime number greater than 3, but by comparison, the optimal unitary T count for the qubit-based Toffoli gate is 7. It remains an area of further investigation to discover other such 'super-optimal' qudit circuits and to quantify the relative compilation efficiency of different qudit paradigms over varying values of d. It should be noted that qubit-based Toffoli gates can be synthesised using only 4 T gates through non-unitary methods that involve ancilla and measurements [46]. However, methods for incorporating non-unitary processes into a general framework for quantum compilation remains poorly understood and open to future development. The aforementioned 4 M gate unitary implementation of the qudit-based Toffoli gate raises the question as to whether the dependence of the optimal T (or M) count on the inclusion of non-unitary gates is a property exclusive to the qubit paradigm, or at least a property that is absent for qudits of prime dimension greater than 3. Such a question is open and subject to further study. Despite these successes, we were unable to devise a qudit compiler which reduces the non-Clifford gate count that is computationally efficient, so this also remains an area of further study.

It is unclear at this stage whether qudits or qubits will become the dominant paradigm for the first generation of quantum computers, but hopefully the quantum circuits and optimization methods presented in this thesis will take us one step closer to witnessing the outcome first-hand.

Bibliography

[1] Further references can be found on page 20 of Chapter 4 and page 13 of Chapter 5.

[2] Peter Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Rev.* 41.2 (1999), pp. 303–332.

[3] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-key Cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: http://doi.acm.org.sheffield.idm.oclc.org/10.1145/359340.359342.

[4] Daniel Gottesman. "Stabilizer Codes and Quantum Error Correction". PhD thesis. Pasadena, California: California Institute of Technology, 1997.

[5] Emanuel Knill, Raymond Laflamme, and Wojciech H. Zurek. "Resilient Quantum Computation". In: *Science* 279.5349 (1998), pp. 342–345.

[6] Daniel Gottesman. "Theory of fault-tolerant quantum computation". In: *Physical Review A* 57.1 (1997).

[7] D.S. Wang et al. "Threshold Error Rates for the Toric and Surface Codes". In: *Quant. Inf. Comput.* 10.5 & 6 (2010), pp. 456–469. DOI: 10.26421/QIC10.5-6-6. URL: http://www.rintonpress.com/xxqic10/qic-10-56/0456-0469.pdf.

[8] Eric Dennis et al. "Topological Quantum Memory". In: *Journal of Mathematical Physics* 43.4452 (2002).

[9] Daniel Gottesman and Isaac L. Chuang. "Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations". In: *Nature* 402 (1999), pp. 390–393.

[10] Sergey Bravyi and Alexei Kitaev. "Universal quantum computation with ideal Clifford gates and noisy ancillas". In: *Physical Review A* 71.022316 (2005).

[11] Sergey Bravyi and Jeongwan Haah. "Magic-state distillation with low overhead". In: *Physical Review A* 86.052329 (2012).

[12] Joe O'Gorman and Earl T. Campbell. "Quantum computation with realistic magic-state factories". In: *Phys. Rev. A* 95 (3 2017), p. 032338. DOI: 10.1103/PhysRevA.95.032338. URL: https://link.aps.org/doi/10.1103/PhysRevA.95.032338.

[13] R Raussendorf, J Harrington, and K Goyal. "Topological fault-tolerance in cluster state quantum computation". In: *New Journal of Physics* 9.6 (2007), pp. 199–199. DOI: 10.1088/1367-2630/9/6/199.

[14] Austin G. Fowler, Simon J. Devitt, and Cody Jones. "Surface code implementation of block code state distillation". In: *Scientific Reports* 3.1 (2013). DOI: `10.1038/srep01939`.

[15] Austin G. Fowler and Simon J. Devitt. *A bridge to lower overhead quantum computation.* 2013. arXiv: `1209.0510 [quant-ph]`.

[16] M. Amy, D. Maslov, and M. Mosca. "Polynomial-Time T-Depth Optimization of Clifford+T Circuits Via Matroid Partitioning". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.10 (2014), pp. 1476–1489. DOI: `10.1109/TCAD.2014.2341953`.

[17] M. Amy and M. Mosca. "T-Count Optimization and Reed–Muller Codes". In: *IEEE Transactions on Information Theory* 65.8 (2019), pp. 4771–4784. DOI: `10.1109/TIT.2019.2906374`.

[18] Abraham Lempel. "Matrix Factorization over GF(2) and Trace-Orthogonal Bases of GF(2^n)". In: *SIAM J. Comput.* 4.2 (1975), pp. 175–186.

[19] Earl T Campbell and Mark Howard. "Unifying Gate Synthesis and Magic State Distillation". In: *Phys. Rev. Lett.* 118.060501 (2017).

[20] Christopher Chamberland et al. *Building a fault-tolerant quantum computer using concatenated cat codes.* 2020. arXiv: `2012.04108 [quant-ph]`.

[21] Sergey Bravyi, Graeme Smith, and John A. Smolin. "Trading Classical and Quantum Computational Resources". In: *Physical Review X* 6.2 (June 2016). ISSN: 2160-3308. DOI: `10.1103/physrevx.6.021043`. URL: `http://dx.doi.org/10.1103/PhysRevX.6.021043`.

[22] Daniel Litinski. "A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery". In: *Quantum* 3 (Mar. 2019), p. 128. ISSN: 2521-327X. DOI: `10.22331/q-2019-03-05-128`. URL: `http://dx.doi.org/10.22331/q-2019-03-05-128`.

[23] Daniel Litinski. "Magic State Distillation: Not as Costly as You Think". In: *Quantum* 3 (Dec. 2019), p. 205. ISSN: 2521-327X. DOI: `10.22331/q-2019-12-02-205`. URL: `http://dx.doi.org/10.22331/q-2019-12-02-205`.

[24] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. "Fast and Efficient Exact Synthesis of Single-Qubit Unitaries Generated by Clifford and T Gates". In: *Quantum Info. Comput.* 13.7–8 (July 2013), pp. 607–630. ISSN: 1533-7146.

[25] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. "Asymptotically Optimal Approximation of Single Qubit Unitaries by Clifford andTCircuits Using a Constant Number of Ancillary Qubits". In: *Physical Review Letters* 110.19 (May 2013). ISSN: 1079-7114. DOI: `10.1103/physrevlett.110.190502`. URL: `http://dx.doi.org/10.1103/PhysRevLett.110.190502`.

[26] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. "Practical Approximation of Single-Qubit Unitaries by Single-Qubit Quantum Clifford and T Circuits". In: *IEEE Transactions on Computers* 65.1 (Jan. 2016), pp. 161–172. ISSN: 0018-9340. DOI: 10.1109/tc.2015.2409842. URL: http://dx.doi.org/10.1109/TC.2015.2409842.

[27] Neil J. Ross and Peter Selinger. "Optimal ancilla-free Clifford+T approximation of z-rotations". In: *Quantum Inf. Comput.* 16.11&12 (2016), pp. 901–953. URL: http://www.rintonpress.com/xxqic16/qic-16-1112/0901-0953.pdf.

[28] M. Amy et al. "A Meet-in-the-Middle Algorithm for Fast Synthesis of Depth-Optimal Quantum Circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32.6 (2013), pp. 818–830. DOI: 10.1109/TCAD.2013.2244643.

[29] Earl T Campbell and Mark Howard. "Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost". In: *Phys. Rev. A* 95.022316 (2017).

[30] Luke E Heyfron and Earl T Campbell. "An efficient quantum compiler that reduces T count". In: *Quantum Science and Technology* 4.1 (2018), p. 015004. DOI: 10.1088/2058-9565/aad604. URL: https://doi.org/10.1088%2F2058-9565%2Faad604.

[31] David Gosset et al. "An algorithm for the T-count". In: *Quantum Information & Computation* 14 (Nov. 2014), pp. 1261–1276. URL: https://www.microsoft.com/en-us/research/publication/an-algorithm-for-the-t-count/.

[32] A Yu Kitaev. "Quantum computations: algorithms and error correction". In: *Russian Mathematical Surveys* 52.6 (1997), pp. 1191–1249. DOI: 10.1070/rm1997v052n06abeh002155. URL: https://doi.org/10.1070%2Frm1997v052n06abeh002155.

[33] Christopher M. Dawson and Michael A. Nielsen. "The Solovay-Kitaev Algorithm". In: *Quantum Info. Comput.* 6.1 (2006), pp. 81–95. ISSN: 1533-7146.

[34] Adriano Barenco et al. "Elementary gates for quantum computation". In: *Phys. Rev. A* 52 (5 Nov. 1995), pp. 3457–3467. DOI: 10.1103/PhysRevA.52.3457. URL: https://link.aps.org/doi/10.1103/PhysRevA.52.3457.

[35] W. K. Wootters and W. H. Zurek. "A single quantum cannot be cloned". In: *Nature* 299.5886 (1982), pp. 802–803. DOI: 10.1038/299802a0.

[36] Andrew Steane. "Multiple-particle interference and quantum error correction". In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 452.1954 (1996), pp. 2551–2577. DOI: 10.1098/rspa.1996.0136. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1996.0136.

[37] A. M. Steane. "Error Correcting Codes in Quantum Theory". In: *Phys. Rev. Lett.* 77 (5 1996), pp. 793–797. DOI: `10.1103/PhysRevLett.77.793`. URL: `https://link.aps.org/doi/10.1103/PhysRevLett.77.793`.

[38] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* 10th. New York, NY, USA: Cambridge University Press, 2011. ISBN: 1107002176, 9781107002173.

[39] Benjamin J. Brown et al. "Poking Holes and Cutting Corners to Achieve Clifford Gates with the Surface Code". In: *Phys. Rev. X* 7 (2 2017), p. 021029. DOI: `10.1103/PhysRevX.7.021029`. URL: `https://link.aps.org/doi/10.1103/PhysRevX.7.021029`.

[40] Sergey Bravyi and Robert König. "Classification of Topologically Protected Gates for Local Stabilizer Codes". In: *Physical Review Letters* 110.17 (2013). DOI: `10.1103/physrevlett.110.170503`.

[41] Bryan Eastin and Emanuel Knill. "Restrictions on Transversal Encoded Quantum Gate Sets". In: *Physical Review Letters* 102.11 (2009). DOI: `10.1103/physrevlett.102.110502`.

[42] Luke Heyfron. *Source code for "TOpt"*. DOI: `10.5281/zenodo.1345084`.

[43] Luke Heyfron and Earl T. Campbell. "A quantum compiler for qudits of prime dimension greater than 3". In: *arXiv* arXiv:1902.05634 [quant-ph] (2019). URL: `https://arxiv.org/abs/1902.05634`.

[44] Aleks Kissinger and John van de Wetering. "Reducing the number of non-Clifford gates in quantum circuits". In: *Phys. Rev. A* 102 (2 Aug. 2020), p. 022406. DOI: `10.1103/PhysRevA.102.022406`. URL: `https://link.aps.org/doi/10.1103/PhysRevA.102.022406`.

[45] Niel de Beaudrap, Xiaoning Bian, and Quanlong Wang. "Techniques to Reduce $\pi/4$-Parity-Phase Circuits, Motivated by the ZX Calculus". In: *Electronic Proceedings in Theoretical Computer Science* 318 (May 2020), pp. 131–149. ISSN: 2075-2180. DOI: `10.4204/eptcs.318.9`. URL: `http://dx.doi.org/10.4204/EPTCS.318.9`.

[46] Cody Jones. "Low-overhead constructions for the fault-tolerant Toffoli gate". In: *Phys. Rev. A* 87.022328 (2013).