Efficient Near-Optimal Clifford + T Gate Synthesis

Luke Heyfron, and Earl T. Campbell

October 2017

Abstract

1 Introduction

2 Workflow Overview

The high level workflow of the protocol begins by converting an n qubit input Clifford + T circuit, C_{in}, (fig. 1.1) to a CNOT + T circuit plus additional operations that are all Clifford (fig. 1.2). The purpose of this is to isolate the non-Clifford behaviour within a subcircuit that has properties required for our optimization framework (see section 3). The main obstacle to this conversion process is the presence of Hadamard gates in the original circuit. External Hadamards, found at the beginning and end of the circuit, can be left in place. In order to remove internal Hadamards we use an approach similar to that of [7], [4] where a unitary Hadamard gate is replaced with a Hadamard gadget that involves only CNOT + T gates as well as preparation of ancillas in the $|+\rangle$ state and measurement in the Pauli-X basis (see figure 2). In doing so we are trading internal Hadamards for external Hadamards and space-like overhead.

Once the Hadamards are removed, we decompose the CNOT + T circuit into two partitions such that one part consists of CNOT gates only and the other is diagonal in the computational basis (fig. 1.3). We then extract the *signature tensor* of the diagonal CNOT + T circuit, which is an object that uniquely characterizes the unitary U_f up to a Clifford factor.

The core layer of the algorithm, *T-optimiser*, takes as input the signature tensor of the diagonal CNOT + T circuit and outputs a *gate synthesis matrix*, A, which represents a particular circuit decomposition for the input unitary. The number of columns of A is equal to the T count of its corresponding circuit, so *T-optimiser* ideally outputs a gate synthesis matrix with minimal column number. It is believed that finding optimal solutions for this problem is NP-hard [3], so we must use efficient heuristics that yield near-minimal column number.

The output A matrix is mapped to a circuit of the same form as U_f, with which the original U_f circuit is replaced. The result is an output circuit that implements the full Clifford + T input circuit on the first n qubits and has near optimal T count.

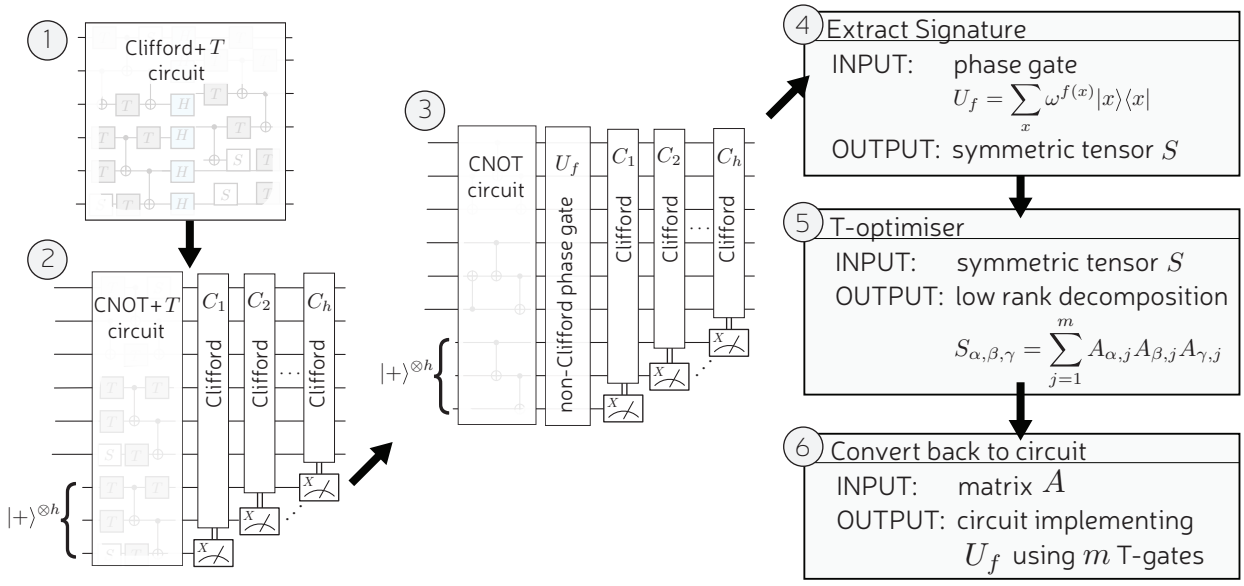

Figure 1: The high level workflow of the T gate optimization protocol is shown. A Clifford + T circuit is converted to the CNOT+T gate set by introducing ancillas and performing classically controlled Clifford gates. A non-Clifford phase gate is extracted, which maps to a signature tensor upon which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

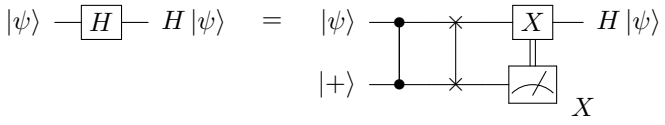

Figure 2: Internal Hadamards are removed by replacing each one according to this circuit identity. The Hadamards on the left-hand circuit are now external and act on an ancilla, therefore they no longer obstruct the *T-optimiser* algorithms.

3 Diagonal CNOT+T Framework

This section describes how we map the gate synthesis problem from the quantum circuit picture to an abstract mathematical picture involving binary tensors. We proceed by borrowing the observation from [2],[1] that the action of any n-qubit diagonal CNOT + T unitary U_f on a computational basis state $|\mathbf{x}\rangle$ is given by

$$U_f |\mathbf{x}\rangle = \omega^{f(\mathbf{x})} |\mathbf{x}\rangle , \tag{1}$$

where $\omega = e^{i\frac{\pi}{4}}$, and U_f is completely characterized by a polynomial $f : \mathbb{Z}_2^n \mapsto \mathbb{Z}_8$, known as the phase function of U_f. A phase function can be decomposed into a sum of linear, quadratic and cubic monomials on the x_i where each monomial of order r has a coefficient in $\mathbb{Z}_8$ and is weighted by a factor 2^{r-1}, as in the following:

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

where $l_\alpha, q_{\alpha,\beta}, c_{\alpha,\beta,\gamma} \in \mathbb{Z}_8$. In [5] such decompositions of f are referred to as *weighted polynomials* and it was proven that any two unitaries with weighted polynomials whose coefficients all have the same parity are Clifford equivalent. Note the weighted polynomial can be lifted directly from the circuit definition of U_f if we work in the $\{T, CS, CCZ\}$ basis, as each kind of gate corresponds to the linear, quadratic and cubic terms, respectively. We define the *signature tensor*, $S \in \mathbb{Z}_2^{(n,n,n)}$, of U_f to be a symmetric tensor of order 3 whose elements are equal to the parity of the weighted polynomial coefficients of U_f according to the following relations:

$$l_\alpha \pmod 2 = S_{\sigma(\alpha,\alpha,\alpha)} = S_{a,a,a} \tag{3}$$
$$q_{\alpha,\beta} \pmod 2 = S_{\sigma(\alpha,\beta,\beta)} = S_{\sigma(\alpha,\alpha,\beta)} \tag{4}$$
$$c_{\alpha,\beta,\gamma} \pmod 2 = S_{\sigma(\alpha,\beta,\gamma)} \tag{5}$$

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

We recall the definition of gate synthesis matrices from [5], where a matrix, A in $\mathbb{Z}_2^{(n,m)}$, is a gate synthesis matrix for a unitary U_f if it satisfies,

$$f(\mathbf{x}) = |A^T\mathbf{x}| \pmod 8 \tag{6}$$

where f is the phase function of U_f and $|.|$ is the Hamming weight of a binary vector. An A matrix can be efficiently extracted from a diagonal CNOT + T circuit by tracking the action of each gate on the computational basis states through the circuit. Conversely, given an A matrix one can generate a circuit that implements U_f using m T gates. We can determine the signature tensor of U_f from an A matrix of U_f using the following relation,

$$S_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} A_{\alpha,j} A_{\beta,j} A_{\gamma,j} \pmod 2 \tag{7}$$

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

Problem 3.1. *Given a unitary U_f whose signature tensor is S, find an A matrix with minimal number of columns that satisfies equation 7.*

Note that pairs of identical columns and all-zero columns will never appear in an optimal A matrix as these have no effect on the signature tensor.

4 *T-optimiser*

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

4.1 Reed-Muller decoder (RM)

It has been shown that problem 3.1 can be mapped in polynomial time to the minimum distance decoding problem of the punctured Reed-Muller code of order $n-4$ and length n, compactly written as $RM^*(n-4,n)$, and is believed to be a hard problem [3]. This imposes a practical upper bound on the number of qubits n_{RM} over which circuits can be optimally synthesized with respect to T count. Using an author-written implementation of the brute force decoder, we have found this limit to be $n_{RM} = 6$. To gain insight into the complexity of the problem, consider the following. The number of codespace generators for $RM^*(n-4,n)$ is equal to $N_G = \sum_{r=1}^{n-4} \binom{n}{r}$. Therefore the size of the search space is $N_{\text{search}} = 2^{N_G}$. On a processor with a clock speed of 3.20GHz, assuming we can check one codeword per clock cycle, it would take over 91 years to exhaustively search this space and therefore determine optimality for a general CNOT+T circuit for $n = 7$. It would take $\approx 6 \times 10^{31}$ years for $n = 8$. Clearly, we need develop heuristics for this problem.

4.2 Recursive Expansion (RE)

The simplest means of efficiently obtaining an A matrix for a given signature tensor S is to make use of the modulo identity $2ab = a+b-a\oplus b$. This is applied recursively to each non-linear term in the weighted polynomial associated with S to yield a decomposition of the form in equation 6. We call this the *recursive expansion* (RE) algorithm, which has been shown to yield worst-case T counts of $\mathcal{O}(n^3)$.

4.3 Temporary Order Reduction (TOR)

An efficient heuristic that requires only $\mathcal{O}(n^2)$ T gates was proposed in [5], which involves breaking down an input CNOT + T circuit into a sequence of control-U operators where each successive operator acts on fewer qubits than the previous, eventually allowing the optimal Reed-Muller decoder to be used when $n \leq n_{RM}$ for a particular control-U. Each target U corresponds to a weighted polynomial with only up to quadratic terms, and the control qubit maps to a particular variable in the weighted polynomial that has been factorized out from all terms in which it appears. The reduction in order from a cubic to a quadratic weighted polynomial allows for an optimal decomposition U' for each target U to be found efficiently using Lempel's matrix factoring algorithm [6]. Additional steps are required to obtain a decomposition of the form 6 from the control-U' sequence, which leads to sub-optimality in the general case but with T count scaling of $\mathcal{O}(n^2)$, a factor n improvement over RE. This algorithm we refer to as the *temporary order reduction* (TOR) algorithm efficiently yields optimal T counts for the special case where every gate in the CNOT + T circuit shares a control.

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

4.4 Third Order Duplicate and Destroy (TODD)

In this section, we present an algorithm based in principle on Lempel's matrix factoring algorithm [6] that is 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 in section 5 we present numerical evidence that it is the best efficient solver of problem 3.1 developed so far.

The algorithm works by identifying a subset of columns of an input gate synthesis matrix, A, to which we can add an arbitrary column vector, $\mathbf{z}$. This allows us to force pairs of columns to be identical, and identical columns can be removed without changing the unitary that it implements up to a Clifford factor. We call this the *Third Order Duplicate and Destroy* (TODD) algorithm because, much like the villainous Victorian barber, it shaves away at the columns of the input A matrix iteratively until the algorithm finishes execution.

The rest of this section is consists of three parts: the first part shows that we can add an arbitrary column vector 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 $\mathbf{z}$ results in a new gate synthesis matrix that has two identical columns, which can be subsequently eliminated. The third and final part describes the flow of the algorithm itself.

We begin by defining the χ matrix as a function of a gate synthesis matrix and the column vector $\mathbf{z}$.

Definition 4.1. χ **Matrix.** Given some gate synthesis matrix, A, and a column vector $\mathbf{z} \in \mathbb{Z}_2^n$ let

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

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

Lemma 4.1. *Let A and $A' = A \oplus xy^T$ be two gate synthesis matrices where $\mathbf{z}$, $\mathbf{y}$ are arbitrary column vectors of dimension n and m, respectively. $S(A) = S(A')$ as long as the following conditions hold true:*

1. $|\mathbf{y}| = 0 \pmod 2$
2. $A\mathbf{y} = \mathbf{0}$
3. $\chi(A, \mathbf{z})\ \mathbf{y} = \mathbf{0}$.

Proof. We begin the proof by finding an expression for $S(A')$ using equation 7,

$$S^{(A')}_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} (A_{\alpha,j} \oplus z_\alpha y_j)(A_{\beta,j} \oplus z_\beta y_j)(A_{\gamma,j} \oplus z_\gamma y_j) \pmod 2, \tag{8}$$

and expanding the brackets,

$$\begin{aligned} S^{(A')}_{\alpha,\beta,\gamma} = \sum_{j=1}^{m} (&A_{\alpha,j}A_{\beta,j}A_{\gamma,j} \oplus z_\alpha z_\beta z_\gamma y_j \\ &\oplus z_\alpha z_\beta A_{\gamma,j} y_j \oplus z_\beta z_\gamma A_{\alpha,j} y_j \oplus z_\gamma z_\alpha A_{\beta,j} y_j \\ &\oplus z_\alpha A_{\beta,j} A_{\gamma,j} y_j \oplus z_\beta A_{\gamma,j} A_{\alpha,j} y_j \oplus z_\gamma A_{\alpha,j} A_{\beta,j} y_j) \pmod 2. \end{aligned} \tag{9}$$

We can see that the first term of equation 9 summed over all j is equal to $S(A)$, by definition. The task is to show that the remaining terms sum to zero under the specified conditions. Next, we use the definitions of $|\mathbf{y}|$, $A\mathbf{y}$ and $\chi(A, \mathbf{z})\ \mathbf{z}$ to simplify the result,

$$S^{(A')}_{\alpha,\beta,\gamma} = S^{(A)}_{\alpha,\beta,\gamma} \oplus z_\alpha z_\beta z_\gamma |\mathbf{y}| \oplus z_\alpha z_\beta \left[A\mathbf{y}\right]_\gamma \oplus z_\beta z_\gamma \left[A\mathbf{y}\right]_\alpha \oplus z_\gamma z_\alpha \left[A\mathbf{y}\right]_\beta \oplus \left[A\chi(A, \mathbf{z})\ \mathbf{y}\right]_{\iota(\alpha,\beta,\gamma)}, \tag{10}$$

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

By applying condition (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 $A' = A \oplus \mathbf{z}\mathbf{y}^T$ where $\mathbf{z}$ and $\mathbf{y}$ are column vectors on $\mathbb{Z}_2$ of length n and m, respectively, defined such that $z_i = A_{i,a} \oplus A_{i,b}$ for some $a, b \in [1, m]$ and $y_a \oplus y_b = 1$. 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, $\mathbf{z}$ and $\mathbf{y}$,

$$A'_{i,j} = A_{i,j} \oplus z_i y_j, \tag{11}$$

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

$$A'_{i,j} = A_{i,j} \oplus (A_{i,a} \oplus A_{i,b}) y_j. \tag{12}$$

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

$$A'_{i,a} = A_{i,a} \oplus (A_{i,a} \oplus A_{i,b})y_a, \tag{13}$$

$$A'_{i,b} = A_{i,b} \oplus (A_{i,a} \oplus A_{i,b})y_b. \tag{14}$$

We substitute in the condition $y_b = y_a \oplus 1$ into 14,

$$\begin{aligned} A'_{i,b} &= A_{i,b} \oplus (A_{i,a} \oplus A_{i,b})(y_a \oplus 1) \\ &= A_{i,a} \oplus (A_{i,a} \oplus A_{i,b})y_a \\ &= A'_{i,a}. \end{aligned} \tag{15}$$

□

Now we are ready to describe the TODD algorithm. Given an input gate synthesis matrix A with signature tensor S, we begin by iterating through all column pairs a, b of A where we construct the vector $\mathbf{z}$ according to lemma 4.2. We check to see if the conditions in lemma 4.1 are satisfied for $\mathbf{z}$ by forming the matrix $\tilde{A}$ by concatenating A with $\chi(A, \mathbf{z})$. We then calculate a basis for the right null space of $\tilde{A}$. Any vector in the null space, $\mathbf{y}$, 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 $\mathbf{y}$ such that $y_a \oplus y_b = 1$. At this stage we know that we can remove at least one column from A. We force condition 1 of lemma 4.1 to be satisfied by appending a 1 to $\mathbf{y}$ and an all-zero column to A if $\mathbf{y}$ has odd-weight. Finally, we add the value of $\mathbf{z}$ 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 now start again from the beginning, iterating over columns of the new A matrix. The algorithm terminates if every column pair has been exhausted without success.

Note that this algorithm takes as input a gate synthesis matrix whereas *T-Optimize* requires a signature tensor. This means TODD requires some initial A matrix to be generated by another algorithm such as RE or TOR.

Third Order Duplicate-then-Destroy (TODD) Algorithm

Input: Gate synthesis matrix $A \in \mathbb{Z}_2^{(n,m)}$.

Output: Gate synthesis matrix $A' \in \mathbb{Z}_2^{(n,m')}$ such that $m' \leq m$ and $S^{(A')} = S^{(A)}$.

- Let $\text{col}_j(A)$ be a function that returns the j^{th} column of A.
- Let $\text{cols}(A)$ be a function that returns the number of columns of A.
- Let nullspace(A) be a function that returns a matrix whose columns generate the right nullspace of A.
- Let simplify(A) be a function that removes every pair of identical columns and every all-zero column from A.

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

---

5 Results

Our T gate optimization protocol was implemented in C++ in order to achieve two goals: first benchmark test our different *T-Optimise* algorithms against one another using random circuits (see figure 3); and second to evaluate the performance of the full Clifford + T protocol against the best known circuit decompositions for useful quantum algorithms, the results of which are listed in table 1. We see from both results that TODD produces quantum circuits with smallest T count.

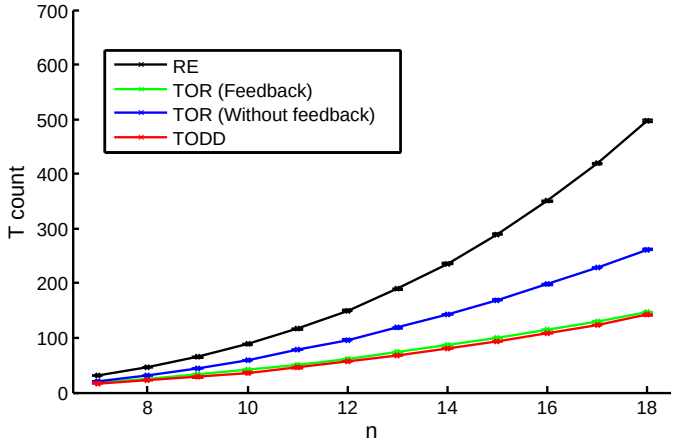

Figure 3: Circuits generated by the CNOT and T gate were randomly generated for varying number of qubits n then optimized by our implementations of RE, TOR and TODD. The average T-count for each n over many random circuits are shown on the vertical axis. TODD produces circuit decompositions with the smallest T-counts on average but scales the same as the next best algorithm, TOR (Feedback). Both of these algorithms are better than RE by a factor n. The difference between the T-counts for TODD and TOR (Feedback) converge on a constant 5.5 ± 0.7 for large n.

Table 1: T-counts for various universal Clifford + T benchmark circuits as synthesized by the TODD and RE algorithm are shown. T_{original} are the best known results produced by the works cited in the *Circuit* column, and T_{TODD} is the result for *TODD*. $n_{\textbf{original}}$ is the number of qubits of the original circuit and $n_{\textbf{out}}$ is the total number of qubits of the output circuit including ancillas used to implement multiply controlled Toffoli gates as well as Hadamards using path variables [**?**]. The total execution time in seconds for *TODD* run on an *Intel i7* 2.40Gz processor is given in column *Time*.

Circuit	$T_{\textbf{original}}$	$T_{\textbf{TODD}}$	$T_{\textbf{RE}}$	n	h	**Time (s)**
hwb6_47_107 [3]	71	55	102	6	43	91.713
hwb6-42-150 [3]	71	46	140	6	43	160.198
nth_prime6_inc_55_667 [3]	400	263	354	6	39	228.025
ham15-109-214 [3]	97	28	65	15	47	27.756
ham15-70 [3]	230	103	148	15	47	100.899
ham15tc1 [3]	1019	258	359	15	50	270.862
#*0117* [8]	79	13	63	6	116	118.64
#*017F* [8]	80	23	59	6	59	20.618
#*0001* [8]	40	21	46	6	39	5.2
#*001F* [8]	43	22	55	6	39	4.382
#*0007* [8]	47	13	31	6	64	10.141
#*007F* [8]	40	19	46	6	27	1.267

6 Discussion

7 Conclusion

8 Acknowledgements

References

[1] M. Amy, D. Maslov, and M. Mosca. Polynomial-time T-depth optimization of Clifford+T circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1476–1489, Oct 2014.

[2] M. Amy, D. Maslov, M. Mosca, and M. Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, June 2013.

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

[4] M. J. Bremner, R. Jozsa, and D. J. Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 2010.

[5] 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.

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

[7] A. Montanaro. Quantum circuits and low-degree polynomials over $\mathbb{F}_2$. *Journal of Physics A: Mathematical and Theoretical*, 50(8):084002, 2017.

[8] M. Soeken, M. Roetteler, N. Wiebe, and G. D. Micheli. Logic synthesis for quantum computing. *arXiv preprint*, arXiv:1706.02721, 2017.