Polynomial-time T-depth Optimization of Clifford+T circuits via Matroid Partitioning

Matthew Amy,[1] Dmitri Maslov,[2] and Michele Mosca[3,4]

[1] *Department of Computer Science*
University of Toronto, Toronto, Ontario, Canada

[2] *National Science Foundation*
Arlington, Virginia, USA

[3] *Institute for Quantum Computing, and Dept. of Combinatorics & Optimization*
University of Waterloo, Waterloo, Ontario, Canada

[4] *Perimeter Insitute for Theoretical Physics*
Waterloo, Ontario, Canada

December 13, 2013

Abstract

Most work in quantum circuit optimization has been performed in isolation from the results of quantum fault-tolerance. Here we present a polynomial-time algorithm for optimizing quantum circuits that takes the actual implementation of fault-tolerant logical gates into consideration. Our algorithm re-synthesizes quantum circuits composed of Clifford group and T gates, the latter being typically the most costly gate in fault-tolerant models, e.g., those based on the Steane or surface codes, with the purpose of minimizing both T-count and T-depth. A major feature of the algorithm is the ability to re-synthesize circuits with additional ancillae to reduce T-depth at effectively no cost. The tested benchmarks show up to 65.7% reduction in T-count and up to 87.6% reduction in T-depth without ancillae, or 99.7% reduction in T-depth using ancillae.

1 Introduction

Quantum computers have the potential to efficiently solve important computational problems, including integer factorization [29] and quantum simulation [18], for which there are no known efficient classical algorithms. However, even with recent advances in quantum information processing technologies [6, 7, 8, 26], the prospects of scalable quantum computing without some systematic way of mitigating physical errors and noise are bleak.

The active and vibrant fields of quantum error correction and fault-tolerance provide such tools for constructing scalable quantum computers. By combining physical qubits through the use of error correcting codes and providing fault-tolerant logical operations, larger computations can be achieved with high fidelity – by concatenating codes, or in topological codes by increasing code distance – provided the physical operations achieve a certain threshold fidelity. With recent improvements to fault-tolerant thresholds [5, 14, 15], scalable quantum computation is becoming

more and more viable, resulting in a growing need for efficient automated design tools targeting fault-tolerant quantum computers.

Quantum circuit synthesis and optimization is particularly important, given the prevalence of the circuit model of quantum computation, but previous work has been largely isolated from the unique concerns of fault-tolerance. While at the physical level, coupled gates are generally the hardest to perform, most of the common quantum error-correcting codes have efficient $CNOT$ implementations. Moreover, for fault-tolerant models based on (double even, self-dual) CSS codes, e.g., the popular Steane code, as well as the promising surface codes, the Clifford group can be implemented as logical gates with little cost [14, 33].

For universal quantum computing, however, at least one non-Clifford group gate is needed, which typically requires large ancilla factories and gate teleportation to implement fault-tolerantly. As the non-Clifford T gate has known constructions in most of the common error correction schemes, the standard universal fault-tolerant gate set is taken to be "Clifford + T". Given the high cost of the fault tolerant implementations of the T gate [1, 14], exceeding the cost of Clifford group gates by as much as a factor of a hundred or more, it has recently been proposed that efficient circuits should minimize the number of T gates, and more specifically the number of T gates that cannot be performed in parallel [2, 13] – we define these metrics as a circuit's T-count and T-depth, respectively. Indeed, Fowler [13] shows how to perform fault-tolerant computations in time proportional to one round of measurement per layer of T gates, and as a result the T-depth directly determines a circuit's runtime. Likewise, reducing the number of T gates reduces the number of ancilla states that require preparation, vastly reducing circuit volume and at the same time increasing the fidelity of the computation. While the primary purpose of our work is to optimize T-depth (circuit runtime), our algorithm also provides significant reductions to T-count (circuit volume).

Some recent work has been done concerning minimization of T-depth [2, 28], though these previous results focus on finding small optimal two- and three-qubit circuits [2], and on classes of circuits that can be parallelized to T-depth 1 by adding ancillae [2, 28]. By contrast, we report a scalable automated tool for the optimization of T-depth that functions with or without ancillae, and is not limited to a few qubits or a specific class of circuits. In particular, we present a polynomial-time algorithm for optimizing both the T-depth and T-count of quantum circuits composed of Clifford group and T gates. The algorithm also makes automatic use of ancillae to optimize T-depth, with the addition of ancillae typically decreasing the runtime of our software implementation. Our experiments show on average 61.1% reduction in T-depth and 39.9% reduction in T-count without adding any ancillae, using the available benchmarks. When the use of ancillae is allowed, the average T-depth reduction is demonstrated to be as high as 80.7% (the more ancillae are allowed the more parallelization becomes possible, in some cases reducing T-depth by as much as 99.7%).

The rest of this paper is structured as follows: Section 2 reviews some background on quantum and reversible computation, and introduces the notations we will use. Sections 3 and 4 describe the algorithmic core – a procedure that optimally parallelizes the T gates in a circuit composed of $CNOT$ and T gates by performing matroid partitioning. Section 5 develops a heuristic extending the optimal $\{CNOT, T\}$ core to a universal gate set, while Section 6 describes the final algorithm. Section 7 reports our experimental results, and Section 8 concludes the paper.

2 Preliminaries

We begin by reviewing some basic facts about quantum and reversible circuits necessary for this paper.

In the classical circuit model, the state of a system of n bits is represented as a binary string of length n, with classical gates corresponding to operators that map length-n binary strings to length-m binary strings. More precisely, length-n binary strings are vectors of $\mathbb{F}_2^n$, where $\mathbb{F}_2$ is the two-element finite field with addition corresponding to Boolean exclusive-OR (EXOR, $\oplus$) and multiplication corresponding to Boolean AND ($\wedge$). We then represent classical gates as operators $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$, and we typically refer to f as a (classical) function. For brevity, if $m = 1$ we call f *Boolean*.

The quantum circuit model, one of the prominent models of quantum computation [24], generalizes the classical circuit model to deal with quantum effects. In particular, it describes the state space of a system of n qubits as a vector in a 2^n-dimensional complex vector space $\mathcal{H}$ spanned by the (classical) n bit states. By convention, we refer to the classical states as the standard or computational basis of $\mathcal{H}$ and write them in Dirac notation: $|x\rangle, x \in \mathbb{F}_2^n$.

In contrast to the classical circuit model, quantum gates are restricted to a subset of all operators on $\mathcal{H}$ – specifically, quantum gates are linear operators $U : \mathcal{H} \to \mathcal{H}$ that preserve the L_2 norm. Such operators U satisfy $U^\dagger U = UU^\dagger = I$, where $U^\dagger$ denotes the adjoint of U, and are known as *unitary*. Given that unitary operators are invertible, we see that the subset of quantum transformations that permute the computational basis states are exactly the set of invertible classical transformations – we call such functions *reversible*, with the intuition that any computation performed by reversible functions can be undone or reversed. The *Toffoli* gate,

$$\Lambda_2(X) : |x\rangle|y\rangle|z\rangle \mapsto |x\rangle|y\rangle|z \oplus (x \wedge y)\rangle,$$

is an example of a reversible function.

We can also have classical/quantum computations that use *ancillae* – being bits/qubits that can be initialized to the $0/|0\rangle$ or $1/|1\rangle$ state and act as a temporary register. Without loss of generality, we require that all ancillae are initialized in the $0/|0\rangle$ state. In the case of a circuit with n bits, m of which are data bits (i.e. $n - m$ is the number of ancillae), we describe the state space as some subspace V of $\mathbb{F}_2^n$ with dimension m. We will typically use n to represent the total number of bits in a system, and m to refer to the number of data bits.

While all reversible classical gates are linear as operators over $\mathcal{H}$, they need not be linear as operators over $\mathbb{F}_2^n$. In particular, we call $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ *linear* if $f(x \oplus y) = f(x) \oplus f(y)$. For instance, the reversible *controlled-NOT* gate

$$CNOT : |x\rangle|y\rangle \mapsto |x\rangle|x \oplus y\rangle$$

is linear over $\mathbb{F}_2^n$. It is a known result that the set of all linear reversible functions are those that can be computed by a circuit consisting of only $CNOT$ gates [25].

Throughout this paper we will also be interested in linear Boolean functions and their relation to linear reversible functions. For convenience, we refer to the set of n-ary linear Boolean functions $\mathbb{F}_2^n \to \mathbb{F}_2$ as the *dual vector space* $(\mathbb{F}_2^n)^*$ of $\mathbb{F}_2^n$, and note that a linear Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be represented as a row vector over $\mathbb{F}_2^n$ – i.e., x^T for some $x \in \mathbb{F}_2^n$. Furthermore, for a set of linear Boolean functions $S \subseteq (\mathbb{F}_2^n)^*$, we define $\text{rank}(S)$ as the maximum number of independent (row) vectors in S, or equivalently the dimension of the subspace V^* spanned by S.

As this paper is concerned with the optimization of quantum circuits, we also define some quantum gates commonly used in fault tolerant models. In particular, we define the T and Hadamard gates,

$$T : |x\rangle \mapsto e^{\frac{i\pi x}{4}}|x\rangle, \quad H : |x\rangle \mapsto \frac{|0\rangle + (-1)^x|1\rangle}{\sqrt{2}}.$$

We will show that circuits over the gate set $\{CNOT, T\}$ implement linear reversible functions with discrete phases corresponding to the eighth roots of unity. A side result of this paper is a proof that $\{CNOT, T\}$ circuits can be simulated on a classical computer in polynomial time. If this set is further extended with the Hadamard gate we achieve a gate set that is universal for quantum computation. We call $\{H, CNOT, T\}$ the "Clifford + T" gate set, as it contains the Clifford group generators $\{H, P := T^2, CNOT\}$ along with the T gate. Moreover, $\{H, CNOT, T\}$ is a minimal generating set for the Clifford + T gate set.

Since quantum gates are commonly defined by unitary matrices, we provide the equivalent matrix definitions below:

$$H := \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, CNOT := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

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

2.1 Computable sets of linear Boolean functions

While every linear reversible function f over n inputs can be written as n linear Boolean functions, i.e.

$$f(a_1, a_2, ..., a_n) = (f_1(a_1, ..., a_n), ..., f_n(a_1, ..., a_n)),$$

not every set of linear Boolean functions defines a reversible function. For instance, $f(a_1, a_2, a_3) = (a_1, a_2, a_1 \oplus a_2)$ is irreversible since the input a_3 is effectively destroyed. It is easy to verify that a set of n linear Boolean functions forms the outputs of an n-ary linear reversible function if and only if they have rank equal to the dimension of the input space.

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

$$f(a_1, a_2, \dots, a_n) = (f_1(a_1, \dots, a_n), \dots, f_n(a_1, \dots, a_n))$$

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

Since the unitary quantum circuit model is reversible, a set of linear Boolean functions S can only be computed simultaneously (i.e. there exists a quantum circuit implementing the transformation $|a_1 a_2 ... a_n\rangle \mapsto |b_1 b_2 ... b_n\rangle$ where for each $f \in S$, $f(a_1, a_2, ..., a_n) = b_i$ for some i) if it defines a reversible function. We call such a set of linear Boolean functions *(reversibly) computable* over a particular input space V – as we will be concerned strictly with reversible computations, we frequently omit the qualifier *reversible*.

We may also want compute a set S of linear Boolean functions with a linear reversible function on $n > |S|$ qubits. In this case, since every n-ary linear reversible function corresponds to some

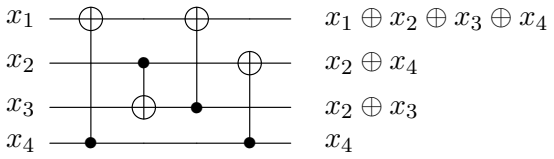


Figure 1: A circuit computing the functions $x_1 \oplus x_2 \oplus x_3 \oplus x_4$, $x_2 \oplus x_4$, and $x_2 \oplus x_3$.

set of n linear Boolean functions, a linear reversible function computes S if and only if there exists some computable superset S' of S. Equivalently from Lemma 1, a set $S \subseteq (\mathbb{F}_2^n)^*$ is (reversibly) computable over a subspace V of $\mathbb{F}_2^n$ if and only if there exists a superset S' of S with cardinality n such that $\text{rank}(S') = \dim(V)$.

Before moving on, we establish the condition under which a computable superset of linear Boolean functions exists. Given a set $S = \{f_1, f_2, ..., f_k\} \subseteq (\mathbb{F}_2^n)^*$ and subspace V, we can observe that we only need to find some $f_{k+1}, ..., f_n \in (\mathbb{F}_2^n)^*$ such that $\text{rank}(\{f_1, f_2, ..., f_n\}) = \dim(V)$. It is not hard to see that such $f_{k+1}, ..., f_n$ exist if and only if the number of linearly independent vectors in $(\mathbb{F}_2^n)^*$ needed is at most $n - k$.

Lemma 2. *Given a subspace V of $\mathbb{F}_2^n$ and a set of linear Boolean functions $S \subseteq (\mathbb{F}_2^n)^*$, there exists a superset S' of S with cardinality n such that* $\text{rank}(S') = \dim(V)$ *if and only if*

$$\dim(V) - \text{rank}(S) \leq n - |S|. \tag{1}$$

We can note that inequality (1) implies $|S| = \text{rank}(S)$, i.e., S is linearly independent, when $\dim(V) = n$.

3 $\{CNOT, T\}$ circuits

We first consider circuits over the gate set $\{CNOT, T, P := T^2, Z := T^4, T^\dagger := T^7, P^\dagger := T^6\}$, as they have a particular property that will be crucial to synthesizing low T-depth circuits. We remind the reader that the even powers of the T gate are all Clifford gates, whereas all odd powers lie outside the Clifford group. This is an essential observation for practical considerations. Furthermore, no power of the T gate requires more than a single non-Clifford T gate to implement. We usually omit the extraneous gates and refer to this gate set by the generating set $\{CNOT, T\}$.

It can be observed that since $CNOT|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle$ and $T|x\rangle = e^{\frac{i\pi x}{4}}|x\rangle$ for $x, y \in \mathbb{F}_2$, a $\{CNOT, T\}$ circuit can be described as computing a linear reversible function on the input basis state, with an added phase that is some power of $\omega := e^{i\pi/4}$. Stated more precisely [2],

Lemma 3. *A unitary $U \in U(2^n)$ is exactly implementable by an n-qubit circuit over $\{CNOT, T\}$ if and only if*

$$U|x_1x_2 \ldots x_n\rangle = \omega^{p(x_1,x_2,\ldots,x_n)}|g(x_1, x_2, \ldots, x_n)\rangle$$

where $x_1x_2...x_n \in \mathbb{F}_2^n$ and

$$p(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{l} c_i \cdot f_i(x_1, x_2, \ldots, x_n)$$

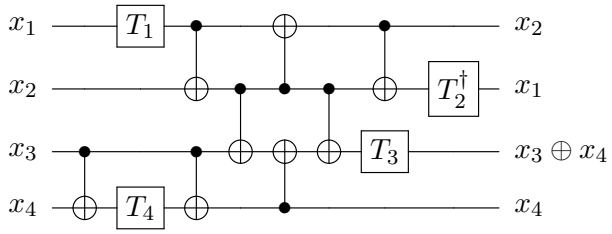
for some linear reversible function $g \in \mathbb{F}_2^n \to \mathbb{F}_2^n$ and linear Boolean functions $f_1, f_2, ..., f_l \in (\mathbb{F}_2^n)^*$ with coefficients $c_1, c_2, ..., c_l \in \mathbb{Z}_8$.

As a result of Lemma 3, we can fully characterize any unitary $U \in U(2^n)$ implementable by a $\{CNOT, T\}$ circuit with a set $S \subseteq \mathbb{Z}_8 \times (\mathbb{F}_2^n)^*$ of linear Boolean functions together with coefficients in $\mathbb{Z}_8$, and a linear reversible output function $g : \mathbb{F}_2^n \to \mathbb{F}_2^n$, with the interpretation

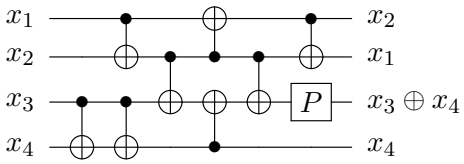
$$U_{\langle S,g \rangle} : |x_1 x_2 ... x_n\rangle \mapsto \omega^{\sum_{(c,f) \in S} c \cdot f(x_1, x_2, ..., x_n)} |g(x_1, x_2, ..., x_n)\rangle.$$

Moreover, S and g are efficiently computable given a $\{CNOT, T\}$ circuit, taking time linear in the number of qubits and gates. In computing S it also becomes apparent when T gates, possibly physically separated within the circuit, are applied to the same value and can thus be replaced by a phase gate such as $P : |x\rangle \mapsto \omega^{2 \cdot x} |x\rangle$. Our experimental results show that a large number of T gates can be removed in this way.

Example 1. Consider the following circuit.



If we track the effect of the $CNOT$ gates we see that T_1 and $T_2^\dagger$ (indices are used to mark different T gates within the circuit) are both applied to qubits in the state $|x_1\rangle$, resulting in a cumulative phase of $\omega^{x_1 + 7x_1} = \omega^{8x_1} = 1^{x_1} = 1$. As such, both gates can be removed. Likewise, T_3 and T_4 are both applied to qubits in the state $|x_3 \oplus x_4\rangle$, and their phases combine to $\omega^{2(x_3 \oplus x_4)}$ – this pair of T gates can thus be replaced with a single P gate. This results in an optimized circuit:



which may be optimized further to the form $SWAP(x_1, x_2)CNOT(x_4, x_3)P(x_3)$ by rewriting the linear reversible section.

Once S and g have been computed, the proof of Lemma 3 [2] gives a constructive method for synthesizing a circuit implementing $U_{\langle S,g \rangle}$. However, this naïve method of re-synthesis may end up with *worse* T-depth than the original circuit, despite possibly reduced T-count.

We can instead recall from Section 2 that if a subset $A \subseteq S$ is reversibly computable (i.e. if the linear Boolean functions in A are reversibly computable), we can construct a linear reversible function with outputs simultaneously computing the functions in A; in this case $|A|$ of the necessary phase factors could then be applied in parallel. Synthesis can thus proceed by partitioning S into computable subsets, then for each partition $A \subseteq S$ first compute a (reversible) superset of A with a stage of $CNOT$ gates; many efficient algorithms exist [4, 25, 20] that can decompose a linear reversible function into $CNOT$ gates. Next we apply the relevant phase gates in parallel to add the phase $\omega^{\sum_{(c,f) \in A} c \cdot f(x_1, x_2, ..., x_n)}$, and finally uncompute by reversing the $CNOT$ stage. Given that at most one T gate is used to implement any integer power of T, every partition will have a T-depth of at most 1.

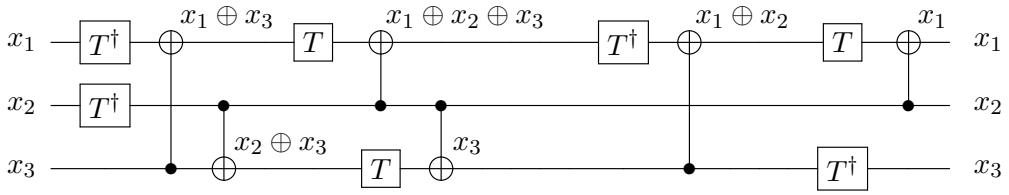


Figure 2: $\{CNOT, T\}$ circuit implementing the doubly controlled Z gate.

As a result, any unitary U implementable over $\{CNOT, T\}$ can be implemented in T-depth k where k is the minimum number of sets partitioning[1] $S_1 = \{(c, f) \in S | c \equiv 1 \mod 2\}$ into computable subsets, as elements in $S_0 = S \setminus S_1$ do not require T gates to implement. In fact, we can trivially see that given a specific set S and output g, k is the minimal T-depth, as any layer of T gates in a circuit implementing $U_{\langle S,g \rangle}$ corresponds to a computable subset of S.

It is important to note that while this method reduces and maximally parallelizes the T gates, it will not necessarily find the optimal T-count and by extension T-depth. Specifically, given a $\{CNOT, T\}$ circuit with phase defined by the set S there may be some distinct S' that defines an equivalent computation using fewer T gates. Consider, for instance, the set $S_\emptyset = \left\{(1, f) | f \in (\mathbb{F}_2^4)^*\right\}$. We note that the integer sum $\sum_{f \in (\mathbb{F}_2^4)^*} f(x_1, x_2, x_3, x_4)$ is equal to 8 for any non-zero $(x_1, x_2, x_3, x_4) \in \mathbb{F}_2^4$, and 0 for $(0, 0, 0, 0)$. Since $\omega^8 = \omega^0 = 1$, $S_\emptyset$ computes the trivial phase on every input and is therefore equivalent to the empty set $\emptyset$. In this case, all $|S_\emptyset| = 15$ T gates can then be removed.

In spite of this, a computer search proves that for $n < 4$, no two sets S, S' requiring distinct numbers of T gates define equivalent computations. As a result, we obtain a direct proof that the doubly controlled-Z gate requires 7 T gates to implement over $\{CNOT, T\}$ with any number of ancillae. For $n \geq 4$ however, the problem of minimizing T-count in $\{CNOT, T\}$ circuits reduces to a minimization problem over degree 1 polynomials in mixed arithmetic; moreover, since every such polynomial defines the global phase for some $\{CNOT, T\}$ circuit, the two problems are in fact equivalent.

3.1 Parallelizing $\Lambda_2(Z)$

To illustrate $\{CNOT, T\}$ re-synthesis, consider the circuit in Figure 2, implementing the doubly controlled-Z gate,

$$\Lambda_2(Z) : |x_1 x_2 x_3\rangle \mapsto \omega^{4 \cdot x_1 \wedge x_2 \wedge x_3} |x_1 x_2 x_3\rangle$$

over $\{CNOT, T\}$. We track the effect of each $CNOT$ gate on the state of the qubits, as annotated in the circuit. When a $T/T^\dagger$ gate is applied, we add/subtract a term in the exponent of ω corresponding to the state of the target qubit. The resulting transformation is then $\Lambda_2(Z) : |x_1 x_2 x_3\rangle \mapsto \omega^{p(x_1, x_2, x_3)} |x_1 x_2 x_3\rangle$, where

$$p(x_1, x_2, x_3) = x_1 + x_2 + x_3 - (x_1 \oplus x_2) - (x_1 \oplus x_3) - (x_2 \oplus x_3) + (x_1 \oplus x_2 \oplus x_3).$$

In fact, since $2 \cdot (x \wedge y) = x + y - x \oplus y$, we see that $\omega^{p(x_1, x_2, x_3)} = \omega^{4 \cdot (x_1 \wedge x_2 \wedge x_3)}$ as expected.

The T-stages in Figure 2 also correspond to a partition of S, specifically

$$\{\{-(x_1), -(x_2)\}, \{x_1 \oplus x_3, x_2 \oplus x_3\}, \{-(x_1 \oplus x_2 \oplus x_3)\}, \{x_1 \oplus x_2, -(x_3)\}\}.$$

[1]For the remainder of this paper we do not separate S_0 and S_1 to simplify the presentation, though our algorithm can be trivially modified to partition S_0 and S_1 separately.

It can easily be verified that for each subset S in this partition, $\dim(V) - \text{rank}(S) \leq n - |S|$ (i.e. S satisfies (1)), since $\dim(V) = n = 3$ and each subset is linearly independent. By contrast, the partition

$$\left\{ \begin{array}{c} \{-(x_1), -(x_2)\}, \{x_1 \oplus x_3, x_2 \oplus \\ x_3, x_1 \oplus x_2\}, \\ \{-(x_1 \oplus x_2 \oplus x_3)\}, \{-(x_3)\} \end{array} \right\}$$

could not have been synthesized as the set $\{x_1 \oplus x_3, x_2 \oplus x_3, x_1 \oplus x_2\}$ has rank 2, thus the mapping $|x_1x_2x_3\rangle \mapsto |(x_1 \oplus x_3)(x_2 \oplus x_3)(x_1 \oplus x_2)\rangle$ is not reversible.

While we haven't yet described how to find a minimal computable partition algorithmically, we can observe that the T-depth 3 partition

$$\left\{ \begin{array}{c} \{-(x_1), -(x_2), x_1 \oplus x_3\}, \{-(x_1 \oplus \\ x_2 \oplus x_3)\}, \\ \{-(x_3), x_1 \oplus x_2, x_2 \oplus x_3\} \end{array} \right\}$$

is computable since each subset satisfies (1), and is also minimal. If, however, we added an ancilla when re-synthesizing Figure 2, we can use the extra qubit to generate a smaller partition. In particular, we know $|x_1x_2x_3\rangle \mapsto |x_1x_2(x_1 \oplus x_3)\rangle$ is reversible so $|x_1x_2x_3\rangle|0\rangle \mapsto |x_1x_2(x_1 \oplus x_3)\rangle|0\rangle$ is as well. We can then compute the value $x_1 \oplus x_2 \oplus x_3$ into the ancilla with 2 $CNOT$ gates and apply 4 T gates (one for each qubit) at the same time. To connect this intuitive idea with equation (1), we observe that $n = 4$, $\dim(V) = 3$ since the input $|x_1x_2x_3\rangle|0\rangle$ spans a space of dimension 3, and

$$\text{rank}(\{-(x_1), -(x_2), x_1 \oplus x_3, -(x_1 \oplus x_2 \oplus x_3)\}) = 3,$$

so $\dim(V) - \text{rank}(S) = 0 \leq 0 = n - |S|$. Figure 3 shows the resulting circuit, implementing $\Lambda_2(Z)$ in T-depth 2.

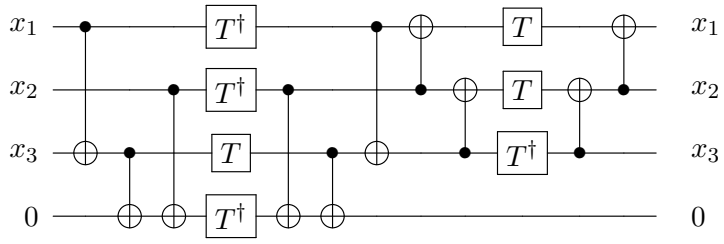


Figure 3: T-depth 2 implementation of Figure 2 with one ancilla.

4 Matroids

We now turn our attention to the problem of determining a minimal partition of a set of linear Boolean functions into computable sets. Due to the connection between the computability condition (1) on sets of linear Boolean functions and linear independence, we are able to phrase the problem as a *matroid partitioning* problem. To do so, we first introduce the concept of a *matroid*, an algebraic structure that generalizes the idea of linear independence in vector spaces.

Definition 1. A finite matroid is a pair (S, I) where S is a finite set and I is a set of subsets of S such that

1. $\emptyset \in I$.

2. For all $A, B \subset S$, if $A \in I$ and $B \subset A$, then $B \in I$.

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

It turns out that a set of linear Boolean functions, together with an independence relation defined by the equality (1), forms a matroid:

Lemma 4. *For any subspace V of $\mathbb{F}_2^n$ with dimension m and set of linear Boolean functions $S = \{f_1, f_2, \ldots, f_k\} \subseteq V^*$, let I denote the set*

$$\{A \subseteq S | m - \text{rank}\,(A) \leq n - |A|\}.$$

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

Proof. We verify that (S, I) satisfies all three conditions of Definition 1.

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

2. Suppose $A, B \subset S$, where $A \in I$ and $B \subset A$. Since $A = B \cup (A \setminus B)$ we see that

$$\begin{aligned} \text{rank}(A) &\leq \text{rank}(B) + \text{rank}(A \setminus B) \\ &\leq \text{rank}(B) + (|A| - |B|), \end{aligned}$$

   and so $\text{rank}(A) - \text{rank}(B) \leq |A| - |B|$. Since $m - \text{rank}(A) \leq n - |A|$, we see that

$$m \leq n + \text{rank}(A) - |A| \leq n + \text{rank}(B) - |B|,$$

   and thus $m - \text{rank}(B) \leq n - |B|$.

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

$$m - \text{rank}(B) \leq m - \text{rank}(A) \leq n - |A| < n - |B|,$$

   and so $m - \text{rank}(B \cup \{s\}) \leq n - |B \cup \{s\}|$ for any $s \in A$.

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

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

□

With Lemmas 2 and 4 we see that the problem of finding a minimal partition of the phase factors in a $\{CNOT, T\}$ circuit reduces to the more general matroid partitioning problem.

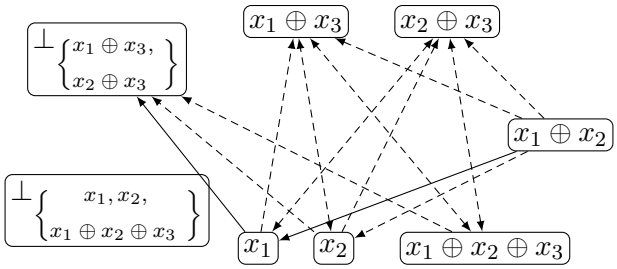


Figure 4: The directed graph G_s constructed when adding $x_1 \oplus x_2$ to the minimal partition $\{\{x_1 \oplus x_3, x_2 \oplus x_3\}, \{x_1, x_2, x_1 \oplus x_2 \oplus x_3\}\}$. A minimum length path is shown in solid lines, resulting in the new partition $\{\{x_1 \oplus x_3, x_2 \oplus x_3, x_1\}, \{x_1 \oplus x_2, x_2, x_1 \oplus x_2 \oplus x_3\}\}$.

4.1 Matroid partitioning

The matroid partitioning problem can be defined as follows:

Definition 2. (Matroid partitioning)
Given a matroid (S, I), find a partition $\{A_1, A_2, \ldots, A_k\}$ of S such that $A_i \in I$ for each $1 \leq i \leq k$ and for any other partition $\{A'_1, A'_2, \ldots, A'_{k'}\}$ into independent subsets, $k' \geq k$.

Matroid partitioning can, perhaps surprisingly, be solved in polynomial time, given an independence oracle for the matroid [12]. As a result, given an oracle for (1), the T gates in a $\{CNOT, T\}$ circuit can be optimally partitioned efficiently. Since the condition in Lemma 2 can be checked by using Gaussian elimination to compute the matrix rank in $O(n^3)$ time,[2] we thus see that a minimal partition can be computed in polynomial time. The rest of this section describes an algorithm for computing such a minimal partition.

The algorithm we use for solving the matroid partitioning problem is based on an algorithm due to Edmonds [12]. Given a matroid (S, I) and a minimal (matroid) partition P of $S' \subset S$, we take an element $s \in S \setminus S'$ not already partitioned and construct a minimal partition of $S' \cup \{s\}$. To create the new partition, we construct a directed graph G_s containing a vertex u for every $u \in S' \cup \{s\}$ as well as a vertex $\perp_p$ for every subset $p \in P$. The edges of G_s represent changes to the partition that are invariant under the property of each subset being independent. In particular, for any $u, v \in S' \cup \{s\}$ there is a directed edge $v \to u$ in G_s if and only if u is contained in some subset $p \in P$ and $(p \setminus \{u\}) \cup \{v\} \in I$, i.e. v can be added to p if we remove u. Additionally, given a subset $p \in P$ and element $u \in S' \cup \{s\}$, there exists an edge $u \to \perp_p$ if and only if $p \cup \{u\} \in I$. A path from s to $\perp_p$ for some subset p gives a set of updates to P that produce a valid partition P' of $S' \cup \{s\}$. Likewise, if there is no such path, there is no partition of size $|P|$ partitioning $S' \cup \{s\}$ (see [12] for a proof), and so a new subset $\{s\}$ is added to P. Figure 4 shows the full graph G_s for one iteration when computing a minimal partition for $\Lambda_2(Z)$ (Figure 2).

Rather than generating the graph G_s explicitly for each element s, we try to construct a path from s to some $\perp_p$ breadth-first (Algorithm 1). The time complexity of breadth-first search is $O(|E| + |V|)$ for a graph with edge set E and vertices V. We can note that there are $|S'| + |P| + 1$ vertices and at most $|S'|^2 + |P| \cdot (|S'| + 1) + (|S'| + |P|)$ edges in G_s, as well as the fact that

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

$|P| \leq |S'|$. Since each edge requires a single test for independence in $O(n^3)$ time, the breadth first search requires time in $O(|S'|^2 \cdot n^3 + |S'|)$.

Algorithm 1 Matroid partitioning algorithm

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

Algorithm 1 details the algorithm for adding an element s to a partition P of matroid (S, I) – the full algorithm follows by iteratively adding each element of the ground set to the initially empty partition, and correctness follows from the property that if P is minimal for (S, I), then the new partition P' is minimal for $(S \cup \{s\}, I)$. Since adding a single element to a partition of i elements takes $O\left((2i)^2 \cdot n^3 + 2i\right)$ time, and $\sum_i^{|S|} i^2 = \frac{|S|^3}{3} + \frac{|S|^2}{2} + \frac{|S|}{6}$, we see that Algorithm 1 can be used to partition the full set S in $O(|S|^3 \cdot n^3)$ time.

5 Extending to a universal gate set

In the previous sections we described a method for re-synthesizing $\{CNOT, T\}$ circuits that removes redundant T-gates by computing the total phase and parallelizing the phase gates through matroid partitioning. However, the usefulness of such an algorithm on its own is marred by the fact that $\{CNOT, T\}$ circuits are a restricted class of quantum circuits – in particular, they do not create superpositions or interference between basis states. To apply the optimization procedure to more

complex quantum circuits, we extend these ideas to deal with the universal gate set $\{H, CNOT, T\}$.

We recall that a Hadamard gate H has the effect

$$H : |x_1\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{x_2 \in \mathbb{F}_2} \omega^{4 \cdot x_1 \cdot x_2} |x_2\rangle$$

on a basis state $x_1 \in \mathbb{F}_2$. We call x_2 a *path variable*, following in the tradition of similar representations of quantum circuits [10, 27], called *sum over path* representations. We note that the state $|x_1\rangle$ effectively ceases to exist, having been replaced with $|x_2\rangle$. Circuits over $\{H, CNOT, T\}$ can then be described by a phase polynomial and set of linear Boolean outputs, similar to Lemma 3.

Lemma 5. *If a unitary $U \in U(2^n)$ is exactly implementable by an n-qubit circuit over $\{H, CNOT, T\}$ with k H gates, then for $x_1 x_2 ... x_n \in \mathbb{F}_2^n$,*

$$U|x_1 x_2 ... x_n\rangle = \frac{1}{\sqrt{2^k}} \sum_{x_{n+1} ... x_{n+k} \in \mathbb{F}_2^k} \omega^{p(x_1, x_2, ..., x_{n+k})} |y_1 y_2 ... y_n\rangle$$

where $y_i = h_i(x_1, x_2, \ldots, x_{n+k})$ and

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

for some linear Boolean functions h_i, f_i, g_i and coefficients $c_i \in \mathbb{Z}_8$. The k path variables $x_{n+1}, ..., x_{n+k}$ result from the application of Hadamard gates.

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

It can be noted that unlike Lemma 3, the converse is not true – some computations of the form in Lemma 5 do not define unitary transformations.

Synthesis of a circuit based on the above representation is more challenging. In particular, the Hadamard gates in effect destroy values and create new ones, changing the state space to some new subspace of $\mathbb{F}_2^{n+k}$, possibly with greater dimension if the destroyed value was linearly dependent with the other qubits (e.g. an initialized ancilla). Each linear Boolean function is likewise computable only in some of the possible state spaces. To re-synthesize we then need to apply Hadamard gates in such a way as to be able to pick up each phase factor and end in the state space span $(\{y_1, y_2, ..., y_n\})$.

Since the application of a Hadamard gate changes the state space, the state of each qubit must be chosen so that the qubits span a suitable space afterwards. As an illustration consider the transformation

$$|x_1 x_2\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{x_3 \in \mathbb{F}_2} \omega^{4 \cdot x_3 \cdot x_2} |(x_1 \oplus x_2) x_3\rangle.$$

We could achieve the correct phase by applying a Hadamard gate on the second qubit first, but the resulting state would be $|x_1 x_3\rangle$, from which we cannot directly construct the output state $|(x_1 \oplus x_2) x_3\rangle$. The simplest way to choose a suitable state for each qubit before applying a Hadamard gate is to use the qubit's state in the original circuit, though there may be other ways

of computing such states. During re-synthesis we then transform the state to match the state in the original circuit before a particular Hadamard gate is applied.

In this sense, the Hadamard gates are fixed by the original circuit and the re-synthesis process needs to place the remaining terms of the phase (i.e. $c_i \cdot f_i(x_1, x_2, ..., x_{n+k})$ in between them. One approach is to use the greedy nature of Algorithm 1 to maintain a partition of those functions that are computable in the current state space of the circuit. Specifically, we iterate through the Hadamard gates and for each one we partition any elements that are in the new state space – this step relies on the fact that Algorithm 1 is greedy to avoid having to repartition the elements that were already in the old state space. For any block in the partition containing functions that will not be computable after the next Hadamard gate, we remove the block and synthesize a $\{CNOT, T\}$ circuit applying those phases. A more detailed description is given in Section 6.

While this method is heuristic, we note that the partition is always minimal for the set of currently computable functions. In particular, Algorithm 1 produces a minimal partition when given a minimal partition, and removing blocks from a partition does not affect minimality – given a subset P' of a minimal partition P, if there existed a partition P_0 of the elements in P' such that $|P_0| < |P'|$, then $P_0 \cup P \setminus P'$ is a partition of the elements in P into fewer sets.

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

Lemma 6. *Given a subspace V of $\mathbb{F}_2^n$ with $\dim(V) = m$ and a set of linear Boolean functions $S \subseteq V^*$, let*

$$I_i = \{A \subseteq S | i - \text{rank}\,(A) \leq n - |A|\}.$$

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

Proof. Suppose there exists some partition P_0 of (S', I_{m+1}) with $|P_0| < |P'|$. We then see that one element of $S \setminus S'$ can be added to any $A \in P_0$ to give a set in I_m. In particular, consider any $A \in P_0$. Since $m + 1 - \text{rank}(A) \leq n - |A|$ we see that

$$m - \text{rank}(A) \leq n - |A| - 1 = n - |A \cup \{s\}|$$

for any $s \in S \setminus S'$. We also note that $n - |A \cup \{s\}| \geq 0$ as required, since any subset of S has rank at most m, so for any $A \in I_{m+1}$, $d + 1 - \text{rank}(A) > 0$ and thus $|A| < n$. Therefore, for any $A \in P_0, s \in S \setminus S'$ we have $A \cup \{s\} \in I_m$.

Next we note that for any $T \subseteq S$ there exists a partition of (T, I_m) with size at most $|T| - 1$. This is a simple result of the fact that $m < n$, as any subset $A \subseteq T$ of size 2 has rank at least 1, so

$$m - \text{rank}(A) \leq m - 1 \leq n - 2 = n - |A|.$$

Additionally, any size 1 subset of T is trivially independent under I_m.

Thus, since we can add one element to every partition in P_0, and we can partition the remaining $|S \setminus S'| - |P_0|$ elements into at most $|S \setminus S'| - |P_0| - 1$ partitions, we see that there exists a partition of (S, I_m) with size at most

$$|P_0| + (|S \setminus S'| - |P_0| - 1) = |S \setminus S'| - 1.$$

Since $|S \setminus S'| - 1 < |P|$ we obtain a contradiction. □

6 The Tpar algorithm

In the last section we described a heuristic for optimizing T-count and depth over the gate set $\{H, CNOT, T\}$. In this section, we present the concrete algorithm, Tpar, and enlarge the gate set to include the Pauli gates

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

We ignore the irrelevant global phase i in $Y = iXY$, though it can be recovered by applying $XPXP = iI$ to any qubit. Appendix A gives a demonstration of the Tpar algorithm on a simple circuit.

Recall that in the computational basis, the input space of a circuit with n qubits, $n-m$ ancillae and k Hadamard gates is a dimension m subspace V of $\mathbb{F}_2^{n+k}$. We represent the state of a qubit as a vector in the dual space of $\mathbb{F}_2^{n+k}$, $\mathbb{F}_2^{(n+k)*}$, along with a Boolean value b, called the parity, which is used to record bit flips. Specifically, we note that $X : |x\rangle \mapsto |1 \oplus x\rangle$, so we can model bit flips with a single parity bit. We denote the set of states $\mathbb{F}_2 \times \mathbb{F}_2^{(n+k)*}$ as $\mathcal{S}$.

Given a Clifford + T circuit C written as a sequence of gates in $\{X, Y, Z, P, P^\dagger, H, CNOT, T, T^\dagger\}$ we first compute a triple $\langle S, Q, H\rangle \in \mathcal{D}$ representing C. $S = \{(c, f) | c \in \mathbb{Z}_8, f \in \mathcal{S}\}$ stores the T^k phase factors as linear Boolean functions with parity and multiplicity, $Q = (g_1, g_2, ..., g_n) \in \mathcal{S}^n$ tracks the state of each qubit, and $H = (h_1, h_2, ...h_k)$ gives a sequence of Hadamard gates where each entry h_i stores the input and output states, $h_i.Q_I$ and $h_i.Q_O$ respectively. We define the initial state of the circuit as $Q_0 = ((0, x_1), (0, x_2), ..., (0, x_m), (0, 0), ..., (0, 0))$, which is understood as the state $|x_1 x_2 ... x_m\rangle |0\rangle^{\otimes n-m}$. To compute $\langle S, Q, H\rangle$, we use the function $[\![U]\!] : \mathcal{D} \to \mathcal{D}$ (Figure5) to sequentially update the triple $\langle S, Q, H\rangle$ for each gate U in the circuit, starting from the initial value $\langle \emptyset, Q_0, \emptyset\rangle$.

The Tpar algorithm (Algorithm 2) proceeds as follows: after computing $\langle S, Q, H\rangle$, we synthesize a new circuit by iterating through the Hadamard gates in H while updating a partition P of the functions of S that are computable in the current subcircuit. In particular, we divide S into S_P and S_{-P}, where S_P are the already partitioned elements and S_{-P} are those not already partitioned. For a given $h_i = \{Q_I, O_O\}$, for every $(c, f) \in S_{-P}$ we check whether $f \in \text{span}(Q_I)$. If so, we add (c, f) to the current partition using Algorithm 1 with the independence relation $A \subseteq S \in I$ if and only if $\text{rank}(Q_I) - \text{rank}(A) \leq n - |A|$. After partitioning, we update S_P and S_{-P} accordingly. The tests for inclusion of each function f in $\text{span}(h_i.Q_I)$ requires $O\left(|S_{-P}| \cdot (n+k)^3\right)$ time, and we can loosely bound the partitioning time as the time to partition the entire set S using Algorithm 1, $O(\left(|S|^3 \cdot (n+k)^3\right)$; since $|S_{-P}| \leq |S|$ the entire step thus takes $O\left(|S|^3 \cdot (n+k)^3\right)$ time. A tighter analysis is possible, though we omit it as the algorithm is heuristic in nature.

We next iterate through P and for each block $A \in P$, if $f \notin \text{span}(Q_O)$ for some $(c, f) \in A$ we remove A from the partition and synthesize a circuit computing the relevant phase factors. While we

Algorithm 2 T-parallelization algorithm

function TPAR(Clifford + T circuit C)
  $C' := \emptyset$
  $\langle S, Q, H\rangle := [\![C_{|C|}]\!] \cdots [\![C_1]\!] \langle \emptyset, Q_0, \emptyset\rangle$
  Set $S_P := \emptyset$; $S_{-P} := S$; $P := \emptyset$
  for each $1 \le i \le k$ **do**
    $I := \{A \subseteq S | \operatorname{rank}(h_i.Q_I) - \operatorname{rank}(A) \le n - |A|\}$
    for each $(c, f) \in S_{-P}$ **do**
      if $f \in \operatorname{span}(h_i.Q_I)$ **then**
        $P :=$Partition$((c, f), P, (S_P, I))$
        $S_P := S_P \cup \{(c, f)\}$; $S_{-P} := S_{-P} \setminus \{(c, f)\}$
      end if
    end for
    for each $A \in P$ **do**
      if $i = k$ or $\exists (c, f) \in A$ s.t. $f \notin \operatorname{span}(h_i.Q_O)$
      then
        Append(C', Synthesize$(A, h_i.Q_I, h_i.Q_I)$)
        $P := P \setminus A$
      else if $\operatorname{rank}(h_i.Q_O) - \operatorname{rank}(A) > n - |A|$ **then**
        Choose $(c, f) \in A$ such that
          $\operatorname{rank}(A) = \operatorname{rank}(A \setminus \{(c, f)\})$
        $A := A \setminus \{(c, f)\}$
        $S_P := S_P \setminus \{(c, f)\}$; $S_{-P} := S_{-P} \cup \{(c, f)\}$
      end if
    end for
    Append(C', Synthesize$(\emptyset, h_i.Q_I, h_i.Q_O)$)
  end for
  return C'
end function

function SYNTHESIZE(A, Q_I, Q_O)
  /* Synthesize a circuit implementing
   $U : |Q_I\rangle \mapsto \omega^{\sum_{(c,(b,f))\in A} c \cdot b \oplus f(x_1, x_2, \ldots, x_{n+k})} |Q_O\rangle$ */
  Compute $A' \supseteq A$ s.t. $\operatorname{rank}(A') = \operatorname{rank}(Q_I), |A'| = n$
  Synthesize $\{CNOT, X\}$ circuit $C_1 : |Q_I\rangle \mapsto |A'\rangle$
  Synthesize $\{Z, P, T\}$ circuit
    $C_2 : |A'\rangle \mapsto \omega^{\sum_{(c,b,f)\in A'} c \cdot b \oplus f(x_1, x_2, \ldots, x_{n+k})} |A'\rangle$
  Synthesize $\{CNOT, X, H\}$ circuit $C_3 : |A'\rangle \mapsto |Q_O\rangle$
  Return $C_1 C_2 C_3$
end function

$$
\begin{aligned}
&[\![X_i]\!]\langle S, Q, H\rangle = \langle S, (g_1, ..., g_{i-1}, 1 \oplus g_i, ..., g_n), H\rangle \\
&[\![Z_i]\!]\langle S, Q, H\rangle = \langle S \uplus \{(4, g_i)\}, Q, H\rangle \\
&[\![Y_i]\!]\langle S, Q, H\rangle = \langle S \uplus \{(4, g_i)\}, Q', H\rangle \qquad \text{where } Q' = (g_1, ..., g_{i-1}, 1 \oplus g_i, ..., g_n) \\
&[\![P_i]\!]\langle S, Q, H\rangle) = \langle S \uplus \{(2, g_i)\}, Q, H\rangle \\
&[\![P_i^\dagger]\!]\langle S, Q, H\rangle) = \langle S \uplus \{(6, g_i)\}, Q, H\rangle \\
&[\![H_i]\!]\langle S, Q, (h_1, h_2, ..., h_j)\rangle) = \langle S, Q', H'\rangle \qquad \text{where } Q' = (g_1, ..., g_{i-1}, (0, x_{j+i}), ..., g_n) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad H' = (h_1, h_2, ..., h_j, \{Q_I = Q, Q_O = Q'\}) \\
&[\![CNOT_{(i,j)}]\!]\langle S, Q, H\rangle) = \langle S, Q', H\rangle \qquad \text{where } Q' = (g_1, ..., g_{j-1}, g_j \oplus g_i, ..., g_n) \\
&[\![T_i]\!]\langle S, Q, H\rangle) = \langle S \uplus \{(1, g_i)\}, Q, H\rangle \\
&[\![T_i^\dagger]\!]\langle S, Q, H\rangle) = \langle S \uplus \{(7, g_i)\}, Q, H\rangle
\end{aligned}
$$

Figure 5: Semantic function $[\![\cdot]\!]$. We define $S \uplus T$ as the union of S and T where any f such that $(c_1, f) \in S, (c_2, f) \in T$ is given coefficient $c_1 + c_2 \mod 8$. U_i denotes the gate U applied to qubit i and $CNOT_{(i,j)}$ specifies i as the control qubit and j as the target.

defer the discussion of the synthesis procedure for now, we remark that it requires $O\left((n+k)^3\right)$ time. Otherwise, if A is no longer an independent set under the new independence relation $A \subseteq S \in I$ if and only if $\text{rank}(Q_O) - \text{rank}(A) \leq n - |A|$, we remove a linearly dependent element from A and update S_P and S_{-P} so that the deleted element will be re-partitioned on the next iteration. As $\text{rank}(A)$ and a linearly dependent element can both be found with one application of Gaussian elimination, this step also requires $O\left((n+k)^3\right)$ time, and so the entire loop takes $O\left(|P| \cdot (n+k)^3\right)$ time.

Finally, we synthesize a circuit applying the Hadamard gate in $O\left((n+k)^3\right)$ time, and repeat the entire process for the next Hadamard. The entire algorithm, shown in Algorithm 2, thus runs in time

$$O\left(|C| \cdot n + k \cdot (n+k)^3 \cdot \left(|S|^3 + |P| + 1\right)\right).$$

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

6.1 Synthesizing partitions

We now describe the general synthesis procedure, SYNTHESIZE(A, Q_I, Q_O), from Algorithm 2. The procedure synthesizes a circuit with inputs Q_I and outputs Q_O applying the phases given by a computable partition A of linear Boolean functions.

The algorithm proceeds by first extending A with $n - |A|$ linear Boolean functions to form a set A' with rank equal to $\text{rank}(Q_I)$ – this is accomplished by using row operations to reduce A to a subset of Q_I, then adding the vectors in $Q_I \setminus A$. Next we synthesize a circuit computing A' by reducing Q_I and A' to the same basis using Gaussian elimination in $O((n+k)^3)$ time, where

addition of two rows corresponds to the application of a $CNOT$ gate, and parity changes correspond to X gates. The circuit reducing Q_I to this basis is applied forwards, while the circuit reducing A' is applied in reverse, giving a circuit mapping $|Q_I\rangle \mapsto |A'\rangle$. The phase factors are applied by constructing a combination of T, P and Z gates, corresponding to the relevant coefficients, then the circuit mapping $|Q_I\rangle$ to $|A'\rangle$ is reversed to compute $|Q_I\rangle$. In the case when $Q_O \neq Q_I$, the corresponding Hadamard gate is applied to compute the output $|Q_O\rangle$.

As alluded to before, we now see that the synthesis procedure has time complexity $O\left((n+k)^3\right)$ since it requires a constant number (three) of applications of Gaussian elimination. Moreover, the T-depth of the resulting circuit is 1.

As an important practical issue, Gaussian elimination based synthesis produces linear reversible circuits that are non-optimal in terms of the number of gates or depth, resulting in a potential increase in the number of $CNOT$ gates after re-synthesizing, as compared to the original design. While our focus was on the optimization of T gates, there exist algorithms, [20, 25], that produce more efficient circuits for linear reversible functions. Specifically, [25] provides an algorithm to synthesize linear reversible circuits with $\Theta(n^2/\log(n))$ gates, and [20] reports an $O(n)$-depth algorithm. More recently, [17] described an optimization procedure for stabilizer circuits that could be applied afterward to further optimize linear reversible and Clifford group subcircuits. In practical implementations an advanced linear reversible synthesis algorithm should be used. Compared to the optimization of T-depth, we considered the optimization of the linear reversible circuit stages to be a second order improvement and did not pursue it in this work.

7 Results

We implemented Algorithm 2 in C++[3] and used it to optimize various quantum circuits, specifically arithmetic and reversible ones, from the literature. Individual circuits were written in the standard fault-tolerant universal gate set $\{X, Y, Z, H, P, P^\dagger, CNOT, T, T^\dagger\}$, using the decompositions found in [2] where applicable. As most arithmetic circuits are dominated by Toffoli gates, we used the lowest T-depth implementation of the Toffoli without ancillae known [2].

Results are reported in Tables 1 and 2. They were generated in Debian Linux running on a quad-core 64-bit Intel Core i7 2.40GHz processor and 8 GB RAM. Table 1 gives gate counts for the circuits before and after optimization. Table 2 shows T-depths before and after optimization using either 0, n, or ∞ ancillae[4] where n denotes the original number of qubits in the circuit. The T-depth for each circuit before optimization was computed by parallelizing the T-gates and Toffoli gates by hand, and writing each group of parallel Toffoli gates in T-depth 3.

With no extra ancillae added, all the tested benchmarks show significant reductions in terms of both T-count and T-depth, with average reductions of 39.9% and 54.3% respectively. The algorithm is particularly effective in cases where adjacent Toffoli gates share either controls or targets, as many of the phases cancel – each of the GF(2^m) multipliers share this structure, and as a result show large reductions in T-count and T-depth. In fact, the Tpar algorithm will parallelize any GF(2^m) multiplication circuit to T-depth 2 when given sufficiently many ancillae, by noting that each such circuit contains two stages of Toffoli gates that result in one $\{CNOT, T\}$ stage each

[3] C++ Source code available at http://code.google.com/p/tpar/.

[4] In order to give an example of the trade-off between number of ancillae and the T-depth for some non-constant number of ancillae, we arbitrarily chose to illustrate results with n ancillae. Our implementation allows any other computable value.

after removing trivial identities. By comparison, since the Toffoli gates *cannot* be all written in parallel, the minimum T-depth achievable using T-depth 1 Toffoli implementations [28] is $12(m-1)$. Those circuits that mix controls and targets between adjacent Toffoli gates are less affected by the optimization, e.g., CSUM-MUX$_9$, as the Hadamard gates create barriers to T parallelization.

The runtimes show that algorithm scales well to large circuits, the largest tested circuit having 192 qubits, 28672 T gates and 8192 Hadamard gates. This stands in contrast to most previous efforts to optimize quantum circuits, which have generally been limited in usefulness to a few qubits and a small number of gates. While the inclusion of Hadamard gates adds significant complexity to the algorithm, it has actually reduced runtime of the algorithm compared to experiments where Tpar was applied only to $\{CNOT, T\}$ subcircuits which is likely a result of the greater T-count reductions. As a result, Tpar appears to be an effective heuristic algorithm for the large-scale optimization of fault-tolerant circuits.

We also tested our algorithm's ability to make use of ancillae to optimize T-depth (Table 2). For each of the benchmark circuits, we applied our algorithm with an added n ancillae, where the original circuit contained n qubits. We also report the minimum T-depth achievable for each circuit using our algorithm. It can be noted that our algorithm usually decreases in running time when ancillae are added, due to the reduced number of partitions and thus faster matroid partitioning. Specifically, when there are many ancillae, for the majority of the time when an item s is partitioned it can be directly added to one of the partitions in time $O(|P| \cdot (n+k)^3)$. The algorithm is thus very flexible, and the experimental data illustrates a great potential for exploring space-time trade-off in quantum circuits.

As an important application, our experiments include instances of the multiple control Toffoli gates, $\Lambda_k(X)$, which are widely used in the construction of reversible circuits. We report the results using two different implementations – the Barenco *et al.* implementation using $k-2$ ancillae with arbitrary initial states [3], and the Nielsen-Chuang implementation using $k-2$ ancillae initialized in the state $|0\rangle$ [24]. In both constructions the ancillae are returned to their initial state. Our optimization of the Barenco *et al.* version reduces the T-count from $7(4k-8)$ to $3(4k-8)+4$ and T-depth from $3(4k-8)$ to $4k-8$ with unbounded ancillae, in the instances we tried. Likewise, our optimization of the Nielsen-Chuang implementation reduced the T-count from $7(2k-3)$ to $4(2k-3)+3$ and T-depth from $3(2k-3)$ to $2k-3$. These formulae in fact hold for every $k \geq 3$, a result of the simple structure of the circuits. As the two versions use $4k-8$ and $2k-3$ sequential Toffoli gates, respectively, we note that Tpar parallelizes each Toffoli to T-depth 1 when sufficiently many ancillae are available. Moreover, the reductions in T-count can be observed to correspond directly to each shared target or control – in this way, Tpar achieves the same T count and depth reductions as the multiply-controlled gate construction reported in [28], but applies to more general cases and does not require implementing controls with $\Lambda_2(\pm iX)$ gates.

As a final remark, we note that our algorithm reproduces many of the previous results regarding the optimization of T-depth. In particular, Figure 6 shows the circuit produced by running Tpar on an implementation of the Toffoli gate. The circuit mirrors the T-depth 1 Toffoli reported in [28]. Moreover, the full range of T-depths possible with different numbers of ancillae can be observed, as seen in Figure 7. We also show a re-synthesized controlled-T gate [2] using one ancilla to reduce the T-depth from 5 to 3 (Figure 8), and a re-synthesized Barenco *et al.* implementation of the $\Lambda_3(X)$ gate using no ancillae (Figure 9).

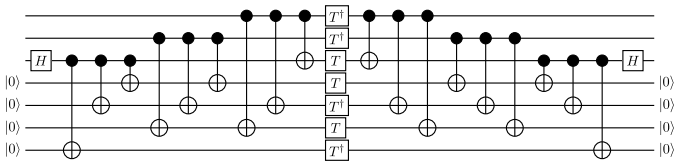


Figure 6: T-depth 1 implementation of the Toffoli gate.

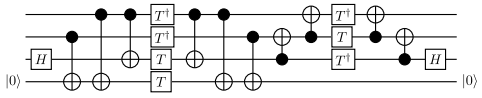


Figure 7: T-depth 2 implementation of the Toffoli gate.

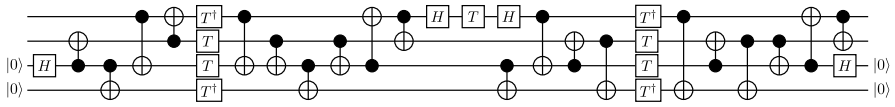


Figure 8: T-depth 3 implementation of the controlled-T gate (CNOT stages optimized by templates [22]).

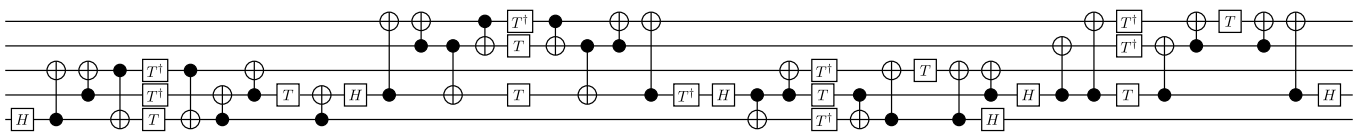


Figure 9: An optimized implementation of the $\Lambda_3(X)$ gate (CNOT stages optimized by templates [22]).

Table 1: T-count benchmarks. We report the gate counts after optimizing circuits with Tpar, using no extra ancillae. N specifies the number of qubits. x_C reports the number of $CNOT$ gates, x_T reports the number of T gates and x_U reports the number of other gates. x' denotes the number of gates after optimization.

Benchmark	N	x_C	x_T	x_g	Time (s)	x'_C	x'_T	x'_g	T-count reduction (%)
Mod 5_4 [21]	5	32	28	9	0.000	48	16	12	42.9
VBE-Adder$_3$ [32]	10	80	70	20	0.001	114	24	23	65.7
CSLA-MUX$_3$ [31]	15	90	70	20	0.001	425	62	21	11.4
CSUM-MUX$_9$ [31]	30	196	196	84	0.005	411	112	70	42.9
QCLA-Com$_7$ [11]	24	215	203	73	0.003	583	95	73	53.2
QCLA-Mod$_7$ [11]	26	441	413	132	0.008	1185	249	138	39.7
QCLA-Adder$_{10}$ [11]	36	273	238	86	0.018	737	162	73	31.9
Adder$_8$ [30]	24	466	399	126	0.007	920	215	153	46.1
RC-Adder$_6$ [9]	14	104	77	30	0.001	234	63	29	18.2
Mod-Red$_{21}$ [19]	11	121	119	58	0.001	301	73	51	38.7
Mod-Mult$_{55}$ [19]	9	55	49	36	0.000	166	37	20	24.5
$\Lambda_3(X)$ – [3]	5	28	28	8	0.000	54	16	12	42.9
– [24]	5	21	21	6	0.000	41	15	9	28.6
$\Lambda_4(X)$ – [3]	7	56	56	16	0.000	90	28	23	50.0
– [24]	7	35	35	10	0.000	63	23	16	34.3
$\Lambda_5(X)$ – [3]	9	84	84	24	0.001	132	40	34	52.4
– [24]	9	49	49	14	0.000	94	31	23	36.7
$\Lambda_{10}(X)$ – [3]	19	224	224	64	0.004	328	100	89	55.4
– [24]	19	119	119	34	0.002	232	71	58	40.3
GF(2^4)-Mult [23]	12	115	112	32	0.001	324	68	27	39.3
GF(2^5)-Mult [23]	15	179	175	50	0.004	535	111	36	36.6
GF(2^6)-Mult [23]	18	257	252	72	0.008	649	150	43	40.5
GF(2^7)-Mult [23]	21	349	343	98	0.031	992	217	36	36.7
GF(2^8)-Mult [23]	24	468	448	128	0.052	1256	264	40	41.1
GF(2^9)-Mult [23]	27	575	567	162	0.110	1701	351	44	38.1
GF(2^{10})-Mult [23]	30	709	700	200	0.227	2176	410	69	41.4
GF(2^{16})-Mult [23]	48	1856	1792	512	5.079	6592	1040	82	42.0
GF(2^{32})-Mult [23]	96	7291	7168	2048	602.577	33269	4128	166	42.4
GF(2^{64})-Mult [23]	192	28860	28672	8192	95447.466	180892	16448	334	42.6
Average									39.9
Maximum									65.7

8 Conclusion

We have described an algorithm for re-synthesizing Clifford + T circuits with reduced T-count and depth. The algorithm uses a representation of circuits based on linear Boolean functions, allowing T gates to be combined and then parallelized through the use of matroid partitioning algorithms. The algorithm has worst case runtime that is cubic in the number of T gates, qubits, and Hadamard gates, though our experiments show that the algorithm is sufficiently fast for practical circuit sizes.

Our benchmarks (Tables 1 and 2) show that large gains can be obtained in reducing the T-count and T-depth of quantum circuits. In some cases, T-count was reduced by as much as 65.7%, while using no additional ancillae the T-depth could be reduced by up to 87.6%. Furthermore, the benchmarks illustrate that ancillae can be used to parallelize T gates further, and given the runtimes reported the algorithm can be seen to provide substantial flexibility in exploring the trade-

Table 2: T-depth benchmarks. We report the T-depth after no optimization (original) and after optimization with 0 (cf. Table 1), n, or unbounded ancillae.

Benchmark	T-depth original	T-depth 0 ancilla	Red. (%)	Time (s)	T-depth N ancilla	Red. (%)	Time (s)	T-depth ∞ ancilla	Red. (%)
Mod 5_4 [21]	12	6	50.0	0.000	3	75.0	0.000	3	75.0
VBE-Adder$_3$ [32]	24	9	62.5	0.000	5	79.2	0.000	5	79.2
CSLA-MUX$_3$ [31]	21	8	61.9	0.004	4	81.0	0.001	4	81.0
CSUM-MUX$_9$ [31]	18	9	50.0	0.003	4	77.8	0.005	3	83.3
QCLA-Com$_7$ [11]	27	12	55.6	0.003	7	74.1	0.004	7	74.1
QCLA-Mod$_7$ [11]	57	29	49.1	0.008	14	75.4	0.010	14	75.4
QCLA-Adder$_{10}$ [11]	24	11	54.2	0.005	6	75.0	0.006	6	75.0
Adder$_8$ [30]	69	30	56.5	0.007	15	78.3	0.008	15	78.3
RC-Adder$_6$ [9]	33	22	33.3	0.002	11	66.7	0.001	11	66.7
Mod-Red$_{21}$ [19]	48	25	47.9	0.002	15	68.8	0.011	15	68.8
Mod-Mult$_{55}$ [19]	15	7	53.3	0.000	4	73.3	0.005	4	73.3
$\Lambda_3(X)$ – [3]	12	8	33.3	0.001	4	66.7	0.000	4	66.7
– [24]	9	6	33.3	0.000	3	66.7	0.001	3	66.7
$\Lambda_4(X)$ – [3]	24	13	45.8	0.001	8	66.7	0.002	8	66.7
– [24]	15	9	40.0	0.001	5	66.7	0.000	5	66.7
$\Lambda_5(X)$ – [3]	36	18	50.0	0.001	12	66.7	0.004	12	66.7
– [24]	21	12	42.9	0.001	7	66.7	0.001	7	66.7
$\Lambda_{10}(X)$ – [3]	96	43	55.2	0.005	32	66.7	0.032	32	66.7
– [24]	51	27	47.1	0.003	17	66.7	0.008	17	66.7
GF(2^4)-Mult [23]	36	6	83.3	0.001	4	88.9	0.001	2	94.4
GF(2^5)-Mult [23]	48	9	81.3	0.002	5	89.6	0.002	2	95.8
GF(2^6)-Mult [23]	60	9	85.0	0.005	5	91.7	0.003	2	96.7
GF(2^7)-Mult [23]	72	12	83.3	0.026	7	90.3	0.004	2	97.2
GF(2^8)-Mult [23]	84	13	84.5	0.035	7	91.7	0.006	2	97.6
GF(2^9)-Mult [23]	96	15	84.4	0.058	7	92.7	0.010	2	97.9
GF(2^{10})-Mult [23]	108	16	85.2	0.157	7	93.5	0.012	2	98.1
GF(2^{16})-Mult [23]	180	24	86.7	3.128	12	93.3	0.061	2	98.9
GF(2^{32})-Mult [23]	372	47	87.4	644.189	23	93.8	1.246	2	99.5
GF(2^{64})-Mult [23]	756	94	87.6	127287.329	44	94.2	78.641	2	99.7
Average			61.1			78.5			80.7
Maximum			87.6			94.2			99.7

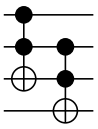
off between ancilla usage and T-depth. In the most extreme case and using unbounded ancillae, we were able to reduce the T-depth of GF(2^m)-Mult circuits from $12(m-1)$ to a constant of just 2. While the benchmarks demonstrated were all arithmetic or otherwise reversible operations, such operations typically require the majority of the resources in circuits for quantum algorithms of interest [16].

We close by noting that as a consequence of the Tpar algorithm, reducing the number of terms in the mixed arithmetic polynomials describing the phase corresponds directly to reducing the T-complexity of quantum circuits; in fact, it was observed that minimization of T-count in $\{CNOT, T\}$ circuits is equivalent to minimizing the number of odd coefficients in the phase polynomial. A natural avenue of future work is then to develop methods for optimizing such polynomials for T-count and depth. This work also represents the first instance, to the authors' knowledge, of the use of sum over paths style representations in quantum circuit synthesis and optimization. While this representation has already proven effective in optimizing circuit T-count and T-depth, the questions

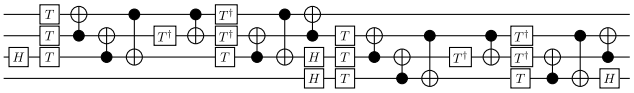
of synthesis for more general phases, e.g. the sum over paths representation of $\{H, CNOT, T\}$, and of the precise form of phase polynomials synthesizeable over the "Clifford + T" gate set remain. Moreover, we leave it as a topic of future research to find new applications to optimization over different gate sets and cost metrics. Efficient practical synthesis of linear reversible circuits is another important direction that would directly contribute to improving the results of this work.

A Parallelizing $(\Lambda_2(X) \otimes I)(I \otimes \Lambda_2(X))$

In this section we illustrate the workings of the Tpar algorithm using the following circuit:



We first expand this circuit by using the T-depth 3 Toffoli gate implementation [2]:



Next we compute $\langle S, Q, H \rangle$ by applying the function $[\![\cdot]\!]$ to each gate in sequence, starting with $\langle \emptyset, (x_1, x_2, x_3, x_4)), \emptyset \rangle$. The result is

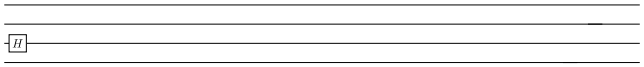
$$S = \left\{ \begin{array}{c} x_1, 2 \cdot x_2, x_5, 7 \cdot (x_1 \oplus x_2), 7 \cdot (x_1 \oplus x_5), \\ 7 \cdot (x_2 \oplus x_5), (x_1 \oplus x_2 \oplus x_5), x_6, x_7, 7 \cdot \\ (x_2 \oplus x_6), \\ 7 \cdot (x_2 \oplus x_7), 7 \cdot (x_6 \oplus x_7), (x_2 \oplus x_6 \oplus x_7) \end{array} \right\},$$

$$Q = (x_1, x_2, x_6, x_8),$$

$$H = (h_1, h_2, h_3, h_4) \quad \text{where}$$

$$h_1 = \{Q_I = (x_1, x_2, x_3, x_4), Q_O = (x_1, x_2, x_5, x_4)\},$$
$$h_2 = \{Q_I = (x_1, x_2, x_5, x_4), Q_O = (x_1, x_2, x_6, x_4)\},$$
$$h_3 = \{Q_I = (x_1, x_2, x_6, x_4), Q_O = (x_1, x_2, x_6, x_7)\},$$
$$h_4 = \{Q_I = (x_1, x_2, x_6, x_7), Q_O = (x_1, x_2, x_6, x_8)\}.$$

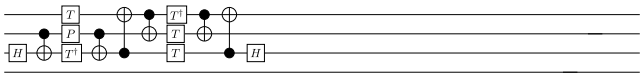
Starting with h_1, we see that the terms $x_1, 2 \cdot x_2, 7 \cdot (x_1 \oplus x_2)$ are computable, so we partition them into blocks satisfying $4 - \text{rank}(A) \leq 4 - |A|$, giving $P = \{\{x_1, 2 \cdot x_2\}, \{7 \cdot (x_1 \oplus x_2)\}\}$. As neither partition will become uncomputable after h_1, we simply apply the first Hadamard gate:



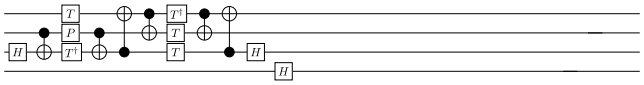
At the second Hadamard gate, the path variable x_5 is available, so $x_5, 7 \cdot (x_1 \oplus x_5), 7 \cdot (x_2 \oplus x_5), x_1 \oplus x_2 \oplus x_5$ are now computable. We add them to the partition to get

$$P = \left\{ \begin{array}{c} \{x_1, 2 \cdot x_2, 7 \cdot (x_2 \oplus x_5)\}, \{7 \cdot \\ (x_1 \oplus x_2)\} \\ \{x_1 \oplus x_2 \oplus x_5, 7 \cdot (x_1 \oplus x_5), x_5\} \end{array} \right\}.$$

We trivially see that $x_2 \oplus x_5 \notin \text{span}(\{x_1, x_2, x_6, x_4\})$ and likewise neither is x_5, so we synthesize a circuit computing the partitions $\{x_1, 2 \cdot x_2, 7 \cdot (x_2 \oplus x_5)\}$ and $\{x_1 \oplus x_2 \oplus x_5, 7 \cdot (x_1 \oplus x_5), x_5\}$ and allow $\{7 \cdot (x_1 \oplus x_2)\}$ to move past the second Hadamard gate.



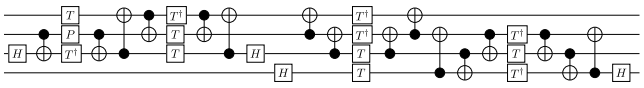
Again, x_6 is now available, so insert the newly computable terms $x_6, 7 \cdot (x_2 \oplus x_6)$ into the current partition $P = \{\{7 \cdot (x_1 \oplus x_2)\}\}$ to get $P = \{\{7 \cdot (x_1 \oplus x_2), x_6, 7 \cdot (x_2 \oplus x_6)\}\}$. As the single partition block will still be computable after applying h_3, we apply the next Hadamard gate:



We now add the last of the terms to the partition, $x_7, 7 \cdot (x_6 \oplus x_7), x_2 \oplus x_6 \oplus x_7$, giving

$$\left\{ \begin{array}{c} \{7 \cdot (x_1 \oplus x_2), x_6, 7 \cdot (x_2 \oplus \\ x_6), x_7\}, \\ \{7 \cdot x_2 \oplus x_7), 7 \cdot (x_6 \oplus \\ x_7), x_2 \oplus x_6 \oplus x_7\} \end{array} \right\}.$$

As both partitions contain the value x_7 which will be destroyed by h_4, we apply the remaining partitions, followed by the last Hadamard gate:



The final circuit, shown above, reduces the original circuit by 2 T gates (from 14 to 12) and 2 levels of T-depth (from 6 to 4).

Note that the partitions used in the above are minimal, but are not the partitions Algorithm 1 actually produces. Instead, these partitions have been created to best demonstrate the algorithm. Additionally, for simplicity, we described states without parity, as there are no bit flip gates in this example.

Acknowledgments

We would like to thank Martin Rötteler and Bill Cunningham for many helpful discussions.

This material is based upon work partially supported by the National Science Foundation (NSF), during D. Maslov's assignment at the Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Michele Mosca is also supported by Canada's NSERC, MPrime, CIFAR, and CFI. IQC and Perimeter Institute are supported in part by the Government of Canada and the Province of Ontario.

References

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

[2] M. Amy, D. Maslov, M. Mosca, and M. Rötteler, "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, `arXiv:1206.0758`.

[3] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Phys. Rev. A*, vol. 52, pp. 3457–3467, 1995, `quant-ph/9503016`.

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

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

[6] J. W. Britton, B. C. Sawyer, A. C. Keith, C.-C. J. Wang, J. K. Freericks, H. Uys, M. J. Biercuk, and J. J. Bollinger, "Engineered two-dimensional ising interactions in a trapped-ion quantum simulator with hundreds of spins," *Nature*, no. 7395, pp. 489–492, 2012.

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

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

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

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

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

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

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

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

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

[16] IARPA Quantum Computer Science Program, 2011-2013, http://www.iarpa.gov/Programs/sso/QCS/qcs.html.

[17] V. Kliuchnikov and D. Maslov, "Optimization of Clifford circuits," *Phys. Rev. A*, vol. 88, p. 052307, arXiv:1305.0810.

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

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

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

[21] D. Maslov, "Reversible logic synthesis benchmarks page," http://webhome.cs.uvic.ca/$\tilde{\text{d}}$maslov/, last accessed October 2013.

[22] D. Maslov, G. W. Dueck, D. M. Miller, and C. Negrevergne. "Quantum Circuit Simplification and Level Compaction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 3, pp. 436–444, 2008, quant-ph/0604001.

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

[24] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

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

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

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

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

[29] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," *Foundations of Computer Science*, pp. 124–134, 1994, quant-ph/9508027v2.

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

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

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

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