

# Optimal Layout-Aware CNOT Circuit Synthesis with Qubit Permutation

Irfansha Shaik<sup>a,b</sup> and Jaco van de Pol<sup>a</sup>

<sup>a</sup>Department of Computer Science, Aarhus University, Denmark

<sup>b</sup>Kvantify ApS, DK-2300 Copenhagen S, Denmark

ORCID (Irfansha Shaik): <https://orcid.org/0000-0002-7404-348X>, ORCID (Jaco van de Pol):

<https://orcid.org/0000-0003-4305-0625>

**Abstract.** CNOT optimization plays a significant role in noise reduction for Quantum Circuits. Several heuristic and exact approaches exist for CNOT optimization. In this paper, we investigate more complicated variations of optimal synthesis by allowing qubit permutations and handling layout restrictions. We encode such problems into Planning, SAT, and QBF. We provide optimization for both CNOT gate count and circuit depth. For experimental evaluation, we consider standard T-gate optimized benchmarks and optimize CNOT sub-circuits. We show that allowing qubit permutations can further reduce up to 56% in CNOT count and 46% in circuit depth. In the case of optimally mapped circuits under layout restrictions, we observe a reduction up to 17% CNOT count and 19% CNOT depth.

## 1 Introduction

Quantum Computing promises speedup in solving computationally hard and classically intractable problems. Logical formulations of such problems are compiled to enable execution on quantum processors. The Quantum compilation pipeline broadly consists of two main stages, Circuit Synthesis and Layout Synthesis. Circuit Synthesis mainly focuses on the decomposition of abstract circuits to a target gate set. Layout Synthesis instead focuses on satisfying hardware restrictions. For instance, not all physical qubits interact with each other in some current quantum processors. Thus, quantum gates that act on 2 qubits can only be scheduled on adjacent physical qubits. In the current Noisy Intermediate Scale Quantum (NISQ) era, noise is inherent to quantum computers. Every execution of a gate can increase the error in the computation. For practical quantum computing, error reduction is of utmost importance. Optimization techniques are applied throughout the compilation pipeline. In particular, reducing gate count and circuit depth can directly impact the error rate.

While an optimal synthesis for the whole compilation pipeline is ideal, it is an extremely hard problem. For instance, [23] proposes SMT-based synthesis under hardware connectivity restrictions and target gate set. From [23], it is clear that synthesis beyond 4 qubits is impractical. Essentially, to optimize a  $n$  qubit circuit one needs to consider its  $2^n \times 2^n$  unitary matrix. The alternative is to optimize error-prone gates like 1-qubit T-gates and 2-qubit CNOT-gates for error reduction. Several approaches are applied for T-gate optimization [2, 1] in tools like T-par<sup>1</sup> and Feynman.<sup>2</sup> While such tools reduce

T-gate count and depth, they can significantly increase CNOT-gate count. As a result, CNOT optimization without changing the T-gate count has been proposed, based on Gaussian elimination [3], Greedy algorithms [8, 9], Steiner tree [20, 15], SAT [21], and ASP [27, 26]. While heuristic techniques are well studied, exact approaches are still unexplored in many variations.

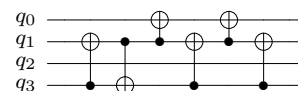
**Contributions** We consider two variations of CNOT synthesis, one with qubit permutation and one with CNOT restrictions. For qubit permutation, we define weak equivalence (W) where the order of output qubits is free. This allows for more – often smaller – solutions than exact synthesis with strong equivalence (S). For CNOT restrictions (R), we only allow CNOT gates on adjacent qubits, i.e., layout-aware synthesis. We are in particular interested in 4 variants: S, S+R, W, and W+R. Adding restrictions makes the problem NP-hard [19], while the complexity without them is still open [19]. We encode such hard problems into Classical Planning, Propositional satisfiability (SAT), and Quantified Boolean Formulas (QBF). For the first time, we provide optimal encodings for W and W+R synthesis variants. For the S, S+R and W variants, we experiment with peephole optimization on arbitrary quantum circuits, in which individual CNOT-slices are optimized. We validate this on standard T-gate optimal benchmarks. We extended our open source tool Q-Synth v3<sup>3</sup> to include all encoding variants of CNOT synthesis mentioned above.

## 2 Preliminaries

### 2.1 CNOT circuits

In this paper, we focus on special circuits called CNOT circuits, which consist solely of 2-qubit CNOT gates (controlled-NOT). A CNOT gate takes two inputs, a control qubit with  $a$  and a target qubit with  $b$ , and outputs  $a \oplus b$  on the target qubit. For example, Table 1 shows a CNOT circuit with 6 CNOT gates.

**Table 1.** Original CNOT Circuit



<sup>1</sup> <https://github.com/meamy/t-par>

<sup>2</sup> <https://github.com/meamy/feynman>

<sup>3</sup> Q-Synth v3, available at <https://github.com/irfansha/Q-Synth>

**Table 2.** Column additions for respective CNOT gates in the example circuit

$$\begin{pmatrix} q_0 & q_1 & q_2 & q_3 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \xrightarrow{q_3, q_1} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \xrightarrow{q_1, q_3} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \xrightarrow{q_1, q_0} (\dots) \xrightarrow{q_3, q_1} (\dots) \xrightarrow{q_1, q_0} (\dots) \xrightarrow{q_3, q_1} \begin{pmatrix} q_0 & q_1 & q_2 & q_3 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

CNOT sub-circuits appear frequently in quantum circuits, since CNOT is the only binary gate in many quantum platforms. Optimizing such sub-circuits directly impacts the overall error. Every  $n$ -qubit CNOT circuit can be represented by a so-called *parity* matrix i.e., a full-rank  $n \times n$  matrix in  $GL_n(F_2)$  [3, 23]. Adding column  $i$  to column  $j$  (modulo 2) in this matrix corresponds to applying a CNOT gate with control qubit  $i$  to target qubit  $j$ . So the minimal CNOT circuit corresponds to finding the shortest series of column additions to obtain the goal matrix from the Identity matrix. The parity matrix formulation for CNOT circuits is much more compact than the usual  $2^n \times 2^n$  complex unitary matrix for arbitrary quantum circuits.

**Definition 1.** Given a CNOT circuit  $C$  on  $n$  qubits, we define  $M_C$  to be its parity matrix in  $GL_n(F_2)$  generated by applying all CNOT gates in  $C$  to the  $n \times n$  Identity matrix.

The columns of the parity matrix are labeled with the qubits. Given a circuit, one can transform the Identity matrix to the final parity matrix by applying column additions corresponding to the CNOT gates. For example, Table 2 shows such a transformation via column additions of our example circuit. The right-most matrix in Table 2 shows the equivalent matrix for the CNOT circuit. In the parity matrix, each column represents the output of the corresponding qubit. For instance, column  $q_2$  has the bit sequence 0, 0, 1, 0 representing the untouched qubit  $q_2$ . On the other hand, column  $q_0$  with bit sequence 1, 1, 0, 0 represents  $q_0 \oplus q_1$ .

## 2.2 Classical planning

Given a description of a world, finding a sequence of actions that transform an initial state to some goal state is Automated Planning. In Classical Planning [14], the actions are deterministic and there exists a single initial state. Any reachability encoding can be elegantly encoded in such a specification. The problem is specified using Domain and Problem files in the Planning Domain Definition Language (PDDL) [13]. A domain file defines the predicates that describe the world and lists schematic actions that can change the world. A problem file specifies the objects used, the initial state, and the goal state. One can then use existing State-of-the-art domain-independent planners to solve problems. Layout Synthesis of Quantum circuits has been successfully encoded before in classical planning [31].

## 2.3 Propositional satisfiability

Given a boolean formula, finding an assignment that makes the formula true is a propositional satisfiability (SAT) problem. In recent years, many (NP-complete and NP-hard) problems have been successfully encoded and solved using SAT [5, 12]. Several synthesis-related problems in Quantum Computing have been encoded in SAT [29, 25, 21, 32]. Since we are interested in optimal solutions, SAT-based solving is a promising technique for proving optimality.

## 2.4 Quantified boolean formulas

Quantified Boolean Formula (QBF) Logic [4] is an extension of propositional logic with universal and existential quantifiers. One can encode a propositional formula in a more compact way taking advantage of inherent structure. When propositional formulas get too large to encode, encoding in QBF is an alternative. For instance, using QBF-based encodings helped in avoiding large Organic Synthesis encodings based on Planning in [30].

## 3 Optimal CNOT synthesis

In this section, we discuss CNOT optimization and its variations with synthesis. We will first establish different notions of equivalence between CNOT circuits. Then we discuss layout-aware synthesis, in the presence of connectivity restrictions. Finally, we discuss the relevant combinations of synthesis variants for the encodings in this paper. In section 8, we present related work and compare it with our approach.

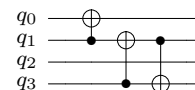
### 3.1 CNOT circuit equivalence

For optimal synthesis, we first need to establish equivalence between CNOT circuits. In general, two quantum circuits are equivalent if their unitary matrices are the same. Intuitively, equivalent circuits have the same input-output behavior. Note that equivalent circuits can have different gate counts and circuit depths. The general idea for optimization is to compute an equivalent circuit with either a lower gate count or circuit depth.

**Strong equivalence (S)** Every parity matrix has a corresponding unitary matrix. For CNOT circuits, we can directly use the parity matrix representation for this equivalence relation. If two CNOT circuits have the same parity matrix, then they are strongly equivalent [3, 23].

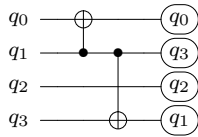
**Definition 2.** Two CNOT circuits  $C, C'$  are strongly equivalent if and only if  $M_C = M_{C'}$ .

For example, consider the CNOT circuit in Table 1 with the final matrix in Table 2. Synthesizing optimal column operations to reach  $M_C$  is equivalent to synthesizing an optimal circuit  $C'$ . One can synthesize the same parity matrix by using only three column additions on  $(q_1, q_0)$ ,  $(q_3, q_1)$ , and  $(q_1, q_3)$ . Thus, the resulting equivalent circuit as in Table 3 only has 3 CNOT gates (optimal) as instead of 6.

**Table 3.** Optimized circuit via S, with 3 CNOT gates only

**Weak equivalence (W)** While one can use strong equivalence for optimal synthesis, the definition is somewhat restrictive. The order of input qubits and output qubits is the same in the optimized circuit in Table 3. However, one could allow permutations of the output qubits within a circuit, as long as one keeps track of the final “physical” position of the “logical” output qubits. For example, Table 4 shows an equivalent circuit where the order of output qubits is changed.

**Table 4.** Optimized circuit using W (weak equivalence), with 2 CNOTs. The final position of permuted output qubits are represented using circles.



In CNOT circuits, the output qubit permutation simply corresponds to the permutation of columns in the parity matrix.

**Definition 3.** Given a permutation  $P$  and a matrix  $M_C$ , we define  $P(M_C)$  be the column permuted matrix.

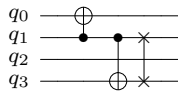
Intuitively, it might be possible to reach some permutation of the matrix in fewer steps. We now define the weak equivalence based on the permutation of matrices.

**Definition 4.** Two CNOT circuits  $C$  and  $C'$  are weakly equivalent if and only if there exists a permutation  $P$  such that  $P(M_C) = M_{C'}$ .

In our example, by using weak equivalence only 2 column operations are required to reach a permuted matrix. Table 4 shows such an optimal circuit with only 2 CNOT gates. We use circles to denote the permuted positions of output qubits.

To convert a weakly equivalent circuit to a strongly equivalent circuit, one can add tailing swaps to model the permutation of output qubits. When there are no connectivity restrictions on CNOT gates, these swap gates have *zero-cost*. One can remove swaps by simply relabelling gates in constant time. In our example, using swaps results in a strongly equivalent optimized circuit as shown in Table 5.

**Table 5.** Optimized circuit using W with 2 CNOTs and one SWAP



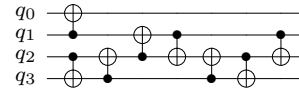
### 3.2 Restricted CNOT connections (R)

We also explore layout-aware CNOT optimization. In some quantum platforms, not all qubit pairs are connected thus restricting 2-qubit gate execution. One can only apply CNOT gates on adjacent pairs of qubits based on some coupling graph. Usually, Circuit Synthesis and Layout Synthesis are separated, leading to suboptimal results. For instance, most Layout Synthesis techniques do not take CNOT gate cancellation opportunities into account when transpiling.

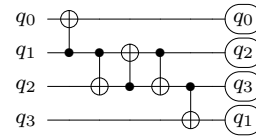
We integrate these phases, by adapting our CNOT synthesis to respect connectivity constraints. Given a restricted set of CNOT connections, we only allow column additions that correspond to adjacent qubits when synthesizing the final matrix (or a permutation of it). We now obtain the minimal CNOT circuit that satisfies the restrictions.

Suppose, we want to optimize our example circuit in Table 1, allowing CNOT gates only on qubit pairs (0, 1), (1, 2), and (2, 3). If we insist on strong equivalence, we need 8 CNOT gates (optimal) as shown in Table 6. Note that we need more than the original 6 CNOT gates, due to connectivity restrictions. Allowing weak equivalence requires only 5 CNOT gates (optimal) as shown in Table 7.

**Table 6.** Optimized circuit with S+R, using 8 CNOTs



**Table 7.** Optimized circuit with W+R, using 5 CNOTs. The final placement of permuted output qubits are represented using circles.



### 3.3 Metrics and relevant synthesis variants

We consider three metrics on quantum circuits in this paper, CNOT-gate count, depth, and CNOT depth. The *CNOT count* is simply the number of CNOT gates in a circuit. The *depth* of a circuit is the length of the longest path in the dependency graph of its gates, connected by direct input-output dependencies. The *CNOT depth* of a circuit is the largest number of CNOT gates on any dependency chain. For instance, the circuit in Table 6 has CNOT count 8, but its CNOT depth is only 7 (since the first two CNOT gates are applied in parallel). For CNOT circuits, depth and CNOT depth always coincide.

For CNOT and depth optimization, optimal synthesis is a computationally hard problem. Variants with CNOT restrictions have been proven to be NP-hard for both gate count and depth optimization [19] metrics. In fact, for synthesis with CNOT restrictions, even finding approximate solutions is NP-Hard [18]. We encode such hard problems in Classical Planning, SAT, and QBF. In particular, we are interested in 4 synthesis variants:

- S: Synthesis with strong equivalence (and no restrictions).
- W: Synthesis with weak equivalence (and no restrictions).
- S+R: Synthesis with strong equivalence and CNOT restrictions.
- W+R: Synthesis with weak equivalence and CNOT restrictions.

Not all variants can be encoded efficiently in every solving technique. For instance, we found an efficient encoding of the W variant in SAT, but it seems more difficult in classical planning and QBF. So we encode selected variants for each technique:

- Classical Planning: only S and S+R with CNOT gate optimization.
- SAT: All 4 variations for both CNOT count and CNOT depth.
- QBF: S and S+R for both CNOT count and depth optimization.

## 4 CNOT synthesis as planning

In this section, we first describe the encoding for the S+R variant in Classical Planning in PDDL. We encode the synthesis as a reachability problem, where nodes of a graph represent the state of the matrix

and edges represent column additions. Given a circuit, we first compute its parity matrix which corresponds to the goal node. The shortest path from the initial node with the Identity matrix to the goal node corresponds to the optimal number of CNOT gates.

All our objects, which label rows and columns, are of type `qubit`. We use the following two predicates to represent the state:

- `(m ?r ?c - qubit)`: represents a matrix element with `?r` row and `?c` column parameters.
- `(connected ?a ?b - qubit)`: static predicate to represent connected qubit parameters `?a` and `?b`.

To apply a CNOT gate on two qubits, we encode the corresponding column addition as an action. In preconditions, we specify that a CNOT gate can be applied only on different qubits and the qubits must be connected. In effects, for each row, the element in the target column is flipped if the control column element is true. We use conditional effects in PDDL to encode the effects. Listing 1 is the corresponding domain file with all predicates and actions in PDDL.

**Listing 1.** Domain for S+R CNOT synthesis in PDDL Format

```
(:predicates
(m ?r ?c - qubit)(connected ?a ?b - qubit))
(:action cnot
:parameters (?c ?t - qubit)
:precondition (and
(not(= ?c ?t))(connected ?c ?t))
:effect (and
(forall(?r - qubit)
(when (and (m ?r ?c) (m ?r ?t))
(not(m ?r ?t))))
(forall(?r - qubit)
(when (and (m ?r ?c)(not(m ?r ?t))
(m ?r ?t))))))
```

For any CNOT synthesis instance, one can use the same Domain file. The Problem file, on the other hand, defines instance-specific information i.e., objects, initial and goal states. Listing 2 shows snippets of the problem file for our example (see Table 1). For CNOT synthesis, we define one object per qubit. In the initial state, we encode the Identity matrix i.e., only diagonal elements are set to true. We specify which qubits are connected based on an input coupling graph. In the initial state specification, one only specifies the true propositions, and unspecified propositions are negated by default. For the goal state, we encode the final matrix for the given circuit.

**Listing 2.** Problem snippets in PDDL for the example circuit

```
(:objects q0 q1 q2 q3 - qubit)
(:init
(m q0 q0)(m q1 q1)(m q2 q2)(m q3 q3)
(connected q0 q1)(connected q1 q0)
(connected q1 q2)(connected q2 q1)
(connected q2 q3)(connected q3 q2))
(:goal (and
(m q0 q0) ... (not(m q0 q2))(not(m q0 q3))
(m q1 q0) ... (not(m q1 q2)) (m q1 q3)
(not(m q2 q0))... (m q2 q2) (not(m q2 q3))
(not(m q3 q0))... (not(m q3 q2))(not(m q3 q3))))
```

An optimal plan i.e., a plan with minimal actions, corresponds to an optimal circuit. We can then use any off-the-shelf optimal planners to synthesize optimal CNOT circuits. One could also use heuristic planners for fast synthesis in case of large instances. For S synthesis without restrictions, we simply drop the `connected` predicate from the domain and problem files.

## 5 CNOT synthesis as SAT

Encoding in classical planning is elegant and easy to understand. However, classical planners are good at finding fast heuristic plans but face scalability issues for computing optimal plans. Since our synthesis problem is encoded as a bounded reachability problem, a SAT encoding for optimal synthesis is promising.

### 5.1 Gate optimal encoding

For CNOT gate optimality, we apply a standard one-hot reachability encoding. First, we define variables for matrices which represent the state at each time step. We represent the matrix element in row  $r$  and column  $c$  at time step  $t$  as  $m_{r,c}^t$ . At each time step, we apply a single column addition on some control and target columns. We represent the control column as  $ctrl^t$  and the target column as  $trg^t$  at time step  $t$ . For a plan length of  $k$ , we define  $k$  copies of action variables and  $k + 1$  copies of state variables.

**S+R synthesis** The initial state corresponds to the Identity matrix. We encode it using Exactly-One (EO) constraints on row elements and unit clauses for diagonal elements.

$$\bigwedge_{r=0}^{n-1} \text{EO}(m_{r,0}^0, \dots, m_{r,n-1}^0) \wedge \bigwedge_{q=0}^{n-1} m_{q,q}^0 \quad (1)$$

For each transition step, exactly one control and target column is chosen (see Equation 2). Given a coupling graph with a set of connected qubit pairs  $CP$ , we only allow corresponding column pairs (see Equation 3). For state updates, we encode constraints:

- For every row, we update target column matrix variables based on control column matrix variables. If the control variable is:
  - true, then the target variable is flipped (see Equation 4)
  - false, then the target variable is propagated (see Equation 5)
- All untouched column variables are propagated (see Equation 6).

For time steps  $t \in \{0, \dots, k-1\}$ , we specify:

$$\text{EO}(ctrl_0^t, \dots, ctrl_{n-1}^t) \wedge \text{EO}(trg_0^t, \dots, trg_{n-1}^t) \quad (2)$$

$$\bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{n-1} (\{\neg ctrl_i^t \vee \neg trg_j^t \mid (i, j) \notin CP\}) \quad (3)$$

$$\bigwedge_{(i,j) \in CP} \bigwedge_{r=0}^{n-1} (ctrl_i^t \wedge trg_j^t \wedge m_{r,i}^t) \implies (m_{r,j}^t \neq m_{r,j}^{t+1}) \quad (4)$$

$$\bigwedge_{(i,j) \in CP} \bigwedge_{r=0}^{n-1} (ctrl_i^t \wedge trg_j^t \wedge \neg m_{r,i}^t) \implies (m_{r,j}^t = m_{r,j}^{t+1}) \quad (5)$$

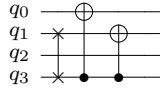
$$\bigwedge_{i=0}^{n-1} \bigwedge_{r=0}^{n-1} \neg trg_i^t \implies (m_{r,i}^t = m_{r,i}^{t+1}) \quad (6)$$

For a given circuit  $C$ , we encode the goal state with the corresponding final matrix  $M_C$ . For every 1 in the matrix, we add positive unit clauses in the goal state matrix and negative ones for every 0 (see Equation 7). In synthesis variant S, all different qubit pairs are connected.

$$\bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{n-1} \left( \bigwedge_{M_C[i,j]=1} m_{i,j}^k \wedge \bigwedge_{M_C[i,j]=0} \neg m_{i,j}^k \right) \quad (7)$$

**W synthesis** One can encode column permutation of goal matrix for weak equivalence. However, such a permutation would result in many clauses. Instead, we observe that every circuit with permuted output qubits has an equivalent circuit with permuted input qubits. An input-permuted circuit has swaps at the start instead of the end of the circuit. Removing initial swaps simply relabels CNOT gates and results in the permutation of output qubits. Note that such relabelling does not change the number of CNOTs in the circuit. For example, Table 8 shows an example circuit with initial swaps instead of the end as in Table 5. Identity matrix permutation can be encoded elegantly

**Table 8.** Optimized circuit with initial swaps via W, with 2 CNOTs



using exactly-one constraints on the time step 0 state variables. This can be achieved by dropping unit clauses in Equation 1 and adding exactly-one constraints on column variables. Essentially, we replace Equation 1 with Equation 8.

$$\bigwedge_{r=0}^{n-1} \text{EO}(m_{r,0}^0, \dots, m_{r,n-1}^0) \wedge \bigwedge_{c=0}^{n-1} \text{EO}(m_{0,c}^0, \dots, m_{n-1,c}^0) \quad (8)$$

**W+R synthesis** In the presence of CNOT restrictions, removing initial swaps can result in CNOT gates applied on restricted qubit pairs. To circumvent this problem, we encode symbolic qubit pair restrictions based on the initial permutation. Exactly-one constraints in the Initial matrix encodes the permutation. If  $m_{i,p}^0$  is true then it implies that qubit  $i$  is mapped to qubit  $p$ . We use such information to specify the restricted qubit pairs after the permutation. Essentially, if a restricted qubit pair  $(i, j)$  is mapped to  $(p, q)$  then  $(p, q)$  is restricted. We replace Equation 3 with Equation 9.

$$\bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{n-1} \bigwedge_{p=0}^{n-1} \bigwedge_{q=0}^{n-1} (\{m_{i,p}^0 \wedge m_{j,q}^0 \implies \neg \text{ctrl}_p^t \vee \neg \text{trg}_q^t \mid (i, j) \notin \text{CP}\}) \quad (9)$$

## 5.2 Depth optimal encoding with parallel plans

CNOT depth is another important metric in the optimization of quantum circuits. CNOTs acting on different qubits can be applied at the same depth. Depth-based synthesis can be encoded in SAT by allowing parallel CNOTs at each time step. The makespan of such an encoding corresponds to the depth of the synthesized circuit. We only discuss the S+R synthesis variant; the other 3 variants directly follow from the above gate-optimal encoding. Similar to the gate-optimal encoding, we define the same matrix variables to represent the state. Both the initial and goal constraints are exactly the same, i.e., Equations 1 and 7 stay the same. To allow parallel CNOT gates, we define one variable  $\text{cnot}_{i,j}^t$  for each qubit pair  $(i, j)$  at time step  $t$ . To respect CNOT restrictions, we disable the CNOT variables on restricted pairs (see Equation 10). We also use target variables  $\text{trg}_q^t$  as before for propagation of untouched column variables. So propagation constraints as in Equation 6 stay the same for depth optimal encoding, now Equation 16. Note that multiple target columns can be changed due to parallel CNOTs. We handle parallel CNOT operations by specifying (see Equations 11-15):

- Almost-One (AMO) CNOT gate is applied on a qubit.
- Atleast-One (ALO) CNOT gate is applied at each time step. Only for efficiency, dropping them would not affect correctness.
- Target column  $\text{trg}_j$  is set to true iff some CNOT on  $(i, j)$  is true.
- For every CNOT variable and every row, we update target column variables based on control column variables:
  - if the control variable is true, then the target variable is flipped
  - if the control variable is false, the target variable is propagated

For time steps  $t \in \{0, \dots, k-1\}$ , we specify:

$$\bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{n-1} (\{\neg \text{cnot}_{i,j}^t \mid (i, j) \notin \text{CP}\}) \quad (10)$$

$$\bigwedge_{q=0}^{n-1} \text{AMO}(\{\text{cnot}_{i,j}^t \mid (i = q \text{ or } j = q \text{ and } (i, j) \in \text{CP})\}) \quad (11)$$

$$\text{ALO}(\{\text{cnot}_{i,j}^t \mid (i, j) \in \text{CP}\}) \quad (12)$$

$$\bigwedge_{i=0}^{n-1} ((\bigvee_{j=0}^{n-1} \text{cnot}_{i,j}^t) = \text{trg}_i^t) \quad (13)$$

$$\bigwedge_{(i,j) \in \text{CP}} \bigwedge_{r=0}^{n-1} (\text{cnot}_{i,j}^t \wedge m_{r,i}^t) \implies (m_{r,j}^t \neq m_{r,j}^{t+1}) \quad (14)$$

$$\bigwedge_{(i,j) \in \text{CP}} \bigwedge_{r=0}^{n-1} (\text{cnot}_{i,j}^t \wedge \neg m_{r,i}^t) \implies (m_{r,j}^t = m_{r,j}^{t+1}) \quad (15)$$

$$\bigwedge_{i=0}^{n-1} \bigwedge_{r=0}^{n-1} \neg \text{trg}_i^t \implies (m_{r,i}^t = m_{r,i}^{t+1}) \quad (16)$$

## 6 CNOT synthesis as QBF

Even for the simplest synthesis variant S, the SAT encoding uses  $O(n^2)$  variables and  $O(n^3)$  clauses. For moderately large  $n$  the encoding sizes can get massive. In CNOT synthesis, the column updates are the same for every row. One can use universal quantification in QBF to capture this structure and generate a compact encoding. While QBF solvers are not as mature as SAT solvers, in some cases well-structured QBF encodings can help. In this section, we focus on the S+R synthesis variant for CNOT count optimization. The other variants (S for CNOT count and S, S+R for CNOT depth) follow directly. We drop W and W+R variants for QBF, as encoding column permutation symbolically is difficult.

The action variables, i.e., control and target variables are the same as in the SAT encoding. Instead of defining column matrix variables for each row, we define a symbolic row with universal variables. We use binary encoding for the universal variables, we define  $R$  as  $\{R_0, \dots, R_{\lceil \log(n) \rceil - 1}\}$ . For better propagation, we add one-hot encoding for symbolic row variables with existential variables  $r_0, \dots, r_{n-1}$ . The idea is to set the existential variables based on binary row variables. We can directly use existential row variables for state update constraints similar to our SAT encoding. We only need one set of column matrix variables to represent the complete matrix:  $c_i$  represents the  $i$ th column variable (for the symbolic row  $R$ ).

We define the prefix of our QBF encoding as follows:

$$\exists \text{ctrl}_0^0, \dots, \text{ctrl}_{n-1}^0 \exists \text{trg}_0^0, \dots, \text{trg}_{n-1}^0 \quad (17)$$

$$\dots \quad (18)$$

$$\exists \text{ctrl}_0^{k-1}, \dots, \text{ctrl}_{n-1}^{k-1} \exists \text{trg}_0^{k-1}, \dots, \text{trg}_{n-1}^{k-1} \quad (19)$$

$$\forall R \quad \exists r_0, \dots, r_{n-1} \quad (20)$$

$$\exists c_0^0, \dots, c_{n-1}^0 \dots \exists c_0^k, \dots, c_{n-1}^k \quad (21)$$

First, we imply existential row variables from binary-encoded symbolic row variables. Exactly one existential row variable is true.

$$\bigwedge_{i=0}^{n-1} (\text{bin}(R, i) \implies r_i) \wedge \text{EO}(r_0, \dots, r_{n-1}) \quad (22)$$

For the initial state, we encode the identity matrix where only diagonal matrix variables are true.

$$\bigwedge_{i=0}^{n-1} r_i = c_i^0 \quad (23)$$

For the goal state, we encode the final matrix  $M_C$  again using existential row variables.

$$\bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{n-1} \bigwedge_{M_C[i,j]=1} r_i \implies c_j^k \wedge \bigwedge_{M_C[i,j]=0} r_i \implies \neg c_j^k \quad (24)$$

Transition constraints are similar to those in our SAT encoding (see Equations 2 to 6), but here we simply drop the row indices from the SAT encoding. For time steps  $t \in \{0, \dots, k-1\}$ , we specify:

$$\text{EO}(\text{ctrl}_0^t, \dots, \text{ctrl}_{n-1}^t) \wedge \text{EO}(\text{trg}_0^t, \dots, \text{trg}_{n-1}^t) \quad (25)$$

$$\bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{n-1} (\{\neg \text{ctrl}_i^t \wedge \neg \text{trg}_j^t \mid (i, j) \notin \text{CP}\}) \quad (26)$$

$$\bigwedge_{(i,j) \in \text{CP}} (\text{ctrl}_i^t \wedge \text{trg}_j^t \wedge c_i^t) \implies (c_j^t \neq c_j^{t+1}) \quad (27)$$

$$\bigwedge_{(i,j) \in \text{CP}} (\text{ctrl}_i^t \wedge \text{trg}_j^t \wedge \neg c_i^t) \implies (c_j^t = c_j^{t+1}) \quad (28)$$

$$\bigwedge_{i=0}^{n-1} \neg \text{trg}_i^t \implies (c_i^t = c_i^{t+1}) \quad (29)$$

## 7 Implementation and evaluation

We are mainly interested in evaluating the following aspects:

- The quality improvement due to qubit permutation (W vs S).
- The overhead of imposing connectivity restrictions (S+R, W+R).
- The performance of the Planning, SAT, and QBF techniques.

**Peephole optimization** To allow CNOT optimization in arbitrary circuits, we employ *Peephole optimization* using a standard *slice-and-replace* approach. Given a quantum circuit in QASM format, we extract the CNOT slices from the circuit's dependency DAG as follows: We start from the top, such that each slice has one maximal CNOT sub-circuit followed by an arbitrary number of non-CNOT gates. For each CNOT sub-circuit, we optimize its gate count or depth. Finally, we replace each CNOT sub-circuit with its optimal counterpart.

Once the slicing is fixed, the order in which slices are treated does not matter for S, S+R, and W variants. Furthermore, the optimal

number of CNOTs is fixed for a given slicing. The W+R encoding cannot be used directly in our peephole optimization. Since a permutation in one slice can break CNOT connections in subsequent slices, the order of slice optimization matters. While solving slice-by-slice from top to bottom gives correct results, the final CNOT count might be sub-optimal, even for the given slicing. Hence, we do not apply peephole optimization with the W+R variant in this paper.

### 7.1 Experimental setup

Our tool Q-Synth v1<sup>4</sup> and v2<sup>5</sup> solve layout synthesis using classical planning and SAT solving, respectively. We extended Q-Synth to solve CNOT synthesis with Planning, SAT, and QBF. We provide an open-source tool Q-Synth v3<sup>6</sup> that implements all encoding variants discussed, including peephole optimization. For experimental evaluation, we consider standard T-gate optimized benchmarks [8] generated by T-par. We consider all benchmarks with up to 14 qubit circuits and at most 200 CNOT gates resulting in 11 instances. We propose two experiments to address our research questions.

**Experiment 1** We optimize CNOT count and depth on the benchmarks with S variant encodings of classical planning (CP), SAT, and QBF. To investigate the impact of qubit permutation, we compare S encodings with W encodings in SAT. Further, we compare our results with the state-of-the-art heuristic CNOT optimization tool DaCSynth (DS) in both gate [8] and depth [9] optimization. DaCSynth applies the same slice-and-replace approach with greedy heuristic algorithms for CNOT optimization. We use the results from Table 2 in [8] and Table 3 in [9] on our benchmarks for a fair comparison. Since DaCSynth is not an open-source tool, we can only compare the reported CNOT count and depth but not time and memory costs.

**Experiment 2** To investigate the overhead of connectivity restrictions, we take the W-optimized circuits from Experiment 1. We optimally map the circuits with Q-Synth v2 [32] onto the 14-qubit platform IBM Melbourne. Q-Synth v2 maps the circuits by inserting the optimal number of swaps. We then apply S+R optimization to the result. Since the input circuits are already optimized with W, any reduction in CNOT count or CNOT depth with S+R is significant.

**Tools and resources** For CP, we use the state-of-the-art optimal planner FastDownward [16] with merge-and-shrink (fd-ms) heuristic. Among the optimal planners that handle conditional effects, fd-ms performed the best in our preliminary experiments. In the case of SAT-based solving we use Cadical-1.53 [6] as SAT solver, and CAQE [28] with Blokker preprocessor [17] as QBF solver. In both the above experiments, for each slice in the peephole optimization, we give 600 seconds time and 8 GB memory limits. If a timeout occurs we leave the unoptimized slice untouched. All computations for the experiments are run on a cluster.<sup>7</sup>

**Metrics for comparison** We report and compare techniques on three metrics, CNOT count, depth, and CNOT depth. Two-qubit gates are more error-prone than 1-qubit gates, so tools like TKET [33] and T-par [8] mainly focus on CNOT depth. For Experiment 1, we compare with circuit depth as only circuit depths are reported in DaCSynth paper [9]. For Experiment 2, we report and compare the CNOT depth for our different techniques.

<sup>4</sup> <https://github.com/irfansha/Q-Synth/releases/tag/Q-Synth-v1.0-ICCAD23>

<sup>5</sup> <https://github.com/irfansha/Q-Synth/releases/tag/Q-Synth-v2.0-SAT2024>

<sup>6</sup> <https://github.com/irfansha/Q-Synth/releases/tag/Q-Synth-v3.0-ECAI24>

<sup>7</sup> <http://www.csc.aau.dk/grendel>, Huawei FusionServer Pro V1288H V5, with 384 GB main memory, using one 3.0 GHz Intel Xeon Gold 6248R core.

**Table 9.** Experiment 1: S vs W variants peephole synthesis on T-gate optimal circuits.

Circuit (#CNOTs/Depth)	#n	CNOT Optimization						Depth Optimization			
		CNOT count				Depth		Depth			
		CP(S)	SAT(S)	QBF(S)	SAT(W)	DS [8]	SAT(W)	SAT(S)	QBF(S)	SAT(W)	DS [9]
barencotof3 (52/60)	5	41	41	41	26	26	35	46	45	35	35
barencotof4 (96/96)	7	87	87	87	<b>48</b>	50	<b>60</b>	85	84	73	61
barencotof5 (134/123)	9	118	118	118	<b>71</b>	73	87	118	114	108	87
mod54 (48/57)	5	42	42	42	32	32	41	48	49	41	<b>40</b>
modmult55 (106/75)	9	82	82	82	<b>71</b>	73	50	54	56	53	50
qft4 (96/185)	5	84	84	84	57	<b>56</b>	<b>147</b>	172	172	164	149
rcadder6 (165/157)	14	141	141	141	<b>94</b>	100	95	129	129	95	95
tof3 (35/46)	5	30	30	30	<b>19</b>	21	<b>29</b>	41	41	32	31
tof4 (63/71)	7	55	55	55	37	37	45	59	57	56	<b>43</b>
tof5 (97/104)	9	81	81	81	50	50	<b>62</b>	78	80	86	63
vbadder3 (120/88)	10	86	86	86	<b>53</b>	61	48	66	68	49	<b>45</b>
Mean Reduction(%)		16.3	16.3	16.3	<b>44.9</b>	43.8	<b>34.2</b>	15.6	15.7	22.1	<b>34.2</b>
Max Reduction(%)		28.3	28.3	28.3	<b>55.8</b>	50.0	45.5	28.0	25.3	44.3	<b>48.9</b>

## 7.2 Results and discussion

**Experiment 1** Table 9 shows the data on Experiment 1. Under “CNOT optimization”, we report the CNOT count for Planning, SAT, and QBF. Here all three techniques with S synthesis performed similarly. While the CNOT reduction is the same, classical planning had 4 timeout slices whereas SAT and QBF based optimization had 3 timeout slices. Comparing S and W, we observe that SAT with W synthesis results in significantly more reduction (up to 55.8%). Since the T-par tool adds additional CNOT gates to route T gates for optimization [2], permuting qubits can avoid such extra CNOTs. SAT encoding with W optimally solved all slices, thus the reported results are optimal for the given circuit slicing.

In comparison with DaCSynth (column DS), SAT with W synthesis performs well and guarantees the optimal CNOT count. We indeed report better CNOT count compared to DaCSynth. Surprisingly, we observed one instance (`qft4`) where DaCSynth reports a lower CNOT count. Either the slicing in DaCSynth is different or their reported count is a mistake.

Under “Depth Optimization”, all CNOT slices are replaced by depth-optimal slices in all our variants. But note that, even though CNOT slices have optimal depth locally, the global circuit depth need not be optimal for a given slicing. Surprisingly, we observed that CNOT count optimization results in overall better depth (right-most column under “CNOT Optimization”). The SAT(W) variant with CNOT optimization results in a mean depth reduction of 34.2% (only 21.1% with Depth optimization). We observed that local depth optimization adds extra parallel CNOTs, thus resulting in a higher global depth. The mean depth reduction achieved by Q-Synth and DaCSynth is the same (34.2%).

**Experiment 2** Table 10 reports the results of Experiment 2. With S+R synthesis, we observe CNOT count reduction (up to 17.1%) in 9 out of 11 already optimally mapped circuits. Only classical planning reported 1 timeout slice. For SAT and QBF, the CNOT reduction we report is optimal for the given slicing.

In the case of depth optimization, we observe CNOT depth reduction (up to 11.9%) in 4 out of 11 instances. In Experiment 1, we observed that CNOT optimization results in better global CNOT depth reduction. Similarly, as reported in Table 10, all three techniques report better CNOT depth reduction with local CNOT optimization.

**SAT vs QBF efficiency** Tables 11 and 12 show the time and memory taken by all our encodings. For CNOT optimization with S (Table 11, Experiment 1), we observe that SAT and QBF techniques per-

form similarly in terms of time and memory. In most cases, Cadical (SAT) is slightly faster and takes less memory than CAQE (QBF). Interestingly, on a large slice from the 14-qubit circuit, CAQE is slightly faster than Cadical. This can happen because the QBF encoding is only linear in variables and quadratic in constraints, while our SAT encoding is quadratic in variables and cubic in constraints. So the QBF encoding is promising for instances with many qubits.

In case of S+R synthesis (Table 11, Experiment 2), adding CNOT restrictions seems to boost the performance of SAT encoding compared to QBF. Note that the coupling graph is typically planar, with a low out-degree, so the SAT encoding with restrictions becomes quadratic instead of cubic. For depth optimization, both SAT and QBF techniques take only a few seconds for each slice (see Table 12). Since the optimal depths of slices are small, the memory footprint is negligible (close to zero, not shown here). Only QBF takes around 125 MB memory for the 14-qubit instance `rcadder6`.

Both W and S+R variants are practical for optimization: most instances are solved within a minute in our benchmark set. While we cannot apply peephole optimization with W+R, we optimized the individual slices from experiment 2 with W+R. The W+R encoding performs well. It optimally solves all slices and never takes more than a minute for any slice.

**CP vs SAT and QBF** In general, fd-ms (CP) is slower than using SAT and QBF solvers (Table 11). Overall it results in 5 timeout slices compared to 3 for SAT and QBF. We also noticed that fd-ms uses more memory (up to 6 GB) compared to the other two.

Note that CP-based solving techniques are orthogonal to SAT and QBF. For instance, CP results in maximum CNOT depth reduction in Experiment 2 with the instance `qft4`. Another advantage of the CP approach is being able to use fast heuristic planners. Just using any heuristic planner results in heuristic CNOT optimization. For large circuits (with hundreds of qubits), such an approach is more feasible than the SAT and QBF based approaches.

## 8 Related work

CNOT synthesis has been studied before, also in the context of qubit permutation and CNOT restrictions. In this section, we discuss some CNOT synthesis approaches that are close to our approach.

**S and S+R synthesis** Several techniques are applied for CNOT synthesis such as Gaussian elimination [3], Steiner trees [20], rewrite rules [18], and asymptotically optimal algorithms [24, 20]. Optimal

**Table 10.** Experiment 2: S+R variant peephole synthesis for optimally mapped circuits on to 14-qubit Melbourne platform.

Circuit (#CNOTs/CNOT depth)	CNOT Optimization						Depth Optimization	
	CNOT count			CNOT depth			CNOT depth	
	CP	SAT	QBF	CP	SAT	QBF	SAT	QBF
barecotof3 (44/41)	39	39	39	35	35	35	41	41
barecotof4 (78/70)	74	74	74	66	66	66	70	70
barecotof5 (110/95)	108	108	108	93	93	93	95	95
mod54 (56/49)	48	48	48	43	<b>40</b>	<b>40</b>	45	45
modmult55 (131/79)	117	<b>115</b>	<b>115</b>	<b>69</b>	71	71	76	78
qft4 (105/101)	87	87	87	<b>82</b>	83	83	89	89
rcadder6 (145/102)	137	137	137	99	98	<b>97</b>	102	102
tof3 (34/33)	34	34	34	33	33	33	33	33
tof4 (61/55)	61	61	61	55	55	55	55	55
tof5 (80/71)	77	77	77	67	<b>66</b>	67	71	71
vbeadder3 (86/75)	79	79	79	67	67	67	74	74
Mean reduction(%)	7.4	<b>7.6</b>	<b>7.6</b>	8.0	<b>8.3</b>	<b>8.3</b>	2.6	2.3
Max reduction(%)	17.1	17.1	17.1	<b>18.8</b>	18.4	18.4	11.9	11.9

**Table 11.** Time t (in seconds) and Memory m (in MB; – means negligible) taken with CNOT optimization.

Circuit	Experiment 1 (S vs W)						Experiment 2 (S+R)							
	CP(S)		SAT(S)		QBF(S)		SAT(W)		CP		SAT		QBF	
	t	m	t	m	t	m	t	m	t	m	t	m	t	m
barecotof3	7	–	5	–	7	–	5	–	343	247	9	–	14	–
barecotof4	608	6090	303	131	307	159	6	–	588	247	10	–	19	–
barecotof5	41	299	8	–	22	–	6	–	759	241	11	–	23	–
mod54	7	–	5	–	7	–	5	–	338	233	10	–	17	–
modmult55	11	–	23	–	51	91	11	–	956	2950	265	141	576	179
qft4	10	–	10	–	19	–	5	–	628	237	11	–	24	–
rcadder6	1588	2830	635	195	652	238	66	110	953	225	143	110	161	130
tof3	13	–	11	–	6	–	5	–	301	241	9	–	13	–
tof4	11	–	6	–	13	–	5	–	368	244	10	–	18	–
tof5	612	5840	609	187	611	204	6	–	524	227	10	–	21	–
vbeadder3	621	4390	607	186	619	217	7	–	434	247	10	–	23	–

**Table 12.** Time taken in seconds for Depth optimization.

Circuit	Experiment 1			Experiment 2	
	SAT(S)	QBF(S)	SAT(W)	SAT	QBF
barecotof3	5	6	5	9	10
barecotof4	7	15	5	10	12
barecotof5	10	26	6	10	13
mod54	5	6	5	9	12
modmult55	6	15	9	11	18
qft4	5	10	5	10	16
rcadder6	33	98	14	11	15
tof3	5	6	5	9	10
tof4	6	13	5	10	12
tof5	9	24	6	10	13
vbeadder3	10	32	6	10	14

CNOT synthesis is mainly considered in a broader context i.e., in the presence of either T gates or RZ gates. Here instead of synthesis on  $n \times n$  matrix, synthesis so-called phase polynomial is applied which also keeps track of phase rotation by T gates. Synthesis is applied in some polynomial representation using Steiner trees in [15], as SAT in [21], and as Answer Set Programming (ASP) in [27, 26]. Giving CNOT circuits without T or RZ gates as inputs for such encodings results in S and S+R variant encodings.

**W and W+R synthesis** In [7], authors proposed heuristic W and W+R variants based on the Syndrome Decoding Problem. The same authors proposed greedy algorithms for W in DaCSynth, which we compared with in this paper. Qubit permutations are applied in the

TKET compiler [34], but only without CNOT restrictions. In all variations, allowing qubit permutations results in further reduction in both CNOT count and depth. To our knowledge, W and W+R variants have not been handled optimally before.

**Beyond CNOT synthesis** SAT-based Synthesis of Clifford circuits with CNOT, H, and S gates has been proposed with both gate [29] and depth optimization [25] in the QMAP tool. Using CNOT circuits as input in the QMAP tool is similar to our S variant synthesis.

Instead of peephole optimization with circuit slicing, one can apply global CNOT synthesis using so-called holes as in [22]. CNOT synthesis is sometimes integrated with Layout Synthesis to achieve further reduction as in the heuristic approaches of [10, 11, 35].

## 9 Conclusion

In this paper, we considered optimal CNOT synthesis with two extensions, qubit permutation and layout restrictions. To our knowledge, we provide the first optimal CNOT synthesis variants with qubit permutation. We have encoded variations of optimal CNOT synthesis in Classical Planning, SAT, and QBF. We handled both CNOT count and CNOT depth metrics for optimization. By applying peephole optimization, we validated our techniques on standard T-gate optimal benchmarks. Our results show the effectiveness of qubit permutation on CNOT count and depth reduction. Finally, we showed further reduction in already optimally mapped benchmarks.

We leave integrated Layout + CNOT Synthesis, including optimal initial mapping, as a challenge for future work.



## Acknowledgements

This research was partially funded by the Innovation Fund Denmark. 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 ECAI 2024 for their useful suggestions.

## References

- [1] M. Amy, D. Maslov, M. Mosca, and M. Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 32(6):818–830, 2013. doi: 10.1109/TCAD.2013.2244643.
- [2] M. Amy, D. Maslov, and M. Mosca. Polynomial-time T-depth optimization of Clifford+T circuits via matroid partitioning. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 33(10):1476–1489, 2014. doi: 10.1109/TCAD.2014.2341953.
- [3] T. Beth and M. Rötteler. Quantum algorithms: applicable algebra and quantum physics. *Quantum information*, pages 96–150, 2001.
- [4] O. Beyersdorff, M. Janota, F. Lonsing, and M. Seidl. Quantified Boolean Formulas. In *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1177–1221. IOS Press, 2021. doi: 10.3233/FAIA201015.
- [5] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021. ISBN 978-1-64368-160-3. doi: 10.3233/FAIA336.
- [6] A. Biere et al. 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.
- [7] T. G. de Brugière, M. Baboulin, B. Valiron, S. Martiel, and C. Allouche. Quantum CNOT circuits synthesis for NISQ architectures using the syndrome decoding problem. In *Reversible Computation - 12th International Conference, RC 2020, Oslo, Norway, July 9-10, 2020, Proceedings*, volume 12227 of *Lecture Notes in Computer Science*, pages 189–205. Springer, 2020. doi: 10.1007/978-3-030-52482-1\_11.
- [8] T. G. De Brugière, M. Baboulin, B. Valiron, S. Martiel, and C. Allouche. Gaussian elimination versus greedy methods for the synthesis of linear reversible circuits. *ACM Transactions on Quantum Computing*, 2(3), sep 2021. doi: 10.1145/3474226.
- [9] T. G. De Brugière, M. Baboulin, B. Valiron, S. Martiel, and C. Allouche. Reducing the depth of linear reversible quantum circuits. *IEEE Transactions on Quantum Engineering*, 2:1–22, 2021.
- [10] A. M. de Griend and S. M. 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.
- [11] J. Ding and S. Yamashita. Exact synthesis of nearest neighbor compliant quantum circuits in 2-D architecture and its application to large-scale circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39:1045–1058, 2020. URL <https://api.semanticscholar.org/CorpusID:132241251>.
- [12] J. K. Fichte, D. L. Berre, M. Hecher, and S. Szeider. The silent (r)evolution of SAT. *Commun. ACM*, 66(6):64–72, 2023. doi: 10.1145/3560469.
- [13] M. Fox and D. Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, 20:61–124, 2003. doi: 10.1613/jair.1129.
- [14] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.
- [15] V. Gheorghiu, J. Huang, S. M. Li, M. Mosca, and P. Mukhopadhyay. Reducing the CNOT count for Clifford+T circuits on NISQ architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(6):1873–1884, 2023. doi: 10.1109/TCAD.2022.3213210.
- [16] M. Helmert, G. Röger, and E. Karpas. Fast downward stone soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, volume 2835, 2011.
- [17] M. Heule, M. Järvisalo, F. Lonsing, M. Seidl, and A. Biere. Clause elimination for SAT and QSAT. *Journal of Artificial Intelligence Research (JAIR)*, 53:127–168, 2015.
- [18] K. Iwama, Y. Kambayashi, and S. Yamashita. Transformation rules for designing CNOT-based quantum circuits. In *Proceedings of the 39th annual Design Automation Conference*, pages 419–424, 2002.
- [19] J. Jiang, X. Sun, S. Teng, B. Wu, K. Wu, and J. 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.
- [20] A. Kissinger and A. M. de Griend. CNOT circuit extraction for topologically-constrained quantum memories. *Quantum Inf. Comput.*, 20(7&8):581–596, 2020. doi: 10.26421/QIC20.7-8-4.
- [21] G. Meuli, M. Soeken, and G. D. Micheli. SAT-based {CNOT, T} quantum circuit synthesis. In *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings*, volume 11106 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2018. doi: 10.1007/978-3-319-99498-7\_12.
- [22] E. Murphy and A. Kissinger. Global synthesis of CNOT circuits with holes. In *Proceedings of the Twentieth International Conference on Quantum Physics and Logic, QPL 2023, Paris, France, 17-21st July 2023*, volume 384 of *EPTCS*, pages 75–88, 2023. doi: 10.4204/EPTCS.384.5.
- [23] H. Nagarajan, O. Lockwood, and C. 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. URL <https://api.semanticscholar.org/CorpusID:244488668>.
- [24] K. N. Patel, I. L. Markov, and J. P. Hayes. Optimal synthesis of linear reversible circuits. *Quantum Inf. Comput.*, 8(3):282–294, 2008. doi: 10.26421/QIC8.3-4-4.
- [25] T. Peham, N. Brandl, R. Kueng, R. Wille, and L. Burgholzer. Depth-optimal synthesis of Clifford circuits with SAT solvers. *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 01:802–813, 2023. URL <https://api.semanticscholar.org/CorpusID:258461565>.
- [26] C. Piazza and R. Romanello. Synthesis of CNOT minimal quantum circuits with topological constraints through ASP. In *Proceedings of the International Workshop on AI for Quantum and Quantum for AI (AIQxQIA 2023)*, volume 3586 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL <https://ceur-ws.org/Vol-3586/paper2.pdf>.
- [27] C. Piazza, R. Romanello, and R. Wille. An ASP approach for the synthesis of CNOT minimal quantum circuits. In *Proceedings of the 38th Italian Conference on Computational Logic, Udine, Italy, June 21-23, 2023*, volume 3428 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL <https://ceur-ws.org/Vol-3428/paper18.pdf>.
- [28] M. N. Rabe and L. Tentrup. CAQE: A certifying QBF solver. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 136–143, Sept. 2015.
- [29] S. Schneider, L. Burgholzer, and R. Wille. A SAT encoding for optimal Clifford circuit synthesis. *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 190–195, 2022. URL <https://api.semanticscholar.org/CorpusID:251800203>.
- [30] I. Shaik and J. van de Pol. Classical planning as QBF without grounding. In *ICAPS*, pages 329–337. AAAI Press, 2022.
- [31] I. Shaik and J. van de Pol. Optimal layout synthesis for quantum circuits as classical planning. In *ICCAD'23, California, USA, 2023*. IEEE/ACM.
- [32] I. Shaik and J. 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.
- [33] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan. tket): a retargetable compiler for nisq devices. *Quantum Science and Technology*, 6(1):014003, nov 2020. doi: 10.1088/2058-9565/ab8e92. URL <https://dx.doi.org/10.1088/2058-9565/ab8e92>.
- [34] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan. tket>: a retargetable compiler for nisq devices. *Quantum Science and Technology*, 6(1):014003, 2020.
- [35] B. Wu, X. He, S. Yang, L. Shou, G. Tian, J. Zhang, and X. Sun. Optimization of CNOT circuits on limited-connectivity architecture. *Physical Review Research*, 5(1):013065, 2023.