PhD Project First Year Report
Optimal Designs for Fault-Tolerant Quantum Computers

Author: Luke Heyfron

Supervisor: Dr. Earl T. Campbell

June 2017

Abstract

In order to realise practical quantum computation, it is important to consider the resource cost associated with implementing quantum circuits. The T gate requires several hundred times more elementary operations than Clifford group operators to implement fault-tolerantly for certain high-threshold quantum error correction codes such as the surface code. Therefore, effective methods for minimizing the number of T gates in a given quantum circuit are instrumental in reducing the overall cost of quantum computation. Finding the minimum number of T gates required to implement an n-qubit circuit generated by CNOT and T has been shown to be equivalent to minimum distance decoding of the punctured Reed-Muller code of length n and order n-4, which is believed to be a hard problem. In this report, we present two computationally efficient heuristics for estimating the minimal number of T gates required for multi-qubit circuits generated by CNOT and T gates. Both algorithms output circuits with T counts that have asymptotic scaling $\mathcal{O}(n^2)$, which is the same scaling as the upper bound for the inefficient optimal decoder and improves upon the worst-case scaling of $\mathcal{O}(n^3)$ for a naive fast decoder. The algorithms can be adapted for circuits over the universal $\{H, CNOT, T\}$ gate set by allowing for ancilla preparation and classical feedforward, and they have been used to estimate the T count for known practical quantum circuits.

1 Introduction

For many types of quantum error correction code including the surface code, a special class of so-called Clifford operations have *transversal* implementations [8], meaning the physical gates used to implement a logical operator act independently on each qubit with the same code block [10]. A transversal operator is natively protected by the error correction code against noise and the number of physical gates required for its implementation are linear in the number of physical qubits. Therefore, such operators are assumed to be cheap, perfect resources. However, Clifford operations alone are insufficient for universal quantum computation. The T gate is the common choice for the canonical non-Clifford gate, whose inclusion in the gate set allows for universal quantum computation. The T gate is not transversal for the surface code as well as many other codes and requires many more steps to implement fault-tolerantly. An established method for implementing a T gate requires the distillation of many noisy resource states called magic states into few high-fidelity states over several rounds [6],[5]. The state can then be teleported into a circuit using a scheme such as [11]. The distillation and teleportation stages can be performed with physical Clifford operations but involve many more physical Clifford operations than those of logical Cliffords, which means T gates are considered as high costing gates compared to Clifford operations. Therefore, it is of interest to develop protocols for synthesizing quantum circuit decompositions that minimize the number of T gates used.

Much work has already been done to develop T gate optimization algorithms. Algorithms for optimizing the T depth, another resource metric for T gates, were developed by Amy, Maslov, Mosca and Roetteler in [3] and by Amy, Maslov and Mosca in [2]. Gosset, Kliuchnikov, Mosca and Russo developed an algorithm that finds T count optimal decompositions for multi-qubit Clifford and T circuits [9]. More recently, Amy and Mosca [4] proved a relationship between the problem of finding T gate minimal decompositions of circuits composed of CNOT and T gates and minimum distance decoding of the punctured Reed-Muller code of length n and order $n-4$. This was followed by Campbell and Howards work [7], in which a method for simultaneously performing gate synthesis and magic state distillation was proposed that features a fast Reed-Muller decoder based on Lempel's matrix factoring algorithm with improved T count scaling compared to [4] (see sections 4.1 and 5.1).

In this report an algorithm is presented for reducing the T count for n-qubit quantum circuits composed of CNOT and T gates. It is shown how it is possible to generalize this algorithm for the universal gate set $\{H, CNOT, T\}$. The rest of the paper is structured as follows: in section 2 we define the terminology and notation conventions used for the rest of the paper; in section 3 we define the gate synthesis problem that our algorithm solves, first in the quantum circuit picture, then in terms of a special kind of matrix representation that facilitates the description of our solution. In section 4 we describe the algorithm presented in this paper for reducing the T count; in section 5 we present the results of this algorithm compared to previous works for both randomized circuits over CNOT and T as well as practical circuits of a universal gate set. Finally, we consolidate our results and discuss future work in section 6.

2 Theory and Terminology

2.1 Quantum Theory

We define the set of n-qubit computational basis states to be $\{|x\rangle\}$ for all $x \in \{0,1\}^n$, the set of binary tuples of length n. The single qubit Pauli operators are defined in the computational basis as follows,

$$\sigma_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{1}$$

The n-qubit Pauli group $\mathcal{P}^n$ is defined as the n-fold tensor product of single qubit Pauli operators, along with multiplicative factors of ± 1 and $\pm i$ i.e. $\mathcal{P}^n = \{(-1)^a\, i^b \sigma_c \mid a, b \in \{0,1\}, c \in \{0, x, y, z\}\}^n$. The operators CNOT, H and T are given by

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{2}$$

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \tag{3}$$

$$T = \begin{pmatrix} 1 & 0 \\ 0 & \omega \end{pmatrix}, \tag{4}$$

where we define $\omega = e^{i\frac{\pi}{4}}$. A single qubit operator G applied to the i^{th} qubit of an n-qubit system is expressed as $G^{(i)}$. A controlled-G operator where G is a single qubit target operator is expressed as control-$G^{(t,c)}$ where the t^{th} and c^{th} qubits are the target and control qubit, respectively.

The n-qubit Clifford group $\mathcal{C}^n$ is defined as the set of operators that maps $\mathcal{P}^n$ to itself under conjugation, $\mathcal{C}^n = \{U \mid UPU^\dagger = P', P, P' \in \mathcal{P}^n\}$. The k^{th} level of the Clifford hierarchy on n-qubits, $\mathcal{C}_k^n$, is defined recursively,

$$\mathcal{C}_k^n = \{U \mid UPU^\dagger = Q, P \in \mathcal{P}^n, Q \in \mathcal{C}_{k-1}^n\}, \tag{5}$$

with $\mathcal{C}_1^n = \mathcal{P}^n$. We define $\mathcal{D}_k^n \subseteq \mathcal{C}_k^n$ as the set of diagonal elements of the k^{th} level of the Clifford hierarchy.

3 The Gate Synthesis Problem

An n-qubit quantum circuit C of length k is a time-ordered set of operators taken from an elementary gate set G on n qubits. In particular, if C is such a quantum circuit, then $C = \{C_1, C_2, ..., C_k\}$ where $C_t \in G^n$ is the operator

applied at the t^{th} time step. C is said to implement the unitary U_C, which is given by

$$U_C = \prod_{t=k}^{1} C_t. \tag{6}$$

Two circuits are said to be equivalent if they implement the same unitary, i.e. $C \equiv_U C'$ if $U_C = U_{C'}$. We define $E(g)$ to be the cost function of applying a particular gate $g \in G$ to a logical qubit. The total cost of a circuit is therefore,

$$E(C) = \sum_{t=1}^{k} E(C_t). \tag{7}$$

Problem 3.1. *Gate Synthesis Problem (version 1). Given a unitary U, find some circuit C such that $E(C) = min\{E(C') \mid \forall C', U = U_{C'}\}$.*

According to our cost model for quantum gates, $E(\text{Clifford}) = 0$ and $E(T) = 1$. The total cost function for a circuit C is therefore,

$$E(C) = \sum_{t=1}^{k} E(C_t) = \tau(C), \tag{8}$$

where we define the T count of a circuit, $\tau(C)$, to be the number of T gates in circuit C. Therefore, the gate synthesis problem is equivalent to minimizing the number of T gates. We define the T count of a unitary as follows.

Definition 3.1. *T count.* Given a unitary U, the T count is defined as $\tau(U) = \min\{\tau(C) \mid \forall C, U = U_C\}$

3.1 Circuits Composed of CNOT and T Gates

In this section we establish the notions of phase polynomials and gate synthesis matrices following the analyses of [4] and [7]. It is known that any unitary U generated by the CNOT and T gate can be decomposed as,

$$U = VW, \tag{9}$$

where V is a member of the Clifford group with $\tau(V) = 0$ and $W \in \mathcal{D}_3^n$. It follows that $\tau(W) = \tau(U)$. Due to V not contributing to the T count of U, we will assume that U is already diagonal to simplify the analysis. The action of a unitary $U \in \mathcal{D}_3^n$ on a computational basis state can be written as follows,

$$U |x\rangle = \omega^{f_U(x)} |x\rangle, \tag{10}$$

where $f_U(x)$ is known as the *phase function* for U. The phase function can be determined by tracking the state of each qubit through the circuit, updating appropriately after each CNOT gate, and adding a term after each T gate. See figure 1 for an example. This construction leads to the following decomposition for phase function known as the *phase polynomial*,

$$f_U(x) = \sum_{y \in \{0,1\}^n \setminus 0} \mathbf{a}_y \bigoplus_{i=1}^{n} x_i^{y_i} \mod 8, \tag{11}$$

where $\mathbf{a} \in \mathbb{Z}_8^{2^n}$ is called an *implementation vector*. Modulo 8 arithmetic is due to ω being the eight root of unity. Once an implementation vector for U is found, it is possible to construct a circuit C that implements U using $\tau(C) = |\mathbf{a} \pmod 2|$ T gates. There exist multiple implementation vectors that give rise to the same phase functions. This is due to the existence of non-trivial implementation vectors that implement the all-zero phase function. We denote the set of all implementation vectors that implement the all-zero phase function as $\mathcal{R}$. In [4], Amy and Mosca proved that the set $\mathcal{R}' = \{r \pmod 2 \mid r \in \mathcal{R}\}$ is exactly the punctured Reed-Muller code of length n and order $n-4$, denoted $RM^*(n-4, n)$.

We now define the gate synthesis matrix, which is central to the description of the algorithms presented in this report.

Definition 3.2. *Gate Synthesis Matrix.* Let $f_U(x)$ be a phase function for a unitary U. A gate synthesis matrix is a matrix, A, whose elements $A_{i,j} \in GF(2)$, that satisfies $f_{(CUC')}(x) = |A^T x|$ where $C, C' \in \mathcal{C}_2^n$.

It should be noted that the number of columns of a gate synthesis matrix is equal to $|\mathbf{a} \pmod 2|$, where $\mathbf{a}$ is the implementation vector for which A is a gate synthesis matrix, and therefore is also equal to the T count of the circuit. This means we can rewrite problem 3.1,

Problem 3.2. *Gate Synthesis Problem (version 2). Given a unitary U, find a gate synthesis matrix A such that $cols(A) = min\{cols(A') \mid \forall A', f_U(x) = |A'^T x|\}$, where $cols(A)$ is the number of columns of matrix A.*

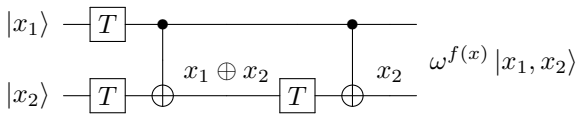


Figure 1: For circuits composed of CNOT and T gates, the phase polynomial is determined by updating the state of the target qubit of each CNOT gate and adding the appropriate term after each T gate. This circuit has phase polynomial $f(x) = x_1 + x_2 + x_1 \oplus x_2$.

4 Solutions to the Gate Synthesis Problem

4.1 Lempel's Factoring Algorithm

The synthesis algorithm described in this section is based on the Lempel algorithm developed by Abraham Lempel in [12]. The algorithm itself provides a minimal factorization of a symmetric matrix $A = BB^T$ defined on GF(2) such that B has minimal columns. The process involves finding an initial matrix B that satisfies $A = BB^T$ known as the *elementary factor* of A, which does not necessarily have minimal columns. Algorithm 1 describes the procedure for finding the elementary factor, which is based on the supplementary MATLAB code for [7]. Once we have an elementary factor, we can find the minimal factor using Lempel's factoring algorithm described in algorithm 2. Within the main loop structure is a procedure that reduces the number of columns of B by 1 or 2 each iteration. The algorithm terminates when a certain condition is met that depends on the number of columns of B, the rank of A and the diagonal elements of A. Note that the version of the Lempel factorization algorithm described here works for non-singular A matrices only. The general algorithm for singular matrices can be found in section 4 of [12] and is omitted from this work for brevity.

The means of applying the algorithm to gate synthesis for circuits over CNOT and T gates is proposed by Campbell and Howard in [7]. It involves breaking up the synthesis process into a number of rounds where after each round, a part of the circuit is synthesized and the remaining part acts on one fewer qubit than the previous round. In this way we reduce the problem size until it is feasible to use the computationally inefficient optimal Reed-Muller decoder, which requires searching over every element of $\mathcal{R}$. The cardinality of $\mathcal{R}$ increases This enforces a practical limitation on the number of qubits

We define two different versions of gate synthesis via Lempel's factoring algorithm as "with feedback" and "without feedback". The following analysis describes what is meant by this. Given that at synthesis round t, we have remaining weighted polynomial f_t with signature tensor S,

$$f_t(x) = \sum_{i=1}^{n} S_i x_i + 2 \sum_{i<j}^{n} S_{i,j} x_i x_j + 4 \sum_{i<j<k}^{n} S_{i,j,k} x_i x_j x_k \tag{12}$$

$$f_t(x) = S_h x_h + 2x_h \left(\sum_{i \neq h}^{n} S_{i,h} x_i + 2 \sum_{i<j \neq h}^{n} S_{i,j,h} x_i x_j \right) + f_{t,\bar{h}}(x \backslash x_h) \tag{13}$$

Where we define $f_{t,\bar{h}}$ to be the weighted polynomial containing all the terms of $f_t(x)$ that do not depend on x_h. We define A to be a symmetric matrix on GF(2) such that,

$$A_{i,j} = S_{i,j,h}, \tag{14}$$

for which we find the minimal factor, B, using algorithm 2. This leads to the phase polynomial,

$$f'_h(x \backslash x_h) = |B^T x| = \sum_{j=1}^{m} \left(\bigoplus_{i=1}^{n} B_{i,j} x_i \right). \tag{15}$$

Now we need to account for the factor of $2x_h$ from equation 13 in order for it to form a valid contribution to the phase polynomial of f_t.

$$f_h(x) = 2x_h \left(\sum_{i \neq h}^{n} S_{i,h} x_i + 2 \sum_{i<j \neq h}^{n} S_{i,j,h} x_i x_j \right) = 2x_h \sum_{j=1}^{m} \left(\bigoplus_{i=1}^{n} B_{i,j} x_i \right) + \Delta(x \backslash x_h) \tag{16}$$

$$= mx_h + \sum_{j=1}^{m} \left(x_h \oplus \bigoplus_{i=1}^{n} B_{i,j} x_i \right) + f'_h(x \backslash x_h) + \Delta(x \backslash x_h), \tag{17}$$

where Δ is a correction polynomial for the cubic terms not accounted for in Lempel's matrix factoring algorithm, which does not depend on x_h.

Now we can define the difference between the "with feedback" and "without feedback. The next round of Lempel synthesis will act on weighted polynomial,

$$f_{t+1}(x \backslash x_h) = \begin{cases} f'_h(x \backslash x_h) + f_{t,\bar{h}}(x \backslash x_h) + \Delta(x \backslash x_h) & \text{with feedback} \\ f_{t,\bar{h}}(x \backslash x_h) + \Delta(x \backslash x_h) & \text{without feedback,} \end{cases} \tag{18}$$

where the other terms of f_t are added to the output phase polynomial of the Lempel gate synthesis algorithm.

Algorithm 1 Elementary Factor

Input: A symmetric matrix A with n rows and m columns whose elements are members of GF(2).
Output: A matrix B such that $A = BB^T$.

procedure ELEMENTARY
  $B \leftarrow$ empty $n \times 0$ matrix.
  for all $1 \leq i < j \leq n$ **do**
    $c \leftarrow \mathbf{0}$, length n all-zero vector.
    if $(A_{i,j} = 1)$ AND $(\sum_{k=1}^{n} B_{i,k} B_{j,k} = 0 \pmod 2)$ **then**
      $c_i \leftarrow 1$
      $c_j \leftarrow 1$
      $B \leftarrow \begin{pmatrix} B & c \end{pmatrix}$
  for all $1 \leq i \leq n$ **do**
    $c \leftarrow \mathbf{0}$, length n all-zero vector.
    if $(A_{i,i} = 1)$ AND $(\sum_{k=1}^{n} B_{i,k} = 0 \pmod 2)$ **then**
      $c_i \leftarrow 1$
      $B \leftarrow \begin{pmatrix} B & c \end{pmatrix}$

Algorithm 2 Lempel Algorithm

Input: A symmetric matrix A with n rows and m columns whose elements are members of GF(2).
Output: A matrix B such that $A = BB^T$ that B has minimal columns.

- Let $\delta(A)$ be a function on matrix A such that $\delta = \begin{cases} 1 & \text{if } A_{i,i} = 0 \text{ for all } i \\ 0 & \text{otherwise} \end{cases}$
- Let $\text{col}_j(A)$ denote the j^{th} column of matrix A.
- Let $\text{cols}(A)$ denote the number of columns of matrix A.
- Let $\text{rank}(A)$ denote the rank of matrix A.

procedure LEMPEL
  $B \leftarrow$ ELEMENTARY(A)
  if $\text{cols}(B) = \text{rank}(A) + \delta(A)$ **then**
    exit LEMPEL
  else
loop:
    Find any vector y such that $By = \mathbf{0}$, the all zero vector.
    if $|y| = 1 \pmod 2$ **then**
      $y \leftarrow \begin{pmatrix} y \\ 1 \end{pmatrix}$
      $B \leftarrow \begin{pmatrix} B & \mathbf{0} \end{pmatrix}$
    Find any pair of columns i, j such that $y_i = 0$ and $y_j = 1$.
    $x \leftarrow \text{col}_i(B)$
    $B \leftarrow B + xy^T$
    Remove columns i and j from B.
    if $\text{cols}(B) = \text{rank}(A) + \delta(A)$ **then**
      exit LEMPEL
    else
      goto *loop*

4.2 The Extended Lempel Algorithm

In this section we present an algorithm based in principle on Lempel's matrix factoring algorithm but extended to work for tensors of order 3. This algorithm allows us to reduce the number of columns of a gate synthesis matrix directly and is the key result of this report. The first part of this section shows that we can add an arbitrary column vector x to a subset of columns of the gate synthesis matrix without altering the signature tensor. The second part shows that choosing a specific value of x results in a new gate synthesis matrix that has two identical columns, which can be subsequently removed.

Definition 4.1. *Signature tensor.* Let A be an $n \times m$ gate synthesis matrix. The signature tensor of A is an $n \times n \times n$ symmetric tensor on GF(2) defined as,

$$(S(A))_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} A_{\alpha,j} A_{\beta,j} A_{\gamma,j}. \tag{19}$$

Note that as $S(A)$ is symmetric, it is invariant under any permutation of indices. Hence, we shall omit any repeated index for the sake of brevity, e.g. $S_{\alpha,\alpha,\beta} = S_{\alpha,\beta}$.

Let $\mathcal{I} = \{(\alpha,\beta,\gamma) \mid 1 \leq \alpha \neq \beta \neq \gamma \leq n\}$ be the set of all 3-tuples such that each element falls in the range $[1,n]$ and is unique. We define $\chi(A,x)$ to be an $|\mathcal{I}| \times m$ matrix that is a function of A and x, a $n \times m$ matrix and a column vector of length n, respectively:

$$(\chi(A,x))_{i,j} = x_\alpha x_\beta A_{\gamma,j} + x_\beta x_\gamma A_{\alpha,j} + x_\gamma x_\alpha A_{\beta,j}, \tag{20}$$

where (α,β,γ) is the i^{th} element of $\mathcal{I}$.

Lemma 4.1. *Let A and $A' = A + xy^T$ be two gate synthesis matrices where x, y are arbitrary column vectors of dimension n and m, respectively. $S(A) = S(A')$ if all of the following conditions are met:*

1. $|y| = 0$
2. $Ay = 0$
3. $\chi(A,x)\ y = 0$.

Proof. We begin the proof by finding an expression for $S(A')$ using equation 19,

$$(S(A'))_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} (A_{\alpha,j} + x_\alpha y_j)(A_{\beta,j} + x_\beta y_j)(A_{\gamma,j} + x_\gamma y_j), \tag{21}$$

and expanding the brackets,

$$\begin{aligned}(S(A'))_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} (&A_{\alpha,j}A_{\beta,j}A_{\gamma,j} + x_\alpha x_\beta x_\gamma y_j \\ &+ x_\alpha x_\beta A_{\gamma,j} y_j + x_\beta x_\gamma A_{\alpha,j} y_j + x_\gamma x_\alpha A_{\beta,j} y_j \\ &+ x_\alpha A_{\beta,j} A_{\gamma,j} y_j + x_\beta A_{\gamma,j} A_{\alpha,j} y_j + x_\gamma A_{\alpha,j} A_{\beta,j} y_j).\end{aligned} \tag{22}$$

If we sum the first term over all j, we find that it becomes equal to $S(A)$. The task is to show that the remaining terms sum to zero under the specified conditions. Next, we use the definitions of $|y|$, Ay and $\chi(A,x)\ y$ to simplify the result,

$$(S(A'))_{\alpha,\beta,\gamma} = (S(A))_{\alpha,\beta,\gamma} + x_\alpha x_\beta x_\gamma |y| + x_\alpha x_\beta (Ay)_\gamma + x_\beta x_\gamma (Ay)_\alpha + x_\gamma x_\alpha (Ay)_\beta + (\chi(A,x)\ y)_i, \tag{23}$$

where we define i such that (α,β,γ) is the i^{th} element of $\mathcal{I}$.

By applying condition (1), the second term is eliminated; by applying condition (2), the next three terms are eliminated, and by applying condition (3), the final term is eliminated. □

Lemma 4.2. *Let A be an $n \times m$ gate synthesis matrix where all columns are unique and non-zero. Let $A' = A + xy^T$ where x and y are column vectors on GF(2) of length n and m, respectively, defined such that $x_i = A_{i,a} + A_{i,b}$ for some $a, b \in [1,m]$ and $y_a + y_b = 1$. The columns a and b of A' are identical.*

Proof. We begin the proof by finding expressions for the matrix elements of A' in terms of A, x and y,

$$A'_{i,j} = A_{i,j} + x_i y_j, \tag{24}$$

and substitute the definition of x,

$$A'_{i,j} = A_{i,j} + (A_{i,a} + A_{i,b})y_j. \tag{25}$$

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

$$A'_{i,a} = A_{i,a} + (A_{i,a} + A_{i,b})y_a, \tag{26}$$
$$A'_{i,b} = A_{i,b} + (A_{i,a} + A_{i,b})y_b. \tag{27}$$

We substitute in the condition $y_b = y_a + 1$ into 27,

$$\begin{aligned} A'_{i,b} &= A_{i,b} + (A_{i,a} + A_{i,b})(y_a + 1) \\ &= A_{i,a} + (A_{i,a} + A_{i,b})y_a \\ &= A'_{i,a}. \end{aligned} \tag{28}$$

□

Now we are ready to describe the extended Lempel algorithm, abbreviated as *LempelX*. Given a unitary U and an initial gate synthesis matrix A that implements U, we commence by iterating through all column pairs a, b of A where we construct the vector x according to lemma 4.2. We check to see if the conditions in lemma 4.1 are satisfied for x by extending the matrix A with rows constructed according to equation 20, for all triplets $(\alpha, \beta, \gamma) \in \mathcal{I}$ and call this extended matrix $\tilde{A}$. We then calculate a basis for the right null space of $\tilde{A}$. Any vector in the null space, y, is an incidence vector for a subset of columns of A that satisfy conditions 2 and 3 of lemma 4.1. We scan through the null space basis until we find a y such that $y_a + y_b = 1 \pmod 2$. At this stage we know that we can remove at least one column from A. We force condition 1 of lemma 4.1 to be satisfied by appending a 1 to y and an all-zero column to A if y has odd-weight. Finally, we add the value of x to every column j for which $y_j = 1$, then eliminate columns a and b. This reduces the number of columns of A and therefore the T count of U. We have now described a single round LempelX (see algorithm 3). The full LempelX algorithm is described in algorithm 4 and entails repeating the procedure in algorithm 3 until every column pair in a single round has been exhausted without the required conditions of lemmas 4.1 and 4.2 being met. At this stage, we regard A as a minimal gate synthesis matrix with respect to LempelX and terminate the algorithm.

5 Experimental Results

5.1 Random Circuits Composed of CNOT and T Gates

The Lempel and LempelX algorithms were implemented in C++ and benchmark tested against randomized circuits over CNOT, T. The circuits were generated such that each element of the signature tensor is 1 with probability 1/2. The naive decoder is based on expanding each term in the weighted polynomial according to the identity $x_a \oplus x_b = x_a + x_b - 2x_a x_b$. We see in figure 2a that both Lempel and LempelX perform much better than the naive decoder. This was expected for Lempel as the naive decoder yields worse-case T counts that scale asymptotically as $\mathcal{O}(n^3)$ with the number of qubits n, whereas Campbell and Howard showed that Lempel (feedback) scales as $\mathcal{O}(n^2)$ [7]. The experimental data in 2b shows that LempelX a lower T count than Lempel (feedback) by a value that converges to a constant of 5.5 ± 0.7 from $n \geq 10$. This implies the T count of LempelX has the same asymptotic scaling as Lempel (feedback), however an analytical argument for this scaling has not yet been found.

5.2 Universal Practical Circuits

The Lempel and LempelX algorithms were tested on universal circuits composed from CNOT, H and T gates and the results are compiled in table 1. These circuits were taken from Maslov's reversible benchmark website [13]. Multiply controlled Toffoli gates are constructed using a standard method demonstrated in figure 3. Hadamards are eliminated using 4, which has the effect of swapping the Boolean variable associated with the qubit to which the Hadamard is applied with a completely new variable representing the "Hadamard-ancilla". This trick for Hadamard removal using so called *path variables* is due to [2]. The addition of potentially hundreds of Hadamard-ancillas for large circuits could cause practical issues for the synthesis algorithms regarding time constraints. A full investigation

Algorithm 3 Extended Lempel Algorithm (Base)

Input: A matrix A with n rows and m columns whose elements are members of $\mathbb{Z}_2$.
Output: A matrix A' with n rows and $p = m - q$ columns such that $a \in [0, 2]$ and $S(A') = S(A)$.

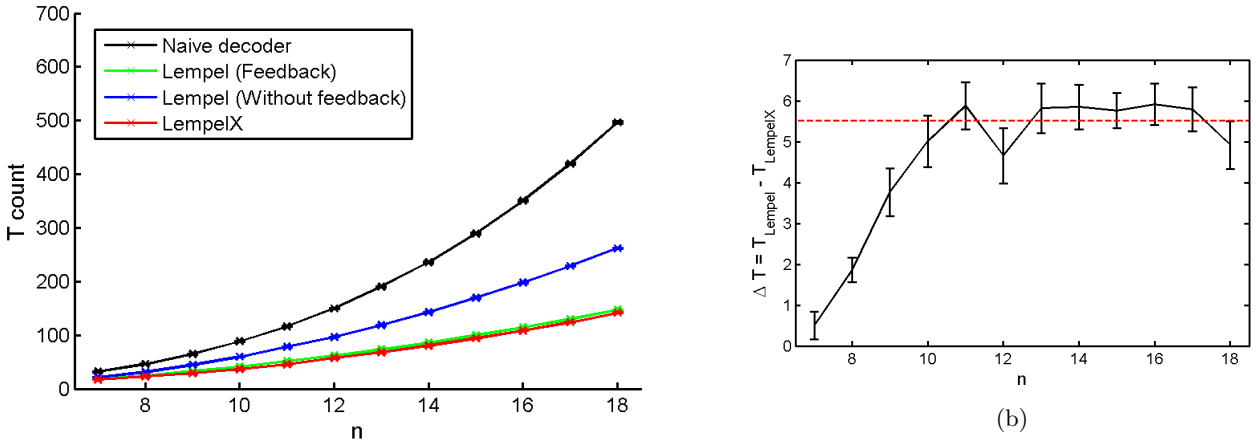
- Let $\mathrm{col}_j(A)$ be a function that returns the j^{th} column of A.
- Let $\mathrm{cols}(A)$ be a function that returns the number of columns of A.
- Let $\mathrm{nullspace}(A)$ be a function that returns a matrix whose columns generate the right nullspace of A.

procedure LempelXbase
  Initialize $A' \leftarrow A$
  for all $1 \leq a < b \leq m$ **do**
    $x \leftarrow \mathrm{col}_a(A) + \mathrm{col}_b(A)$
    $\tilde{A} \leftarrow \begin{pmatrix} A \\ \chi(A, x) \end{pmatrix}$
    $N \leftarrow \mathrm{nullspace}(\tilde{A})$
    for all $1 \leq k \leq \mathrm{cols}(N)$ **do**
      $y = \mathrm{col}_k(N)$
      if $y_a + y_b = 1$ **then**
        if $|y| = 1 \pmod 2$ **then**
          $A' \leftarrow \begin{pmatrix} A' & \mathbf{0} \end{pmatrix}$
          $y \leftarrow \begin{pmatrix} y \\ 1 \end{pmatrix}$
        $A' \leftarrow A' + xy^T$
        Remove columns a and b from A'
        exit LempelXbase

Algorithm 4 Extended Lempel Algorithm (Full)

Input: A matrix A with n rows and m columns whose elements are members of $\mathbb{Z}_2$.
Output: A matrix A' with n rows and $p \leq m$ columns such that p is minimal with respect to algorithm 1 and $S(A') = S(A)$.

procedure LempelX
start:
  $A' \leftarrow$ LempelXbase(A)
  if $A' = A$ **then**
    exit LempelX
  else
    goto start



(a) (b)

Figure 2: T counts for random circuits composed of CNOT and T gates synthesized by Lempel and LempelX.

is yet to be conducted into the practical limits on space and time resources for our implementation of Lempel and LempelX. Until this is done, functionality has been included in our implementations to optionally divide the input circuit into partitions, each of which can only contain up to a specifiable maximum number of Hadamards.

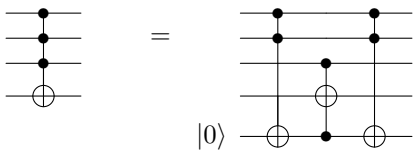


Figure 3: Circuit showing a method to construct the the 4-control Toffoli gate with standard Toffoli gates. The generalized n-controlled Toffoli can be constructed similarly with additional ancillas.

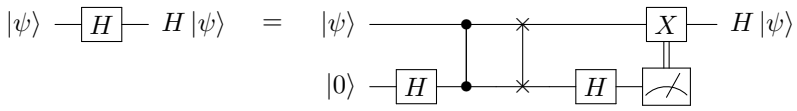


Figure 4: Internal Hadamards are eliminated by delegating them to act on ancillas according to the above circuit identity. The Hadamards on the left-hand circuit are now external, therefore they no longer obstruct the Lempel and LempelX synthesis algorithms.

Table 1: T counts for various benchmark circuits composed of CNOT, H and T gates as synthesized by the extended Lempel algorithm. T_{original} are the results produced my Amy and Mosca in [4] and are listed for comparison. N_H is the size of the ancilla registry for Hadamard-ancillas and $N_{\text{partitions}}$ is the number of partitions. All circuits are taken from [13].

Circuit	T_{original}[4]	T_{LempelX}	T_{naive}	$N_{partitions}$	N_H	Time (s)
Cycle17_3:	1944	567	567	24	30	562.719
hwb6_47_107	71	55	102	1	36	91.713
hwb6-42-150	71	46	140	1	36	160.198
mod5d4	16	8	8	1	1	0.019
nth_prime6_inc_55_667	400	263	354	5	30	228.025
ham15-109-214	97	28	65	1	30	27.756
ham15-70	230	103	148	3	30	100.899
ham15tc1	1019	258	359	7	30	270.862

6 Discussion, Conclusions and Plans for Future Work

To summarize, in this report an algorithm was presented for reducing the T count of circuits over CNOT and T gates that extends the principle of Lempel's factoring algorithm to symmetric tensors of order 3. This was benchmark tested on randomized CNOT and T circuits and was found to perform marginally better than then next best algorithm. Finally, it was tested on universal circuits for which decompositions were found that use between 25% and 77% T gates compared to previous constructions from [4]. At this stage, how much of this decrease is attributable to the synthesis algorithm is unqualified, however it seems likely that it is mostly caused by discrepancies in other stages of circuit processing. For example, in this work path variables are used to effectively "remove" Hadamards but other methods exist for Hadamard avoidance such as breaking the circuit into partitions that alternate between columns of Hadamards and regions composed only of CNOT and T gates.

There are many avenues for future work on T count optimization. Over the remaining two years of this project I plan to investigate the following:

- The results for Lempel depend on the order that qubits are factorized and for LempelX, the order that columns pairs are searched. I wish to investigate whether there are any means of characterizing these variations in order to optimize both algorithms with respect to execution sequence.
- One avenue for improving T count optimization, especially for the universal construction, is to combine T count optimization algorithms with H count optimization such as in [1]. It seems intuitive that solutions to both problems would complement each other as Hadamards and T gates serve as mutual obstacles during gate synthesis.
- There may be other conditions for which columns of gate synthesis matrices can be eliminated. I wish to investigate this further.
- It is known that minimum distance decoding of $RM^*(n-4,n)$ is equivalent to determining the symmetric tensor rank of symmetric tensors of order 3. I will research the best available algorithms for performing this task.
- In its current form, LempelX does not use the optimal Reed-Muller decoder at any stage. I will modify LempelX in order to switch to the optimal decoder once the rank of the gate synthesis matrix is reduced to $\leq n_{RM}$ where n_{RM} is the practical limit on the number of qubits for the Reed-Muller decoder.

References

[1] N. Abdessaied, M. Soeken, and R. Drechsler. *Quantum Circuit Optimization by Hadamard Gate Reduction*, pages 149–162. Springer International Publishing, Cham, 2014.

[2] M. Amy, D. Maslov, and M. Mosca. Polynomial-time t-depth optimization of clifford+t circuits via matroid partitioning. *arXiv*, arXiv:1303.2042v2 [quant-ph], 2013.

[3] M. Amy, D. Maslov, M. Mosca, and M. Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *arXiv*, arXiv:1206.0758v3 [quant-ph], 2013.

[4] M. Amy and M. Mosca. T-count optimization and reed-muller codes. *arXiv*, arXiv:1601.07363v1 [quant-ph], 2016.

[5] S. Bravyi and J. Haah. Magic-state distillation with low overhead. *Physical Review A*, 86(052329), 2012.

[6] S. Bravyi and A. Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A*, 71(022316), 2005.

[7] E. T. Campbell and M. Howard. Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost. *Phys. Rev. A*, 95(022316), 2017.

[8] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill. Topological quantum memory. *arXiv*, arXiv:quant-ph/0110143, 2001.

[9] D. Gosset, V. Kliuchnikov, M. Mosca, and V. Russo. An algorithm for the t-count. *arXiv*, arXiv:1308.4134v1 [quant-ph], 2013.

[10] D. Gottesman. Theory of fault-tolerant quantum computation. *Physical Review A*, 57(1), 1997.

[11] D. Gottesman and I. L. Chuang. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature*, 402:390–393, 1999.

[12] A. Lempel. Matrix factorization over gf(2) and trace-orthogonal bases of gf(2). *SIAM J. Comput.*, 4(2), 1975.

[13] D. Maslov. "reversible logic synthesis benchmarks page". http://webhome.cs.uvic.ca/~dmaslov/. 2011.