# Depth-Optimal Quantum Layout Synthesis as SAT

**Anna B. Jakobsen**
Department of Computer Science, Aarhus University, Denmark

**Anders B. Clausen**
Department of Computer Science, Aarhus University, Denmark

**Jaco van de Pol**
Department of Computer Science, Aarhus University, Denmark

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

## Abstract

Quantum circuits consist of gates applied to qubits. Current quantum hardware platforms impose connectivity restrictions on binary CX gates. Hence, Layout Synthesis is an important step to transpile quantum circuits before they can be executed. Since CX gates are noisy, it is important to reduce the CX count or CX depth of the mapped circuits.

We provide a new and efficient encoding of Quantum-circuit Layout Synthesis in SAT. Previous SAT encodings focused on gate count and CX-gate count. Our encoding instead guarantees that we find mapped circuits with minimal circuit depth or minimal CX-gate depth. We use incremental SAT solving and parallel plans for an efficient encoding. This results in speedups of more than 10-100x compared to OLSQ2, which guarantees depth-optimality. But minimizing depth still takes more time than minimizing gate count with Q-Synth.

We correlate the noise reduction achieved by simulating circuits after (CX)-count and (CX)-depth reduction. We find that minimizing for CX-count correlates better with reducing noise than minimizing for CX-depth. However, taking into account both CX-count and CX-depth provides the best noise reduction.

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

**Keywords and phrases** Quantum Layout Synthesis, Transpiling, Circuit Mapping, Incremental SAT, Parallel Plans

**Digital Object Identifier** 10.4230/LIPIcs.SAT.2025.16

**Related Version** *Extended Version*: `https://arxiv.org/abs/2506.06752` [8]

**Supplementary Material** *Software (Github)*: `https://github.com/anbclausen/quills`
archived at `swh:1:dir:ae986beceee3051ca8ec16feed3ddc0b2afd6aab`
*Software (Zenodo link)*: `https://doi.org/10.5281/zenodo.15606051` [9]

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

**Acknowledgements** This paper is based on the MSc-thesis of Anna B. Jakobsen and Anders B. Clausen. They contributed equally to this work. We thank Kostiantyn V. Milkevych for porting Quills to Q-Synth v4.

## 1 Introduction

With recent advances in quantum hardware, there is a surge of interest in scalable compilation techniques. In order to execute a quantum program, it must be compiled to a circuit with unary and binary gates, while handling hardware restrictions. The quantum compilation pipeline broadly consists of two steps: Circuit Synthesis and Layout Synthesis. In this paper, we focus on the Quantum Layout Synthesis (QLS) problem [13, 24, 15, 21] that transforms a "logical" quantum circuit consisting of native gates into an equivalent circuit that respects the connectivity of a physical quantum platform. Many quantum platforms do not support all-to-all connectivity of physical qubits, thus restricting the application of binary gates to only neighboring qubits. We assume that the only gates in the input circuit are unary gates and the binary CX (Conditional NOT) gate. The physical platform is specified by an (undirected) graph on physical qubits, the coupling map. The output is an initial mapping of the logical qubits onto the physical qubits, as well as a new, equivalent circuit, in which CX gates are only applied to physical qubits that are connected in the coupling map. To ensure this, so-called SWAP gates may be inserted to route interacting logical qubits towards connected physical qubits. During routing, the order of the gates must also be respected.

Note that the QLS problem may admit many possible solutions. One can easily schedule all gates by eagerly adding SWAP gates to bring distant logical qubits closer together. Although the primary objective of QLS is to handle connectivity restrictions, the secondary objective is to maximize hardware performance. In the current Noisy Intermediate Scale Quantum (NISQ) era, qubits are noisy, and every gate adds to the error. Reducing noise is crucial for practical quantum computing. While there does not exist a single metric that predicts actual hardware performance, there exist some simple metrics like gate-count that are indicative. Since binary CX gates can be up to 10 times more error-prone than unary gates, the number of CX gates (CX-count metric) is more relevant than just gate count [14]. Since SWAP gates are usually implemented by 3 CX gates each, solutions with too many SWAPs should be avoided. Several heuristic approaches have been proposed that reduce the additional SWAP gates. The most famous is SABRE [13] (available in IBM's Qiskit), which implements a very fast heuristic to find a correct layout mapping. The tool MQT QMAP provides a heuristic approach based on the A*-algorithm [27]. There exist a myriad of other heuristic approaches that focus on CX-count optimization, we refer to [23] for a detailed survey. While heuristic approaches are fast, their solutions are often far from optimal [15]. Unfortunately, the optimal QLS problem is NP-hard [23]. The problem remains hard when considering natural restrictions on the coupling graph, like planar graphs [5] and bipartite graphs [11], and it is even conjectured to be NP-hard for simple linear neighbor graphs [7]. Thus, unless P=NP, one needs exhaustive exact approaches for (near)-optimal solutions.

Several exact approaches based on SMT [24, 15], Classical Planning [20], and SAT [21, 26] have been proposed to optimize CX-count. Synthesizing a $k$-SWAP optimal circuit involves refuting the existence of a mapped circuit of at most $k-1$ SWAPs. Recent SAT approaches proposed encodings whose makespan corresponds to the number of inserted SWAP gates instead of the number of CX-gates. Since there are typically fewer SWAP gates than CX gates, such SAT encodings significantly outperform earlier approaches [21].

Despite the progress in CX-count optimization, CX-count alone is not sufficient to predict the hardware performance. Note that qubits become unstable over time due to so-called *decoherence*. It is essential to execute gates in the circuit before the qubits become unstable. To reduce execution time, multiple gates can be executed in parallel, provided they are independent, i.e., do not act on the same qubits. This makes circuit depth an important metric, since it captures this notion of parallel execution. Since CX-gates are up to 10x slower compared to unary gates, we focus on reducing the CX-depth of the circuit. This metric counts the longest chain of dependent CX-gates.

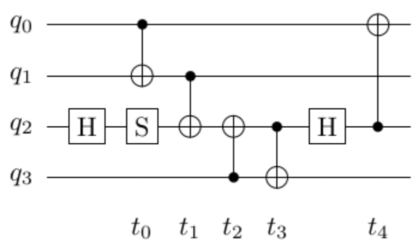
Optimization of circuit depth has been explored in [15] in the context of SMT solving. Their tool OLSQ2 is the state-of-the-art tool for depth-optimal synthesis. In practice, optimization of (CX)-depth is harder than (CX)-count. Unlike (CX)-count optimization, the makespan of an encoding optimizing (CX)-depth must be at least the (CX)-depth of the initial circuit. Since the (CX)-depth is typically higher than the number of SWAPs, depth optimal synthesis becomes much harder in practice. Compared to the SAT based state-of-the-art tool Q-Synth v2 for CX-count optimization, OLSQ2 is indeed up to 5 orders of magnitude slower [21, 15]. Even the best SMT-based near-optimal tool TB-OLSQ2 is up to 2 orders of magnitude slower compared to Q-Synth v2 [21]. SAT-based tools are shown to significantly outperform SMT-based tools for QLS [21, 26]. The natural next step is to investigate if an efficient SAT-based depth optimal encoding is feasible.

## Contributions

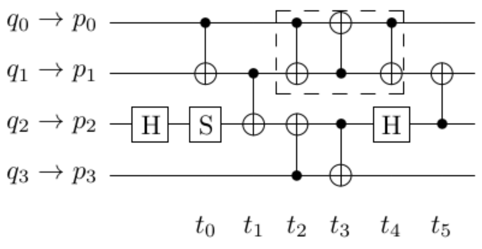
In this paper, we provide an encoding of (CX)-depth-optimal QLS in SAT for the first time. Additionally, we provide a tool, QuilLS[1], which implements the encoding. Our approach is based on increasing the time horizon, where in each time step one layer of independent gates is added. The first SAT solution found corresponds to a mapped circuit with optimal depth. We use incremental SAT solving (reusing partial information from previous calls) [16] and parallel plans to encode depth as makespan for efficient solving.

A second contribution is an extensive experiment, comparing various tools with respect to their running time and their performance in actual (CX)-count and (CX)-depth metrics. We compare our new tool with SABRE, Q-Synth and (TB)-OLSQ2. We run these tools on a standard set of quantum circuit benchmarks, obtained from [15, 21] and on the coupling graph of a number of standard quantum platforms (with 5-80 qubits). Our tool is often more than 10-100x faster compared to previous tools with guaranteed depth-optimality, but still slower than size-optimal tools.

Finally, we investigate how the various optimization metrics correlate with a reduction in noise, which is the ultimate goal of optimal QLS. Following the approach in [12, 25], we simulate all mapped circuits on a quantum platform with and without simulated noise, using the Qiskit simulator, and compute the Hellinger distance [3, 6, 19] of these runs. Surprisingly, we find that a reduction in CX-count correlates better with noise reduction than CX-depth. However, taking both measures, CX-count + CX-depth, into account leads to the best noise reduction, motivating the importance of our contribution.



**(a)** An Example circuit.



**(b)** CX-depth optimal circuit with 1 extra SWAP.

**Figure 1** Example circuit, before and after mapping on a linear neighbour graph. The dotted box represents the extra SWAP gate implemented as 3 CX gates. Time steps $t_0$ to $t_5$ indicate the CX gate parallel steps.

[1] Available at `https://github.com/anbclausen/quills`, also ported to Q-Synth v4 at `https://github.com/irfansha/Q-Synth`

## 2 Preliminaries

### 2.1 Layout Synthesis of an Example Circuit

As discussed previously in the introduction, the Quantum Layout Synthesis (QLS) problem takes as inputs a quantum circuit and a platform coupling map. For example, consider the 4-qubit circuit in Figure 1a with 5 CX-gates and 3 unary gates. For simplicity, consider a quantum platform of 4 qubits $\{p_0, p_1, p_2, p_3\}$ with linear connections i.e., only $(p_0, p_1), (p_1, p_2), (p_2, p_3)$ are connected. Notice that CX gates at $t_0, t_1, t_4$ form a triangle in Figure 1a, however our linear platform does not have a triangle. Thus, there is no mapping from logical to physical qubits such that all the CX gates can be scheduled. In such cases, SWAP gates can be inserted into the circuit, which allow physical qubits to swap their states. A SWAP gate can be constructed of three CX gates [17]. Indeed, as shown in Figure 1b, by using one extra SWAP gate we can schedule all gates respecting connectivity restrictions.

Layout synthesis can be optimized using various metrics. One is size optimality, where the number of gates in the output circuit is minimized. In practice, this means minimizing the number of SWAP gates. Another is minimizing the depth of the output circuit, which is defined as the longest path in the dependency graph of the gates in the circuit. A similar metric is the CX-depth, where only the dependencies between the CX gates are considered. Our original circuit has CX-depth 5 where as the mapped circuit has optimal CX-depth of 6. In Figure 1, we indicate the parallel steps with CX-gates before and after mapping. The number of parallel steps with CX-gates essentially corresponds to CX-depth. Optimization of CX-count differs from CX-depth largely due to the interleaving of SWAP gates with other CX gates. The choice of optimization goal can have a large impact. For example, one of the benchmark circuits used in our experimental evaluation (‘`tof_5`’), has a depth-optimal solution on the platform Guadalupe of depth 64 using 31 SWAP gates, while a size-optimal circuit uses only 5 SWAP gates, but has depth 71.

### 2.2 Layout Synthesis as SAT

Given a propositional formula, the SAT problem is to find an assignment to the variables such that the formula evaluates to true. While the general problem is NP-complete, in practice several instances can be solved efficiently by modern SAT solvers [2]. This motivates encoding other NP-complete problems to SAT. However, finding an efficient problem encoding is highly non-trivial. As discussed earlier, recent SAT based approaches [21, 26] perform well on CX-count metric. Unlike (CX)-depth optimization, a SAT encoding optimizing (CX)-count only requires a makespan of 2 on the example in Figure 2. The key idea is to group CNOTs at each parallel step and add a single SWAP gate from time step $t_1$. Thus, the makespan scales with the number of SWAPs which is typically much lower than the (CX)-count or (CX)-depth.

While the disadvantage in the larger required makespan is unavoidable, we can still apply parallel plans [10] to schedule parallel gates in a mapped circuit. It is possible to encode depth in a sequential encoding, however allowing independent gates in the same time step can reduce the makespan. The makespan refers to the circuit (CX)-depth of the solution. Reducing makespan indeed avoids many variable copies, thus impacting performance. We adapt another main technique, incremental SAT solving, used in the earlier SAT encodings for our (CX)-depth optimal encoding. Remember that an optimal synthesis of $k$ (CX)-depth circuit requires refuting the existence of $k-1$ or less (CX)-depth circuits. Instead of checking for circuits with (CX)-depth from 0 to $k-1$ independently, we reuse information from earlier SAT calls. By using incremental SAT solving, we can maintain a single CNF formula and

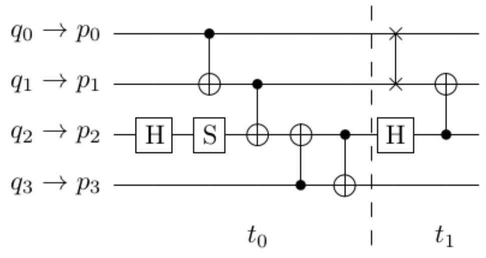


**Figure 2** Mapped circuit with only 2 parallel steps in the context of CX-count optimization.

"turn clauses on or off" instead of rebuilding the entire CNF formula every time. This also increases the performance during *inprocessing* (the actual work of the SAT solver). Finally, for a (CX)-depth optimal encoding, we need to handle the execution of SWAP gates over 3 time steps instead of just 1.

## 3 Depth-optimal SAT encoding

In the previous sections, we observed that SAT-based encodings in Q-Synth perform very well in the case of size-optimal synthesis. In this section, we formulate a similar SAT encoding of depth-optimal layout synthesis. In contrast to size-optimal synthesis, depth-optimal synthesis might produce solutions with a sub-optimal number of CX gates to find a solution with optimal circuit depth. Circuit depth refers to the longest path in the gate dependency graph. We employ two techniques, parallel plans, and incremental SAT solving for an efficient encoding. Unlike Q-Synth, a parallel step in our encoding encodes a single layer in a circuit, consisting of parallel gates applied to different qubits. Thus, the makespan of the encoding corresponds to the depth of the mapped circuit. The novelty of our encoding comes from the handling of SWAP gates while maintaining depth-optimality, which is non-trivial.

The overall structure of the encoding can be seen in Algorithm 1. Since the output cannot have smaller depth than the input, we only test the existence of a mapping when the makespan is at least as large as the input depth. The first time step where the instance is satisfied is the optimal depth. Much of the structure of the encoding follows that of [21]. We will focus the in-depth explanation to those parts that are different.

**Algorithm 1** Building and solving the SAT encoding.

```
Input: Coupling map $\mathcal{P}$ and circuit $\mathcal{C}$
  let $D$ be the depth of $\mathcal{C}$
  let instance be an empty CNF
  for $t = 1, 2, \ldots$ do
      Add $MappingConstraints(t)$ to instance
      Add $ConnectivityConstraints(t)$ to instance
      Add $GateConstraints(t)$ to instance
      Add $SwapConstraints(t)$ to instance
      if $t \geq D$ then
          Add $Assumptions(t)$ to instance
          Solve instance with assumption $asm^t$
          if instance is satisfied then
              return satisfying assignment
```

**Table 1** Encoding variables and descriptions.

| Variable | Description |
|---|---|
| $\mathrm{mp}^t_{q,p}$ | logical qubit $q$ is mapped to the physical qubit $p$ at time $t$. |
| $\mathrm{oc}^t_p$ | physical qubit $p$ is occupied by some logical qubit at time $t$. |
| $\mathrm{e}^t_{q,q'}$ | logical qubits $q, q'$ are mapped to connected qubits at time $t$. |
| $\mathrm{u}^t_p$ | physical qubit $p$ is not part of a SWAP gate at time $t$. |
| $\mathrm{c}^t_g$ | gate $g$ is currently applied at time $t$. |
| $\mathrm{a}^t_g$ | gate $g$ was applied at a time strictly before $t$. |
| $\mathrm{d}^t_g$ | gate $g$ is delayed and has not been applied yet at time $t$. |
| $\mathrm{sw}^t_{p,p'}$ | physical qubits $p, p'$ are swapped at times $t$, $t-1$ and $t-2$. |
| $Sw(p)^t$ | the set of variables $\mathrm{sw}^t_{p,p'}$ where $(p,p') \in P_{conn}$ |
| $\mathrm{st}^t_p$ | physical qubit $p$ is touched by a SWAP at $t$, $t-1$ and $t-2$. |
| $\mathrm{asm}^t$ | assumption to indicate that $t$ is the final time step. |

Table 1 presents the variables used in the encoding. For each time step, a set of variables is generated for the constraints of that time step. The $\mathrm{asm}^t$ variable is special and is used for incremental SAT solving; it provides context to the SAT solver about previous attempts to find a solution, essentially reducing the search space of the solver. SWAP gates are scheduled over 3 time steps since they are commonly implemented as three consecutive CX gates. In the following, we describe each part of the constraints added for each time step.
The Mapping and Connectivity constraints are exactly as in [21], but are repeated here for the sake of completeness. We use $P$ for the set of physical qubits and $Q$ for the set of logical qubits. $P_{conn}$ indicates the edges between physical qubits in the connectivity graph. We use $g.q$ to indicate the logical qubit that a unary gate must be applied on, and $g.q_0, g.q_1$ for the qubits that a CX gate must be applied on. $G_{CX}$ denotes the set of CX gates in the input circuit while $G_{unary}$ denotes the set of unary gates.

*MappingConstraints*

$$\bigwedge_{q\in Q} \mathrm{ExactlyOne}(\mathrm{mp}^t_{q,p_0}, \ldots, \mathrm{mp}^t_{q,p_n}) \tag{1}$$

$$\bigwedge_{p\in P} \mathrm{AtMostOne}(\mathrm{mp}^t_{q_0,p}, \ldots, \mathrm{mp}^t_{q_m,p}) \tag{2}$$

$$\bigwedge_{p\in P} \left(\mathrm{oc}^t_p \iff \big(\bigvee_{q\in Q} \mathrm{mp}^t_{q,p}\big)\right) \tag{3}$$

The above constraints ensure that the $\mathrm{mp}^t_{q,p}$ and $\mathrm{oc}^t_p$ variables are set correctly. Here $\mathrm{ExactlyOne}(t_1, \ldots, t_n) \triangleq \sum_{i=1}^n t_i = 1$ denotes that exactly one of $\{t_1, \ldots, t_n\}$ is true while $\mathrm{AtMostOne}(t_1, \ldots, t_n) \triangleq \sum_{i=1}^n t_i \leq 1$ denotes that at most one of $\{t_1, \ldots, t_n\}$ is true.

*ConnectivityConstraints*

$$\bigwedge_{q,q'\in Q} \left( \bigwedge_{(p,p')\in P_{conn}} \big((\mathrm{mp}^t_{q,p} \wedge \mathrm{mp}^t_{q',p'}) \Rightarrow \mathrm{e}^t_{q,q'}\big) \quad \wedge \bigwedge_{p,p'\notin P_{conn}} \big((\mathrm{mp}^t_{q,p} \wedge \mathrm{mp}^t_{q',p'}) \Rightarrow \neg \mathrm{e}^t_{q,q'}\big)\right) \tag{4}$$

$$\bigwedge_{g\in G_{CX}} (\mathrm{c}^t_g \implies \mathrm{e}^t_{g.q_0,g.q_1}) \tag{5}$$

Constraint 4 ensures that the $e^t_{q,q'}$ predicates are true if and only if the physical qubits assigned to $q$ and $q'$ at time $t$ are connected. Constraint 5 ensures that the CX gates are applied to connected physical qubits.

*GateConstraints*

The Gate constraints follow a similar pattern to those in [21], but are substantially different because in this encoding at most one gate can be scheduled on each qubit at a given time step. For a given time $t$, a gate is either currently scheduled denoted $\mathrm{c}_g^t$, it has been scheduled for some $t' < t$ denoted $\mathrm{a}_g^t$ (advanced), or it will be scheduled for some $t' > t$ denoted $\mathrm{d}_g^t$ (delayed) as encoded in constraint 6. Furthermore, no gate can be "advanced" on the first time step since nothing has been applied yet as encoded in constraint 7.

$$\bigwedge_{g \in G} \mathrm{ExactlyOne}(\mathrm{c}_g^t, \mathrm{a}_g^t, \mathrm{d}_g^t) \tag{6}$$

$$\bigwedge_{g \in G} \neg \mathrm{a}_g^1 \tag{7}$$

The main idea from the size-optimal encoding was that of two-way propagation. This is adapted to a depth-optimal encoding by ensuring that if a gate is "current" or "delayed" its direct successors must be "delayed", while if a gate is "current" or "advanced" its direct predecessors must be "advanced". This information is propagated throughout the circuit, giving rise to the name. Constraint 8 enforces that the dependencies of the input circuit are preserved in the output circuit and that two gates are not applied on the same qubit at the same time. We use $S(g)$ (and $P(g)$) to denote the successors (predecessors) of gate $g$ in the dependency graph of the input circuit.

$$\bigwedge_{g \in G} \Big( \bigwedge_{g' \in S(g)} (\mathrm{c}_g^t \vee \mathrm{d}_g^t) \implies \mathrm{d}_{g'}^t \;\wedge \bigwedge_{g' \in P(g)} (\mathrm{c}_g^t \vee \mathrm{a}_g^t) \implies \mathrm{a}_{g'}^t \Big) \tag{8}$$

To ensure that gates are not skipped or applied twice, constraint 9 ensures that a gate is "advanced" if and only if it was "current" or "advanced" in the previous time step. Likewise, a gate is "current" or "delayed" if and only if it was "delayed" in the previous time step.

$$\bigwedge_{g \in G} \Big( \big( (\mathrm{c}_g^{t-1} \vee \mathrm{a}_g^{t-1}) \iff \mathrm{a}_g^t \big) \wedge \big( \mathrm{d}_g^{t-1} \iff (\mathrm{c}_g^t \vee \mathrm{d}_g^t) \big) \Big) \tag{9}$$

Constraint 10 is added to speed up solving time by reducing the search space. This constraint express that when a gate is "current" none of its predecessors and successors can be "current". We use *fullS*$(g)$ (and *fullP*$(g)$) to denote the transitive closure of the successor (predecessor) relation in the dependency graph. We do this to improve the automatic propagation of the dependency constraints.

$$\bigwedge_{g \in G} \bigwedge_{g' \in \mathit{fullS}(g) \cup \mathit{fullP}(g)} \mathrm{c}_g^t \implies \neg \mathrm{c}_{g'}^t \tag{10}$$

Whenever a gate is "current", the physical qubit(s) that it is being applied on must be "usable", i.e., not undergoing a SWAP. This is encoded for binary gates in constraint 11 and for unary gates in constraint 12. These constraints ensure that a gate and a SWAP are not applied at the same time step.

$$\bigwedge_{g \in G_{CX}} \bigwedge_{p \in P} \mathrm{c}_g^t \implies \big( (\mathrm{mp}_{g.q_0,p}^t \vee \mathrm{mp}_{g.q_1,p}^t) \implies \mathrm{u}_p^t \big) \tag{11}$$

$$\bigwedge_{g \in G_{unary}} \bigwedge_{p \in P} \mathrm{c}_g^t \implies (\mathrm{mp}_{g.q,p}^t \implies \mathrm{u}_p^t) \tag{12}$$

*SwapConstraints*

Constraint 13 encodes that a given physical qubit can participate in at most one SWAP gate for any three consecutive time steps. This is to ensure that SWAP gates are not applied in conflict with each other. We use $Sw(p)^t$ to mean the set of variables $\mathrm{sw}^t_{p,p'}$ where $(p, p') \in P_{conn}$.

$$\bigwedge_{p\in P} \mathrm{AtMostOne}(Sw(p)^t \cup Sw(p)^{t-1} \cup Sw(p)^{t-2}) \tag{13}$$

The "swap" variables indicate the last time step of a SWAP gate. To ensure that gates are not scheduled on top of SWAPs we require that if the "swap touched" variable is set for a qubit, it is not available for gates to be scheduled on it for this and the preceding two time steps as seen in constraint 14. Also, as encoded in constraint 15, the "swap" variables must all be set to false for the first three time steps.

$$\bigwedge_{p\in P} \bigwedge_{t'\in\{t,t-1,t-2\}} (\mathrm{st}^t_p \implies \neg \mathrm{u}^{t'}_p) \tag{14}$$

$$\bigwedge_{p,p'\in P_{conn}} (\neg \mathrm{sw}^1_{p,p'} \wedge \neg \mathrm{sw}^2_{p,p'} \wedge \neg \mathrm{sw}^3_{p,p'}) \tag{15}$$

Nothing stops the SAT solver from swapping unmapped physical qubits. Such SWAPs are referred to as 'spurious SWAPs'. To ensure that no spurious SWAPs are scheduled, we require that at least one physical qubit in a SWAP must have a logical qubit mapped to it in constraint 16. We denote a SWAP with only one mapped physical qubit 'ancillary SWAP'. Ancillary SWAPs can be disabled by requiring that both physical qubits participating in a SWAP be occupied. We only present results allowing ancillary SWAPs.

$$\bigwedge_{p,p'\in P_{conn}} (\mathrm{sw}^t_{p,p'} \implies (\mathrm{oc}^t_p \vee \mathrm{oc}^t_{p'})) \tag{16}$$

Three final constraints are added to encode the effect of a SWAP gate. Constraint 17 says that if a "swap" variable is set which includes the physical qubit $p$, the "swap touched" variable for $p$ is also set. Constraint 18 says that if the swap touched variable is not set for a qubit, its mapping is preserved from the last time step. Finally, constraint 19 says that when the swap variable for two qubits is set, the mappings for them switch compared to the previous time step.

$$\bigwedge_{p\in P} (\mathrm{st}^t_p \iff (\bigvee_{p'\in conn(p)} \mathrm{sw}^t_{p,p'})) \tag{17}$$

$$\bigwedge_{p\in P} \bigwedge_{q\in Q} (\neg \mathrm{st}^t_p \implies (\mathrm{mp}^{t-1}_{q,p} \iff \mathrm{mp}^t_{q,p})) \tag{18}$$

$$\bigwedge_{p,p'\in P_{conn}} \bigwedge_{q\in Q} \Big(\mathrm{sw}^t_{p,p'} \implies ((\mathrm{mp}^{t-1}_{q,p} \iff \mathrm{mp}^t_{q,p'}) \wedge (\mathrm{mp}^{t-1}_{q,p'} \iff \mathrm{mp}^t_{q,p}))\Big) \tag{19}$$

*Assumptions*

We also encode an assumption for incremental SAT solving in constraint 20: For each $t$ exceeding the depth of the input circuit, we add this constraint expressing that no gate is delayed at the last time step, which ensures that all gates from the input circuit are scheduled. This tells the SAT solver that we expect a solution at time step $t$, and consequently not at any time step prior to $t$.

$$\bigwedge_{g\in G} (\mathrm{asm}^t \implies \neg \mathrm{d}^t_g) \tag{20}$$

## 3.1 Encoding variations

There are two goal variations that are easily implemented. The first one is finding optimal CX-depth instead of depth. This can be achieved by the same encoding, by preprocessing the circuit to only include CX gates. After the optimal placement of SWAPs has been found using the encoding, the unary gates can be reinserted to get the final circuit.

The other variation is finding a circuit with minimal depth and locally minimal SWAP count. After a circuit with minimal depth has been found, we count the number of SWAPs $S$ and force it to decrease in increments of one by adding new assumption variables $\mathrm{sasm}^i$ for $i = S - 1, \dots, 0$. To force the number of SWAPs to be at most $i$, the following constraint is added to the instance: $\mathrm{sasm}^i \implies \mathrm{AtMostN}(i, \mathrm{sw}^0_{p_0,p_1}, \dots, \mathrm{sw}^t_{p_{n-1},p_n})$, where $\mathrm{sw}^0_{p_0,p_1}, \dots, \mathrm{sw}^t_{p_{n-1},p_n}$ represents all "swap" variables that have been created in total over all layers. The first $i$ where it is not possible to solve an instance, tells us that the locally minimal number of SWAP gates is $i + 1$. This is because the search space is monotone in the number of SWAP gates [15].

## 3.2 Count Optimal Encoding vs Depth Optimal Encoding

We have employed several effective techniques from SAT based SWAP-count optimal synthesis in [21] for a scalable depth optimal encoding in this paper. In this Section, we summarize the similarities and differences between the count optimal and depth optimal encodings.

The following list describes the techniques adapted from count optimal encoding:

- Incremental solving to reuse learned clauses from earlier bounds with simple assumptions.
- Bi-directional propagation of gate constraints using "delayed" and "advanced" variables.
- Similar cardinality and connectivity constraints in both encodings.

The following list summarizes the main differences from the count optimal encoding:

- A SWAP gate (with depth 3) can interleave with other 1-qubit and 2-qubit gates in depth optimal encoding unlike a count optimal setting. Equations 13 to 19 present the new constraints for enabling interleaving of SWAP gates.
- The makespan of a count optimal encoding corresponds to the number of SWAPs needed. However, the makespan of a depth optimal encoding is at least the depth of the original circuit. Since the circuit depth/CNOT-depth is often several times more than the number of SWAPs, scalability of encodings differs greatly.
- As discussed in Section 3.1, a depth optimal encoding also enables local SWAP count optimization with simple additional constraints. On the other hand, one cannot easily optimize local depth in a count optimal encoding.

## 4 Experimental evaluation

Our SAT encoding is implemented in the tool QuilLS[2]. Remember that optimizing depth is harder than optimizing count due to larger makespan. For an evaluation, we are interested in the performance penalty by depth optimization instead of count. Further, we are also interested in the quality of results and their impact on noise reduction which is the ultimate goal for circuit optimization. In particular, we structure our experiments to answer the following research questions:

[2] QuilLS is available at `https://github.com/anbclausen/quills`. It is also ported to Q-Synth v4 at `https://github.com/irfansha/Q-Synth`. The experiments reported here are obtained with QuilLS.

**A** How does the runtime performance of QuilLS compare to Q-Synth, TB-OLSQ2, and OLSQ2?

**B** How do the depth, CX-depth, SWAP count of QuilLS compare to SABRE, Q-Synth, TB-OLSQ2, and OLSQ2?

**C** How do different optimization strategies affect the quality of the synthesized circuits in terms of noise?

Our experiments are run on a Linux machine with a 2.4GHz CPU and 60GB of memory. We always use a timeout of 2 hours (7200 seconds). For our benchmarks, we use a set of 33 standard benchmarks (15 arithmetic and 18 QUEKO [23]) with up to 54 qubits (q) and 270 CX gates from the literature [15, 21]. Furthermore, we use 10 8q Variational Quantum Eigensolver (VQE) circuits with up to 79 CX gates from [21] representing practically applicable quantum circuits. Each of these benchmark circuits is synthesized on the following platforms: Tenerife (5q), Melbourne (14q), Guadalupe (16q), Tokyo (20q), and Cambridge (32q) by IBM; Google's Sycamore (54q); and Rigetti's Aspen-M (80q). Given a test circuit, we only synthesize it on a platform if the platform has the required number of qubits. The exact data on the circuits can be found in Section A.

For our evaluation, we use the following tools:

- QuilLS: Our (CX)-depth-optimal tool, with CaDiCaL (v1.5.3) [1] SAT solver as our backend.
- SABRE [13]: A heuristic algorithm that is part of Qiskit. We use the same experimental setting as in [21].
- Q-Synth (v2) [21]: A SAT based state-of-the-art size-optimal tool with CaDiCal (v1.5.3) as backend.
- OLSQ2 [15]: An SMT-based state-of-the-art depth-optimal tool with Z3 (v4.13.0.0) [4] as a backend.
- TB-OLSQ2 [15]: A near-optimal version of OLSQ2 with better runtime performance. We report a timeout if the tool does not terminate within the time limit.

For both OLSQ2 and TB-OLSQ2, we use the best settings, i.e., allowing them to use SABRE to get an upper bound.

All measurements are available on Zenodo [9][3] and an overview of the data is given in Appendix B in the full technical report [8]. Next, we will discuss the setup and the results of each experiment in detail.

## 4.1 Performance of QuilLS Compared to Existing Tools

### Experimental setup

We are interested in both depth and CX-depth optimization. As mentioned in Section 3.1, we can optimize either depth or CX-depth in our SAT encoding. We also ensure that QuilLS always computes locally minimal SWAPs. The OLSQ2 and TB-OLSQ2 tools optimize depth, and we can make them optimize CX-depth instead, by removing all unary gates from the input circuit before passing it to these tools. For a fair comparison, we set options such that OLSQ2 and TB-OLSQ2 also compute locally minimal SWAP count. We run SABRE, Q-Synth and the (CX)-depth-optimal versions of (TB)-OLSQ2 and QuilLS on all the benchmarks on every platform and measure running time.

[3] Full code and data is available at `https://doi.org/10.5281/zenodo.15606051`

## Results

First, we focus on depth optimization for tools QuilLS and OLSQ2. The times for computing a heuristic solution with SABRE are excluded since they are all smaller than 0.5 seconds. Figure 3 shows the running times comparison of QuilLS and the other tools. The full data set with running times for all tools can be found in the appendix.

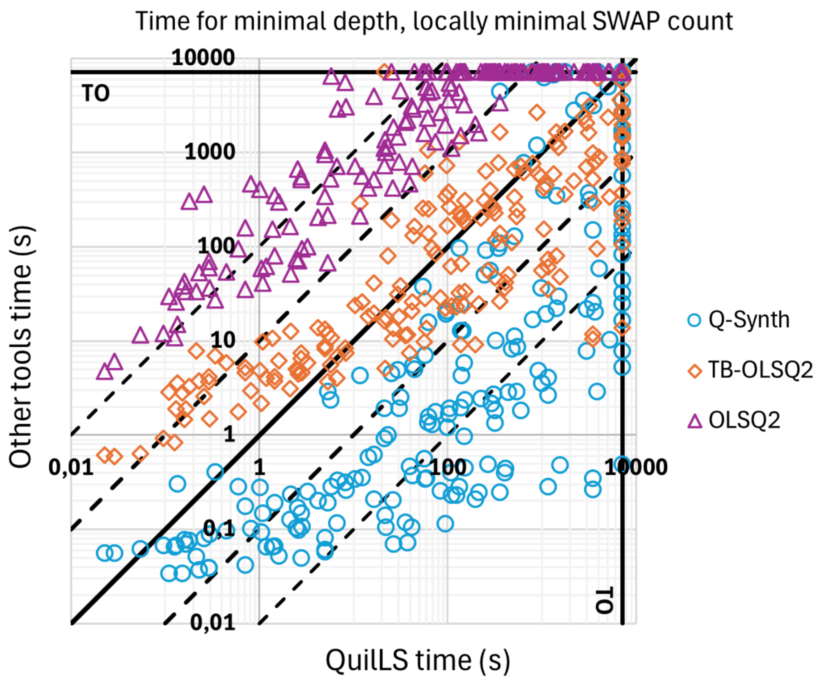


**Figure 3** Time comparison between QuilLS and other tools (with their own optimization metrics). A mark on the bold diagonal line shows equal performance between the tools, a mark above shows better performance by QuilLS, and a mark below shows worse performance by QuilLS than the other tool. Dotted diagonal lines indicate performance ratios of 10 and 100. Bold horizontal and vertical lines indicate timeouts (TO).

In Table 2 the number of solved instances, and time-outs are shown for each tool/metric. We also include the results with CX-depth optimization by QuilLS and OLSQ2.

## QuilLS vs OLSQ2 and TB-OLSQ2

OLSQ2 is our direct competitor that optimizes the same metric, i.e., depth and locally minimal SWAP count. From the results, it is clear that we significantly outperform OLSQ2. OLSQ2 is almost always at least 10 times slower than QuilLS, and sometimes up to 3 orders of magnitude slower. TB-OLSQ2 on the other hand, being a near-optimal tool performs better on circuits with larger synthesis times. This is consistent with the observations between optimal and near-optimal synthesis in the literature [15].

## QuilLS vs Q-Synth

Q-Synth is generally 10 times faster than QuilLS and more than 100 times faster on many circuits. Despite being similar encodings and using the same SAT solver as a backend, we see the hardness of optimizing depth compared to SWAP count. Interestingly, on some circuits with large synthesis times Q-Synth uses more time than QuilLS.

**Table 2** Timeouts (TO), and solved instances for QLS tools.

| Tool | TO | Solved | Solved % |
|---|---|---|---|
| SABRE | 0 | 220 | 100% |
| TB-OLSQ2 (CX-depth) | 26 | 194 | 88% |
| TB-OLSQ2 (depth) | 30 | 190 | 86% |
| Q-Synth (SWAP count) | 31 | 189 | 86% |
| QuilLS (CX-depth) | 35 | 185 | 84% |
| QuilLS (depth) | 48 | 172 | 78% |
| OLSQ2 (CX-depth) | 102 | 118 | 54% |
| OLSQ2 (depth) | 125 | 95 | 43% |

### Depth vs CX-depth optimization in QuilLS

It is faster in general to compute a CX-depth-optimal circuit than a depth-optimal circuit. This is expected, since the instances are smaller. This is also the case for (TB)-OLSQ2, and the performance for CX-depth-optimal synthesis gives a picture very similar to Figure 3.

Q-Synth, TB-OLSQ2, and the CX-depth-optimal version of QuilLS solve 84-88% of instances, while the depth-optimal version of QuilLS solves 78% of instances. OLSQ2 solves around 50% of instances. SABRE solves every instance within a second. However, in the next experiment we will see that the quality of mappings from SABRE are indeed far from optimal.

## 4.2 Quality of Results for QuilLS Compared to Existing Tools

In the previous experiment, we have seen a performance penalty on optimizing depth instead of SWAP count. In this experiment, we look at the quality of the results reported by all the tools. We are interested in finding a trade-off between performance and quality of results. So far, we have considered the two metrics SWAP count and depth with locally optimal SWAP count. Since it is quite cheap to compute SWAP optimal mappings, one could also consider optimizing depth for the optimal SWAP count. To do this, one can use e.g. Q-Synth to find the optimal SWAP count, and then use our tool QuilLS for optimizing depth given a bound on SWAP count (we call this combination “Q-S+Qui”). The experimental setup is the same as for the previous section, except that we include this new variant: size optimal with locally minimal (CX)-depth. We now measure the depth, CX-depth, and SWAP count for each synthesized circuit.

### Results

Table 3 shows for each tool and optimization metric the average ratio of the depth, CX-depth and CX count compared to optimal tools, over a set of benchmarks that all tools could solve (93 for all tools, 129 if we discard OLSQ2). We also show the success rate (number of solved circuits over all 220 benchmarks). We report the number of CX gates in the synthesized circuit rather than the SWAP gates, since the CX gate count also reflects the size of the original circuit. We set the tool and metric combination that guarantees optimal values to **1.00** and provide the ratios of the results from other tools. A higher ratio means a worse result.

**Table 3** Average ratios between depth (d), CX-depth (CX-d) and CX gate count (CX) of circuits mapped by various tool/metric combinations. Bold numbers indicate guaranteed optimality. 93 circuits are solved by all tools; 129 circuits are solved by all tools except OLSQ2.

| | | | Avg. over N=129 | | | Avg. over N=93 | | |
|---|---|---|---|---|---|---|---|---|
| Tool/metric/%success | | | d. | CX-d | CX | d. | CX-d | CX |
| SABRE | heur | 100 | 1.48 | 1.46 | 1.15 | 1.29 | 1.27 | 1.10 |
| TB-O2 | CX-d | 88 | 1.09 | 1.06 | 1.01 | 1.08 | 1.06 | 1.01 |
| TB-O2 | d. | 86 | 1.08 | 1.07 | 1.01 | 1.08 | 1.07 | 1.01 |
| Q-Synth | CX | 86 | 1.06 | 1.05 | **1.00** | 1.04 | 1.03 | **1.00** |
| QuilLS | CX-d | 84 | 1.03 | **1.00** | 1.04 | 1.02 | **1.00** | 1.03 |
| QuilLS | d. | 78 | **1.00** | 1.03 | 1.09 | **1.00** | 1.02 | 1.04 |
| Q-S+Qui | CX-d | 67 | 1.04 | 1.01 | **1.00** | 1.03 | 1.01 | **1.00** |
| Q-S+Qui | d. | 60 | 1.02 | 1.02 | **1.00** | 1.01 | 1.02 | **1.00** |
| OLSQ2 | CX-d | 54 | – | – | – | 1.02 | **1.00** | 1.03 |
| OLSQ2 | d. | 43 | – | – | – | **1.00** | 1.01 | 1.04 |

### Discussion

For all three measures, SABRE gets the worst ratios, which is to be expected since SABRE is a heuristic tool; it is also the only tool that can map all instances. Comparing N=129 vs N=93, it is also clear that for harder instances, SABRE inserts relatively more SWAP gates.

Only OLSQ2 and QuilLS guarantee minimal (CX)-depth. Q-Synth and TB-OLSQ2 get (CX)-depths that are less than 10% off from the optimal depths. Q-Synth (including Q-S+Qui) guarantees optimal CX-count. QuilLS (optimizing for depth) is on average 9% off from the CX-count optimum (for N=129). It performs the same on CX-count as OLSQ2 (for N=93). The other tool combinations have 1-4% worse CX-count than Q-Synth, except the heuristic tool SABRE (15% off).

We conclude that QuilLS finds 5-9% better depths and CX-depths on average than the optimal tools with different optimization goals, while finding 46% better (CX)-depths on average than SABRE. This comes at the cost of larger CX-counts than other tools. We observe that first optimizing SWAP count and then depth produces very good results overall.

## 4.3 Impact of Optimization Metrics on Noise Reduction

Our ultimate motivation for circuit optimization is noise reduction. The noise in a quantum circuit comes from noisy gates, interference from the environment, and measurement errors. One way to evaluate the impact of these errors is by running the same circuit many times and comparing the distribution of the measurement outcomes to the expected distribution. These distributions can be obtained, for instance by Qiskit's simulator with and without noise models.

Each quantum platform in the Qiskit library comes with a noise model, providing noise probabilities for native gates. To use these, we first transpile our circuits to IBM's native gates, using Qiskit (without optimization).

The noise profile for gates varies significantly between the different qubits of the platform. This poses a problem for us: The same synthesized circuit with different initial mappings would have very different levels of noise. Noise models are represented as a set of probability distributions over quantum operations, so we created an averaged version of each noise model by averaging the noise distribution per operation over all qubits. In this way, the noise model is still realistic, but does not depend on the initial mapping.

Qiskit's noise models specify gate errors and measurement errors, but not the interference from the environment when qubits are idle (also called decoherence). To incorporate this, we use the same approach as in [12]: Qiskit provides a noise distribution for the `I`-gate, which is meant to model decoherence [18]. Hence, before simulating a circuit, we insert I-gates for all idle positions in that circuit.

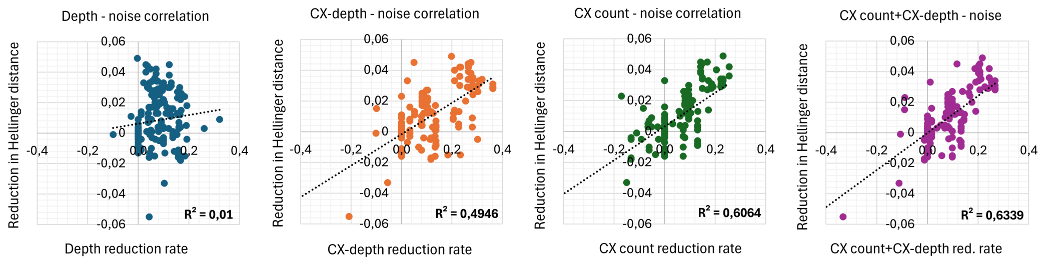
We apply all the different QLS tools (with the variations discussed in earlier experiments) to the transpiled circuits, and simulate the mapped circuits under the averaged noise model.

## Experimental setup

We simulate each synthesized circuit 100,000 times with and without noise. When simulating a circuit, we measure the mapped qubits at the very end of the circuit to obtain a probability distribution. We use Qiskit to compute the Hellinger distance between the two distributions obtained from simulations with and without the noise model. This is standard in the literature [3, 6, 19]. Given the resource intensity of simulation, we only use the medium-sized platform IBM Guadalupe (16q) with the standard and VQE benchmarks.

## Results

Figure 4 shows the correlations between reduction rates for depth, CX-depth, CX count, and the sum of CX-depth and CX count and the reduction in the Hellinger distance as a measure for noise. We consider SABRE as baseline and report reductions in various tools. The depth reduction rate for tool X is normalized as $(d(SABRE) - d(X))/d(SABRE)$, and similar for the other metrics. This presentation and analysis style follows the example of [25].



**(a)** Noise coefficient 1%. **(b)** Noise coefficient 49%. **(c)** Noise coefficient 61%. **(d)** Noise coefficient 63%.

**Figure 4** Noise correlation coefficients between reduction rate for the optimization goal and reduction in Hellinger distance.

## Discussion

Figure 4 shows that the depth reduction rate is not correlated with noise reduction at all, while the CX-depth reduction rate and the CX gate count reduction rate are quite strongly correlated. Surprisingly, CX gate count reduction correlates more with noise than CX-depth reduction. The correlation is strengthened further when looking at the sum of CX-depth and CX gate count.

This result is surprising since it is generally assumed that noise scales with depth [25, 12, 14, 15]. For example, the authors of [14] stress the importance of considering unary gates when minimizing depth. In [22], it is argued that minimizing CX-depth provides a better noise reduction than minimizing depth, since the binary CX gates are more noisy. But seeing no correlation between depth and noise is unexpected.

We conclude that reducing CX gate count and CX-depth reduces noise in output circuits. Further, CX gate count correlates better than CX-depth. CX count + CX-depth provides the best correlation so far. Surprisingly, we see no correlation between reducing depth and reducing noise.

## 5 Conclusion

In this paper, we have presented an efficient SAT encoding of (CX)-depth-optimal quantum layout synthesis. The SAT encoding has variations that allow ancillary SWAPs and finding locally minimal SWAP counts. We have implemented these encodings in the tool QuilLS and tested them on standard benchmarks. We compare QuilLS to the state-of-the-art depth-optimal tool, OLSQ2 [15], its near-optimal variant TB-OLSQ2, the state-of-the-art size optimal tool, Q-Synth [21], and a heuristic, fast but sub-optimal tool, SABRE [13]. QuilLS and OLSQ2 synthesize circuits with better (CX)-depths than the other tools, and QuilLS vastly outperforms the previous best depth-optimal tool OLSQ2, by solving more instances, and often being more than 10-100x faster.

We also evaluate the amount of noise in the circuits synthesized by each tool, by measuring the Hellinger distance between ideal and noisy simulations of the circuits. We see that reducing the number of SWAPs and reducing CX-depth (and in particular their combination) is strongly correlated with reducing noise. Contrary to what is generally assumed in the literature, we do not see a correlation between reducing depth and reducing noise. However, this should be explored further, by testing whether these results carry over to larger quantum platforms and when executing the circuits on real quantum computers.

## References

1 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`.

2 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021. `doi:10.3233/FAIA336`.

3 Samudra Dasgupta and Travis S. Humble. Characterizing the reproducibility of noisy quantum circuits. *Entropy*, 24(2):244, February 2022. `doi:10.3390/e24020244`.

4 Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In *TACAS Proceedings*, LNCS 4963, pages 337–340. Springer, 2008. `doi:10.1007/978-3-540-78800-3_24`.

5 M. R. Garey, D. S. Johnson, and R. Endre Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976. `doi:10.1137/0205049`.

6 Konstantinos Georgopoulos, Clive Emary, and Paolo Zuliani. Modeling and simulating the noisy behavior of near-term quantum computers. *Physical Review A*, 104(6), December 2021. `doi:10.1103/physreva.104.062432`.

7 Yuichi Hirata, Masaki Nakanishi, Shigeru Yamashita, and Yasuhiko Nakashima. An efficient conversion of quantum circuits to a linear nearest neighbor architecture. *Quantum Inf. Comput.*, 11:142–166, 2011. `doi:10.26421/QIC11.1-2-10`.

8 Anna B. Jakobsen, Anders B. Clausen, Jaco van de Pol, and Irfansha Shaik. Depth-Optimal Quantum Layout Synthesis as SAT (full version). *CoRR*, 2025. `arXiv:2506.06752`.

9 Anna Blume Jakobsen, Anders B. Clausen, J.C. van de Pol, and Irfansha Shaik. Code, benchmarks and data for "depth-optimal quantum layout synthesis as sat" paper at sat25, June 2025. `doi:10.5281/zenodo.15606051`.

10 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`.

11 M. S. Krishnamoorthy. An NP-hard problem in bipartite graphs. *SIGACT News*, 7(1):26, January 1975. `doi:10.1145/990518.990521`.

12 Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. QASMBench: A low-level quantum benchmark suite for NISQ evaluation and simulation. *ACM Transactions on Quantum Computing*, 4(2), February 2023. `doi:10.1145/3550488`.

13 Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *ASPLOS*, pages 1001–1014. ACM, 2019. `doi:10.1145/3297858.3304023`.

14 Sanjiang Li, Ky Dan Nguyen, Zachary Clare, and Yuan Feng. Single-qubit gates matter for optimising quantum circuit depth in qubit mapping. In *ICCAD*, pages 1–9. IEEE, 2023. `doi:10.1109/ICCAD57390.2023.10323863`.

15 Wan-Hsuan Lin, Jason Kimko, Bochen Tan, Nikolaj Bjørner, and Jason Cong. Scalable optimal layout synthesis for NISQ quantum processors. In *DAC'23*, pages 1–6, July 2023. `doi:10.1109/DAC56929.2023.10247760`.

16 Joao Marques-Silva, Ines Lynce, and Sharad Malik. Conflict-driven clause learning sat solvers. *Handbook of Satisfiability*, 336:133–182, 2021. `doi:10.3233/FAIA200987`.

17 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, 2010.

18 Qiskit contributors. Qiskit documentation, 2023. URL: `https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.IGate`.

19 Salonik Resch and Ulya R. Karpuzcu. Benchmarking quantum computers and the impact of quantum noise. *ACM Comput. Surv.*, 54(7):142:1–142:35, 2022. `doi:10.1145/3464420`.

20 Irfansha Shaik and Jaco van de Pol. Optimal layout synthesis for quantum circuits as classical planning. In *2023 IEEE/ACM International Conference on Computer Aided Design*, ICCAD'23. IEEE, 2023. `doi:10.1109/ICCAD57390.2023.10323924`.

21 Irfansha Shaik and Jaco van de Pol. Optimal layout synthesis for deep quantum circuits on NISQ processors with 100+ qubits. In *SAT'24*, volume 305, pages 26:1–18. LIPIcs, 2024. `doi:10.4230/LIPIcs.SAT.2024.26`.

22 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`.

23 Bochen Tan and Jason Cong. Optimality study of existing quantum computing layout synthesis tools. *IEEE Transactions on Computers*, 70(9):1363–1373, 2021. `doi:10.1109/TC.2020.3009140`.

24 Robert Wille, Lukas Burgholzer, and Alwin Zulehner. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19. ACM, June 2019. `doi:10.1145/3316781.3317859`.

25 Xin-Chuan Wu, Marc Grau Davis, Frederic T. Chong, and Costin Iancu. Qgo: Scalable quantum circuit optimization using automated synthesis. *CoRR*, 2022. `arXiv:2012.09835`.

26 Jiong Yang, Yaroslav A. Kharkov, Yunong Shi, Marijn J. H. Heule, and Bruno Dutertre. Quantum Circuit Mapping Based on Incremental and Parallel SAT Solving. In *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*, volume 305 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.SAT.2024.29`.

27 Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 38(7):1226–1236, 2019. `doi:10.1109/TCAD.2018.2846658`.

## A Benchmarks Information

The circuit data for all the circuits used in the experiments can be found in Table 4.

**Table 4** Circuit data for all standard, VQE and QUEKO benchmarks.

| Circuit name | Qubits | Gates | CX gates | Depth | CX-depth |
|---|---|---|---|---|---|
| 4gt13_92 | 5 | 66 | 30 | 38 | 26 |
| 4mod5-v1_22 | 5 | 21 | 11 | 12 | 10 |
| adder | 4 | 23 | 10 | 11 | 6 |
| barenco_tof_4 | 7 | 72 | 34 | 68 | 34 |
| barenco_tof_5 | 9 | 104 | 50 | 95 | 48 |
| mod_mult_55 | 9 | 91 | 40 | 47 | 25 |
| mod5mils_65 | 5 | 35 | 16 | 21 | 16 |
| or | 3 | 17 | 6 | 8 | 6 |
| qaoa5 | 5 | 22 | 8 | 14 | 8 |
| qft_8 | 8 | 106 | 56 | 42 | 26 |
| rc_adder_6 | 14 | 140 | 71 | 83 | 40 |
| tof_4 | 7 | 55 | 22 | 46 | 21 |
| tof_5 | 9 | 75 | 30 | 61 | 28 |
| toffoli | 3 | 15 | 6 | 11 | 6 |
| vbe_adder_3 | 10 | 89 | 50 | 58 | 33 |
| vqe_8_0_5_100 | 8 | 96 | 52 | 79 | 44 |
| vqe_8_0_10_100 | 8 | 129 | 63 | 92 | 51 |
| vqe_8_1_5_100 | 8 | 43 | 18 | 32 | 16 |
| vqe_8_1_10_100 | 8 | 100 | 47 | 76 | 41 |
| vqe_8_2_5_100 | 8 | 90 | 48 | 80 | 44 |
| vqe_8_2_10_100 | 8 | 153 | 79 | 136 | 75 |
| vqe_8_3_5_100 | 8 | 76 | 40 | 61 | 35 |
| vqe_8_3_10_100 | 8 | 151 | 78 | 119 | 68 |
| vqe_8_4_5_100 | 8 | 76 | 39 | 60 | 32 |
| vqe_8_4_10_100 | 8 | 136 | 71 | 102 | 56 |
| 16QBT_05CYC_TFL_0 | 16 | 37 | 15 | 5 | 5 |
| 16QBT_10CYC_TFL_0 | 16 | 73 | 29 | 10 | 7 |
| 16QBT_15CYC_TFL_0 | 16 | 109 | 44 | 15 | 11 |
| 16QBT_20CYC_TFL_0 | 16 | 145 | 58 | 20 | 14 |
| 16QBT_25CYC_TFL_0 | 16 | 180 | 72 | 25 | 15 |
| 16QBT_30CYC_TFL_0 | 16 | 217 | 87 | 30 | 18 |
| 16QBT_35CYC_TFL_0 | 16 | 253 | 101 | 35 | 25 |
| 16QBT_40CYC_TFL_0 | 16 | 289 | 116 | 40 | 27 |
| 16QBT_45CYC_TFL_0 | 16 | 325 | 130 | 45 | 30 |
| 54QBT_05CYC_QSE_0 | 54 | 192 | 54 | 5 | 5 |
| 54QBT_10CYC_QSE_0 | 54 | 384 | 108 | 10 | 10 |
| 54QBT_15CYC_QSE_0 | 54 | 576 | 162 | 15 | 12 |
| 54QBT_20CYC_QSE_0 | 54 | 767 | 216 | 20 | 16 |
| 54QBT_25CYC_QSE_0 | 54 | 959 | 270 | 25 | 21 |
| 54QBT_30CYC_QSE_0 | 54 | 1151 | 324 | 30 | 24 |
| 54QBT_35CYC_QSE_0 | 54 | 1342 | 378 | 35 | 32 |
| 54QBT_40CYC_QSE_0 | 54 | 1534 | 432 | 40 | 33 |
| 54QBT_45CYC_QSE_0 | 54 | 1727 | 487 | 45 | 38 |