


# CNOT-Optimal Clifford Synthesis as SAT

Irfansha Shaik ✉ 

Department of Computer Science, Aarhus University, Denmark  
Kvantify Aps, Copenhagen S, Denmark

Jaco van de Pol ✉ 

Department of Computer Science, Aarhus University, Denmark

---

## Abstract

Clifford circuit optimization is an important step in the quantum compilation pipeline. Major compilers employ heuristic approaches. While they are fast, their results are often suboptimal. Minimization of noisy gates, like 2-qubit CNOT gates, is crucial for practical computing. Exact approaches have been proposed to fill the gap left by heuristic approaches. Among these are SAT based approaches that optimize gate count or depth, but they suffer from scalability issues. Further, they do not guarantee optimality on more important metrics like CNOT count or CNOT depth. A recent work proposed an exhaustive search only on Clifford circuits in a certain normal form to guarantee CNOT count optimality. But an exhaustive approach cannot scale beyond 6 qubits.

In this paper, we incorporate search restricted to Clifford normal forms in a SAT encoding to guarantee CNOT count optimality. By allowing parallel plans, we propose a second SAT encoding that optimizes CNOT depth. By taking advantage of flexibility in SAT based approaches, we also handle connectivity restrictions in hardware platforms, and allow for qubit relabeling. We have implemented the above encodings and variations in our open source tool Q-Synth.

In experiments, our encodings significantly outperform existing SAT approaches on random Clifford circuits. We consider practical VQE and Feynman benchmarks to compare with TKET and Qiskit compilers. In all-to-all connectivity, we observe reductions up to 32.1% in CNOT count and 48.1% in CNOT depth. Overall, we observe better results than TKET in the CNOT count and depth. We also experiment with connectivity restrictions of major quantum platforms. Compared to Qiskit, we observe up to 30.3% CNOT count and 35.9% CNOT depth further reduction.

**2012 ACM Subject Classification** Hardware → Quantum computation; Computing methodologies → Planning for deterministic actions

**Keywords and phrases** Circuit Synthesis, Circuit Optimization, Quantum Circuits, Propositional Satisfiability, Parallel Plans, Clifford Circuits, Encodings

**Digital Object Identifier** 10.4230/LIPIcs.SAT.2025.28

**Related Version** *Extended Version*: <https://arxiv.org/abs/2504.00634> [26]

**Supplementary Material** *Software (Github)*: <https://github.com/irfansha/Q-Synth>

archived at `swh:1:dir:f934bdca528231c8f64a232248ae97bed6d26102`

*Software (Zenodo Link)*: <https://doi.org/10.5281/zenodo.15575214>

**Funding** This research was partially funded by the Innovation Fund Denmark project “Automated Planning for Quantum Circuit Optimization”. This research was also partially funded by the European Innovation Council through Accelerator grant no. 190124924.

**Acknowledgements** We thank L. Burgholzer for his generous help with settings of the QMAP software for Clifford Synthesis. The experiments were carried out on the Grendel cluster at the Centre for Scientific Computing, Aarhus (<http://www.cscaa.dk/grendel/>). We thank the anonymous reviewers of SAT 2025 for their useful suggestions.



© Irfansha Shaik and Jaco van de Pol;

licensed under Creative Commons License CC-BY 4.0

28th International Conference on Theory and Applications of Satisfiability Testing (SAT 2025).

Editors: Jeremias Berg and Jakob Nordström; Article No. 28; pp. 28:1–28:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

Quantum Computing promises an alternative solution to some challenging computational problems that are out-of-reach for classical computers. While several competing quantum platforms exist in the current Noisy Intermediate Scale Quantum (NISQ) era, they all come with different strengths and weaknesses. Quantum programs must be compiled to low level quantum circuits satisfying target hardware requirements before execution. While most quantum platforms accept 1-qubit and 2-qubit gates, their native gate-sets can vary. Currently, no quantum platform has the best qubit count, fidelity, and latency all together. Circuit optimization can play a crucial role for practical quantum computing in both current NISQ or future fault-tolerant processors. For example, [10] observed that IonQ’s Aria quantum platform had a limit of 950 single qubit gates. Only by using circuit optimization techniques they were able to run the required circuits. For satisfying hardware requirements, Layout Synthesis and Circuit Synthesis are two main steps. In Layout Synthesis, circuits are synthesized to handle hardware layout restrictions. Often, not all qubits are connected in a quantum platform thus 2-qubit quantum gates can only be scheduled on neighboring qubits. Circuit Synthesis involves either synthesizing to required gate-set or optimizing some circuit metric for practical quantum computing.

We consider circuit optimization in this paper. Optimal synthesis of an arbitrary  $n$ -qubit circuit requires considering  $2^n \times 2^n$  unitary matrices of complex numbers. While optimal circuit synthesis is ideal, it is a challenging computational problem [16]. Instead, peephole optimization is often used where *easier* sub-circuits are optimized [24]. In this paper, we focus on an interesting subclass of circuits called *Clifford circuits*. Any circuit composed of 1-qubit Hamard (H) and Phase (S) gates, and 2-qubit Conditional-Not (CNOT) gates is a Clifford circuit. While polynomially simulatable, Clifford circuits capture important phenomena like entanglement and superposition, and have applications in teleportation and dense quantum encoding [1]. Clifford circuits are also key in quantum error correction, required for the future fault-tolerant hardware. Synthesis of Clifford circuits only requires  $(2n) \times (2n + 1)$  Boolean matrices, using the *stabilizer* formalism [1], instead of full unitary matrices. Using peephole optimization, one can replace Clifford sub-circuits with their optimized counterparts.

There is no single optimization metric that can *predict* actual hardware performance perfectly. Quantitative metrics, like qubit fidelity, provide good predictions, but lead to numerical optimization problems. Simple metrics like gate count, circuit depth are preferred for optimization [15, 23, 28, 29]. In NISQ processors, 2-qubit CNOT gates are up to 10 times more error-prone than 1-qubit gates. Industrial compilers, like Qiskit [21] by IBM and TKET [28] by Quantinuum, apply Clifford optimization reducing CNOT gate-count (cx-count) and CNOT depth (cx-depth). Given a circuit with unary gates and CNOT gates, cx-depth is the maximum number of CNOT gates that are executed in series. In a recent survey paper [17], the authors extensively compared major compilers on cx-count/cx-depth reduction. While several approaches exist for Clifford optimization, industrial compilers mainly focus on heuristic approaches. Exact optimization approaches unfortunately suffer from scalability problems. Optimization of Clifford circuits is NP-hard [13]. Even approximation of optimal synthesis is NP-hard [12], thus there is no efficient algorithm unless  $P = NP$ . This undesirable gap between heuristic and exact approaches is well established in the literature [30, 20].

Thus, there is a need to improve the scalability of exact approaches. In recent years, several SAT based approaches were proposed for problems like Layout Synthesis [30, 25] and CNOT synthesis [24]. In [22], authors proposed a SAT encoding for cx-count optimization in Clifford circuits based on bounded reachability. Unfortunately, their encoding does not

guarantee optimality due to the use of asymptotic cx-count upper bound for termination criteria. Despite being a near-optimal encoding, their approach still suffers from scalability issues (even for  $n > 3$  qubits). In [20], the authors proposed an improved SAT encoding instead optimizing circuit depth that can handle up to 5-qubit circuits. While this improved scalability, their synthesis is focused on circuit depth, rather than cx-count and cx-depth. Our experiments (Section 5) show that synthesized circuits with minimal circuit depth can have worse cx-count/cx-depth, even compared to heuristic tools.

Using some special properties of Clifford circuits, one can guarantee cx-count optimality. In [5], authors proposed circuit synthesis restricted to certain normal forms. They observed that by ignoring so-called phase updates one can guarantee cx-count optimality. Using brute force search, they successfully generated a 2.1TB database for all cx-count optimal 6-qubit circuits using up to 100 days of computation. While useful, generating such a database for beyond 6-qubits is not practical [5]. Further, such an approach is not flexible i.e., a new database needs to be generated for each optimization metric/criteria. For example, layout aware optimal Clifford circuits can vary depending on the platform layout restrictions. Alternatively, flexibility of SAT like approaches can be used for efficient on-demand computation.

## Our Contribution

In this paper, we incorporate the ideas proposed in [5] to obtain efficient SAT encodings that guarantee optimal Clifford circuits. We provide the first *cx-count* optimal SAT encoding based on bounded reachability. Restricting the search to circuits in normal form reduces the search space (and consequently the makespan of the encoding) significantly. Further, we propose the first *cx-depth* optimal approach to Clifford circuit synthesis, by adapting the SAT encoding. We have extended our open source tool Q-Synth<sup>1</sup> with version 5 implementing the above encodings. For an experimental comparison, we propose three experiments. In Experiment 1, we compare existing SAT encodings with ours on random Clifford circuits of 3 to 7 qubits. For both cx-count/cx-depth optimization, we significantly outperform previous approach while solving optimally. We show that our approach can optimally solve 4 out of 5 7-qubit circuits for the first time for cx-depth optimization. In Experiment 2, we demonstrate our effectiveness on practical VQE and Feynman [2] benchmarks via peephole optimization. We generate 12 VQE benchmarks of 8 and 16 qubits, and 28 T-gate optimized Feynman benchmarks of 5 to 24 qubits. Given a 10-minute time limit, we consistently outperform TKET, and we observe the best results when TKET+Q-Synth are used together. In Experiment 3, we focus on layout aware re-synthesis of practical VQE and Feynman benchmarks. We first use Qiskit [21] to map VQE and Feynman benchmarks onto 54-qubit Sycamore [3], 80-qubit Rigetti [7], and 127-qubit Eagle [6] platforms. We then re-synthesize each benchmark with Q-Synth giving a 10-minute time limit. Overall, we observe significant reduction on all platforms in both cx-count and cx-depth optimizations. In VQE benchmarks, we observe a reduction of up to 19.3% cx-count and 27.4% cx-depth. In Feynman benchmarks, we observe a reduction of up to 30.3% cx-count and 35.9% cx-depth. Our experiments indicate that there is a place for SAT like approaches in the quantum compilation pipeline.

---

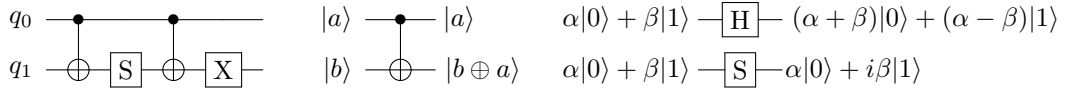
<sup>1</sup> Q-Synth v5 tool with source code, benchmarks, and scripts: <https://github.com/irfansha/Q-Synth>

## 2 Preliminaries

### 2.1 Clifford Circuits

For a textbook description of quantum states, gates and circuits we refer to [18]. We also provide a short summary of quantum notation in Section A.

In classical computing, the fundamental unit of information, a classical bit, has only two states 0 and 1. In Quantum computing, a quantum bit instead is a superposition of 0 and 1. One can represent a qubit state as a vector  $\alpha|0\rangle + \beta|1\rangle$  where  $\alpha, \beta \in \mathbb{C}$  and  $|\alpha|^2 + |\beta|^2 = 1$ . States over  $n$ -qubits live in the tensor product space on all vectors on  $n$  qubits. Quantum gates change the state by acting on one or more qubits. Due to practical difficulties of implementing muti-qubit quantum gates, quantum platforms typically only apply 1-qubit and 2-qubit gates. High level quantum programs are decomposed to low level circuits with 1- and 2-qubit gates before execution. In this paper, we focus on Clifford circuits made of  $\{\text{CNOT}, \text{H}, \text{S}\}$  gates (cf. Section A). Figure 1b shows how the Clifford gates change the state of qubits. A binary CNOT gate (Conditional NOT, also abbreviated as CX) negates the target qubit ( $b$ ) when the control qubit ( $a$ ) is on. This entangling gate can also be expressed as an exclusive OR. The H gate brings a qubit in a superposition, and the S gate applies a phase change to a qubit. Pauli gates  $\{X, Y, Z\}$  are composed of Clifford gates, where  $X = \text{HSSH}$ ,  $Z = \text{SS}$  and  $Y = iXZ$ . We will use Pauli gates later in the paper for so-called relative phase recovery. Figure 1a shows an example Clifford circuit with  $\{\text{CNOT}, \text{S}, \text{X}\}$  gates.



(a) A 2-qubit Clifford circuit. (b) State updates by CNOT, H, and S gates.

■ **Figure 1** Example Clifford circuit, and an illustration of state updates by Clifford gates.

Interestingly, Clifford gates are not universal and can be simulated in polynomial time and space. In [1], Aaronson and Gottesman proposed *stabilizer* formalism to efficiently represent Clifford circuits. For an  $n$ -qubit Clifford circuit, we only need a  $(2n) \times (2n + 1)$  Boolean matrix, called *tableau*. Two Clifford circuits are equivalent (up to global phase) if their corresponding tableaux are equal [5]. The following matrix shows the structure of a  $n$ -qubit tableau:

$$\left( \begin{array}{ccc|ccc|c} q_0 & \dots & q_{n-1} & q_0 & \dots & q_{n-1} & \\ x_{00} & \dots & x_{0(n-1)} & z_{00} & \dots & z_{0(n-1)} & r_0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{(2n-1)0} & \dots & x_{(2n-1)(n-1)} & z_{(2n-1)0} & \dots & z_{(2n-1)(n-1)} & r_{2n-1} \end{array} \right)$$

An  $n$ -qubit tableau is made of  $x_{2n \times n}$ ,  $z_{2n \times n}$ , and  $r_{2n \times 1}$  Boolean matrices. A tableau essentially represents so-called *stabilizer* and *destabilizer* generators. We refer to [1] for a detailed explanation on stabilizer generators, Clifford circuits, and relevant proofs. For the scope of this paper, it is sufficient to understand how a tableau can be computed given a Clifford circuit. The initial tableau, representing an empty circuit, is defined as  $x_{ii} = z_{(n+i)i} = 1$  (for  $0 \leq i < n$ ) and every other cell is 0.

■ **Table 1** Tableau update rules for every row  $i$ , when applying Clifford gates to qubits  $a$  and  $b$ .

	Base Gates			Pauli gates		
	$\text{CNOT}_{a,b}$	$H_a$	$S_a$	$X_a$	$Y_a$	$Z_a$
$x_{ia}$	$x_{ia}$	$z_{ia}$	$x_{ia}$	$x_{ia}$	$x_{ia}$	$x_{ia}$
$z_{ia}$	$z_{ia} \oplus z_{ib}$	$x_{ia}$	$x_{ia} \oplus z_{ia}$	$z_{ia}$	$z_{ia}$	$z_{ia}$
$x_{ib}$	$x_{ia} \oplus x_{ib}$	-	-	-	-	-
$z_{ib}$	$z_{ib}$	-	-	-	-	-
$r_i$	$r_i \oplus x_{ia} z_{ib} (x_{ib} \oplus z_{ia} \oplus 1)$	$r_i \oplus x_{ia} z_{ia}$	$r_i \oplus x_{ia} z_{ia}$	$r_i \oplus z_{ia}$	$r_i \oplus x_{ia} \oplus x_{ib}$	$r_i \oplus x_{ia}$

For each gate in a given circuit, we update every row of the tableau according to the rules in Table 1; this corresponds to column additions modulo 2. For our example Clifford circuit from Figure 1a, the following sequence of matrices represent tableau updates:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \xrightarrow{q_0, q_1} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \xrightarrow{q_1} \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \xrightarrow{q_0, q_1} \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \xrightarrow{q_1} \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

The leftmost matrix corresponds to the initial tableau and the rightmost matrix is our target tableau. Optimal synthesis of Clifford circuits can now be transformed to a graph search problem. Each node of the graph is labelled with a tableau, and edges labeled with Clifford gates correspond to the update rules. Synthesizing the optimal number of gates boils down to finding the shortest path in this graph. On the other hand, optimizing the number of CNOT gates is not so straightforward. We will revisit this in Section 3.

## 2.2 Layout Restrictions and Qubit Relabeling

In high-level quantum circuits, binary CNOT gates can be applied to any pair of qubits. However, in many physical realizations (in particular on superconducting quantum computers), binary gates can only be applied to physical neighbour qubits. We assume that this is specified in some coupling graph  $(Q, E)$ , where  $Q$  are the qubits and  $q E q'$  if a CNOT is possible between qubits  $q$  and  $q'$ . This graph is usually quite sparse and mostly planar.

Most existing Clifford synthesis approaches [5, 28, 21, 20] assume all-to-all qubit connectivity. Under this assumption, local rewrite rules can be used effectively. While useful, such structure breaks down in case of restricted qubit connectivity. Thus, compilers like Qiskit and TKET apply Clifford synthesis before the layout synthesis phase, leading to suboptimal results overall. Some heuristic approaches [9] exist that handle connectivity restrictions in related problems like CNOT synthesis. In [24], we showed that connectivity restrictions can be encoded elegantly using SAT in CNOT synthesis. Given a platform connectivity graph, we will only allow 2-qubit gates on neighboring qubits. Thus, our approach can be used post layout synthesis, allowing further reduction.

In tableau representation, notice that the columns are labelled with qubits. Allowing permutation of columns essentially corresponds to relabeling qubits. In a relaxed notion of equivalence, two Clifford circuits are equivalent if their tableau are equivalent up to a column permutation. Allowing any column permutation of initial tableau can result in better circuits. Major compilers like TKET also allow qubit relabeling in the context of all-to-all connectivity. In [24], we showed that permutation of a similar matrix can be elegantly encode using Exactly-One constraints. We adapt the same idea for Clifford synthesis, we encode permutation of columns in initial tableau using cardinality constraints.

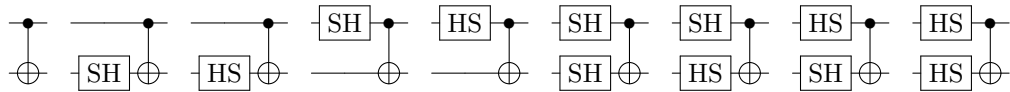
### 3 Revisiting Clifford Normal Forms for Optimal CNOT Synthesis

In earlier sections, we briefly discussed that a simple bounded reachability encoding is sufficient for gate count optimality. For cx-count optimization, existing SAT approach in [22] proposed a MAXSAT like formulation to optimize cx-count for the given gate count. While they can synthesize circuits with better cx-count, they do not ensure cx-count optimality. Consider our example circuit 1a with 4 gates and 2 CNOTs. If we run QMAP tool which implements SAT encoding in [22] on our circuit, it fails to produce a circuit with better cx-count. This makes sense as there does not exist a 4 gate circuit with better cx-count. However, if we consider circuits with higher gate count, there does exist a circuit with a single CNOT gate as shown in Figure 3a. So the given number of Clifford gates in the input circuit is not a valid upperbound. Also, the upperbound of  $\Theta(n^2/\log(n))$  gates for an  $n$ -qubit Clifford circuit [19, 1] is only asymptotically sharp, so it doesn't provide a reliable upper bound for concrete  $n$ .

To ensure cx-count optimality, we need to reformulate the search problem. Let  $q, q'$  be two qubits, we define the set of *entangling* sequences  $E = \{\Psi \text{CNOT}_{q,q'} \mid \Psi \in \{H_q, H_{q'}, S_q, S_{q'}\}^*\}$ . Each entangling sequence, by definition, adds a single CNOT gate to the circuit. A  $d$ -CNOT optimal circuit implies that there does not exist a Clifford circuit with  $< d$  entangling sequences. This modification results in a graph search problem where nodes are still labelled with tableaux. Edges on the other hand are labelled with entangling sequences instead of individual gates. Finding an optimal CNOT circuit corresponds to finding the shortest path in the new graph. While this new formulation guarantees CNOT optimality, the problem of arbitrary single gate sequences still remains. Taking advantage of equivalences between Clifford circuits, [5] proposed a normal form that considers only 9 unique entangling sequences. In the following section, we will revisit observations made in [5] and adapt in the context of a SAT encoding.

#### 3.1 Clifford Normal Forms

Remember that a tableau is made of  $x, z, r$  matrices. [5] observed that two tableau with same  $x$  and  $z$  matrices have same optimal cx-count. From Table 1, we can see that Pauli gates only update  $r$  column. Indeed, one can synthesize any circuit with differing  $r$  column by appending Pauli gates at the front. The authors in [5] consider all circuits with differing  $r$  column as an equivalence class. We will revisit  $r$  column updates in detail later in this Section. We ignore the  $r$  column updates aka relative phase updates for now. This makes the tableau update rules simpler, resulting in fewer unique 1-qubit gate sequences. Turns out there are not many sequences of H and S gates that result in unique tableau state. Clearly, HH and SS cancel out which is equivalent to an Identity gate I. Thus only alternating H and S sequences of gates are interesting. So the interesting sequences are  $\{I, H, S, HS, SH, HSH, SHS, HSHS, SHSH, \dots\}$ . Further, HSH and SHS result in the same tableau state. Using the equivalence of HSH and SHS, we can simplify any non-trivial 1-q gate sequence to a sequence of length less than 4. For example, consider HSHS gate sequence and replace HSH with SHS resulting in SHSS. Since SS cancel out, we are left with SH sequence. Any such sequence can be replaced with one of the 6  $\{I, H, S, HS, SH, HSH\}$  sequences, defined as 1-q unique sequences. Considering these 6 sequences on each qubit followed by a CNOT results in 36 sequences. [5] showed that only the following 9 sequences are unique entangling sequences.





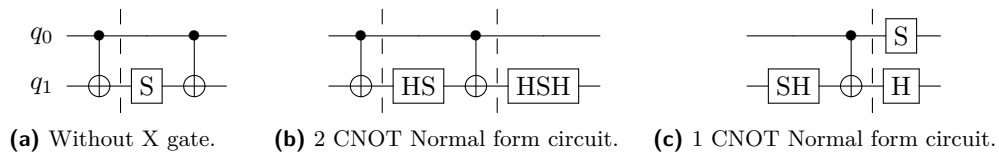
They showed that each of the 36 entangling sequences can be rewritten as one of the above 9 sequences followed by one of the 1-q unique sequences on each qubit. Note that, a CNOT gate  $\text{CNOT}_{j,i}$  can be rewritten as  $H_i H_j \text{CNOT}_{i,j} H_i H_j$ . Thus, we can rewrite all CNOT gates in a given circuit such that the control qubit is less than its target qubit. Now, let us suppose, we have an entangling sequence in the form  $A_i B_j \text{CNOT}_{i,j}$  where  $A, B$  are elements of 1-q unique sequences. Such a sequence can be rewritten to the form  $A'_i B'_j \text{CNOT}_{i,j} C_i D_j$  where  $A', B'$  are in  $\{I, HS, SH\}$  and  $C, D$  are 1-q unique sequences. Using these rewrite rules, from start to end of the circuit, one can push 1-q sequences that are not  $\{I, HS, SH\}$  to the last layer. The final rewritten circuit will only have the above 9 unique entangling sequences followed by 1-q sequences on every qubit.

Consider our example circuit without X gate as in 2a. First CNOT is part of unique entangling sequence, however, S gate followed by second CNOT is not. One can replace  $S_{q_1} \text{CNOT}_{q_0,q_1}$  by  $HS_{q_1} \text{CNOT}_{q_0,q_1} HSH_{q_1}$ . The resulting circuit as shown in 2b has both CNOTs within unique sequences followed by 1-q unique sequences. Figure 2c shows an equivalent circuit with only 1 CNOT. We illustrate entangling sequences in Figure 2 with dotted lines. Authors in [5] consider all circuit with same entangling sequences as part of an equivalence class. However, for a SAT formulation we need to encode the final layer of 1-q sequences explicitly.

Using the above observations, synthesizing  $d$ -CNOT circuit corresponds to finding  $d$  entangling sequences followed by 1-q unique sequences. For synthesizing  $d$ -CNOT optimality, we need to show there does not exist a normal form circuit with  $< d$  entangling sequences. We can easily adapt these observations to handle CNOT-depth optimal synthesis. Allowing parallel entangling sequences on independent qubits at each step allows CNOT-depth optimization. This notion is exactly same as  $\forall$ -step parallel plans [14] in the context of SAT based planning. In other words, sequential encoding corresponds to cx-count optimization whereas parallel encoding corresponds to cx-depth optimization. In Section 4, we present encodings for both cx-count and cx-depth optimization.

### Phase Recovery for Peephole Setting

Earlier in this Section, we ignored phase updates when synthesizing Clifford gates. Thus, the optimized Clifford circuit (either by cx-count or cx-depth) will have a different  $r$ -column in the tableau matrix. While relative phase can be ignored for pure Clifford circuit optimization, for general quantum circuits (i.e., in particular for peephole-optimization), we need to reconstruct the relative phase ( $r$ -column). This can be achieved separately for each qubit, by appending single Pauli gates to the optimized circuit. Appending Pauli gates at the beginning of the circuit is same as so-called *Pauli left multiplication*. In plain words, adding either X or Z Pauli gates at the beginning of the circuit flips the target tableau  $r$ -column. Remember that Pauli gates do not change  $x, z$  matrices but only act on the  $r$ -column. Phase recovery is then figuring out which Pauli gates to be applied to recover our target  $r$ -column. We observe that applying a  $Z_i$  gate flips  $r_i$  whereas  $X_i$  gate flips  $r_{n+i}$  in the target tableau.



■ **Figure 2** Equivalent Clifford circuits of Figure 1a, ignoring relative phase.

Let  $r, r'$  be  $r$ -columns of optimized and target tableau of an  $n$ -qubit circuit. Here, we present a simple algorithm that computes the sequence of X and Z gates that recovers the relative phase. Algorithm 1 clearly runs in linear time in the number of qubits. Consider the CNOT optimal example circuit in Figure 2c: the tableau deviates in the  $r$ -column. Appending X and Z gates as shown in Figure 3a recovers the  $r$ -column. This transformation is shown in Figure 3b. In our tool, we apply the recovery algorithm after synthesizing the optimal circuit. After every synthesis run, we compare the tableau of input and output circuits for correctness.

■ **Algorithm 1** Relative Phase Recovery.

---

```

1: SEQ is empty
2: for all  $i \in [0 \dots n-1]$  do
3:   if  $r_i \oplus r'_i = 1$  then
4:     APPEND  $Z_i$  to SEQ
5:   if  $r_{n+i} \oplus r'_{n+i} = 1$  then
6:     APPEND  $X_i$  to SEQ

```

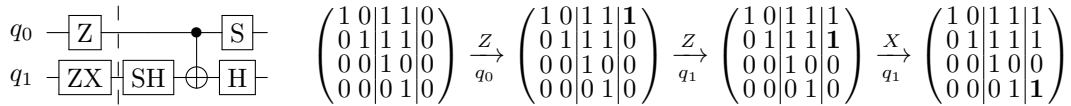
---

## 4 CNOT Optimal SAT Encodings

We now present bounded reachability SAT encodings for cx-count and cx-depth optimization based on the ideas discussed above. For cx-count optimization, we need to reach the target tableau in  $d$  entangling steps from the initial tableau. Encoding tableau updates for the entangling sequences across two time steps is quite complex. Instead, we define a *layer* that acts across three time steps. To encode layer  $k$  constraints, we encode 1-qubit gate updates from time step  $2k$  to  $2k + 1$  and CNOT updates from time step  $2k + 1$  to  $2(k + 1)$ . In each layer, we choose a control and target qubit on which the entangling sequence is applied. At the end, we need to apply a last layer of all 6 1-q sequences. In total, we need  $2d + 2$  time steps i.e., from  $t = 0$  to  $t = 2d + 1$  for a  $d$ -CNOT circuit encoding. Algorithm 2 presents the overall outline of the SAT encoding for a  $d$ -CNOT,  $n$ -qubit circuit. For cx-depth optimization, instead of exactly one control and target, we will allow any subset of independent (i.e., parallel) qubit pairs. Essentially, optimizing for the number of layers corresponds to optimizing for the chosen cx-count or cx-depth metric.

### 4.1 Gate optimal encoding

We define the variables as shown in Table 2. The variables in top half represent chosen control and target qubits, and applied gates. Variables in bottom half represent states of matrix elements. In addition, we define auxiliary variables to encode XOR constraints elegantly. For time steps  $t = 0$  to  $t = 2k$ , we set  $\text{px}_{i,a}^t$  true iff  $\text{x}_{i,a}^t$  and  $\text{x}_{i,a}^{t+1}$  are equal. Similarly, we



(a) Optimal circuit.

(b) Tableau transforms to correct phase.

■ **Figure 3** Phase recovery applied to the Clifford circuit of Figure 2c.



■ **Algorithm 2** SAT Encoding Outline for  $d$ -CNOT Circuit.

1: InitialStateConstraints	▷Encode initial tableau at $t = 0$ as in 4.1.4
2: GoalStateConstraints	▷Encode target tableau at $t = 2d + 1$ as in 4.1.4
3: <b>for all</b> $k \in [0 \dots d - 1]$ <b>do</b>	
4:   Encode1qConstraints( $k$ )	▷Encode I,HS,SH constraints in $k$ th layer as in 4.1.1
5:   EncodeCNOTConstraints( $k$ )	▷Encode CNOT constraints in $k$ th layer as in 4.1.2
6: Last1qConstraints	▷Encode last 1q constraints at $t = 2d$ as in 4.1.3

■ **Table 2** Encoding variables and descriptions.

Variable	Description
$\text{id}_a^k / \text{hs}_a^k / \text{sh}_a^k$	apply I/HS/SH gate on $a$ th qubit in the layer $k$
$\text{cnot}_{a,b}^k$	apply CNOT gate on $a, b$ qubits ( $a < b$ ) in the layer $k$
$\text{ctrl}_a^k / \text{trg}_a^k$	choose $a$ as a control/target qubit in the layer $k$
$\text{id}_a^L / \text{h}_a^L / \text{s}_a^L / \text{hs}_a^L / \text{sh}_a^L / \text{hsh}_a^L$	apply I/H/S/HS/SH/HSH gates on $a$ th qubit in the last layer
$x_{i,a}^t / z_{i,a}^t$	state of $x_{ia}/z_{ia}$ matrix element at time step $t$
$\text{px}_{i,a}^t / \text{pz}_{i,a}^t$	propagate $x_{ia}/z_{ia}$ matrix element from time step $t$ to $t + 1$

set  $\text{pz}_{i,a}^t$  true iff  $z_{i,a}^t$  and  $z_{i,a}^{t+1}$  are equal. We essentially represent propagation of  $x, z$  matrix states from  $t$  to  $t + 1$  time steps. In the following Subsections, we present constraints for each macro in Algorithm 2.

#### 4.1.1 Encoding 1-q I, HS, SH constraints

For a given layer number  $k$ , we define  $t := 2k$ . Encode1qConstraints( $k$ ) is then defined by the following constraints. Exactly one of the 3 single qubit gates is chosen on each qubit. We apply sequential counter ExactlyOne constraints from PySAT [11], referred as EO from here on. If a qubit is neither a control nor target, then we apply identity gate. Instead, if HS or SH gate is applied, the qubit must be either a control or target.

$$\bigwedge_{a=0}^{n-1} \text{EO}(\text{id}_a^k, \text{hs}_a^k, \text{sh}_a^k) \wedge \bigwedge_{a=0}^{n-1} (\neg \text{ctrl}_a^k \wedge \neg \text{trg}_a^k) \implies \text{id}_a^k \wedge \bigwedge_{a=0}^{n-1} (\text{hs}_a^k \vee \text{sh}_a^k) \implies (\text{ctrl}_a^k \vee \text{trg}_a^k) \quad (1)$$

For each of the three gates I, HS, SH, we encode following constraints:

$$\bigwedge_{a=0}^{n-1} \bigwedge_{i=0}^{2n-1} ( \quad \quad \quad ) \quad (2)$$

$$\begin{aligned} \text{id}_a^k &\implies (\text{px}_{i,a}^t \wedge \text{pz}_{i,a}^t) && \triangleright \text{I: } x_{ia} := x_{ia}; z_{ia} := z_{ia} \\ \text{hs}_a^k &\implies ((z_{i,a}^t \iff x_{i,a}^{t+1}) \wedge (x_{i,a}^t \iff \neg \text{pz}_{i,a}^t)) && \triangleright \text{HS: } x_{ia} := z_{ia}; z_{ia} := x_{ia} \oplus z_{ia} \\ \text{sh}_a^k &\implies ((x_{i,a}^t \iff z_{i,a}^{t+1}) \wedge (z_{i,a}^t \iff \neg \text{px}_{i,a}^t)) && \triangleright \text{SH: } z_{ia} := x_{ia}; x_{ia} := z_{ia} \oplus x_{ia} \end{aligned}$$

### 4.1.2 Encoding CNOT constraints

Let  $k$  be the layer number, then we define  $t := 2k + 1$ .  $\text{EncodeCNOTConstraints}(k)$  defines the CNOT constraints as follows. For cx-count optimization, we schedule exactly one CNOT gate in every layer and the chosen CNOT gate defines the control and target qubits.

$$\text{EO}(\{\text{cnot}_{a,b}^k \mid 0 \leq a < n; a < b < n\}) \wedge \bigwedge_{a=0}^{n-1} \bigwedge_{b=a+1}^{n-1} \text{cnot}_{a,b}^k \iff (\text{ctrl}_a^k \wedge \text{trg}_b^k) \quad (3)$$

Similar to 1q constraints, we define the tableau updates rules as below.

$$\bigwedge_{a=0}^{n-1} \bigwedge_{b=a+1}^{n-1} \bigwedge_{i=0}^{2n-1} \text{cnot}_{a,b}^k \implies ((x_{i,a}^t \iff \neg px_{i,b}^t) \wedge (z_{i,b}^t \iff \neg pz_{i,a}^t)) \quad (4)$$

$$\triangleright x_{ib} := x_{ia} \oplus x_{ib}; \quad z_{ia} := z_{ib} \oplus z_{ia}$$

$$\bigwedge_{a=0}^{n-1} \bigwedge_{i=0}^{2n-1} (\neg \text{trg}_a^k \implies px_{i,a}^t) \wedge \bigwedge_{b=0}^{n-1} \bigwedge_{i=0}^{2n-1} (\neg \text{ctrl}_b^k \implies pz_{i,b}^t) \quad \triangleright x_{ia} := x_{ia}; \quad z_{ib} := z_{ib} \quad (5)$$

Given a coupling, for every non-neighbor  $(a, b)$  qubit pair we add  $\neg \text{cnot}_{a,b}^t$ .

### 4.1.3 Final Layer 1q Constraints

Let  $t = 2d$ ,  $\text{Last1qConstraints}$  is defined by the following constraints. We apply exactly one of the 6 1-q unique gate sequences for each qubit  $a$  as  $\bigwedge_{a=0}^{n-1} \text{EO}(\text{id}_a^L, h_a^L, s_a^L, \text{hs}_a^L, \text{sh}_a^L, \text{hsh}_a^L)$ . The constraints for gate sequences  $\{I, \text{HS}, \text{SH}\}$  are exactly same as transition constraints presented earlier as in 4.1.1. We now give constraints for rest of the three gate sequences.

$$\bigwedge_{i=0}^{n-1} \bigwedge_{r=0}^{2n-1} ( \quad (6) \quad$$

$$\begin{aligned} h_i^L &\implies ((z_{i,a}^t \iff x_{i,a}^{t+1}) \wedge (x_{i,a}^t \iff z_{i,a}^{t+1})) && \triangleright \text{H: } x_{ia} := z_{ia}; \quad z_{ia} := x_{ia} \\ s_i^L &\implies (px_{i,a}^t \wedge (x_{i,a}^t \iff \neg pz_{i,a}^t)) && \triangleright \text{S: } x_{ia} := x_{ia}; \quad z_{ia} := x_{ia} \oplus z_{ia} \\ \text{hsh}_i^L &\implies (pz_{i,a}^t \wedge (z_{i,a}^t \iff \neg px_{i,a}^t)) && \triangleright \text{HSH: } z_{ia} := z_{ia}; \quad x_{ia} := z_{ia} \oplus x_{ia} \end{aligned}$$

### 4.1.4 Initial and Goal Constraints

Recall our discussion on tableau structure and initial tableau in Section 2.1. Let us divide the tableau without r-column into 4 equal quadrants. The left quadrants corresponds to the x matrix and the right quadrants correspond to the z matrix, where columns are labelled with qubits. Initially, the top-left and bottom-right quadrants correspond to the Identity matrix, whereas the other quadrants are zero matrices. Permuting columns of x and z matrices corresponds to qubit relabeling. Exactly-One constraints on rows and column of top-left quadrant represents column permutation.

$$\bigwedge_{i=0}^{n-1} ( \bigwedge_{a=0}^{n-1} ((x_{i,a}^0 \iff z_{n+i,a}^0) \wedge \neg x_{n+i,a}^0 \wedge \neg z_{i,a}^0) \wedge \text{EO}(\{x_{i,a}^0 \mid 0 \leq a < n\}) \wedge \text{EO}(\{x_{a,i}^0 \mid 0 \leq a < n\}) ) \quad (7)$$

We can disable column permutation (qubit relabeling) by forcing top-left quadrant as the Identity matrix i.e., adding clauses  $\bigwedge_{i=0}^{n-1} x_{i,i}^0$ .

For goal constraints, we first define the target tableau  $T$  where  $T[x_{ia}]/T[z_{ia}]$  returns the value of  $x/z$  matrix element in row  $i$  and column  $a$ . For a  $d$ -CNOT circuit, the following clauses encode the goal constraints.

$$\bigwedge_{i=0}^{2n-1} \bigwedge_{a=0}^{n-1} \left\{ \begin{array}{ll} x_{i,a}^{2d+1} & \text{if } T[x_{ia}] = 1 \\ \neg x_{i,a}^{2d+1} & \text{if } T[x_{ia}] = 0 \end{array} \right\} \wedge \left\{ \begin{array}{ll} z_{i,a}^{2d+1} & \text{if } T[z_{ia}] = 1 \\ \neg z_{i,a}^{2d+1} & \text{if } T[z_{ia}] = 0 \end{array} \right\} \quad (8)$$

#### 4.1.5 Improvements due to Search Space Reduction

We observe that the order between two parallel entangling sequences does not matter. Let  $\text{cnot}_{a,b}^{k-1}$  and  $\text{cnot}_{a',b'}^k$  be two parallel gates, i.e., they do not share control or target qubits. We order them based on the control qubits, i.e., if  $a > a'$ , then we add  $\neg \text{cnot}_{a,b}^{k-1} \vee \neg \text{cnot}_{a',b'}^k$ . Note that gate ordering constraints increases the number of clauses from  $O(dn^3)$  to  $O(dn^4)$ . Yet, these extra binary clauses reduce the search space and help the SAT solver.

In an optimal makespan, a valid solution can never repeat the state in any layer i.e., we need to consider simple paths only. Given layers  $k, k'$ , we use indicator variables  $\text{dr}_a^{k,k'}$ ,  $\text{dx}_a^{k,k'}$ ,  $\text{dz}_a^{k,k'}$  representing which row and x-column or z-column is different, respectively. The following constraints encode that if the  $i$ th row and the  $a$ th column is chosen, then the corresponding matrix variable is different. We need to find exactly one such matrix variable which is different.

$$\bigwedge_{a=0}^{n-1} \bigwedge_{i=0}^{2n-1} ((\text{dr}_i^{k,k'} \wedge \text{dx}_a^{k,k'}) \implies (x_{i,a}^{2k+2} \neq x_{i,a}^{2k'+2})) \wedge ((\text{dr}_i^{k,k'} \wedge \text{dz}_a^{k,k'}) \implies (z_{i,a}^{2k+2} \neq z_{i,a}^{2k'+2}))$$

$$\wedge \text{EO}(\{\text{dr}_i^{k,k'} \mid 0 \leq i < 2n\}) \wedge \text{EO}(\{\text{dx}_i^{k,k'} \mid 0 \leq i < n\} \cup \{\text{dz}_i^{k,k'} \mid 0 \leq i < n\})$$

To break any  $m$ -cycle, we instantiate the above constraints for every layer pair  $k, k'$  that differs in at most  $m$ . By default, we break all 3-cycles, since it doesn't increase the asymptotic variable and clause count.

#### 4.1.6 Redundant Clauses and Auxiliary Variables for Propagation

Finally, we optionally allow some redundant clauses and auxiliary variables to help our backend SAT solver. In SAT encodings, there is often a trade-off between search and propagation. In [25], we observed that adding redundant clauses sometimes improve performance of SAT solving. Giving explicit constraints that trigger unit clause propagation can avoid some unnecessary search. For example, Equation 4 explicitly encodes that exactly-one CNOT variable can be true at time step  $t$ . It also implicitly encodes that exactly-one control or target variable can be true in the same time step. If a SAT solver sets a CNOT variable to true, then all control, target, and rest of the CNOT variables are propagated. However, if a SAT solver instead sets a control variable to true it does not learn any more information. By giving (redundant) exactly-one constraints on control and target variables can avoid this unnecessary search.

Similarly, using auxiliary variables can impact performance of SAT solving. In Equation 1, recall that propagation variable  $\text{px}_{i,a}^t$  is true then  $x_{i,a}^t$  is propagated else  $x_{i,a}^t$  is flipped. We experimented with using additional auxiliary variable  $\text{fx}_{i,a}^t$  to indicate the flipping instead of  $\neg \text{px}_{i,a}^t$ . We add an option to use the new flipping variables for both  $x, z$  state variables as part of XOR constraints in our encoding. We provide these variations as an option, we turn them on default as they seem to help our backend SAT solver Cadical [4].

## 4.2 Depth optimal encoding

With minimal changes, we can encode optimal depth instead of optimal gate count. Instead of allowing a single CNOT gate, we allow multiple independent gates in each layer. In other words, we allow at most one CNOT gate at any control or target qubit. We use sequential counter AtMostOne (AMO) and AtLeastOne (ALO) constraints from Pysat [11]. We essentially replace Equation 4 with following Equation 9.

$$\bigwedge_{q=0}^{n-1} \text{AMO}(\{\text{cnot}_{a,b}^k \mid a = q \text{ or } b = q \text{ and } 0 \leq a, b < n\}) \wedge \text{ALO}(\{\text{cnot}_{a,b}^k \mid 0 \leq a, b < n\}) \wedge \bigwedge_{q=0}^{n-1} ((\text{ctrl}_q^k \implies \bigvee_{b=0}^{n-1} \text{cnot}_{q,b}^k) \wedge (\text{trg}_q^k \implies \bigvee_{a=0}^{n-1} \text{cnot}_{a,q}^k)) \quad (9)$$

Gate ordering restrictions are simpler for Depth optimal encoding. We eagerly schedule entangling gates without losing optimality. If a CNOT is applied on qubits  $i, j$ , we restrict that some entangling gate is applied on  $i$  or  $j$  qubits in the previous layer. Note that we only require  $O(dn^3)$  clauses even with the additional gate ordering constraints.

$$\bigwedge_{i=0}^{n-1} \bigwedge_{j=i+1}^{n-1} \text{cnot}_{i,j}^k \implies (\text{ctrl}_i^{k-1} \vee \text{ctrl}_j^{k-1} \vee \text{trg}_i^{k-1} \vee \text{trg}_j^{k-1})$$

## 5 Experiments and Results

We implemented above encodings and their variations in version 5 of our open source tool Q-Synth<sup>2</sup> [27]. Given a quantum circuit in OPENQASM 2.0 [8] format, we return the optimized circuit in the same format. For *pure* Clifford circuits, we return CNOT-optimal circuits for a given metric. Given a mapped circuit and a coupling graph of a quantum platform, we optimize respecting the coupling restrictions. Similarly, we generate circuits with a possible qubit relabeling when enabled. On the other hand, we employ peephole synthesis for arbitrary circuits. From start to end of the circuit, we first group non-Clifford and Clifford gates greedily. Then we replace each obtained Clifford slice by its optimized counterpart. Note that for general circuits this does not guarantee global optimality either in cx-count or cx-depth optimization.

### Search Strategies

For each Clifford optimization call, we provide two search strategies i.e., forward and backward. In forward search, we search for circuits with  $k$  CNOT gates, increasing from  $k = 0$  until we find the optimal circuit. Forward search either produces an optimal circuit or a timeout. In backward search, we search for circuits with  $\leq k$  CNOT gates, decreasing from the number of CNOT gates in the initial circuit until we reach UNSAT. Within the given time limit, we either synthesize an optimal circuit or report the best circuit found so far. In our initial experiments, we observed that search space reduction techniques help forward search. Thus, only in forward experiments, we add gate ordering constraints both for cx-count and cx-depth optimization. For forward cx-depth optimization, we additionally include simple path constraints as well. In Tables, we denote forward search with additional constraints as qsynth. Similarly, backward search is denoted as qsynth-b.

<sup>2</sup> Full code, data and logs are available at <https://doi.org/10.5281/zenodo.15575215>

## Research Questions

We are interested in investigating the effectiveness and relevance of our SAT based approach. In particular, we are interested in answering the following three research questions:

- R1: Does restricting search to Clifford normal forms help with scalability?
- R2: What is the effectiveness of SAT encodings on unmapped practical benchmarks?
- R3: What is the effectiveness of SAT encodings on layout mapped practical benchmarks?

We propose the following 3 experiments to answer the above research questions.

### 5.1 Experiment 1: Optimal synthesis

The main difference from existing SAT based approaches is our use of normal forms. In this experiment, we investigate if normal form restricted search helps with scalability. For comparison, we consider QMAP [29] tool that implements previous exact sat-based synthesis of Clifford circuits. QMAP provides near-optimal synthesis for cx-count [22] and optimal synthesis for circuit depth [20]. In [22, 20], QMAP presented results mainly on synthesis of random Clifford circuits from given stabilizers. Random Clifford circuits are one of the standard benchmarks for comparing various approaches. Unfortunately, the random stabilizers used by QMAP in [22, 20] are not accessible. Thus, we generated new random benchmarks of 3 to 7 qubits, with 5 random instances for each qubit. For benchmark generation, we used the standard Qiskit random Clifford stabilizer function. For each such stabilizer, we further optimized with TKET [28] both with and without permutation. Our input benchmarks are essentially the best circuits synthesized by Qiskit and TKET. Any improvement on such benchmarks essentially shows the strength of exact approaches. All generated benchmarks and their scripts are available online (for reproducibility).

For cx-count optimization, we compared with near-optimal cx-count optimization by QMAP [22]. For cx-depth optimization, we compare with optimal circuit depth optimization by QMAP [20] and report cx-depth instead of circuit depth. We chose the best (default) options i.e., binary search, in QMAP v2.8 for a fair comparison. We present results for Q-Synth with forward and backward search, with- and without-permutations. For each instance, we give 3h time limit and 8GB memory limit. For Q-Synth, we use state-of-the-art SAT solver Cadical (v2.1) [4] as a backend. We use Pysat [11] for sequential counter cardinality constraints.

## Results and Discussion

Table 3 presents the results of Experiment 1 with cx-count and cx-depth optimization. For cx-count optimization, we can see that the near-optimal cx-count synthesis by QMAP does not scale well and only solves 3-qubit instances. Q-Synth, on the other hand, with forward search can solve all instances up to 5-qubits and 3 out of 5 6-qubit instances. With backward search, Q-Synth even improves cx-count of 7-qubit instances but uses full 3 hour time limit to do so. In the presence of output permutation, the results are similar to without output permutation while solving fewer 6-qubit instances. For cx-depth optimization, QMAP (without output permutation) can now solve up to 5-qubit instances with optimal depth. Note that QMAP often reports suboptimal cx-depth when optimizing depth. For the hard 6-qubit instances QMAP produces intermediate results, but they are far from optimal. Q-Synth on the other hand, solves all but one 7-qubit instance with forward search. Interestingly with backward search we can improve the unsolved 7-qubit instance but not solve optimally. With output permutation, Q-Synth can optimally solve up to 6-qubit instances but not 7-qubit instances. With backward search, Q-Synth can improve 1 extra 7-qubit instance. We provide the full data in Tables 6 and 7 in Section B.

■ **Table 3** Experiment 1: Optimizing of random Clifford circuits,  $n$ : qubits,  $m$ : optimization metric (average cx-count/cx-depth of 5 instances),  $t$ : average time if all 5 instances are completely solved; else the number of instances solved within the time limit.

	without-permutation								with-permutation							
	tket		qmap		qsynth		qsynth-b		tket		qsynth		qsynth-b			
$n$	m		m	t	m	t	m	t	m		m	t	m	t		
(CNOT-count optimization, $m = \text{cx-count}$ )																
3	3.6		3.6	1002.8	3.6	0.1	3.6	0.1	2.8		2.8	0.1	2.8	0.1		
4	6.8		–	0/5	6.0	0.3	6.0	0.2	6.0		4.8	0.3	4.8	0.4		
5	13.2		–	0/5	8.2	7.1	8.2	21.9	10.2		7.2	19.5	7.2	70.3		
6	17.0		–	0/5	12.6	3/5	11.6	2/5	14.4		13.6	1/5	11.0	0/5		
7	22.8		–	0/5	–	0/5	21.0	0/5	20.4		–	0/5	18.4	0/5		
(CNOT-depth optimization, $m = \text{cx-depth}$ )																
3	3.6		3.6	0.4	3.6	0.1	3.6	0.1	2.8		2.8	0.1	2.8	0.1		
4	5.6		4.8	5.1	3.6	0.1	3.6	0.2	4.8		3.0	0.1	3.0	0.2		
5	11.2		7.2	434.3	4.8	1.2	4.8	16.2	8.2		4.0	2.5	4.0	5.8		
6	13.6		13.2	0/5	5.0	10.5	5.0	839.4	11.2		5.0	82.3	5.0	583.0		
7	18.0		–	0/5	8.6	4/5	17.8	0/5	15.4		–	0/5	15.0	0/5		

From the results, it is clear that restricting search to normal form indeed helps with scalability. We outperform previous SAT approaches by several orders of magnitude in both optimization metrics. Notice that restricting to normal forms achieves two goals. First, the makespan in our encodings correspond to cx-count or cx-depth. On the other hand, the makespan in previous encodings correspond to gate count and circuit depth. Note that gate count or circuit depth can be several times higher than their counterparts. Second, we avoid search through many equivalent circuits thus reducing search space. We conjecture that good performance is due to lower makespan and reduced search space.

Coming to weaknesses of our approach, we could not solve all 6-qubit instances for cx-count optimization. Existing exhaustive based approach [5] could generate a database of all 6-qubit instances. Essentially, an exhaustive approach allows more freedom in avoiding search of equivalent (or symmetric) circuits. It is unclear to us how to include such symmetry breaking in a SAT encoding. On the other hand, our cx-depth optimization can go beyond 6-qubits which is not feasible in exhaustive approach.

## 5.2 Experiment 2: Effect on All-to-All Practical Benchmarks

In practical benchmarks, the Clifford sub-circuits are often shallow. We are interested in investigating the effect of our SAT based optimization on such benchmarks. Here we optimize benchmarks assuming all-to-all qubit connectivity. For practical benchmarks, we consider standard variational quantum eigensolver (VQE) benchmarks as the first benchmark set. Similar to [25], we generated 12 random VQE benchmarks of 8 and 16 qubits with up to 536 cx-depth and 860 cx-count. For the second benchmark set, we used 28 instances from the standard benchmark collection of Feynman tool [2] with up to 24 qubits, 2149 cx-count, and 1878 cx-depth. We consider best cx-count reduction compiler TKET [28, 17] for a comparison. We also experiment with combined optimization of TKET and Q-Synth, we



denote it by TK+QS. In the TK+QS configuration, we first optimize each instance with TKET and then optimize with Q-Synth. In the experiment with qubit permutation, we also enable permutation in TKET. Waiting 3 hours for a single benchmark is not practical. In Experiments 2 and 3, we run Q-Synth only with forward search, with a 600s time limit.

■ **Table 4** Experiment 2: Optimizing practical benchmarks; org: original optimization metric, ch%: change% of optimization metric (lower is better), t: time.

			without-permutation						with-permutation					
			tket		qsynth		TK+QS		tket		qsynth		TK+QS	
org			ch%	ch%	t	ch%	t	ch%	ch%	t	ch%	t		
VQE cx-count	avg	575.8	-3.5	-3.3	321.9	-3.9	318.4	-5.8	-4.4	371.4	-6.1	364.0		
	min	298	-7.7	-8.2	8.3	-8.5	8.2	-8.5	-8.9	16.4	-9.8	16.2		
	max	860	-0.6	-0.6	604	-0.6	604.4	-0.7	-0.7	603.2	-0.7	603.3		
VQE cx-depth	avg	367.6	-2.9	-9.1	20.1	-9.6	19.2	-4	-10.8	72.2	-11.2	58.8		
	min	233	-8.2	-16	3.1	-16.7	3.4	-8.6	-16.7	4.9	-17.1	4.8		
	max	536	0	-3	60.5	-3.8	51.9	0	-3.8	363.5	-4.7	230.6		
Feynman cx-count	avg	222	-8.7	-8.4	207.2	-11.2	208.7	-9.8	-8.8	233.3	-12.4	218.4		
	min	18	-28.6	-32.1	0.2	-32.1	0.2	-28.6	-32.1	0.2	-32.6	0.2		
	max	2149	0	0	612.3	0	613.5	0	0	610.3	0	609.5		
Feynman cx-depth	avg	161	-7.8	-12.3	89.7	-14.4	89.5	-8.3	-12.6	104.5	-14.7	104.2		
	min	16	-29.6	-48.1	0.2	-48.1	0.2	-29.6	-48.1	0.2	-48.1	0.2		
	max	1878	0	1.9	604.3	0	604.2	0	4.9	603.8	1.9	603.7		

## Results and Discussion

Table 4 presents the results of Experiment 2. We present average, minimum, and maximum change in optimization metric and time taken. For cx-count optimization, Q-Synth produces similar reductions compared to TKET in both benchmark sets. We observed up to 8.9% and 32.1% reduction in VQE and Feynman benchmarks respectively by Q-Synth. There does exist some hard slices where Q-Synth timeouts resulting in better results by TKET. For cx-depth reduction, Q-Synth overall outperforms TKET for both benchmarks sets. We observed up to 16.7% and 48.1% reduction in VQE and Feynman benchmarks respectively by Q-Synth. We usually observe the best results with TKET+Q-Synth combination. Full tables are available in [26, Appendix B] for all variations.

From the results, we can see that SAT based approaches perform well also on practical benchmarks. Remember that our benchmarks can have up to 24 qubits, 2149 cx-count, and 1878 cx-depth. Q-Synth produces better cx-depth results up to 48.1% compared to 29.1% by TKET. Such reductions are feasible since TKET focuses on cx-count. Indeed, TKET on an average produces better cx-count reduction than Q-Synth on large instances. Since cx-count and cx-depth are both important metrics, optimizing both is helpful in practice. We see the best results by combining the strengths of TKET and Q-Synth in TK+QS combination. Our SAT based cx-depth optimization appears to nicely complement heuristic cx-count optimization. We conclude that using the combination TK+QS for reduction in both cx-count and cx-depth is feasible in a practical compilation setting.

### 5.3 Experiment 3: Effect on Mapped Practical Benchmarks

Layout synthesis routines in industrial compilers are fast but sub-optimal. Many Clifford optimization techniques do not support layout-aware synthesis, for example TKET. Intuitively, connectivity restrictions break assumptions made in all-to-all connectivity. SAT based approaches can elegantly encode connectivity constraints. In this experiment, we investigate if our approach can further reduce mapped practical circuits by industrial compilers. Our goal is to investigate the reduction in best synthesized benchmarks from industrial compilers. We consider TKET optimized benchmarks with qubit permutations from Experiment 2. We then mapped the benchmarks onto 54-qubit Sycamore [3], 80-qubit Rigetti [7] and 127-qubit Eagle [6] platforms using Qiskit. During Qiskit transpiling, we used the highest optimization “-O3” level. For each benchmark, we optimize with Q-Synth for both cx-count and cx-depth.

■ **Table 5** Experiment 3: Optimizing mapped practical benchmarks; org: original optimization metric, ch%: change% of optimization metric (lower the better), t: time.

		sycamore-54			rigetti-80			eagle-127		
		org	ch%	t	org	ch%	t	org	ch%	t
VQE cx-count	avg	1253.7	-7.7	604.1	1550.2	-8.8	603.8	1842.9	-9.4	568
	min	515	-15.9	602.2	644	-19.3	602.7	731	-18.3	158.1
	max	2179	-2	607.4	2698	-2.3	605.1	3172	-2.5	607.2
VQE cx-depth	avg	818.1	-15.8	277.9	981.3	-17.2	354.2	1150.4	-18.9	344.5
	min	419	-27.4	24	514	-25.9	28.5	597	-24.8	33
	max	1377	-9.1	607.8	1579	-9.3	607.7	1886	-12.3	611.9
Feynman cx-count	avg	433.4	-16.2	264.5	497.4	-18.5	224.3	559.3	-21	219
	min	33	-24.8	0.7	38	-27.3	0.5	39	-30.3	0.6
	max	3564	-7.9	632	4173	-5.9	611.9	4809	-9	622.6
Feynman cx-depth	avg	308.5	-22.1	97.1	349.7	-22.7	119.4	386.2	-23	134.7
	min	30	-32.6	0.7	37	-32.6	0.5	37	-35.9	0.6
	max	2849	-10.7	610.8	3322	-7.9	610.7	3790	-6.6	620.9

### Results and Discussion

Table 5 presents the results for all three platforms. On all platforms, Q-Synth reports good reduction for both cx-count and cx-depth metrics. For VQE benchmarks, we observed reduction of up to 19.3% cx-count and 27.4% cx-depth. For Feynman benchmarks, we observed reduction of up to 30.3% cx-count and 35.9% cx-depth. Full tables are available in [26, Appendix B] for all variations.

The separation of layout and circuit synthesis in current compilers results in suboptimal circuits. We see that SAT based approaches are effective in post layout optimization. From the full tables, we also observe that mapping and optimizing to Google’s Sycamore produces the best results. We conjecture that the denser array structure of the coupling graph is the main factor. The coupling graphs of Rigetti with octagons and Eagle with heavy-hex require many extra CNOT gates to route qubits, at least on our benchmarks. Despite our good reductions, the final cx-count/cx-depth on these platforms are higher than on Sycamore. Overall, the results indicate that SAT based optimization is useful in the compilation pipeline.

## 6 Conclusion

In this paper, we proposed two SAT encodings that synthesize Clifford circuits with optimal CNOT count or depth. We also handle connectivity restrictions and allow qubit permutations. We implemented all encoding variations in the open source tool Q-Synth. The experiments demonstrate scalability compared to existing SAT encodings. For the first time, we are able to synthesize cx-depth optimal 7-qubit random Clifford circuits. On practical VQE and Feynman benchmarks, we overall perform better than TKET in all-to-all connectivity. We also investigated layout-aware optimization on major quantum platforms. We first optimize benchmarks with TKET, then map them using Qiskit and finally optimize the results using Q-Synth. This achieves reductions of up to 30.3% in cx-count and 35.9% in cx-depth. These results show that our SAT based approach nicely complements existing heuristic approaches.

---

## References

- 1 Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5), November 2004. doi:10.1103/physreva.70.052328.
- 2 Matthew Amy. Quantum circuit analysis toolkit, 2016. URL: <https://github.com/meamy/feynman>.
- 3 Frank Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019. doi:10.1038/s41586-019-1666-5.
- 4 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximilian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1, pages 51–53. University of Helsinki, 2020. URL: <https://api.semanticscholar.org/CorpusID:220727106>.
- 5 Sergey Bravyi, Joseph A. Latone, and Dmitri Maslov. 6-qubit optimal clifford circuits. *npj Quantum Information*, 8(1), July 2022. doi:10.1038/s41534-022-00583-7.
- 6 Jerry Chow, Oliver Dial, and Jay Gambetta. Ibm quantum breaks the 100-qubit processor barrier. *IBM Research Blog*, 2, 2021. URL: <https://www.ibm.com/quantum/blog/127-qubit-quantum-processor-eagle>.
- 7 Rigetti Computing. Rigetti computing. URL: <https://www.rigetti.com>.
- 8 Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. Open quantum assembly language, 2017. arXiv:1707.03429.
- 9 Arianne Meijer-van de Griend and Sarah Meng Li. Dynamic qubit routing with CNOT circuit synthesis for quantum compilation. In *Proceedings 19th International Conference on Quantum Physics and Logic, QPL 2022, Wolfson College, Oxford, UK, 27 June - 1 July 2022*, volume 394 of *EPTCS*, pages 363–399, 2022. doi:10.4204/EPTCS.394.18.
- 10 Patrick Ettenhuber, Mads Bøttger Hansen, Pier Paolo Poier, Irfansha Shaik, Stig Elkjær Rasmussen, Niels Kristian Madsen, Marco Majland, Frank Jensen, Lars Olsen, and Nikolaj Thomas Zinner. Calculating the energy profile of an enzymatic reaction on a quantum computer. *Journal of Chemical Theory and Computation*, 21(7):3493–3503, 2025. PMID: 40162965. doi:10.1021/acs.jctc.5c00022.
- 11 Alexey Gnatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018. doi:10.1007/978-3-319-94144-8\_26.
- 12 Kazuo Iwama, Yahiko Kambayashi, and Shigeru Yamashita. Transformation rules for designing cnot-based quantum circuits. In *Proceedings of the 39th Design Automation Conference, DAC 2002, New Orleans, LA, USA, June 10-14, 2002*, pages 419–424. ACM, 2002. doi:10.1145/513918.514026.
- 13 Jiaqing Jiang, Xiaoming Sun, Shang-Hua Teng, Bujiao Wu, Kewen Wu, and Jialin Zhang. Optimal space-depth trade-off of CNOT circuits in quantum logic synthesis. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 213–229. SIAM, 2020. doi:10.1137/1.9781611975994.13.

- 14 Henry A. Kautz, David A. McAllester, and Bart Selman. Encoding plans in propositional logic. In *Proceedings of KR-96*, pages 374–384, November 1996. URL: <https://henrykautz.com/papers/plankr96.pdf>.
- 15 Wan-Hsuan Lin, Jason Kimko, Bochen Tan, Nikolaj Bjørner, and Jason Cong. Scalable optimal layout synthesis for NISQ quantum processors. In *DAC*, 2023. doi:10.1109/DAC56929.2023.10247760.
- 16 Harsha Nagarajan, Owen Lockwood, and Carleton Coffrin. QuantumCircuitOpt: An open-source framework for provably optimal quantum circuit design. *2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS)*, pages 55–63, 2021. doi:10.1109/QCS54837.2021.00010.
- 17 Paul D. Nation, Abdullah Ash Saki, Sebastian Brandhofer, Luciano Bello, Shelly Garion, Matthew Treinish, and Ali Javadi-Abhari. Benchmarking the performance of quantum computing software. *CoRR*, abs/2409.08844, 2025. arXiv:2409.08844.
- 18 Michael A. Nielsen and Isaac L. Chuang. *Quantum circuits*, pages 171–215. Cambridge University Press, 2010. doi:10.1017/CB09780511976667.008.
- 19 K. N. Patel, I. L. Markov, and J. P. Hayes. Efficient synthesis of linear reversible circuits, 2003. arXiv:quant-ph/0302002.
- 20 Tom Peham, Nina Brandl, Richard Kueng, Robert Wille, and Lukas Burgholzer. Depth-optimal synthesis of Clifford circuits with SAT solvers. *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 01:802–813, 2023. doi:10.1109/QCE57702.2023.00095.
- 21 Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023. doi:10.5281/zenodo.2573505.
- 22 Sarah Schneider, Lukas Burgholzer, and Robert Wille. A SAT encoding for optimal Clifford circuit synthesis. *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 190–195, 2022. doi:10.1145/3566097.3567929.
- 23 Irfansha Shaik and Jaco van de Pol. Optimal layout synthesis for quantum circuits as classical planning. In *IEEE/ACM International Conference on Computer Aided Design, ICCAD 2023, San Francisco, CA, USA, October 28 - Nov. 2, 2023*, pages 1–9. IEEE, 2023. doi:10.1109/ICCAD57390.2023.10323924.
- 24 Irfansha Shaik and Jaco van de Pol. Optimal layout-aware CNOT circuit synthesis with qubit permutation. In *ECAI 2024*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pages 4207–4215. IOS Press, 2024. doi:10.3233/FAIA240993.
- 25 Irfansha Shaik and Jaco van de Pol. Optimal layout synthesis for deep quantum circuits on NISQ processors with 100+ qubits. In *27th International Conference on Theory and Applications of Satisfiability Testing SAT*, volume 305 of *LIPICs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.SAT.2024.26.
- 26 Irfansha Shaik and Jaco van de Pol. CNOT-Optimal Clifford Synthesis as SAT (full version). *CoRR*, abs/2504.00634, 2025. arXiv:2504.00634.
- 27 Irfansha Shaik and Jaco van de Pol. Code, benchmarks, scripts and data for “CNOT-optimal Clifford synthesis as SAT” paper at SAT25, June 2025. doi:10.5281/zenodo.15575215.
- 28 Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. t[ket]: a retargetable compiler for NISQ devices. *Quantum Science and Technology*, 6(1):014003, November 2020. doi:10.1088/2058-9565/ab8e92.
- 29 Robert Wille and Lukas Burgholzer. MQT QMAP. In *Proceedings of ISPD-23*. ACM, March 2023. doi:10.1145/3569052.3578928.
- 30 Robert Wille, Lukas Burgholzer, and Alwin Zulehner. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *DAC-19*, page 142. ACM, 2019. doi:10.1145/3316781.3317859.

## A Quantum Computing Notation and Clifford Circuits

We provide a short introduction to the notations used in quantum computing. For a full textbook explanation we refer to [18]. We also shortly summarize the stabilizer formalism for Clifford circuits introduced in [1].

A qubit can be viewed as a 2-dimensional complex vector in the basis  $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$  and  $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ . That is, an arbitrary single qubit is a pair  $\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha|0\rangle + \beta|1\rangle$ , for some  $\alpha, \beta \in \mathbb{C}$  such that  $|\alpha|^2 + |\beta|^2 = 1$ . The equal superposition  $|+\rangle$  is defined as  $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ . A system of  $n$  qubits forms a  $2^n$ -dimensional space. For instance, the base vector  $|010\rangle$  is the tensor product  $|0\rangle \otimes |1\rangle \otimes |0\rangle$  of dimension 8.

A quantum operation on  $n$  qubits is a unitary transformation (i.e., linear and reversible) and can be represented by a  $2^n \times 2^n$  matrix  $U$  such that  $UU^\dagger = I$ . Quantum circuits are used as a concise notation for quantum operations. A quantum circuit is a sequence of quantum gates, where each operation is carried out on a few qubits only.

An important class of 1-qubit operations are the Pauli matrices, which are defined as:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

One can check that  $X|0\rangle = |1\rangle$  and  $X|1\rangle = |0\rangle$ , so  $X$  negates the basis vectors. The Pauli matrices, together with their  $\pm 1, \pm i$  multiples, form the Pauli group  $P_1$  that satisfies many nice equations, e.g.  $XX = I$ ,  $XZ = -ZX$  and  $XY = -iZ$  (this can be easily checked by matrix multiplication). It can also be checked that  $Z|0\rangle = |0\rangle$ , i.e.,  $Z$  stabilizes  $|0\rangle$ . Similarly,  $-Z|1\rangle = |1\rangle$  and  $X|+\rangle = |+\rangle$ . The Pauli group  $P_n$  on  $n$  qubits simply consists of Pauli strings, i.e., a tensor product of  $n$  Pauli matrices multiplied by a constant:  $i^c P_1 \otimes \dots \otimes P_1$  (for  $c \in \{0, 1, 2, 3\}$ ). For instance  $-X \otimes Z \otimes X \in P_3$  stabilizes the 3-qubit state  $|+1+\rangle$ .

The Pauli group forms a small but useful fragment of quantum computing, since it represents the stabilizers of some interesting quantum states. A larger fragment can be generated by the Clifford gates, which consist of the Hadamard gate ( $H$ ), the Phase gate ( $S$ ) and the binary (entangling) gate  $CNOT$  or  $CX$  (conditional NOT).

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

On basis states  $a$  and  $b$ ,  $CNOT(a, b)$  returns  $(a, a \oplus b)$ , i.e., it negates  $b$  if  $a = |1\rangle$ . The Hadamard gate can be used to create a superposition, i.e.  $H|0\rangle = |+\rangle$ , and  $HH = I$ . One can check that  $Z = SS$ ,  $X = HZH$  and  $Y = iXZ$ , so the Pauli gates can be expressed in terms of the Clifford gates. As a matter of fact the Clifford gates correspond exactly to the normalizers of  $P_n$ , that is, those unitary transformations  $U$  such that  $UP_nU^\dagger = P_n$ . For instance,  $SXS^\dagger = Y$  and  $HXH^\dagger = Z$ .

The core of the stabilizer formalism [1] is that quantum circuits consisting of Clifford gates can be simulated in polynomial time by tracking the effect of the gates on the states efficiently. Instead of representing a state as a  $2^n$ -dimensional complex vector, it is represented by  $n$  Pauli strings that together generate all stabilizers of that state. And the effect of each Clifford gate can be defined directly on those stabilizers. This information can be concisely stored in a tableau, using 2 bits for each Pauli matrix,  $00 = I$ ,  $10 = X$ ,  $01 = Z$  and  $11 = Y$ . A row of  $2n + 1$  bits represents a single Pauli string, where the one extra bit is used to encode the sign  $\pm 1$ . For technical reasons, we also keep track of  $n$  so-called destabilizers, explaining the  $2n \times (2n + 1)$  Boolean matrices that are used in the SAT encoding.

**B Experiment 1: Full Tables****Table 6** Experiment 1: Random Clifford cnot-count optimization, m: cx-count t: time, TO: timeout.

Tool: instance	without-permutation								with-permutation			
	tket	qmap		qsynth-f		qsynth-b		tket	qsynth-f		qsynth-b	
	m	m	t	m	t	m	t	m	m	t	m	t
3q05306	4	4	1452.5	4	0.13	4	0.06	2	2	0.1	2	0.09
3q33936	3	3	29.3	3	0.07	3	0.05	3	3	0.07	3	0.06
3q50494	4	4	1804.7	4	0.08	4	0.05	3	3	0.06	3	0.05
3q55125	3	3	77.2	3	0.06	3	0.04	3	3	0.06	3	0.05
3q99346	4	4	1650.2	4	0.08	4	0.05	3	3	0.06	3	0.05
4q05306	6	–	TO	6	0.2	6	0.12	6	5	0.44	5	0.5
4q33936	8	–	TO	6	0.38	6	0.31	8	5	0.35	5	0.71
4q50494	7	–	TO	7	0.49	7	0.38	4	4	0.1	4	0.07
4q55125	6	–	TO	6	0.18	6	0.11	6	5	0.43	5	0.33
4q99346	7	–	TO	5	0.19	5	0.25	6	5	0.43	5	0.35
5q05306	12	–	TO	8	4.74	8	14.1	11	8	34.93	8	151.2
5q33936	13	–	TO	9	11.7	9	33.44	10	8	34.35	8	126.5
5q50494	17	–	TO	8	4.03	8	36.89	11	7	22.35	7	56.6
5q55125	13	–	TO	9	13.89	9	22.23	9	7	4.49	7	9.25
5q99346	11	–	TO	7	1.05	7	2.86	10	6	1.52	6	7.88
6q05306	15	–	TO	–	TO	12	TO	15	15	TO	11	TO
6q33936	20	–	TO	11	2017.7	12	TO	14	14	TO	11	TO
6q50494	17	–	TO	11	964.3	11	4525.9	14	14	TO	12	TO
6q55125	18	–	TO	11	1060.2	11	5059.7	14	10	6452.9	10	TO
6q99346	15	–	TO	–	TO	12	TO	15	–	TO	11	TO
7q05306	25	–	TO	–	TO	–	TO	24	–	TO	19	TO
7q33936	24	–	TO	–	TO	–	TO	20	–	TO	17	TO
7q50494	22	–	TO	–	TO	18	TO	20	–	TO	–	TO
7q55125	22	–	TO	–	TO	17	TO	18	–	TO	17	TO
7q99346	21	–	TO	–	TO	–	TO	20	–	TO	19	TO



■ **Table 7** Experiment 1: Random Clifford cnot-depth optimization, m: cx-depth, t: time, TO: timeout.

Tool:	without-permutation							with-permutation				
	tket	qmap		qsynth-f		qsynth-b		tket	qsynth-f		qsynth-b	
	m	m	t	m	t	m	t	m	m	t	m	t
3q05306	4	4	0.4	4	0.13	4	0.06	2	2	0.1	2	0.09
3q33936	3	3	0.39	3	0.07	3	0.05	3	3	0.07	3	0.05
3q50494	4	4	0.25	4	0.08	4	0.05	3	3	0.06	3	0.04
3q55125	3	3	0.42	3	0.06	3	0.04	3	3	0.06	3	0.05
3q99346	4	4	0.34	4	0.08	4	0.05	3	3	0.06	3	0.05
4q05306	4	5	4.0	4	0.11	4	0.07	4	3	0.12	3	0.1
4q33936	7	5	4.27	4	0.16	4	0.34	7	3	0.14	3	0.57
4q50494	6	6	6.85	4	0.17	4	0.17	3	3	0.07	3	0.05
4q55125	4	4	2.53	3	0.1	3	0.09	4	3	0.12	3	0.1
4q99346	7	4	8.08	3	0.11	3	0.44	6	3	0.21	3	0.36
5q05306	10	8	437.1	5	1.64	5	9.68	8	4	2.57	4	4.56
5q33936	10	7	533.8	5	0.91	5	6.3	8	4	2.81	4	7.0
5q50494	15	7	626.2	5	2.25	5	40.49	9	4	5.71	4	6.98
5q55125	12	8	347.7	5	0.91	5	21.45	8	4	1.27	4	4.98
5q99346	9	6	226.7	4	0.39	4	3.25	8	4	0.37	4	5.41
6q05306	11	–	TO	5	23.59	5	299.27	11	5	168.09	5	607.01
6q33936	16	11	TO	5	6.39	5	994.98	10	5	60.75	5	379.15
6q50494	14	14	TO	5	1.86	5	741.38	11	5	64.86	5	357.08
6q55125	15	17	TO	5	3.03	5	638.7	12	5	35.44	5	587.76
6q99346	12	11	TO	5	17.66	5	1522.43	12	5	82.35	5	984.03
7q05306	19	–	TO	–	TO	18	TO	18	–	TO	–	TO
7q33936	19	–	TO	6	4151.7	–	TO	16	–	TO	–	TO
7q50494	18	–	TO	6	7347.0	–	TO	15	–	TO	–	TO
7q55125	19	–	TO	6	1485.5	–	TO	14	–	TO	12	TO
7q99346	15	–	TO	6	241.0	–	TO	14	–	TO	–	TO