An Efficient Quantum Compiler with Near-Optimal T Count

Luke Heyfron and Earl T. Campbell

October 2017

Abstract

In order to perform quantum algorithms on a real quantum computer, one must first decompose the algorithm into machine-level instructions compatible with the architecture of the quantum computer, a process known as quantum compiling. In principle, there are many different quantum circuit decompositions for the same algorithm and due to the expense of running a quantum computer, it is desirable to compile into circuits that minimize space-time cost metrics. A popular measure of cost is the T count, which emerges from the prevalent Clifford + T magic states model of fault-tolerant quantum computing and closely approximates the full space-time cost for high-threshold quantum error correction codes such as the toric code. The problem of optimizing the T count of a quantum circuit, or gate synthesis, is a relatively young area of research but exciting progress has been made, especially for the single qubit case which is essentially a solved problem. However, multi-qubit gate synthesis is believed to be a hard problem with optimal algorithms requiring classical runtime exponential in the number of qubits, n. Conversely, the best known polynomial-time algorithms either solve special cases that restrict us to a non-universal gate set or make no promises with regards to global optimality. In this paper, we present a quantum compiler for multi-qubit Clifford + T quantum circuits that is both fast with runtime polynomial in n and near-optimal in the sense that the output quantum circuit has T count with worst case scaling of $\mathcal{O}(n^2)$, which is the same scaling as that of the optimal inefficient algorithm and smaller than the best previous heuristic by a factor n. We have implemented several variants of this protocol in C++ and benchmark tested them on random circuits, from which we determine that Third Order Duplicate and Destroy (TODD) yields the lowest T counts on average. Also, we ran our compiler on a library of circuits that implement quantum algorithms in order to compare with previous works and found that in each case our protocol generates circuits with lower T counts than the best known previous algorithm, providing solid evidence that our protocol is the forerunner amongst Clifford + T quantum compilers.

1 Introduction

Compiling is the conversion of an algorithm into a series of hardware level commands or elementary gates. A more optimal compiler 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. At the very dawn of quantum computing as a field, Solovay and Kitaev proposed a general purpose compiler that was compatible with any universal set of elementary gates [1, 2, 3]. Unfortunately, it was far from optimal, generating circuits using many more gates than are necessary.

After a long lull, recent years have brought a series of significant improvements in compiling methods. This new generation of compilers exploit the specific structure of the Clifford+T gate set, reducing quantum circuit depths by several orders of magnitude and also often improving the classical compile time [4, 5, 6, 7]. Focus on this gate set is warranted since it naturally appears as the set of logical gates in almost every fault-tolerant computing architecture [8]. Furthermore, proposed fault-tolerant devices often use techniques, such as magic state distillation [9], where the cost per T gate is several hundreds of Clifford gates [10, 11, 12]. This suggests T count as the key metric of compiler performance.

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 optimal and efficient [4, 7], with a command line implementation freely available [13]. Yet further improvements are possible beyond unitary circuits, by making use of ancilla qubits and measurements [14, 15, 16] or adding an element of randomness to compiling [17, 18]. Whereas, 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]. More recently, a highly efficient compiler has been

developed that is effective at reducing not only the T count but also the CNOT and continuous phase gate counts, but it makes no promises regarding global optimality [19]. Instead, we seek a compiler that runs efficiently with circuit size and yields T counts as close to the global minimum as possible, with some prior attempts at this goal [20].

A useful strategy for multi-qubit synthesis is to take an initial Clifford+T circuit and split it into partitions containing Hadamards and partitions containing CNOT, S and T gates. One can then attempt to reduce the number of T gates within just the latter partitions. Amy and Mosca recently shows that this restricted synthesis problem is formally equivalent to error decoding on a class of Reed-Muller codes [21], which is in turn equivalent to finding the symmetric tensor rank of a 3-tensor [22]. Unfortunately, even this easier sub-problem is NP-complete to solve optimally. Nevertheless, this problem does seem more amenable to efficient solvers that offer reductions in T count. Amy and Mosca argued 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 [23] 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 [24] but suitably modified so that it nearly-optimally solves the required 3-tensor problem.

This paper develops several different compilers that are efficient and near-optimal in the above sense. We provide the first implementations of such compilers and provide performance comparison against: a family of random circuits; and a library of circuits taken from actual quantum algorithms. For random circuits, we find the actual performance follows the worst-case scaling for different compilers. We observe $O(n^2)$ scaling for all variants of our compiling approach compared with $O(n^3)$ scaling for compilers based on earlier work. For actual quantum algorithms, the full Clifford+T gate set is needed and so our compiler also makes use of a gadgetisation tricks to eliminate Hadamards and convert the problem to synthesis over just the CNOT, S and T gate set. Quantum algorithms are highly structured and far from random, so the number of T gates can not be meaningfully compared with the worst case n scaling. Instead, we benchmark against the best previously known results. We find our compilers gave significant reductions in T count for almost every circuit tested and never gave worse T counts.

All of the near-optimal compilers described in this paper look for inspiration in algorithms for the related 2-tensor problem, which we call the 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 [23]. Also proposed 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.

2 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 Clifford group on n qubits $\mathcal{C}^n$ is the normalizer of $\mathcal{P}^n$. The k^{th} level of the *Clifford hierarchy* $\mathcal{C}_k^n$ is defined as follows,

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

with recursion terminated by $\mathcal{C}_1^n = \mathcal{P}^n$. We define $\mathcal{D}_k^n$ to be the diagonal elements of $\mathcal{C}_k^n$. 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 $\{CNOT, H, S\}$. We define the CNOT + T gate set to be $\{\text{CNOT}, S, T\}$, 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 $\mathcal{U}$ 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*. We define $\circ$ to be an operator that composes two circuits together in left-to-right time order.

3 Workflow Overview

We now describe the high level work-flow of our T gate optimization protocol. The input and output of the protocol are quantum circuit decompositions that are defined in terms of two distinct qubit registers, labelled x and y. The x (y) register is composed of n (h) qubits and spans the Hilbert space $\mathcal{H}_{\text{x}}$ ($\mathcal{H}_{\text{y}}$). The input circuit, $\mathcal{U}_{\text{in}} \in \langle \text{Clifford}, T \rangle$, implements a unitary $U \in \mathcal{C}_3^n$. The output, $\mathcal{E}_{\text{out}} \in \langle \text{Clifford}, T, M, |+\rangle, \text{feedforward} \rangle$, is a circuit also composed of

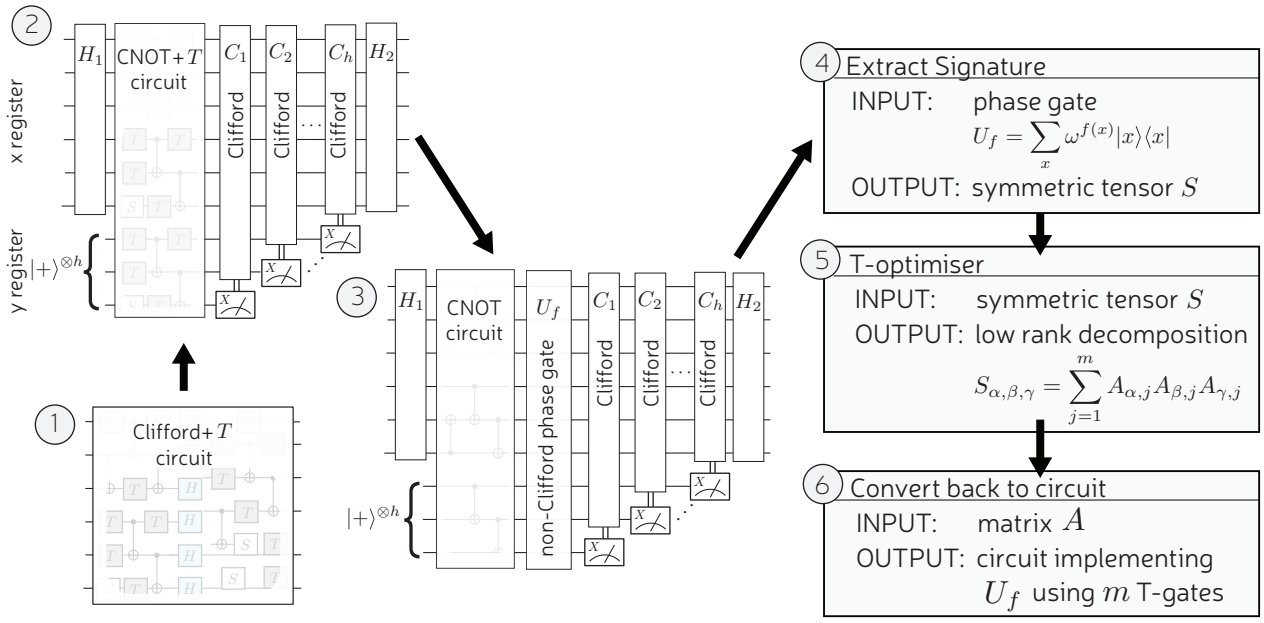


Figure 1: The high level workflow of the T gate optimization protocol is shown. A Clifford + T circuit is converted to the CNOT+T gate set by introducing ancillas and performing classically controlled Clifford gates. A non-Clifford phase gate is extracted, which maps to a signature tensor upon which 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, M, and classical feedforward. $\mathcal{E}_{\text{out}}$ implements the non-unitary quantum operator $\varepsilon_{\text{out}} : \mathcal{H}_{\text{x}} \mapsto \mathcal{H}_{\text{x}} \otimes \mathcal{H}_{\text{y}}$ given by

$$\varepsilon_{\text{out}}(\rho_{\text{x}}) = \varepsilon_{\text{post}}(U_{\text{out}}(\rho_{\text{x}} \otimes \rho_{\text{y}})U_{\text{out}}^{\dagger}), \tag{2}$$

where ρ_{x} is the density matrix for an arbitrary input pure state on $\mathcal{H}_{\text{x}}$, and $\rho_{\text{y}} = (|+\rangle\langle+|)^{\otimes h}$ is a density matrix on $\mathcal{H}_{\text{y}}$. $U_{\text{out}} \in \mathcal{C}_3$ 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, $C_1^{m_1}, C_2^{m_2}, \ldots, C_h^{m_h}$, seen in figure 1. $\mathcal{E}_{\text{out}}$ is synthesized such that, once we trace out the y register from the result of ε_{out} acting on ρ_{x}, we recover the input unitary,

$$\text{Tr}_{\text{y}}(\varepsilon_{\text{out}}(\rho_{\text{x}})) = U\rho_{\text{x}}U^{\dagger}. \tag{3}$$

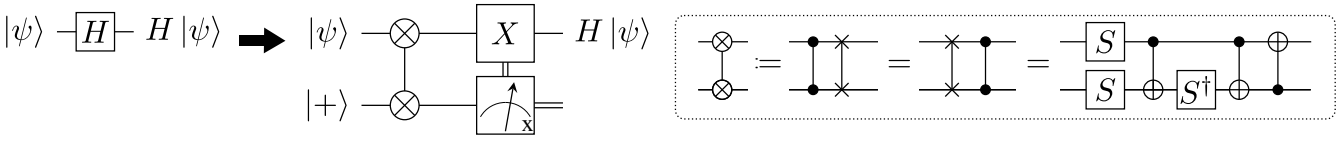
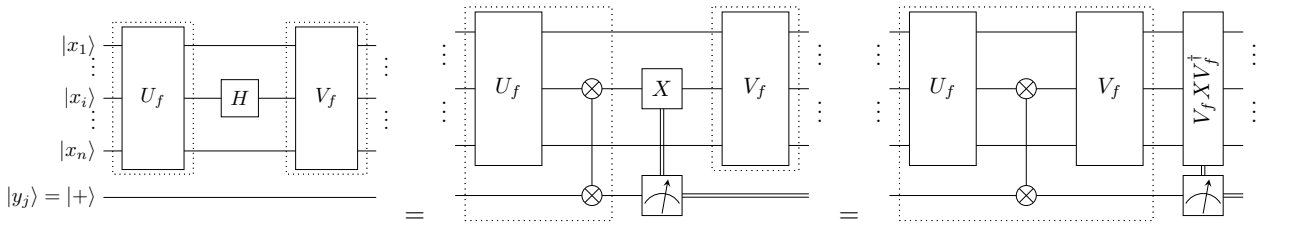


Figure 2: Hadamard gates are replaced by Hadamard-gadgets according to the rewrite rule on the LHS. On the RHS, we define notation for the phase-swap gate and provide an example decomposition into the CNOT + T gate set.

We will describe the process of our protocol in 6 steps. First, we emphasize that *T-Optimiser* makes use of a framework valid only for CNOT + T circuits, which makes Hadamard gates as obstacles. We overcome this in steps 1 and 2 using methods based on reference [25]. First, we remove external Hadamards from $\mathcal{U}_{\text{in}}$ and place them into circuits H_1 and H_2 for those that appear at the beginning and end of the circuit, respectively. Then in step 2, each of the h internal Hadamard gates is replaced by a *Hadamard-gadget* as shown in 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. Each internal Hadamard that appears in $\mathcal{U}_{\text{in}}$ invokes a new Hadamard-ancilla, so the size of the y register is h. After making the initial Hadamard



(4)

Figure 3: Example showing a Hadamard gate swapped for a Hadamard-gadget where the classically controlled Pauli-X gate is commuted through to the end. The CNOT + T -only region increases as shown by the dotted lines. Note that $V_f X V_f^\dagger \in \mathcal{C}_2$ as per equation 1, so has a T-count of 0.

to Hadamard-gadget substitution, 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 figure 3). 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 that of forming Hadamard-bounded partitions as in previous works [26] 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-Optimize* subroutine.

Once the internal Hadamards are removed, we are left with a CNOT + T circuit that implements unitary U_{CNOT+T}, whose action on the computational basis is fully described by two mathematical objects: a *phase function*, $f : \mathbb{Z}_2^n \mapsto \mathbb{Z}_8$, and an invertible matrix $E \in \mathbb{Z}_2^{(n,n)}$, such that

$$U_{CNOT+T} |\mathbf{x}\rangle = \omega^{f(\mathbf{x})} |E\mathbf{x}\rangle \tag{5}$$

where $\omega = e^{i\frac{\pi}{4}}$. It can be shown that a circuit $\mathcal{U}_E$ that implements the transform $U_E |\mathbf{x}\rangle = |E\mathbf{x}\rangle$ needs only the CNOT gate. It is convenient to strip away this Clifford behaviour and focus on *diagonal* CNOT + T circuits that require only the phase function for a full description of the unitary. Hence in step 3, we decompose $\mathcal{U}_{CNOT+T}$ into two partitions: $\mathcal{U}_E$ and $\mathcal{U}_f$, the latter being a diagonal CNOT + T circuit that implements U_f, whose action is given by

$$U_f |\mathbf{x}\rangle = \omega^{f(\mathbf{x})} |\mathbf{x}\rangle . \tag{6}$$

We make use of four different *phase function representations*: *phase polynomials* (PP), *weighted polynomials* (WP), *gate synthesis matrices* (GSM) and *signature tensors* (ST), which are mappable to one another, as well as to and from the unitary and circuit descriptions as shown in figure 4. The mathematical definition of each representation and some of their mappings will be covered in section 4. For now, we will highlight relative merits of each. Phase polynomials map to quantum circuits, where the number of T gates required is a known function of the PP coefficients. There exist many different PPs for the same unitary that in general have different T counts, wherein lies the T optimization problem. On the other hand, weighted polynomials have a one-to-one correspondence with the unitary it implements, a property we exploit to check that our optimization methods leave the input unitary invariant. The signature tensor is a streamlined version of the WP that ignores the Clifford behaviour irrelevant for T optimization. Finally, gate synthesis matrices are streamlined versions of PPs, similarly with the Clifford behaviour ignored. In step 4, we extract the phase function representations of $\mathcal{U}_f$ and store them in memory.

The key subroutine of the algorithm, *T-optimiser*, is performed in step 5. It takes as input the signature tensor, S, of U_f and outputs a gate synthesis matrix, A, with m columns such that $S^{(A)} = S^{(U_f)}$. For T-optimal circuits, the output of *T-optimiser*

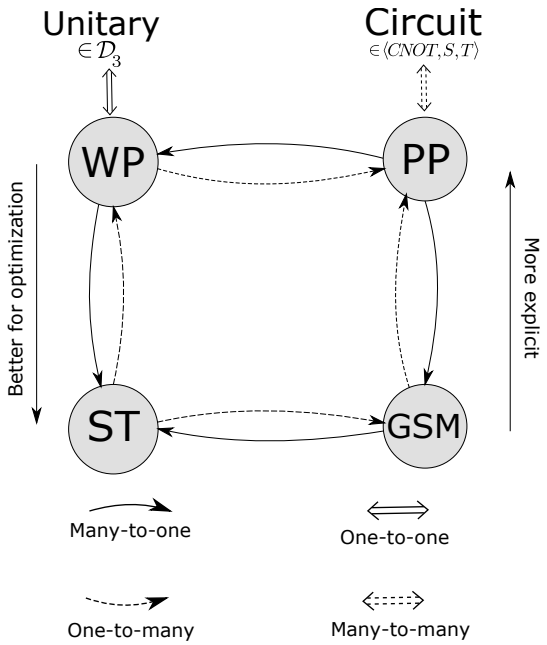


Figure 4: Phase function representations

has minimal m, but finding optimal GSMs is believed to be a hard
problem as we will expand upon in section 5. We have implemented a number of heuristic versions of *T-optimiser* that yield GSMs with near-minimal m, which are described in detail in section 5.

In the final step, we map the output GSM of *T-optimiser* to a diagonal CNOT + T circuit, $\mathcal{U}_{f'}$, that comprises m instances of the T gate using lemma 4.1. The circuit $\mathcal{U}_{f'}$ implements a unitary $U_{f'} = U_f U_{\text{Clifford}}$, where U_{Clifford} is a diagonal Clifford factor. The input WP stored since step 4 contains sufficient information to generate a circuit for $U^\dagger_{\text{Clifford}}$ (see appendix B), hence we recover the original unitary, $U_f = U_{f'} U^\dagger_{\text{Clifford}}$. The final part of step 6 constitutes replacing $\mathcal{U}_f$ with $(\mathcal{U}^\dagger_{\text{Clifford}} \circ \mathcal{U}_{f'})$. At this stage, the protocol terminates returning the final output, $\mathcal{E}_{\text{out}} = (H_1 \circ \mathcal{U}_E \circ \mathcal{U}^\dagger_{\text{Clifford}} \circ \mathcal{U}_{f'} \circ \mathcal{E}_{\text{post}} \circ H_2)$, which has near-minimal T count.

4 Diagonal CNOT+T Framework

In section 3, we described how one can isolate 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 4.1. ***(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 abstract mathematical picture involving binary tensors. We proceed by recalling from equation 6 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:

$$f(\mathbf{x}) = \sum_{\alpha=1}^{n} l_\alpha x_\alpha + 2\sum_{\alpha<\beta}^{n} q_{\alpha,\beta} x_\alpha x_\beta + 4\sum_{\alpha<\beta<\gamma}^{n} c_{\alpha,\beta,\gamma} x_\alpha x_\beta x_\gamma \pmod 8, \tag{7}$$

where $l_\alpha, q_{\alpha,\beta}, c_{\alpha,\beta,\gamma} \in \mathbb{Z}_8$. As in reference [27], we refer to decompositions of f that take the form of equation 7 as *weighted polynomials*. It was proven that $U_{2f} = U_f^2 \in \mathcal{C}_2$ for any WP, f. This implies that any two unitaries with weighted polynomials whose coefficients all have the same parity are Clifford equivalent. Note 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.

We define the *signature tensor*, $S^{(U_f)} \in \mathbb{Z}_2^{(n,n,n)}$, of U_f 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:

$$S_{\sigma(\alpha,\alpha,\alpha)} = S_{a,a,a} = l_\alpha \pmod 2 \tag{8a}$$
$$S_{\sigma(\alpha,\beta,\beta)} = S_{\sigma(\alpha,\alpha,\beta)} = q_{\alpha,\beta} \pmod 2 \tag{8b}$$
$$S_{\sigma(\alpha,\beta,\gamma)} = c_{\alpha,\beta,\gamma} \pmod 2 \tag{8c}$$

for all $\sigma \in \mathcal{S}_3$, the symmetric group on a set with 3 elements. It follows from this definition that any two unitaries with the same signature tensor are Clifford equivalent.

We recall the definition of gate synthesis matrices from reference [27], where a matrix, A in $\mathbb{Z}_2^{(n,m)}$, is a gate synthesis matrix for a unitary U_f if it satisfies,

$$f(\mathbf{x}) = |A^T \mathbf{x}| \pmod 8 \tag{9}$$
$$= \sum_{j=1}^{m} (A_{1,j} x_1 \oplus A_{2,j} x_2 \oplus \cdots \oplus A_{n,j} x_n) \pmod 8 \tag{10}$$

where f is the phase function of U_f and $|.|$ is the Hamming weight of a binary vector. An A matrix can be efficiently extracted from a diagonal CNOT + T circuit by tracking the action of each gate on the computational basis states through the circuit. The signature tensor of U_f can be determined from an A matrix of U_f using the following relation,

$$S^{(A)}_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} A_{\alpha,j} A_{\beta,j} A_{\gamma,j} \pmod 2 \tag{11}$$

In our gate synthesis protocol, we exploit a key property of A matrices described in the following lemma.

Lemma 4.1. *Given an A matrix composed of m unique, non-zero column vectors each of length n, one can efficiently generate a circuit that implements U_f with m uses of the T gate.*

Proof. First, we note from the definition of A in equation 9 that each 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 6, where λ_j is a reversible linear Boolean function given by,

$$\lambda_j(\mathbf{x}) = A_{1,j}x_1 \oplus A_{2,j}x_2 \oplus \cdots \oplus A_{n,j}x_n. \tag{12}$$

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 $x_1, x_2, \ldots, x_n$. Therefore, each λ_j can be implemented using a subcircuit B_j composed of only CNOT gates. We define B_j such that, after applying B_j to the input computational basis state $|\mathbf{x}\rangle$, the i^{th} qubit is in state $|\lambda_j(\mathbf{x})\rangle$. A T gate subsequently applied to qubit i results in the desired phase of $\omega^{\lambda_j(\mathbf{x})}$. We now uncompute B_j by reversing the order of the CNOT gates and applying the resultant $B_j^\dagger$ to the CBS, and so column j has been implemented. We then move on the next column j' until all columns of A have been implemented in this way. This method of generating U_f from A requires steps $\mathcal{O}(\text{poly}(n, m))$ and therefore is efficient. □

We now have the necessary tools to define the gate synthesis problem in the binary tensor picture.

Problem 4.2. ***(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.*

5 *T-optimiser*

Until now the *T-optimiser* 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 will describe the inner workings of the various *T-optimiser*s we have implemented in this work. In reference [28], Amy and Mosca proved that the T-OPT problem is equivalent to minimum distance decoding of the punctured Reed-Muller code of order $n-4$ and length n (often written as $RM^*(n-4, n)$), which is believed to be a hard problem. This imposes a practical upper bound on the number of qubits, n_{RM}, over which circuits can be optimally synthesized with respect to T count.

5.1 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 $n_{\text{RM}} = 6$. 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 $N_{\text{search}} = 2^{N_G}$. On a processor with a clock speed of 3.20GHz, generously assuming we can check one codeword per clock cycle, it would take over 91 years to exhaustively search this space and therefore determine optimality for a general CNOT+T circuit 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.

5.2 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 $2ab = a + b - a \oplus b$. This is applied recursively to each non-linear term in equation 7 of the weighted polynomial associated with S to yield a decomposition of the form in equation 9. Or explicitly, 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:

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

$$2x_\alpha x_\beta \to x_\alpha + x_\beta - x_\alpha \oplus x_\beta, \tag{14}$$

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

from which the corresponding A matrix can be easily extracted. We call this the *recursive expansion* (RE) algorithm, which has been shown to yield worst-case T counts of $\mathcal{O}(n^3)$. It is straightforward to see this, as any proper

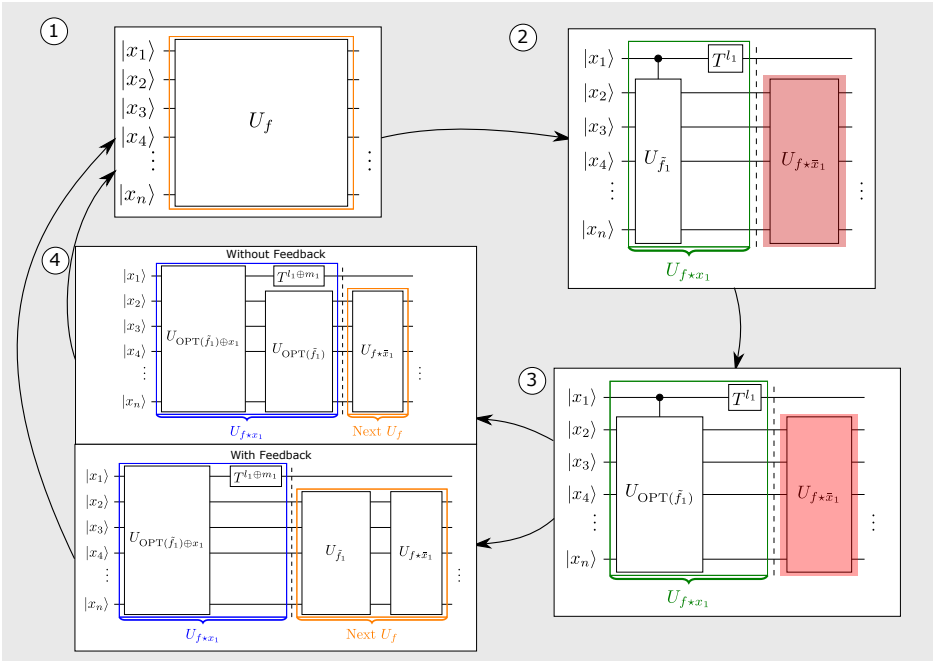


Figure 5: The structure of the TOR algorithm is shown in the quantum circuit picture. The colours of each box have the following meanings: orange border is for subcircuits yet to be pre-processed before the start of optimization and synthesis steps; green border shows subcircuits that are actively being optimized synthesized; blue border is for synthesized subcircuits i.e. they are now decomposed into a sequence of CNOT and T gates; and red fill are for subcircuits that not currently being processed.

gate synthesis matrix resulting from the RE algorithm may include any column with a Hamming weight of $W \in [1, 3]$. There are $\sum_{k=1}^{3} \binom{n}{k} = \mathcal{O}(n^3)$ such columns so from lemma 4.1 there can be at most $\mathcal{O}(n^3)$ T gates in the corresponding circuit decomposition.

5.3 Target Optimal by Order Lowering (TOOL)

An efficient heuristic that requires at most only $\mathcal{O}(n^2)$ T gates was proposed in reference [27], which involves breaking down an input CNOT + T circuit into a sequence of control-U operators where each successive operator acts on fewer qubits than the previous, eventually allowing the optimal Reed-Muller decoder to be used when $n \leq n_{RM}$ for a particular control-U. Each target U corresponds to a weighted polynomial with only up to quadratic terms, and the control qubit maps to a particular variable in the weighted polynomial that has been factorized out from all terms in which it appears. The reduction in order from a cubic to a quadratic weighted polynomial allows for an optimal decomposition U' for each target U to be found efficiently using Lempel's matrix factoring algorithm [29]. Additional steps are required to obtain a decomposition of the form of equation 9 from the control-U' sequence, which leads to sub-optimality in the general case but with T count scaling of $\mathcal{O}(n^2)$, a factor n improvement over RE. This algorithm we refer to as the *Target Optimal by Order Lowering* (TOOL) algorithm efficiently yields optimal T counts for the special case where every gate in the CNOT + T circuit shares a control.

There are two versions of TOOL: with and without feedback. This refers to whether or not extra terms that do not depend on the control qubit of the current iteration are immediately synthesized after calling Lempel's algorithm or if they are fed back into the input signature tensor to be synthesized in the subsequent step.

We now describe a single round of optimization according to the TOOL algorithm. We begin by showing that given some input unitary $U_f \in \mathcal{D}_3$, we can decompose U_f according to panel 2 of figure 5. Recall the definition of the weighted polynomial from equation 7,

$$f(\mathbf{x}) = \sum_{\alpha=1}^{n} l_\alpha x_\alpha + 2 \sum_{\alpha<\beta}^{n} q_{\alpha,\beta} x_\alpha x_\beta + 4 \sum_{\alpha<\beta<\gamma}^{n} c_{\alpha,\beta,\gamma} x_\alpha x_\beta x_\gamma \pmod 8 \tag{16}$$

Now, we group terms in the following way,

$$f(\mathbf{x}) = 2x_h \tilde{f}_h(\mathbf{x} \setminus x_h) + l_h x_h + f'_h(\mathbf{x} \setminus x_h) \pmod 8, \tag{17}$$

where we have defined the following:

Definition 5.1. Target Polynomial. Let $f(\mathbf{x})$ be a weighted polynomial on n qubits. We say $\tilde{f}_h(\mathbf{x} \setminus x_h)$ is the target polynomial of f with respect to x_h defined such that

$$\tilde{f}_h = \frac{f_h - f'_h - l_h}{2}, \tag{18}$$

where f_h and f'_h are the positive and negative Shannon cofactors of f with respect to x_h.

The first term in equation 17 corresponds to the control-$U_{\tilde{f}_h}$ in panel 2 of figure 5, where in our example circuit $h = 1$. The second term corresponds to the T^{l_h} gate. The final term corresponds to the only operator to the right of the dotted line, $U_{f'_h}$, which acts on $n-1$ qubits. The key part of this construction is that by factoring out $2x_h$ from a portion of f, we obtain a lower order target polynomial, $\tilde{f}_h$, that is only defined up to quadratic terms, or explicitly:

$$\tilde{f}_h(\mathbf{x}) = \sum_{\alpha=1,\alpha\neq h}^{n} q_{\alpha,h} x_\alpha + 2 \sum_{\alpha<\beta,\beta\neq h}^{n} c_{\alpha,\beta,h} x_\alpha x_\beta \pmod 8. \tag{19}$$

Operators of the form control-$U_{\tilde{f}_h}$ are uniquely defined (up to Clifford equivalence) by a signature tensor of order 2, rather than order 3 as in the TODD algorithm (see section 5.4). For a given target polynomial, $\tilde{f}_h$, we define the order 2 signature tensor (in a way analogous to equation 8) in terms of the coefficients of the original weighted polynomial, f,

$$S_{\alpha,\alpha} = q_{a,h} \pmod 2 \tag{20}$$

$$S_{\alpha,\beta} = S_{\beta,\alpha} = c_{\alpha,\beta,h} \pmod 2, \tag{21}$$

which can be determined from a gate synthesis matrix, $A \in \mathbb{Z}_2^{(n,m)}$, that implements $\tilde{f}_h$,

$$S_{\alpha,\beta} = \sum_{j=1}^{m} A_{\alpha,j} A_{\beta,j} \pmod 2. \tag{22}$$

The optimization step between panels 2 and 3 of figure 5 is to find an A matrix with minimal columns that satisfies equation 22.

Problem 5.1. ***(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 equation 22 with minimal m.*

It turns out that this problem is exactly equivalent to the matrix factoring problem solved by Abraham Lempel in reference [29]. In an effort to make this paper more self-contained, we provide a description of Lempel's algorithm in appendix A. For now, we treat Lempel's solver as a black box $OPT(\tilde{f})$ that outputs an optimized decomposition for the input target polynomial $\tilde{f}$, as can be seen in panel 3 of figure 5.

5.4 Third Order Duplicate and Destroy (TODD)

In this section, we present an algorithm based in principle on Lempel's matrix factoring algorithm [29] that is extended to work for tensors of order 3. 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 6, we present numerical evidence that it is the best efficient solver of the 3-STR problem developed so far. We call this the *Third Order Duplicate and Destroy* (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.

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 which can be done without changing the signature tensor, as shown in the following lemma.

Lemma 5.1. *Let $A \in \mathbb{Z}^{(n,m)}$ be a gate synthesis matrix whose a^{th} and b^{th} columns are duplicates and $A' \in \mathbb{Z}^{(n,m-2)}$ be a gate synthesis matrix formed from by removing the a^{th} and b^{th} columns of A. It follows that $S^{(A)} = S^{(A')}$ for any such A and A'.*

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

$$S^{(A)}_{\alpha,\beta,\gamma} = \sum_{k=1}^{m} A_{\alpha,k} A_{\beta,k} A_{\gamma,k} \pmod 2, \tag{23}$$

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

$$S^{(A)}_{\alpha,\beta,\gamma} = \sum_{j\in\mathcal{J}} A_{\alpha,j} A_{\beta,j} A_{\gamma,j} + A_{\alpha,a} A_{\beta,a} A_{\gamma,a} + A_{\alpha,b} A_{\beta,b} A_{\gamma,b} \pmod 2, \tag{24}$$

where $\mathcal{J} = [1, m] \setminus \{a, b\}$. As stated in the lemma, the a^{th} and b^{th} columns of A are duplicates and so

$$A_{i,a} = A_{i,b} \;\forall\; i \in [1, n]\,. \tag{25}$$

Now substitute equation 25 into equation 24,

$$\begin{aligned} S^{(A)}_{\alpha,\beta,\gamma} &= \sum_{j\in\mathcal{J}} A_{\alpha,j} A_{\beta,j} A_{\gamma,j} + A_{\alpha,a} A_{\beta,a} A_{\gamma,a} + A_{\alpha,a} A_{\beta,a} A_{\gamma,a} \pmod 2 &(26)\\ &= \sum_{j\in\mathcal{J}} A_{\alpha,j} A_{\beta,j} A_{\gamma,j} \pmod 2 &(27)\\ &= S^{(A')}_{\alpha,\beta,\gamma}, &(28)\end{aligned}$$

where the penultimate step follows from modulo 2 addition and the final step follows from the definition of A' and equation 11. □

Lemma 5.1 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 perform some transformation on the A matrix, which results in a different gate synthesis matrix A' that: a) has duplicate columns; b) preserves the signature tensor of A; and c) does not increase the column number of A by more than 1. In the following lemma we introduce a class of transformations that *duplicate* a particular column of an A matrix such that properties a) and c) are met. We then use lemma 5.3 establish what conditions must be satisfied for the duplication transformation to have property b).

Lemma 5.2. *Let A be an $n \times m$ gate synthesis matrix where all columns are unique and non-zero. Let $A' = A \oplus \mathbf{z}\mathbf{y}^T$ where $\mathbf{z}$ and $\mathbf{y}$ are column vectors on $\mathbb{Z}_2$ of length n and m, respectively, defined such that $z_i = A_{i,a} \oplus A_{i,b}$ for some $a, b \in [1, m]$ and $y_a \oplus y_b = 1$. It follows that the a^{th} and b^{th} columns of A' are duplicates.*

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

$$A'_{i,j} = A_{i,j} \oplus z_i y_j, \tag{29}$$

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

$$A'_{i,j} = A_{i,j} \oplus (A_{i,a} \oplus A_{i,b}) y_j. \tag{30}$$

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

$$A'_{i,a} = A_{i,a} \oplus (A_{i,a} \oplus A_{i,b}) y_a, \tag{31}$$

$$A'_{i,b} = A_{i,b} \oplus (A_{i,a} \oplus A_{i,b}) y_b. \tag{32}$$

We substitute in the condition $y_b = y_a \oplus 1$ into 32,

$$\begin{aligned} A'_{i,b} &= A_{i,b} \oplus (A_{i,a} \oplus A_{i,b})(y_a \oplus 1)\\ &= A_{i,a} \oplus (A_{i,a} \oplus A_{i,b}) y_a &(33)\\ &= A'_{i,a}. \end{aligned}$$

□

Lemma 5.3. *Let A and $A' = A \oplus \mathbf{z}\mathbf{y}^T$ be two gate synthesis matrices where $\mathbf{z}$, $\mathbf{y}$ are arbitrary column vectors of dimension n and m, respectively. $S^{(A)} = S^{(A')}$ as long as 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:*

Definition 5.2. χ **Matrix.** Given some gate synthesis matrix, A, and a column vector $\mathbf{z} \in \mathbb{Z}_2^n$ let

$$\chi(A, \mathbf{z}) = \begin{pmatrix} (z_1\mathbf{r_2} \wedge \mathbf{r_3}) \oplus (z_2\mathbf{r_3} \wedge \mathbf{r_1}) \oplus (z_3\mathbf{r_1} \wedge \mathbf{r_2}) \\ (z_1\mathbf{r_2} \wedge \mathbf{r_4}) \oplus (z_2\mathbf{r_4} \wedge \mathbf{r_1}) \oplus (z_4\mathbf{r_1} \wedge \mathbf{r_2}) \\ \vdots \\ (z_{n-2}\mathbf{r_{n-1}} \wedge \mathbf{r_n}) \oplus (z_{n-1}\mathbf{r_n} \wedge \mathbf{r_{n-2}}) \oplus (z_n\mathbf{r_{n-2}} \wedge \mathbf{r_{n-1}}) \end{pmatrix},$$

where $\mathbf{r_i}$ is the i^{th} row of A, and $\mathbf{x} \wedge \mathbf{y}$ is the element-wise product of vectors $\mathbf{x}$ and $\mathbf{y}$.

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

$$S^{(A')}_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} (A_{\alpha,j} \oplus z_\alpha y_j)\,(A_{\beta,j} \oplus z_\beta y_j)\,(A_{\gamma,j} \oplus z_\gamma y_j) \pmod 2, \tag{34}$$

and expanding the brackets,

$$\begin{aligned} S^{(A')}_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} (&A_{\alpha,j}A_{\beta,j}A_{\gamma,j} \oplus z_\alpha z_\beta z_\gamma y_j \\ &\oplus z_\alpha z_\beta A_{\gamma,j}y_j \oplus z_\beta z_\gamma A_{\alpha,j}y_j \oplus z_\gamma z_\alpha A_{\beta,j}y_j \\ &\oplus z_\alpha A_{\beta,j}A_{\gamma,j}y_j \oplus z_\beta A_{\gamma,j}A_{\alpha,j}y_j \oplus z_\gamma A_{\alpha,j}A_{\beta,j}y_j) \pmod 2. \end{aligned} \tag{35}$$

We can see that the first term of equation 35 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 $\chi(A, \mathbf{z})\ \mathbf{z}$,

$$S^{(A')}_{\alpha,\beta,\gamma} = S^{(A)}_{\alpha,\beta,\gamma} \oplus z_\alpha z_\beta z_\gamma |\mathbf{y}| \oplus z_\alpha z_\beta \left[A\mathbf{y}\right]_\gamma \oplus z_\beta z_\gamma \left[A\mathbf{y}\right]_\alpha \oplus z_\gamma z_\alpha \left[A\mathbf{y}\right]_\beta \oplus \left[\chi(A, \mathbf{z})\ \mathbf{y}\right]_{\iota(\alpha,\beta,\gamma)}, \tag{36}$$

where we $\iota(\alpha, \beta, \gamma)$ is the row index of χ whose elements are given by $(z_\alpha \mathbf{r}_\beta \wedge \mathbf{r}_\gamma) \oplus (z_\beta \mathbf{r}_\gamma \wedge \mathbf{r}_\alpha) \oplus (z_\gamma \mathbf{r}_\alpha \wedge \mathbf{r}_\beta)$. 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. □

Now we have 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 j^{th} column of A, as in lemma 5.2. We check to see if the conditions in lemma 5.3 are satisfied for $\mathbf{z}$ by forming the matrix,

$$\tilde{A} = \begin{pmatrix} A \\ \chi(A, \mathbf{z}) \end{pmatrix}. \tag{37}$$

Any vector, $\mathbf{y}$, in the null space of $\tilde{A}$ simultaneously satisfies *C2* and *C3* of lemma 5.3. 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. If $|\mathbf{y}| = 0 \pmod 2$ then condition *C1* is satisfied and we can perform the duplication transformation from lemma 5.3. Otherwise, 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 duplicate columns by adding the value of $\mathbf{z}$ to every column j for which $y_j = 1$, then destroy the a^{th} and b^{th} columns of A. 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.

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 $m' \leq m$ 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 nullspace(A) be a function that returns a matrix whose columns generate the right nullspace of A.
- Let simplify(A) be a function that removes every pair of identical columns and every all-zero column from A.

procedure TODD
  Initialize $A' \leftarrow A$
start:
  for all $1 \leq a < b \leq \mathrm{cols}(A')$ **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 $1 \leq k \leq \mathrm{cols}(N)$ **do**
      $\mathbf{y} \leftarrow \mathrm{col}_k(N)$
      if $y_a \oplus y_b = 1$ **then**
        if $|\mathbf{y}| = 1 \pmod 2$ **then**
          $A' \leftarrow \begin{pmatrix} A' & \mathbf{0} \end{pmatrix}$
          $\mathbf{y} \leftarrow \begin{pmatrix} \mathbf{y} \\ 1 \end{pmatrix}$
        $A' \leftarrow A' + \mathbf{z}\mathbf{y}^T$
        simplify(A′)
        goto *start*

6 Results & Discussion

We implemented our T gate optimization protocol in C++ and ran 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 four versions of *T-optimiser*: RE, TOOL (feedback), TOOL (without feedback) and TODD. The results for the random benchmark are shown in figure 6. Second, we tested the protocol on a library of benchmark circuits taken from Maslov's Reversible Logic Synthesis Benchmarks Page [30]. These circuits implement useful quantum algorithms including Galois Field multipliers, integer addition, n^{th} prime, Hamming coding functions and the hidden weighted bit functions. Some of these circuits feature the generalised (n-bit) Toffoli gate, which we implement in the Clifford + T gate set using ancillas with a standard method [31]. The results for the quantum algorithm benchmark are listed in table 1.

6.1 Random 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-optimize*. For both versions of TOOL, we find that the numerical results for the T count follows the expected analytical scaling of $\mathcal{O}(n^2)$ and correspondingly the results for RE scales as $\mathcal{O}(n^3)$. We see that TODD outperforms the next best algorithm, TOOL (without feedback), by a constant value of 5.5 ± 0.7 and is therefore the preferred algorithm in settings where classical runtime is not an issue.

While both have a runtime of $\mathcal{O}(\text{POLY}(n))$, the order of the polynomial is much higher for TODD than for TOOL. This means the latter would be preferable for some experimental settings where very large non-Clifford circuits must be generated within a limited time frame e.g. due to decoherence. But at this stage where the disparity between quantum and classical resource costs is so high, reducing the T count is more important than reducing the classical runtime.

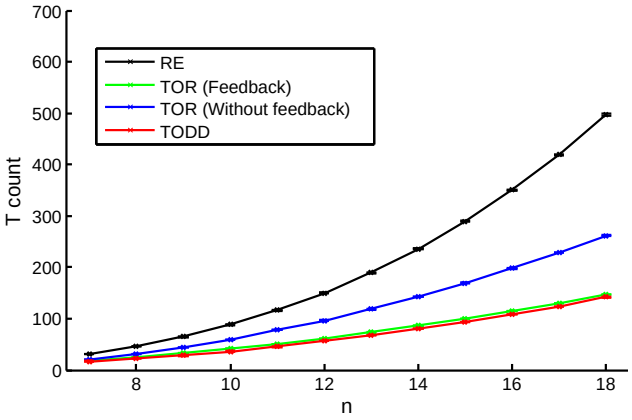


Figure 6: Random benchmark.

The random benchmark effectively benchmarks on random 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 [32, 33, 34]. Low cost designs of IQP circuits provided by our protocol would therefore be an asset for achieving quantum supremacy.

6.2 Quantum Algorithms

The results in table 1 show us that TODD successfully reduced the T count for every algorithm upon which it was tested and has an average and maximum saving of 34% and 79%, respectively. This is immediately useful due to the lower cost associated with solving these problems. However, we acknowledge that the T count does not account for the full space-time cost of quantum computation (often referred to as the aggregate cost). In this work, we ignore the cost of Clifford gates due to the high ratio between the aggregate cost of the T gate and that of Clifford gates. The aggregate cost is highly sensitive to the architecture of the quantum computer, but for the surface code, this ratio is estimated to be between 100 and 1000 [35, 36, 37].

While our protocol leads to circuits with low T count, the final output often has an *increased* CNOT count. This is due to step 6 of our protocol where we map the phase polynomial back to a quantum circuit using a naive approach. As our work concerns T gate optimization, we leave the problem of optimizing CNOT count as an avenue for further work.

Table 1: T-counts for Clifford + T benchmarks circuits. We show the number of qubits n and the T-count for the circuit: before optimization (pre-opt.); after optimization (post-opt.) using the best previous algorithm; and post-optimization using our implementation of TODD. We show the T-count saving for TODD over the best previous algorithm, as well as the total execution time.

Circuit	**Pre-Opt.**		**Post-Opt. (Best Prev.)**			**Post-Opt. (TODD)**			
	n	**T**	**n**	**T**	**Due to**	**n**	**T**	**Saving (%)**	**Runtime (s)**
8-bit adder	24	350	24	213	RM(maj)	108	49	77.00	506.193
GF(2^4)-mult	12	112	12	68	T-par	19	50	26.47	5.408
GF(2^5)-mult	15	175	15	111	T-par	24	97	12.61	43.822
GF(2^6)-mult	18	252	18	150	T-par	29	136	9.333	259.38
GF(2^7)-mult	21	343	21	217	T-par	34	176	18.89	1869.24
HWB_6	7	105	7	71	T-par	41	45	36.62	67.11
QCLA-Adder_{10}	36	238	36	162	T-par	64	59	63.58	123.55
QCLA-COM_7	24	203	24	94	RM(maj)	56	35	62.77	52.006
QCLA-Mod_7	26	413	26	235	Auto (H)	98	58	75.32	989.694
RC-Adder_6	14	77	14	47	RM(maj/rec)	51	36	23.40	37.866
VBE-Adder_3	10	70	10	24	T-par	14	20	16.67	0.101
Grover_5	9	336	9	52	T-par	127	39	25.00	452.677
CSLA-MUX_3	15	70	15	58	RM(rec)	32	50	13.79	15.563
CSUM-MUX_9	30	196	30	76	RM(rec)	81	36	52.63	74.525
Mod-Red_{21}	11	119	11	73	T-par	36	50	31.51	15.52
Mod-Mult_{55}	9	49	9	35	RM(maj/rec)	32	34	2.857	4.904
Hamming_{15} (low)	17	161	17	97	T-par	51	34	64.95	45.282
Hamming_{15} (med)	17	574	17	230	T-par	102	49	78.70	796.282
QFT_4	5	69	5	67	T-par	44	37	44.78	19.547
NC Toff_4	4	21	4	15	T-par	7	13	13.33	< 0.001
NC Toff_5	5	35	5	23	T-par	11	20	13.04	0.046
NC Toff_6	6	49	6	31	T-par	15	25	19.36	0.199
NC Toff_{10}	11	119	11	71	T-par	35	46	35.21	10.877
						Average		**34.08**	
						Max		**78.70**	

7 Conclusions

In this work, we have outlined a framework for optimizing large Clifford + T quantum circuits such that the output has T count near the global optimum and the run time is polynomial in both the number of qubits and the number of gates. This scheme maps the quantum circuit problem to a problem on order 3 symmetric tensors. We have presented an efficient near-optimal approach for solving this problem and reviewed previous methods. We implemented our protocol in C++ and used it to produce circuit decompositions for quantum algorithms with lower T-counts than any previous attempt known to the best of our knowledge. This lowers the cost of quantum computation and takes us closer to the common goal of achieving practical universal fault-tolerant quantum computation.

8 Acknowledgements

We acknowledge support by the Engineering and Physical Sciences Research Council (EPSRC) (Grant No. EP/M024261/1). We thank Mark Howard and Matthew Amy for valuable discussions.

References

[1] A. Y. Kitaev, A. Shen, and M. N. Vyalyi, *Classical and quantum computation*. American Mathematical Society Providence, 2002, vol. 47.

[2] C. M. Dawson and M. A. Nielsen, "The solovay-kitaev algorithm," *arXiv preprint quant-ph/0505030*, 2005.

[3] A. G. Fowler, "Constructing arbitrary steane code single logical qubit fault-tolerant gates," *Quantum Information & Computation*, vol. 11, no. 9-10, pp. 867–873, 2011.

[4] V. Kliuchnikov, D. Maslov, and M. Mosca, "Asymptotically optimal approximation of single qubit unitaries by clifford and t circuits using a constant number of ancillary qubits," *Physical review letters*, vol. 110, no. 19, p. 190502, 2013.

[5] P. Selinger, "Quantum circuits of t-depth one," *Physical Review A*, vol. 87, no. 4, p. 042302, 2013.

[6] D. Gosset, V. Kliuchnikov, M. Mosca, and V. Russo, "An algorithm for the t-count," *Quantum Information & Computation*, vol. 14, no. 15-16, pp. 1261–1276, 2014.

[7] N. J. Ross and P. Selinger, "Optimal ancilla-free clifford+ t approximation of z-rotations," *Quant. Inf. and Comp.*, vol. 16, p. 901, 2016.

[8] E. T. Campbell, B. M. Terhal, and C. Vuillot, "The steep road towards robust and universal quantum computation," *arXiv preprint arXiv:1612.07330*, 2016.

[9] S. Bravyi and A. Kitaev, "Universal quantum computation with ideal Clifford gates and noisy ancillas," *Phys. Rev. A*, vol. 71, p. 022316, 2005.

[10] R. Raussendorf, J. Harrington, and K. Goyal, "Topological fault-tolerance in cluster state quantum computation," *New Journal of Physics*, vol. 9, p. 199, 2007.

[11] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Phys. Rev. A*, vol. 86, p. 032324, Sep 2012. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevA.86.032324

[12] J. O'Gorman and E. T. Campbell, "Quantum computation with realistic magic-state factories," *Physical Review A*, vol. 95, no. 3, p. 032338, 2017.

[13] "Gridsynth command line tool," http://www.mathstat.dal.ca/ selinger/newsynth/.

[14] A. Paetznick and K. M. Svore, "Repeat-until-success: Non-deterministic decomposition of single-qubit unitaries," *Quantum Information & Computation*, vol. 14, no. 15-16, pp. 1277–1301, 2014.

[15] A. Bocharov, M. Roetteler, and K. M. Svore, "Efficient synthesis of probabilistic quantum circuits with fallback," *Physical Review A*, vol. 91, no. 5, p. 052317, 2015.

[16] ——, "Efficient synthesis of universal repeat-until-success quantum circuits," *Phys. Rev. Lett.*, vol. 114, p. 080502, Feb 2015. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevLett.114.080502

[17] E. Campbell, "Shorter gate sequences for quantum computing by mixing unitaries," *Physical Review A*, vol. 95, no. 4, p. 042306, 2017.

[18] M. B. Hastings, "Turning gate synthesis errors into incoherent errors," *arXiv preprint arXiv:1612.01011*, 2016.

[19] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov, "Automated optimization of large quantum circuits with continuous parameters," 2017/10/19 2017. [Online]. Available: https://arxiv.org/abs/1710.07345

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

[21] M. Amy and M. Mosca, "T-count optimization and reed-muller codes," *arXiv preprint arXiv:1601.07363*, 2016.

[22] G. Seroussi and A. Lempel, "Factorization of symmetric matrices and trace-orthogonal bases in finite fields," *SIAM Journal on Computing*, vol. 9, no. 4, pp. 758–767, 1980.

[23] E. T. Campbell and M. Howard, "Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost," *Phys. Rev. A*, vol. 95, p. 022316, Feb 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.95.022316

[24] A. Lempel, "Matrix factorization over gf(2) and trace-orthogonal bases of gf(2ˆn)," *SIAM Journal on Computing*, vol. 4, no. 2, pp. 175–186, 1975.

[25] A. Montanaro, "Quantum circuits and low-degree polynomials over $\mathbb{F}_2$," *Journal of Physics A: Mathematical and Theoretical*, vol. 50, no. 8, p. 084002, 2017. [Online]. Available: http://stacks.iop.org/1751-8121/50/i=8/a=084002

[26] M. Amy, D. Maslov, and M. Mosca, "Polynomial-time T-depth optimization of Clifford+T circuits via matroid partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 10, pp. 1476–1489, Oct 2014.

[27] E. T. Campbell and M. Howard, "Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost," *Phys. Rev. A*, vol. 95, no. 022316, 2017.

[28] M. Amy and M. Mosca, "T-count optimization and Reed-Muller codes," *arXiv*, vol. arXiv:1601.07363v1 [quant-ph], 2016.

[29] A. Lempel, "Matrix factorization over GF(2) and trace-orthogonal bases of GF(2)," *SIAM J. Comput.*, vol. 4, no. 2, 1975.

[30] D. Maslov, "Reversible logic synthesis benchmarks page," http://webhome.cs.uvic.ca/ dmaslov/, 2011.

[31] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 10th ed. New York, NY, USA: Cambridge University Press, 2011.

[32] M. J. Bremner, R. Jozsa, and D. J. Shepherd, "Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy," *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 2010. [Online]. Available: http://rspa.royalsocietypublishing.org/content/early/2010/08/05/rspa.2010.0301

[33] A. W. Harrow and A. Montanaro, "Quantum computational supremacy," *Nature*, vol. 549, p. 203, 2017.

[34] D. Shepherd and M. J. Bremner, "Temporally unstructured quantum computation," *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 465, no. 2105, pp. 1413–1439, 2009. [Online]. Available: http://rspa.royalsocietypublishing.org/content/465/2105/1413

[35] R. Raussendorf, J. Harrington, and K. Goyal, "Topological fault-tolerance in cluster state quantum computation," *New Journal of Physics*, vol. 9, no. 6, p. 199, 2007.

[36] J. O'Gorman and E. T. Campbell, "Quantum computation with realistic magic-state factories," *Phys. Rev. A*, vol. 95, p. 032338, Mar 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.95.032338

[37] A. G. Fowler, S. J. Devitt, and C. Jones, "Surface code implementation of block code state distillation." *Scientific reports*, vol. 3, p. 1939, 2013.

A Lempel's Factoring Algorithm

We describe Lempel's factoring algorithm (originally from reference [29]) 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:

$$S_{\alpha,\beta} = \sum_{k=1}^{m} A_{\alpha,k} A_{\beta,k} \pmod 2. \tag{38}$$

In reference [29], Lempel proves that the minimal value of m is equal to

$$\mu(S) = \rho(S) + (1 \oplus \vee_i^n S_{i,i}), \tag{39}$$

where $\rho(S)$ is the rank of matrix S and $\vee$ is the logical OR operator. Lempel's algorithm solves the problem of finding an A matrix that obeys equation 38 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 $c(A)$, the j^{th} column of A as $c_j(A)$. Lempel's algorithm consists of the following 7 steps:

1. Generate an initial (necessarily suboptimal) A matrix for S.
2. Check if $c(A) = \mu(S)$. If true, exit and output A. Otherwise, perform steps 3 to 7.
3. Find a $y \in \mathbb{Z}_2^m$ such that $Ay = \mathbf{0}$ and $\vee_i^m (y_i \oplus 1) = 1$.
4. If $|y| = 1 \pmod 2$ then update $y \to (y^T, 1)^T$ 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 $A \to A \oplus zy^T$, where $z = c_a(A) \oplus c_b(A)$.
7. Remove columns a and b from A, then go to step 2.

The above algorithm is designed to work for the special case where the input S matrix is non-singular. However, Lempel provides a means of extending to the singular case using a pre-processing step on the input S matrix:

$$\tilde{S} = RSR^T = \begin{pmatrix} S' & 0 \\ 0 & 0 \end{pmatrix}, \tag{40}$$

where S' is a non-singular symmetric matrix with $\rho(S)$ rows. The matrix R is constructed as follows:

$$R = LP, \tag{41}$$

where P is a permutation matrix such that the first $\rho(S)$ rows of PS are linearly independent and L is a self-inverse lower diagonal matrix such that the last $n - \rho(S)$ rows of $TPSP^T$ are all zero vectors. Lempel's algorithm is performed on S' to obtain a minimal factor A'. Finally, we recover the minimal factor of S by applying the transformation:

$$A = P^T T \begin{pmatrix} A' \\ 0 \end{pmatrix}. \tag{42}$$

B Calculating $\mathcal{U}^\dagger_{\text{Clifford}}$

We will now describe how to determine the Clifford correction required to restore the output of *T-Optimize* to the input unitary.

Let the input of *T-Optimize* 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

$$f = f_1 + f_2, \tag{43}$$

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

$$g = g_1 + g_2 = f_1 + g_2, \tag{44}$$

so

$$g = f + (g_2 - f_2). \tag{45}$$

Equation (45) implies that $U_{\text{Clifford}} = U_{g_2 - f_2} \in \mathcal{D}_2$. Therefore, the Clifford correction is $U^\dagger_{\text{Clifford}} = U^\dagger_{g_2 - f_2} = U_{f_2 - g_2}$. We can map $(f_2 - g_2)$ to a phase polynomial and subsequently to a quantum circuit, $\mathcal{U}^\dagger_{\text{Clifford}}$.