Layout Synthesis for Near-Term Quantum Computing: Gap Analysis and Optimal Solution

Bochen Tan and Jason Cong

1 Introduction

The nature of quantum computing (QC) decides that it is much more error-prone than classical computing. The scalable and effective way to resolve this issue is encoding quantum information with quantum error correction (QEC) codes [16, 36], but current technology cannot produce a quantum processor with enough size and fidelity for QEC. In this chapter, we consider near-term QC without QEC. Note that the "near-term" formalism is by no means ephemeral. The early QEC-capable hardware will not be able to run any application on top of the QEC schemes because QEC has huge overheads. It is estimated that application-ready error-corrected quantum computers are at least 10 years away [21, 30], assuming many engineering challenges [1] are solved. Meanwhile, all the QC applications can only use the near-term formalism [32].

Hardware technology draws the upper bound of QC capability, while compilation determines if and how much we can utilize this capability. A QC compiler has to transform the input quantum program to satisfy constraints imposed by the quantum *architecture*. In addition, errors accumulate fast when scaling up QC without QEC, so the compiler should make best efforts to reduce errors. There has been over a decade of QC compilation research, as summarized in Sect. 2, so it is vital to quantitatively examine these existing tools and, if there is still a significant room, improve them.

In general, there are two types of constraints in QC compilation: *logic constraints* and *layout constraints*. Quantum programs are specified as a list of instructions, e.g., Fig. 1a, where each one is an operation on a set of *program qubits*. They can also be

B. Tan · J. Cong (✉)
University of California, Los Angeles, CA, USA
e-mail: bctan@cs.ucla.edu; cong@cs.ucla.edu

R. O. Topaloglu (ed.), *Design Automation of Quantum Computers*,
https://doi.org/10.1007/978-3-031-15699-1_2

```
initialize q[1:5]
g1: CX q[2],q[1]
g2: CX q[3],q[4]
g3: u1 q[3]
g4: u2 q[4]
g5: u3 q[5]
g6: CX q[1],q[5]
g7: u4 q[1]
g8: CX q[4],q[3]
g9: CX q[5],q[2]
```

(a)

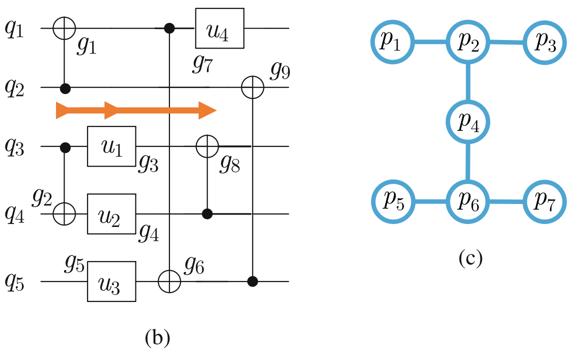


Fig. 1 Inputs of the layout synthesis problem: quantum program and coupling graph. (**a**) A quantum program consisting of 9 gates on 5 program qubits. (**b**) Circuit of the quantum program. Each horizontal wire is a program qubit. Time goes from left to right. The arrow means a dependency chain (g_2, g_3, g_8). (**c**) Coupling graph of “segment H on a Falcon processor” from IBM Quantum [20]. Each vertex is a physical qubit. Entangling two-qubit gates can only be applied to adjacent physical qubits

drawn as circuit diagrams like Fig. 1e where each wire represents a program qubit and each gate corresponds to an operation. A valid gate on n qubits is represented by a unitary matrix of dimension 2^n, e.g.,

$$H = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}, U(\theta, \lambda, \phi) = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & e^{i\lambda+i\phi}\cos\left(\frac{\theta}{2}\right) \end{bmatrix}, CX = \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{bmatrix}, \quad (1)$$

where H and U are *single-qubit gates*, and CX is a *two-qubit gate*. Moreover, U is a *programmable gate* with three parameters. We can tune the parameters to instantiate U to specific gates, e.g., U with $\theta = \pi/2$, $\lambda = \pi$, and $\phi = 0$ is just H. In our example, u_1 to u_4 are instances of U, each black dot is the first qubit for a CX, and each $\oplus$ is the second qubit for a CX. The logic constraint of QC hardware is specified as a *native gate set*, e.g., on IBM quantum computers, the set is $\{U, CX\}$ [20]. Gates not contained in this set, e.g., two-qubit gates other than CX, or gates on three or more qubits, have to be decomposed into a series of gates in the set to be executed on hardware. Fortunately, most near-term quantum algorithms are just written in single-qubit and two-qubit gates [11, 15, 25], and there are canonical decompositions of an arbitrary two-qubit gate into Us and CXs [31, 43] as displayed in Fig. 2. Other *important* multi-qubit gates in QC also have efficient decompositions [5].

An *entangling two-qubit gate* like CX is essential for QC. However, they are not available for all pairs of qubits. The layout constraints of a quantum processor are specified by a *coupling graph* $G = (P, E)$ like Fig. 1c where each vertex is a

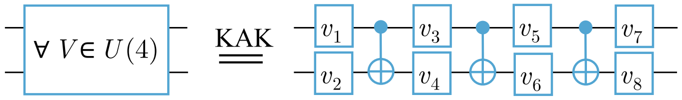


Fig. 2 KAK decomposition. For any V that is a 4-by-4 unitary matrix, the corresponding two-qubit gate can be decomposed into 3 CXs and 8 single-qubit gates

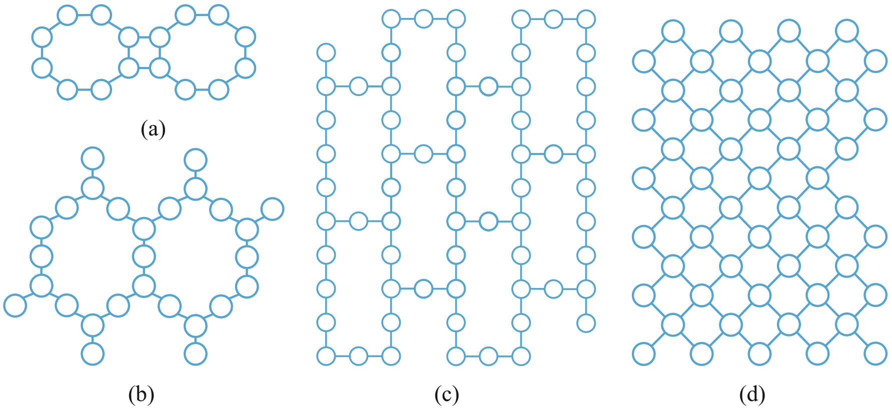


Fig. 3 Coupling graphs of some existing quantum processors. (**a**) Rigetti Aspen-4 [12]. (**b**) IBM Falcon [23]. (**c**) IBM Hummingbird [20]. (**d**) Google Sycamore [3]

physical qubit, and each edge means two-qubit gates can be applied to those two vertices. Thus, we need to map the program qubits in Fig. 1b to physical qubits in Fig. 1c so that the CX gates are on adjacent physical qubits. This is not always possible. In fact, the gates g_1, g_6, and g_9 fully connect the program qubits q_1, q_2, and q_5. No matter how we map the qubits, satisfying these three gates requires a triangle on the coupling graph, which does not exist in Fig. 1c. A way to resolve this issue is by inserting SWAP gates that changes the mapping in the middle of the circuit. However, SWAP gates bring error and may also increase the runtime, so a compiler needs to carefully decide when and where to insert them. To summarize, in the *layout synthesis* i.e., *qubit mapping* phase of compilation, program qubits are mapped to physical qubits, some circuit transformations, e.g., via SWAP gates, are performed, and all the gates are scheduled.

We believe that qubit mapping poses a more serious challenge than gate decomposition. This is because the native gates are decided by fundamental physics of the QC platform, but the coupling graphs have more degrees of design freedom. In fact, the native gate set of IBM quantum computers has not changed greatly since the beginning of their cloud QC service because the fundamental qubit technology is the same, but QC devices with very different coupling graphs have been introduced in the past a few years [20], e.g., Fig. 3b and c. With better understanding of the qubits, new coupling graphs are also being proposed and selected [18].

2 Previous Works

NP-completeness of several versions of the layout synthesis problem has been proven [7, 27, 37, 41]. Therefore, we can categorize previous works into heuristic ones and exact/optimal ones. The runtime of the latter scales exponentially in problem size because of the computational complexity of the problem.

In general, the heuristic works formulate layout synthesis as a search problem [2, 10, 26, 27, 37–39, 46–48]. In the search algorithms, the state is ($\mathcal{G}$, Π) where $\mathcal{G}$ contains the gates that have been considered and Π is the current qubit mapping; the action leading to another state is either changing Π by SWAP(s) or appending some gates into $\mathcal{G}$ if they only act on adjacent qubit(s) under the current mapping; the cost of a Π-change is often evaluated by looking ahead a few more steps in the search tree. At the beginning of the search, $\mathcal{G}$ is just empty and there are a few different ways to find the initial mapping Π_0. References [27, 37, 38] use the earlier two-qubit gates to construct a program graph between program qubits and apply existing graph isomorphism algorithms from this program graph to the coupling graph. In [37], the program graph is additionally weighted by the number of two-qubit gates between this qubit pair. References [2, 10, 39, 46–48] start the search with some Π_0 and expand the search tree a few times. Then, they select the best mapping so far and use it as the real initial mapping. References [26] searches for the final mapping of the reversed program and uses it as Π_0 for the original program.

In theory, one can derive the optimal solution by fully expanding the search tree in the heuristic search approaches, but, in practice, the exact/optimal approaches formulate the layout synthesis into mathematical programming and apply a solver: [28, 40, 42, 45] use satisfiability modulo theories (SMT) solvers, [6, 29, 35] use integer programming (IP) solvers, and [44] uses temporal planners. To reduce runtime, many works compromise by slicing the program and only considering the next slice when inserting SWAPs [6, 28, 29, 35].

3 The Measure-Improve Methodology and Its Application in Classical Circuit Placement

Given the extensive amount of work on layout synthesis, a natural question is if these solutions are close to optimal. However, it remains challenging how to measure their optimality.

The layout synthesis problem summarized above is representative of many problems in design automation or, broadly, in computer science: the complexity is NP-hard, and they can be formulated into some kind of mathematical programming and solved with exponential runtime. To solve large instances of these problems, one approach is to accelerate the solver, often in a domain-specific way. Another approach is to develop heuristic methods that run faster but are not optimal. How do we evaluate these heuristics? A common way is using a set of representative applications as the benchmark and comparing the results by different heuristics. However, because of the complexity of the problem, we do not know the optimal result of these benchmarks, so we do not know how much room of improvement there is. If, after substantial research, the improvements are diminishing, the community may be in a dilemma: is it possible that the current heuristics are very close to optimal and further research will produce diminishing returns; or is there still significant room requiring fresh ideas and more efforts? We cannot be certain about both possibilities since deriving optimal solution for large instances takes astronomical time. In this case, it would be helpful if there are benchmarks with known optimal solution and the size of these benchmarks should be large enough to imitate real applications. With such benchmarks, we can measure the sub-optimality of the heuristics and improve them if there is still significant room.

One such benchmark set in classical circuit design is PEKO [9], *placement examples with known optimal*. Before placement, the circuit is represented as connected *modules* shown as C_i, $i \in [4]$, in Fig. 4a. The input of the problem is thus a *netlist* where each *net* connects two or more pins on different modules. In our example, there is a 2-pin net connecting C_1 and C_4, and a 4-pin net connecting all the modules. After placing the modules on the chip area, manufacturers need to implement the nets with wires, as shown in Fig. 4b. This example is not ideal since the total wirelength can be reduced if we place the modules closer together, e.g., by putting the modules at the four cells in the bottom right corner. Despite that placement is known to be NP-complete [33], the authors of [9] present a way of constructing placement examples with known optimal wirelength from locally optimal nets, as demonstrated in Fig. 4c. That is, one follows the net size distribution specification. For each net of size r, one connects pins from $\lceil\sqrt{r}\rceil \times \lceil\sqrt{r}\rceil$ adjacent modules. Since all the nets are among adjacent modules, the total wirelength cannot

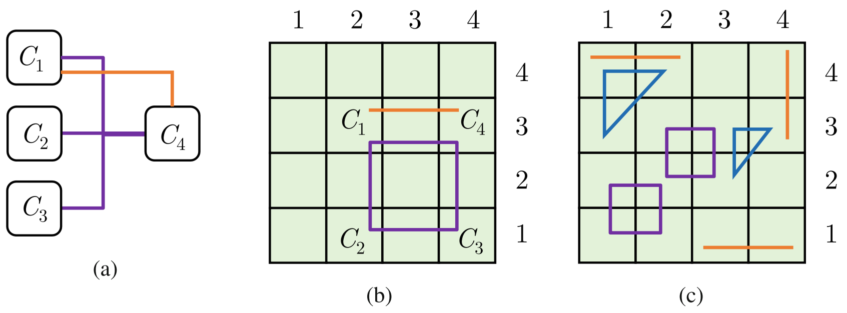
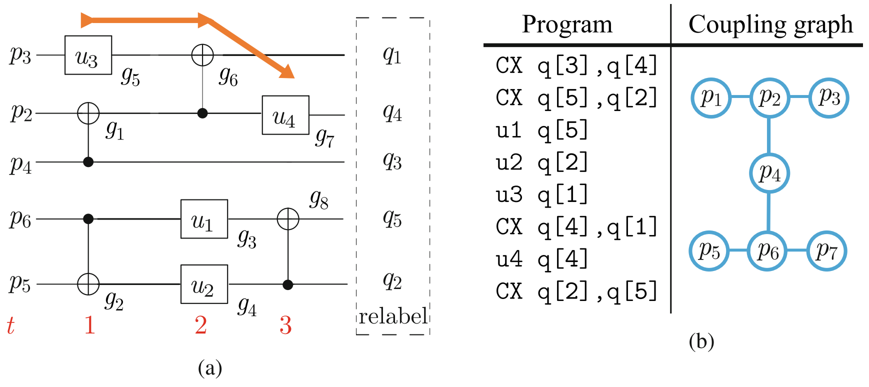


Fig. 4 The placement problem in classical circuit design. (**a**) Netlist, input of the placement problem. Each net connects some pins on modules $C_i, i \in [4]$. There is a 2-pin net (orange) between C_1 and C_4, and a 4-pin net (purple) connecting all modules. (**b**) A placement solution. The modules are placed to cells in the chip area. This solution is not ideal since the modules are placed far away, so longer wires are needed by the nets. (**c**) Construction of PEKO, placement example with known optimal [9]. The nets are all shortest possible, so the shortest wire length of the whole netlist is just the sum of each one



```
CX q[3],q[4]
CX q[5],q[2]
u1 q[5]
u2 q[2]
u3 q[1]
CX q[4],q[1]
u4 q[4]
CX q[2],q[5]
```

Fig. 5 Quantum mapping example with known optimal (QUEKO) [41]. (**a**) Constructing a program with optimal depth 3 from a physically realizable circuit. The orange arrow is a backbone of length 3: (g_5, g_6, g_7). The program is derived by relabeling the wires to program qubits. (**b**) A QUEKO benchmark consisting of a program $\mathcal{P}$ and a coupling graph G. The optimal depth T_O of mapping $\mathcal{P}$ to G is known by construction to be 3. The minimal SWAP count is 0

be reduced. Thus, we know the optimal wirelengths of PEKO benchmarks by construction. The PEKO benchmarks were used to measure optimality of leading placers at that time and showed 2X optimality gap, which spurred the community to invest more efforts into the placement problem. This led to a wirelength reduction equivalent to two generations of hardware scaling in Moore's law [34].

4 Measuring Optimality with QUEKO

To measure the optimality of existing layout synthesis solutions, we developed QUEKO [41]—quantum mapping examples with known optimal, inspired by PEKO.

The depth of a quantum circuit is the total number of time steps it takes to execute the circuit on a specific architecture. In a circuit prior to layout synthesis, e.g., Fig. 1b, the gates are not scheduled, but we can derive some lower bounds of its depth. For instance, there are three gates g_2, g_3, and g_8 subsequently acting on q_3. Since none of them can be executed at the same time, we need at least three time steps to run these gates. Moreover, we cannot change the order of execution, since g_3 depends on g_2, and g_8 depends on g_3. The maximal length of *dependency chains* like (g_2, g_3, g_8) is a lower bound for depth. Of course, it is not always possible to reach this lower bound because we may need to insert SWAPs in layout synthesis which could lengthen the dependency chains. This is indeed the case for the example of Fig. 1, since SWAP insertion is absolutely necessary, as we have explained in Sect. 1.

Table 1 Optimality gaps[a] measured by QUEKO benchmarks of feasible depths[b]

	Small architecture[d] and sparse program[e]				Large architecture and dense program			
Compiler[c]	10	20	30	40	10	20	30	40
JKU [48]	13X	11X	9.3X	8.7X	Process runs out of RAM (128GB)			
Cirq [10]	8.3X	9.3X	7.9X	9.2X	44X	45X	48X	47X
Qiskit [2]	5.4X	4.7X	4.8X	4.6X	12X	12X	11X	11X
t|ket⟩ [38]	1.04X	1.32X	1.30X	1.72X	1.82X	3.1X	3.7X	5.7X

[a] Optimality gap is defined as T/T_O where T_O is the optimal depth and T is the depth of the result produced by a compiler

[b] For each $T_O = 10$, 20, 30, and 40, we generated 10 QUEKO benchmarks and input them to the compilers being examined. The data shown above are geometric means of the 10 corresponding optimality gaps

[c] We used the `Greedy router` in Cirq 0.6.0, `DenseLayout` followed by `StochasticSwap` in Qiskit 0.14.1, `Graph Placement` followed by `Route` in t|ket⟩ 0.4.1

[d] The "small architecture" is Rigetti Aspen-4 (Fig. 3a) and the big one is Google Sycamore (Fig. 3d)

[e] The "sparse program" has the gate density of the Toffoli gate, and the dense one has that of the quantum supremacy experiment [3]

For physically realizable quantum circuits where all the two-qubit gates act on adjacent qubits, the maximal length of dependency chains is indeed the depth, not merely a lower bound. For instance, in Fig. 5a, one of the longest dependency chains is (g_5, g_6, g_7), which means the depth cannot be even lower than 3. We can construct a qubit mapping example with known optimal depth T_O by generating a physically realizable circuit with a *backbone* like (g_5, g_6, g_7) which is a dependency chain of length T_O. Building the backbone does not take too many gates, so we have another degree of freedom in QUEKO named *gate density*: how much of the spacetime is taken by idleness, single-qubit gates, and two-qubit gates. In Fig. 5a, there are three time steps and five qubits, so the spacetime volume of the circuit is 15. Each single-qubit gate takes one unit of volume, and each two-qubit gate takes two units. In Fig. 5a, the single-qubit gates (g_5, g_3, g_4, and g_7) take 4 out of 15 units of volume; the two-qubit gates (g_1, g_2, g_6, and g_8) take 8/15 volume; the reset 3/15 volume is idleness. Thus, the gate density for this circuit is (3/15, 4/15, 8/15). We relabel the physical qubits to program qubits to derive the quantum program that can be input to the compilers, as shown in the dashed box in Fig. 5a, e.g., p_3 is relabeled as q_1, so the single-qubit gate u_3 is on q_1 in Fig. 5b. If a compiler finds out the inverse of our relabeling, it will derive the original physically realizable circuit exactly.

In summary, given a coupling graph G, a depth T_O, and a gate density vector, we can construct a quantum program $\mathcal{P}$ with the known optimal depth T_O if mapped to G, as the demonstrated in Fig. 5b. We can pass the layout synthesis problem $(\mathcal{P}, G)$ to a compiler and check how far is the depth of its result, T, compared to the optimal depth T_O. The backbones in QUEKO benchmarks guarantee that T cannot be smaller than T_O. We can easily derive a depth-optimal solution by reading off the relabeling during QUEKO construction, e.g., for the problem in Fig. 5b, an optimal solution is $q_1 \mapsto p_3$, $q_4 \mapsto p_2$, $q_3 \mapsto p_4$, $q_5 \mapsto p_6$, and $q_2 \mapsto p_5$ in Fig. 5a. Since this solution does not contain any SWAPs, QUEKO benchmarks also have known optimal number of SWAPs, which is 0.

With the help of QUEKO, we find that, despite over a decade long research in layout synthesis, the optimality gaps are still large, as shown in Table 1. The depth of results by compilers other than t|ket⟩ is at least 4.6X the optimal depth. The rightmost column corresponds to QUEKO benchmarks that have the coupling graph, optimal depth, and gate density similar to the quantum supremacy experiment [3], which means that quantum circuits of this size are feasible. On these benchmarks, even the t|ket⟩ result is 5.7X the optimal. These gaps indicate that there is still substantial room for improvements in layout synthesis.

5 SMT Formulation of the Optimal Layout Synthesis Problem

In this section, we would like to present a precise formulation of the layout synthesis problem based on a mathematical programming formulation, which can be solved later by using a satisfiable modulo theories (SMT) solver. We would like to reiterate a few assumptions. (1) The qubit mapping is from program qubits to physical qubits. It is not from basic entities in other problems, like fermions in chemistry simulation [25], to program qubits. (2) Conventionally, the logic constraints have been resolved before layout synthesis, so all the input gates can be executed on hardware; gate cancellation has also been done, so every input gate must be executed. (3) The architecture/coupling graph is fixed. This is true for existing general-purpose quantum processors [17, 19, 20, 22, 24]. However, the formulation can be extended to processors with programmable architectures [8], e.g., using neutral atom arrays [14].

5.1 Variables

A solution of the layout synthesis problem has three sets of variables: *mapping variables*, *schedule variables*, and *SWAP variables*. We shall demonstrate these variables in Fig. 6 which is a solution to the problem instance displayed in Fig. 1. At the end of the day, what we need to inform the hardware is when and where to execute each gate. This is shown as the circuit diagram in Fig. 6. Different from the diagram in Fig. 1b, in this diagram, each wire is a physical qubit, and the gates on the same column are executed at the same time step. On the left of Fig. 6, the physical qubits involved are colored black in the coupling graph.

The value of a mapping variable $\pi_{q,t}$ is the physical qubit where q is mapped at time t. For example, in the beginning q_5 is mapped to p_3, so $\pi_{q_5,1} = p_3$. In Fig. 6, we annotated the program qubit on the wire where it is mapped.

The schedule variable t_g of each gate in the program is just the time step when it is executed, annotated below in Fig. 6, e.g., $t_{g_5} = 1$ and $t_{g_6} = 2$.

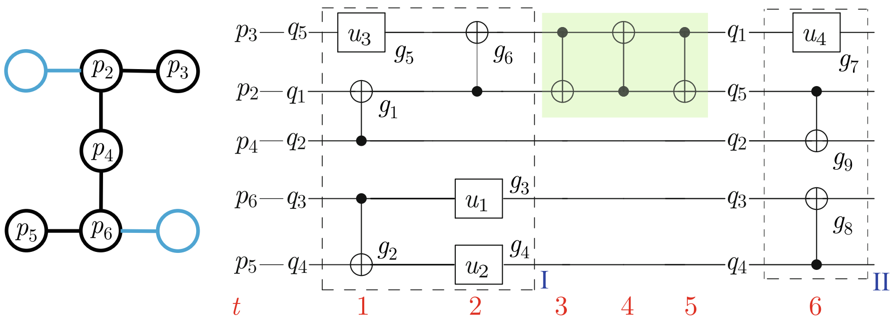


Fig. 6 A valid solution of the layout synthesis problem. On the left, the physical qubits involved are colored black in the coupling graph. On the right, each wire is a physical qubit. The time is shown below. Vertically aligned gates are executed at the same time step. The three green-shaded CX gates consist of a SWAP gate. The initial qubit mapping is shown before step 1, and the mapping after the SWAP is shown before step 6. The circuit can also be seen as two coarse-grain time steps (dashed boxes I and II) separated by a transition consisting of a SWAP on p_2 and p_3.

The SWAP variables $\sigma_{e,t}$ are binary variables that evaluate to 1 if and only if there is a SWAP gate finishing at time t on edge e. There is only one SWAP in Fig. 6, so the only non-zero SWAP variable is $\sigma_{(p_2,p_3),5}$.

We can set any function of the above variables as the objective. Three common objectives are straightforward: *depth*, the number of SWAPs, and *fidelity*. The depth of a quantum circuit is the maximum of all the schedule variables. The number of SWAP gates inserted is the sum of all the SWAP variables. The fidelity of the circuit, assuming a stochastic error model, is the product of all the individual gate fidelity, which can be input as the weights of vertices and edges in the coupling graph. There are four multiplicative terms in the fidelity expression: single-qubit gate fidelity, two-qubit gate fidelity, measurement fidelity, and SWAP fidelity.

5.2 Constraints

There has to be many constraints on the variable assignments in order for a valid solution. We can categorize them into four groups: connections, dependencies, no overlaps, and mapping transformations.

Connections Supposed that a two-qubit gate g at time t_g acts on program qubits q and q'. Then, there should be an edge between π_{q,t_g} and π_{q',t_g}. Otherwise, the physical qubits are not adjacent on the coupling graph and the two-qubit gate cannot be executed. For instance, for gate g_1 acting on q_1 and q_2,

$$t_{g_1} == 1 \;\Rightarrow\; (\pi_{q_1,1},\; \pi_{q_2,1}) \in E, \tag{2}$$

where E is the edge set of the coupling graph.

Dependencies We have introduced the notion of dependency in Sec 4. In general, if two gates subsequently act on the same qubit, the order between them in the input program must be respected, e.g., since g_6 acts on q_5 after g_5,

$$t_{g_6} > t_{g_5}. \tag{3}$$

Note that with domain knowledge, we may relax some of these constraints, as specified later in Sect. 6.2.

No Overlaps Each qubit at each time step can only be involved in one gate, so we need to make sure that there are no overlaps between any gates including the SWAPs we inserted. For instance, $\sigma_{(p_2,p_3),6}$ cannot be 1 since, if so, the SWAP would finish on (p_2, p_3) at time 6 and overlap with g_7. This case is ruled out by constraints like

$$t_{g_7} == 6 \ \wedge \ \big(\pi_{q_1,6} == p_2 \ \vee \ \pi_{q_1,6} == p_3\big) \ \Rightarrow \ \sigma_{(p_2,p_3),6} == 0. \tag{4}$$

Mapping Transformations After a SWAP gate finishes, the qubit mapping should be transformed, e.g., the mapping of q_1 and q_5 exchanged at time 6, i.e.,

$$\pi_{q_5,5} == p_3 \wedge \pi_{q_1,5} == p_2 \wedge \sigma_{(p_2,p_3),5} == 1 \ \Rightarrow \ \pi_{q_1,6} == p_3, \ \pi_{q_5,6} == p_2. \tag{5}$$

On the other hand, if a qubit is not involved in any SWAP gates finishing at time t, its mapping variable should remain the same as time $t-1$, e.g., for q_3 and $t=3$,

$$\pi_{q_3,2} == p_6 \ \wedge \ \sigma_{(p_4,p_6),2} == 0 \ \wedge \ \sigma_{(p_6,p_5),2} == 0 \ \Rightarrow \ \pi_{q_3,3} == p_6. \tag{6}$$

We took a descriptive approach in this section, interested readers can refer to [40] for details. In total, there are $O(NTL)$ constraints where N is the number of physical qubits, T is the total depth, and L is the total number of gates.

6 Closing the Gap with OLSQ—Optimal Layout Synthesis for Quantum Computing

Upon the revelation of optimality gaps by QUEKO, we set out to close these gaps. As a result, we have formulated layout synthesis problem in Sect. 5. The variables and constraints are compatible with a kind of mathematical programming model named SMT, satisfiability modulo theories. Thus, we use an existing SMT solver, z3 [13], to derive the layout synthesis solutions optimally.

In terms of formulation, the main contribution of OLSQ is reducing the number of variables. In previous works like [45], there is a binary variable $x_{\Pi,t}$ at each time step t for a possible qubit mapping $\Pi : Q \to P$. For instance, the initial mapping in Fig. 6 is Π_0: $q_1 \mapsto p_2, q_2 \mapsto p_4, q_3 \mapsto p_6, q_4 \mapsto p_5$, and $q_5 \mapsto p_3$. Thus, $x_{\Pi_0,0} = 1$ and $x_{\Pi,0} = 0$ for any $\Pi \neq \Pi_0$. Note that there are exponentially many possible mappings with respect to the number of qubits, so there are exponentially many variables. In contrast, as Sect. 5.1 has demonstrated, OLSQ captures the mapping with $\pi_{q,t}$ variables and updates the mapping with the SWAP variables $\sigma_{e,t}$. For each time step, there are as many mapping variables as program qubits, *not* as many as possible qubit mappings. The number of SWAP variables is just the number of edges in the coupling graph, which is also linear in the number of physical qubits. Overall, OLSQ only has linearly many variables in the number of qubits, which is a huge improvement over previous works.

We implemented OLSQ in Python 3 and open-sourced the package[1] with BSD-3-Clause license. A user initializes `OLSQ` by choosing whether to use the transition mode, and the objective: depth, the number of SWAPs, or fidelity. Afterwards, the user inputs information about the hardware with `setdevice()`: the number of physical qubits, the edges, how many time steps a SWAP takes. If using fidelity as the objective, the fidelity of individual gates should be input at this stage. Then, the user should input the quantum program with `setprogram()`. Under the assumptions we introduced in the beginning of Sect. 5, OLSQ only needs a list of tuples to represent the program, each tuple for a gate. If it is a single-qubit gate, then the tuple only has one element, which is the index of the involved program qubit; if it is a two-qubit gate, then there are two elements in the tuple. Finally, the user can execute the `solve()` method to acquire the solution. The depth that OLSQ is currently trying will be printed on the screen while it calls z3 solver [13] to solve the SMT model corresponding to this depth. If there is a solution, the quantum circuit after layout synthesis would be returned; otherwise, OLSQ increases the depth and try again.

6.1 *Speeding Up OLSQ with the Transition Mode*

Because of the more efficient formulation, OLSQ shows better scaling in runtime compared to previous work with exact formulation [40]. However, because of the complexity of the problem, the runtimes are still long. Thus, we consider techniques that can accelerate the solving process with some sacrifice on optimality, which leads to the *transition mode*.

In Fig. 6, there are 6 time steps, but the mapping only changes once. So, many mapping variables, e.g., the ones for time steps 2 to 5, take the same value in the previous time step. We would have much less variables if we only keep the mapping

[1] https://github.com/UCLA-VAST/OLSQ.

variables when the mapping changes. In another perspective, this can be seen as if each gate has a "coarse-grain" schedule variable. Between the gates scheduled to coarse-grain time t and $t+1$, there is a transition, which is a set of non-overlapping SWAPs. In Fig. 6, the two dashed boxes are two coarse-grain time steps I and II. The transition between them consists of the SWAP on (p_2, p_3).

To implement the transition mode, we just need to revise the OLSQ formulation by: 1) relaxing the $>$ in dependency constraints like Eq. 3 to $\geq$, 2) changing the duration of SWAP gates to 1, and 3) removing overlap constraints like Eq. 4 since in the transition-based model, the SWAP gates are part of transitions and will not interfere with the other gates. Note that the solver decides which gates go into which coarse-grain time step, so the transition-based (TB-)OLSQ is different from cutting the circuit beforehand and solving the sub-circuits separately. Thus, the results by TB-OLSQ are still much better than heuristics, as shown in Table 2, while achieving over 400X speedup compared to the original OLSQ on the benchmarks used in [40]. This is because the number of transition, $\tilde{T}$, is often much smaller than the circuit depth, T, so the number of variables $O(N\tilde{T}+L)$ and the number of constraints $O(\tilde{T}NL)$ also become much smaller than those of the original OLSQ.

6.2 Exploring Larger Solution Space

Apart from acceleration, we can also improve the solution quality given domain knowledge. One example of this is dropping some dependency constraints. In general, we cannot neglect these constraints for the correctness of results. However, for an important family of circuits named quantum approximate approximation algorithm, QAOA [15], there are many dependency constraints that can be dropped without consequences, because the ZZ gates inside QAOA are commutable, as illustrated in Fig. 7. This is the key factor of the improvements we see in Table 2: even there is a little sacrifice of optimality by using the transition mode, the depths and the numbers of SWAPs are at least halved compared to leading heuristics.

Table 2 Layout synthesis results of QAOA programs for 3-regular graphs[a]

QAOA size	8		10		12		14	
Compiler[b]	Depth	#SWAP	Depth	#SWAP	Depth	#SWAP	Depth	#SWAP
t|ket⟩ [38]	13	6	15	7	20	13	22	14
SABRE [26]	11	5	15	9	13	9	20	11
TB-OLSQ [40]	5	3	6	5	6	5	7	4
OLSQ-GA [42]	4	0	3	0	4	0	4	0

[a] The whole setting is from [4], a leading experimental QAOA work: the QAOA benchmarks are generated from random 3-regular graphs of size 8, 10, 12, and 14, and the coupling graph is part of Google Sycamore. We only use one iteration of QAOA

[b] We used `pytket-cirq` 0.16.0, the Cirq integration of t|ket⟩; SABRE as integrated in Qiskit 0.27.0

$$
\begin{aligned}
ZZ(\gamma) &= \mathrm{diag}(e^{-i\gamma},\ e^{i\gamma},\ e^{i\gamma},\ e^{-i\gamma}) \\
ZZ(\gamma)\otimes I &= \mathrm{diag}(e^{-i\gamma},\ e^{-i\gamma},\ e^{i\gamma},\ e^{i\gamma}, \\
&\qquad e^{i\gamma},\ e^{i\gamma},\ e^{-i\gamma},\ e^{-i\gamma}) \\
I\otimes ZZ(\gamma) &= \mathrm{diag}(e^{-i\gamma},\ e^{i\gamma},\ e^{i\gamma},\ e^{-i\gamma}, \\
&\qquad e^{-i\gamma},\ e^{i\gamma},\ e^{i\gamma},\ e^{-i\gamma})
\end{aligned}
$$

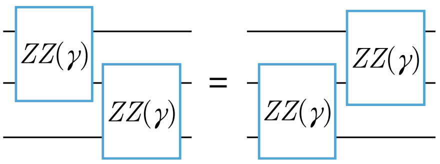


Fig. 7 Commutation of ZZ gates. $ZZ(\gamma)\otimes I$ means applying $ZZ(\gamma)$ gate on the upper two qubits while doing nothing on the bottom qubit. Since $ZZ(\gamma)\otimes I$ and $I\otimes ZZ(\gamma)$ are both diagonal, they are commutable

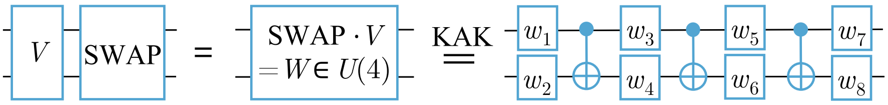


Fig. 8 A SWAP gate absorbed by gate V. In the first step, we compute the matrix product W of SWAP and V; in the second step, we apply KAK decomposition to W

Another example of exploring larger solution space is the technique of gate absorption, resulting in OLSQ-GA [42] where we combine layout synthesis with the synthesis of programmable two-qubit gates using KAK decomposition previously shown in Fig. 2. If a SWAP gate is directly after another two-qubit gate, e.g., in a QAOA circuit or other important circuits for chemistry [25] or machine learning [11], we can combine these two gates by computing the matrix product of them and synthesis this product, as illustrated by Fig. 8. The cost of implementing the gate induced by the product is much less than the cost of implementing the original two gates separately. The "absorption" of SWAP into other gates can be formulated by a set of absorbed SWAP variables $\alpha_{e,t}$ that behaves similarly to $\sigma_{e,t}$ variables, except bundled to other gates by constraints like

$$\alpha_{(p,p'),t} == 1 \quad \Rightarrow \quad \exists g \ \text{ s.t. } \ t_g == t \ \wedge \ \pi_{q,t} == p \ \wedge \ \pi_{q',t} == p', \qquad (7)$$

where two-qubit gate g acts on program qubit q and q'. The results of OLSQ-GA are displayed in the bottom row of Table 2. Compared to TB-OLSQ, the depths decrease further, and more surprisingly, all the SWAPs are absorbed for this set of QAOA benchmarks, so there are no explicit SWAPs in the OLSQ-GA solutions.

7 Conclusion and Future Directions

In this chapter, we apply the measure-improve methodology, which has been successful in classical circuit placement, to the layout synthesis problem which is the center of compilation for near-term QC. We reveal quite large optimality gaps using QUEKO and aim to close these gaps by more efficient formulation, OLSQ, and more customized formulation OLSA-GA.

As future directions, (1) on the “measure” end, we plan to construct optimality benchmarks that are more similar to real QC applications; (2) on the “improve” end, we plan to accelerate the solving further by a top-down decomposition of problem instances, or a bottom-up approach using pre-computed optimal circuit library; 3) with an almost optimal compiler that has feasible runtime, we can evaluate quantum architecture designs without possible bias of the heuristics in compilation.

Acknowledgments This work is partially supported by the Center for Domain-Specific Computing Industrial Partnership Program.

References

1. C.G. Almudever et al., The engineering challenges in quantum computing, in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. (IEEE, Lausanne, Switzerland, 2017), pp. 836–845. https://doi.org/10.23919/DATE.2017.7927104
2. M.S. Anis et al., Qiskit: An open-source framework for quantum computing (2021). [Online]. Available: https://doi.org/10.5281/zenodo.2573505
3. F. Arute et al., Quantum supremacy using a programmable superconducting processor. Nature **574**(7779), 505–510 (2019). arXiv:quant-ph/1910.11333. https://doi.org/10.1038/s41586-019-1666-5
4. F. Arute et al., Quantum approximate optimization of non-planar graph problems on a planar superconducting processor. Nature Physics **17**(3), 332–336 (2021). arXiv:quant-ph/2004.04197
5. A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, H. Weinfurter, Elementary gates for quantum computation. Phys. Rev. A **52**(5), 3457–3467 (1995). https://doi.org/10.1103/PhysRevA.52.3457
6. D. Bhattacharjee, A.A. Saki, M. Alam, A. Chattopadhyay, S. Ghosh, MUQUT: Multi-constraint quantum circuit mapping on NISQ computers: Invited paper, in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (IEEE, Westminster, CO, USA, 2019), pp. 1–7, https://doi.org/10.1109/ICCAD45719.2019.8942132
7. A. Botea, A. Kishimoto, R. Marinescu, On the complexity of quantum circuit compilation, in *Proceedings of the 11th Annual Symposium on Combinatorial Search* (AAAI Press, 2018), p. 5
8. S. Brandhofer, H.P. Büchler, I. Polian, Optimal mapping for near-term quantum architectures based on Rydberg atoms, in *2021 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2021. arXiv:quant-ph/2109.04179
9. C.-C. Chang, J. Cong, M. Romesis, M. Xie, Optimality and scalability study of existing placement algorithms. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **23**(4), 537–549 (2004). https://doi.org/10.1109/TCAD.2004.825870
10. Cirq Developers, Cirq (2021, Aug.). See full list of authors on GitHub: https://github.com/quantumlib/Cirq/graphs/contributors. [Online]. Available: https://doi.org/10.5281/zenodo.5182845
11. I. Cong, S. Choi, M.D. Lukin, Quantum convolutional neural networks. Nature Physics **15**(12), 1273–1278 (2019). arXiv:quant-ph/1810.03787. https://doi.org/10.1038/s41567-019-0648-8
12. A. Cornelissen, J. Bausch, A. Gilyén, Scalable benchmarks for gate- based quantum computers (2021). arXiv:quant-ph/2104.10698
13. L. de Moura, N. Bjørner, Z3: An efficient SMT solver, in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, ed. by C.R. Ramakrishnan, J. Rehof (Springer, Berlin, Heidelberg, 2008), pp. 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

14. S. Ebadi et al., Quantum phases of matter on a 256-atom programmable quantum simulator. Nature **595**(7866), 227–232 (2021). https://doi.org/10.1038/s41586-021-03582-4
15. E. Farhi, J. Goldstone, S. Gutmann, A quantum approximate optimization algorithm (2014). arXiv:quant-ph/1411.4028
16. A.G. Fowler, M. Mariantoni, J.M. Martinis, A.N. Cleland, Surface codes: Towards practical large-scale quantum computation. Phys. Rev. A **86**(3), 032324 (2012). arXiv:quant-ph/1208.0928. https://doi.org/10.1103/PhysRevA.86.032324
17. Google Quantum AI, Quantum computer datasheet (2021). [Online]. Available: https://quantumai.google/hardware/datasheet/weber.pdf
18. J.B. Hertzberg, R.O. Topaloglu, Quantum circuit topology selection based on frequency collisions between qubits. US Patent US20 200 401 925A1 (2020). [Online]. Available: https://patents.google.com/patent/US20200401925A1/en/
19. Honeywell, Honeywell sets new record for quantum computing performance (2020). [Online]. Available: https://www.honeywell.com/us/en/news/2021/03/honeywell-sets-new-record-for-quantum-computing-performance
20. IBM Quantum Processor, [Online]. Available: https://quantum-computing.ibm.com/services/docs/services/manage/systems/processors
21. IBM, 5 things to know about the IBM roadmap to scaling quantum technology (2020). [Online]. Available: https://newsroom.ibm.com/5-Things-About-IBM-Roadmap-to-Scale-Quantum-Technology
22. IONQ, Ionq (2020). [Online]. Available: https://ionq.com/technology
23. P. Jurcevic et al., Demonstration of quantum volume 64 on a superconducting quantum computing system. Quantum Sci. Technol. **6**(2), 025020 (2021). https://doi.org/10.1088/2058-9565/abe519
24. P.J. Karalekas, N.A. Tezak, E.C. Peterson, C.A. Ryan, M.P. da Silva, R.S. Smith, A quantum-classical cloud platform optimized for variational hybrid algorithms. Quantum Sci. Technol. **5**(2), 024003 (2020). arXiv:quant-ph/2001.04449. https://doi.org/10.1088/2058-9565/ab7559
25. I.D. Kivlichan, J. McClean, N. Wiebe, C. Gidney, A. Aspuru-Guzik, G.K.-L. Chan, R. Babbush, Quantum simulation of electronic structure with linear depth and connectivity. Phys. Rev. Lett. **120**(11), 110501 (2018). arXiv:quant-ph/1711.04789. https://doi.org/10.1103/PhysRevLett.120.110501
26. G. Li, Y. Ding, Y. Xie, Tackling the qubit mapping problem for NISQ-era quantum devices, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '19*. (ACM Press, Providence, RI, USA, 2019), pp. 1001–1014. arXiv:cs.ET/1809.02573. https://doi.org/10.1145/3297858.3304023
27. D. Maslov, S.M. Falconer, M. Mosca, Quantum circuit placement. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **27**(4), 752–763 (2008). arXiv:2002.09783. https://doi.org/10.1109/TCAD.2008.917562
28. P. Murali, N.M. Linke, M. Martonosi, A.J. Abhari, N.H. Nguyen, C.H. Alderete, Full-stack, real-system quantum computer studies: Architectural comparisons and design insights, in *Proceedings of the 46th International Symposium on Computer Architecture - ISCA '19* (ACM Press, Phoenix, Arizona, 2019), pp. 527–540. arXiv:quant-ph/1905.11349. https://doi.org/10.1145/3307650.3322273
29. G. Nannicini, L.S. Bishop, O. Gunluk, P. Jurcevic, Optimal qubit assignment and routing via integer programming. arXiv:quant-ph/2106.06446
30. H. Neven, Keynote in Google Quantum Summer Symposium (2020). [Online]. Available: https://youtu.be/HgQOPhNCct0
31. E.C. Peterson, G.E. Crooks, R.S. Smith, Fixed-depth two-qubit circuits and the monodromy polytope. Quantum **4**, 247 (2020). arXiv:1904.10541. https://doi.org/10.22331/q-2020-03-26-247
32. J. Preskill, Quantum computing in the NISQ era and beyond. Quantum **2**, 79 (2018). arXiv:quant-ph/1801.00862. https://doi.org/10.22331/q-2018-08-06-79

33. S. Sahni, A. Bhatt, The complexity of design automation problems, in *Proceedings of the 17th Design Automation Conference*, ser. DAC '80 (Association for Computing Machinery, New York, NY, USA, 1980), pp. 402–411. https://doi.org/10.1145/800139.804562
34. Semiconductors Research Corporation, 'Huge opportunity' in IC design optimization gained by Semiconductor Research Corporation (2007). National Science Foundation: CAD innovation could save industry billions. [Online]. Available: https://www.src.org/newsroom/press-release/2007/41/
35. A. Shafaei, M. Saeedi, M. Pedram, Qubit placement to minimize communication overhead in 2D quantum architectures, in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)* (IEEE, Singapore, 2014), pp. 495–500. https://doi.org/10.1109/ASPDAC.2014.6742940
36. P.W. Shor, Scheme for reducing decoherence in quantum computer memory. Phys. Rev. A **52**(4), R2493–R2496 (1995). https://doi.org/10.1103/PhysRevA.52.R2493
37. M.Y. Siraichi, V.F. dos Santos, S. Collange, F.M.Q. Pereira, Qubit allocation, in *Proceedings of the 2018 International Symposium on Code Generation and Optimization - CGO 2018* (ACM Press, Vienna, Austria, 2018), pp. 113–125. https://doi.org/10.1145/3168822
38. S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, R. Duncan, t|ket⟩: A retargetable compiler for NISQ devices. Quantum Sci. Technol. (2020). arXiv:quant-ph/2003.10611. https://doi.org/10.1088/2058-9565/ab8e92
39. R.S. Smith, E.C. Peterson, M.G. Skilbeck, E.J. Davis, An open-source, industrial-strength optimizing compiler for quantum programs. Quantum Sci. Technol. **5**(4), 044001 (2020). https://doi.org/10.1088/2058-9565/ab9acb
40. B. Tan, J. Cong, Optimal layout synthesis for quantum computing, in *2020 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, ser. ICCAD '20 (Association for Computing Machinery, Virtual Event, USA, 2020). arXiv:quant-ph/2007.15671. https://doi.org/10.1145/3400302.3415620
41. B. Tan, J. Cong, Optimality study of existing quantum computing layout synthesis tools. IEEE Trans. Comput. (2020). arXiv:quant-ph/2002.09783. https://doi.org/10.1109/TC.2020.3009140
42. B. Tan, J. Cong, Optimal qubit mapping with simultaneous gate absorption, in *2021 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, ser. ICCAD '21 (Association for Computing Machinery, Munich, Germany, 2021). arXiv:cs.ET/2109.06445
43. F. Vatan, C. Williams, Optimal quantum circuits for general two-qubit gates. Phys. Rev. A **69**(3), 032315 (2004). https://doi.org/10.1103/PhysRevA.69.032315
44. D. Venturelli, M. Do, E. Rieffel, J. Frank, Compiling quantum circuits to realistic hardware architectures using temporal planners. Quantum Sci. Technol. **3**(2), 025004 (2018). https://doi.org/10.1088/2058-9565/aaa331
45. R. Wille, L. Burgholzer, A. 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 on - DAC '19* (ACM Press, Las Vegas, NV, USA, 2019), pp. 1–6. arXiv:quant-ph/1907.02026. https://doi.org/10.1145/3316781.3317859
46. C. Zhang, A.B. Hayes, L. Qiu, Y. Jin, Y. Chen, E.Z. Zhang, Time-optimal Qubit mapping, in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (ACM, Virtual USA, 2021), pp. 360–374. https://doi.org/10.1145/3445814.3446706
47. A. Zulehner, R. Wille, Compiling SU(4) quantum circuits to IBM QX architectures, in *Proceedings of the 24th Asia and South Pacific Design Automation Conference on - ASPDAC '19* (ACM Press, Tokyo, Japan, 2019), pp. 185–190. https://doi.org/10.1145/3287624.3287704
48. A. Zulehner, A. Paler, R. Wille, Efficient mapping of quantum circuits to the IBM QX architectures, in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, Dresden, Germany, 2018), pp. 1135–1138. arXiv:quant-ph/1712.04722. https://doi.org/10.23919/DATE.2018.8342181