Time-Optimal Qubit Mapping

Chi Zhang*
chz54@pitt.edu
University of Pittsburgh
USA

Ari B. Hayes
arihayes@gmail.com
Rutgers University
USA

Longfei Qiu
lq56@scarletmail.rutgers.edu
Rutgers University
USA

Yuwei Jin
yj243@scarletmail.rutgers.edu
Rutgers University
USA

Yanhao Chen
chenyh64@gmail.com
Rutgers University
USA

Eddy Z. Zhang
eddy.zhengzhang@gmail.com
Rutgers University
USA

ABSTRACT

Rapid progress in the physical implementation of quantum computers gave birth to multiple recent quantum machines implemented with superconducting technology. In these NISQ machines, each qubit is physically connected to a bounded number of neighbors. This limitation prevents most quantum programs from being directly executed on quantum devices. A compiler is required for converting a quantum program to a hardware-compliant circuit, in particular, making each two-qubit gate executable by mapping the two logical qubits to two physical qubits with a link between them. To solve this problem, existing studies focus on inserting SWAP gates to dynamically remap logical qubits to physical qubits. However, most of the schemes lack the consideration of time-optimality of generated quantum circuits, or are achieving time-optimality with certain constraints. In this work, we propose a theoretically time-optimal SWAP insertion scheme for the qubit mapping problem. Our model can also be extended to practical heuristic algorithms. We present exact analysis results by using our model for quantum programs with recurring execution patterns. We have for the first time discovered an optimal qubit mapping pattern for quantum fourier transformation (QFT) on 2D nearest neighbor architecture. We also present a scalable extension of our theoretical model that can be used to solve large quantum circuits.

CCS CONCEPTS

• **Software and its engineering** → **Compilers**; • **Hardware** → **Quantum computation**.

KEYWORDS

Quantum Computing, Qubit Mapping, Noisy Intermediate Quantum Computers, NISQ, Quantum Fourier Transformation, QFT

*This work was done when Chi was visiting Rutgers University.

ASPLOS '21, April 19–23, 2021, Virtual, USA

ACM ISBN 978-1-4503-8317-2/21/04…$15.00
https://doi.org/10.1145/3445814.3446706

ACM Reference Format:
Chi Zhang, Ari B. Hayes, Longfei Qiu, Yuwei Jin, Yanhao Chen, and Eddy Z. Zhang. 2021. Time-Optimal Qubit Mapping. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21), April 19–23, 2021, Virtual, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3445814.3446706

1 INTRODUCTION

Quantum computing is a promising method to speed up important applications. These applications include factoring large numbers [14], searching a database [5], and simulating quantum systems [12]. With around 100 reliable qubits, quantum computers can already solve useful problems that are out of reach for classical computers.

Recently quantum systems with 49-72 qubits have been announced by IBM, Google and Intel. A number of quantum computers with around or less than 20 qubits are available to the public [2] through IBM Q experience. Programmers can run programs on these quantum computers. Despite the existence of certain tools such as IBM Qiskit [1], code must be written with respect to low-level specifications. A complete quantum compiler tool-chain needs to be built such that programmers can develop quantum algorithms that take full advantage of the potentially disruptive computing paradigm without having to worry about low level machine details.

The compilation of a quantum program is decomposed into two levels of translation. First, it converts an algorithm into a logical circuit composed of universal gates. These circuits are formulated independently of the hardware implementation. Second, it converts a logical circuit into a physical circuit with respect to hardware constraints. The problems in the first abstraction layer have been extensively [5, 12, 14] studied by theoreticians. However, less attention has been paid to the second abstraction layer, which is critical to the efficient execution of programs on real quantum computers.

Our paper addresses the second level of translation: from a logical circuit to a hardware-compliant physical circuit. In particular, we tackle the qubit mapping problem. In realistic architecture, it is not possible to establish direct interactions between every pair of qubits. In the superconducting quantum computers, qubits operate in the nearest neighbor manner, in which direct interactions form a bounded degree graph. An example is shown in Fig. 1 (a).

However, a logical circuit independent of hardware implementation assumes an unrestricted architecture, where every two qubits are connected. A logical circuit must be modified to account for the coupling constraints in quantum hardware. The common practice is to dynamically remap logical qubits to physical qubits via SWAP

gates such that each two-qubit gate is applied to two physically connected qubits. An example is shown in Fig. 1. If two logical interacting qubits q_1 and q_4 are respectively mapped to Q_1 and Q_4, one of them has to be "moved" closer to the other. For instance, by swapping q_2 with q_4, we can move q_4 closer to q_1.

Most previous studies on the qubit mapping problem [8, 16, 17, 20, 22, 23] have focused on *gate-optimal* solutions. They minimize the number of inserted SWAP gates. Some [8, 23] enhance the parallelism among the inserted SWAP gates while aiming to minimize the gate count. Zulehner *et al.* [23] proposed an algorithm using the A-star paradigm to minimize the number of swap gates for a local layer of concurrent CNOT gates. Li *et al.* [8] formulate a multi-objective function for exploiting the trade-off between different swap insertion strategies. The study by Siraichi *et al.* [16] models the swap-insertion problem as a subgraph isomorphism problem. Wille *et al.* [20] propose a model for global gate-optimal mapping using the SAT solver. A number of studies [10, 19] note the variability of qubit error rates in the IBM quantum computers and develop variability-aware qubit mapping strategies.

There are very few studies focusing on time-optimal qubit mapping. A time-optimal solution minimizes the depth of the entire transformed circuit rather than the depth or the number of the inserted SWAPs. OLSQ [18] is the only study that solves for optimal depth of the *entire* circuit. It is based on a constraint solver and its mapping overhead depends on the how far the optimal depth is from the ideal depth assuming every two qubits are connected. The work by Childs *et al.* [3] forms the foundation of IBM Qiskit's qubit mapper, however, it uses a heuristic approach to minimize the depth of the inserted SWAPs rather than that of the entire circuit. Lao *et al.* [7] introduces a framework that considers gate selection, qubit mapping, and classical control constraints. Their qubit mapping heuristic aims to improve the depth of the entire circuit by overlapping swaps with original gates in the circuit. But it does not guarantee an optimal mapping solution.

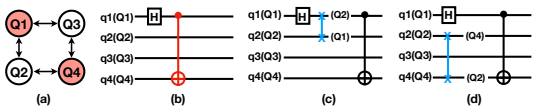

Figure 1: (a) Hardware coupling graph, (b) the logical circuit which cannot run due to qubit coupling constraints, (c) one possible way to make the circuit executable by swapping Q_1 with Q_2, and (d) another possible way to make the circuit executable by swapping Q_2 with Q_4. Upper case Q represents a physical qubit while lower case q represents a logical qubit.

Gate Optimality v.s. Time Optimality. To see why it is important to consider the interaction between inserted swap operations and the original circuit, we use an example in Fig. 1. The original circuit in Fig. 1 (b) is not executable. The transformed circuits in (c) and (d) are both gate optimal. However, only one of them is time optimal. That is because the solution in Fig. 1 (c) inserts a swap involving the slowest qubit, and as a result delays the execution of the entire circuit, while the solution in Fig. 1 (d) does not.

A time-optimal solution not only reduces the execution time, but also improves the reliability of the transformed circuit. It mitigates the *decoherence* effect. Qubits are error prone. A qubit decoheres over time. It gradually lose its state information. The longer a qubit operates, the less reliable it is. A time-optimal solution minimizes the impact of decoherence for the qubits in the circuit, and results in higher fidelity of the circuit as a whole.

Time-optimal qubit mapping is in general challenging. There are many possible permutations of SWAPs to achieve the same desired qubit mapping and there are many possible qubit mappings that can satisfy two-qubit gates in a given circuit. Even properly modeling the problem is a challenge.

In this paper, we tackle the time-optimal qubit mapping problem. We first present a simple and effective model for representing the complete search space. The search space is constructed with respect to an input logical circuit and a bounded degree hardware coupling graph. We then propose a search algorithm that is complete and optimal. Our contributions are summarized as follows:

- We present the first theoretical model for time-optimal qubit mapping without any implicit constraints. The theoretical model can be flexibly extended to practical algorithms.
- We present a search framework based on our time-optimal model. It consists of space pruning, redundancy elimination, and comparative filtering. It significantly reduces the time complexity and makes time-optimal search feasible.
- We discovered time-optimal solutions for quantum fourier transformation (QFT) on both 1D and 2D nearest neighbor architecture (using our search framework). Our solution for 1D nearest neighbor architecture is the same as a manual solution reported by Maslov [9]. But our optimal solution for 2D architecture is for the first time reported. Interested readers for QFT solutions can refer to Section 6.1.1.
- We present a practical extension of our theoretical model. It prunes the branches in the search space unlikely to yield effective mapping solutions. Our practical implementation is not guaranteed to be optimal, however, it still outperforms state-of-the-art qubit mappers with speedups ranging from 0.99X to 1.36X, and on average 1.21x, over representative benchmarks from RevLib, IBM Qiskit, and ScaffCC.
- We implemented our search framework for both optimal and practical solutions. We open sourced the code at GitHub[1].

The remainder of the paper is organized as follows. We start by introducing the background for quantum computing in Section 2. Section 3 presents a motivating example. Section 4 and 5 provide our modeling and implementation details. We present experiment results in Section 6 including both optimal and heuristic results. Related work is in Section 7. Concluding remarks are in Section 8. Appendix includes a detailed optimality proof and a discussion on how to find all optimal solutions when more than one exist.

2 BACKGROUND

2.1 Quantum Gates

There are two types of *elementary* quantum gates. One is single-qubit gate, an unitary quantum operation that can be abstracted as

[1]Our implementation is made publicly available at the GitHub repository: https://github.com/time-optimal-qmapper/TOQM.

the rotations around the axes of the Bloch sphere [11]. The other type of elementary gate is two-qubit gate.

The controlled-NOT (CNOT) is a very important two-qubit gate in quantum computation. A CNOT gate operates on a control qubit and a target qubit. If the control qubit is 0, it leaves the target qubit unchanged; If it is 1, it applies a NOT gate to the target qubit. The CNOT gate entangles qubits and enables communication. There are other types of two-qubit gates, for instance, the iSWAP gate, the cross resonance (CR) gate, the bMap gate, and etc. All of them require the two operand qubits to have direct links.

2.2 Qubit Mapping Problem

To execute a quantum circuit on a real machine, logical qubits must be mapped to physical qubit on the target hardware. When applying a two-qubit gate, the two participating logical qubits must be mapped to physically connected qubits. Due to the bounded degree connectivity of physical qubits on current devices, it is generally considered impossible to find an initial qubit mapping that satisfies all two-qubit gates in the entire circuit.

A common practice is to dynamically map the logical qubits by inserting SWAP gates. A swap gate exchanges the states of the two operand qubits. There are different ways to implement a SWAP. A typical way to implement a SWAP is to use 3 CNOT gates if the links between physical qubits are bidirectional. Our model does not impose constraints on how a SWAP is implemented. Rather, we set the latency of a SWAP as a parameter in our model. We use various SWAP latencies according to the architectures we are evaluating.

The qubit mapping problem takes a logical circuit and a hardware coupling graph as input, outputs a transformed circuit. Only swap operations are allowed to be added into the transformed circuit. After transformation, all two-qubit gates must be hardware compliant. If a two-qubit gate g is performed on logical qubits q_1 and q_2, when running g, q_1 and q_2 must be physically connected.

One logical qubit can be mapped to different physical qubits at different points of circuit execution. Once a two-qubit gate is completed, both of its logical qubits can be remapped to some other physical qubits in the future. An example of qubit mapping is shown in Fig. 1 (c) and (d).

3 MOTIVATION

It is non-trivial to achieve time optimality for the qubit mapping solution. As there are many possible permutations of circuit gates and inserted swaps, the search space is large even for small input. We use quantum fourier transform (QFT) to demonstrate the challenges in finding a time-optimal solution.

QFT is at the heart of integer factorization [15]. It serves as the basis for many important quantum algorithms. QFT is also one of the most challenging benchmarks for qubit mapping. It is because QFT requires all-to-all qubit connection. Each qubit needs to interact with every other qubit in the program.

The QFT circuit has a regular pattern. It has n qubits and $n(n-1)/2$ generic two-qubit gates. We follow the convention by Maslov *et al.* [9] for describing a QFT skeleton circuit. A concrete QFT circuit includes Hadamard (H) gates and controlled phase gates. Following this convention, a single-qubit is absorbed into a nearby two-qubit gate to form a generic two-qubit gate. Any two-qubit gate can be efficiently implemented and we assume they have the same latency as in [9]. Hence the entire circuit is described using generic two-qubit gates of the same latency. We denote the original two-qubit computation gate as GT gates for the simplicity of discussion.

Each logical qubit q_i interacts with every other logical qubit q_j, denoted as $GT(q_i, q_j)$ where $j \neq i$. The logical QFT circuit with 6 qubits is shown in Fig. 2 (b).

LNN. Let's look at the simple linear nearest neighbor (LNN) architecture as shown in Fig. 2 (a), where the capital case Q represents physical qubits. In this architecture, the physical qubits are arranged on a straight line. Each physical qubit only communicates with the qubit on its left or right. However, the logical QFT circuit requires each qubit to interact with every other qubit. The logical circuit in Fig. 2 (b) cannot directly run on Fig. 2 (a) no matter which initial qubit mapping is used.

Due to the all-to-all qubit interaction pattern in QFT, it is hard to obtain an effective qubit mapping on LNN, let alone an optimal solution. Using our search algorithm described in the next section, we find an optimally transformed QFT-6 circuit. The solution is shown in Fig. 2 (c). Although the circuit size is small, it is not difficult to see a butterfly pattern in the transformed circuit represented using physical qubits. With further analysis in Section 6, this pattern can generalize to larger QFT circuits and ensure linear depths.

We note that our solution for QFT in LNN is the same as the manual solution by Maslov [9]. However Maslov [9] cannot find an optimal solution manually for 2D architecture due to the complexity of the architecture, while our mapper can, as described below.

2D Grid. Another prototypical hardware architecture is a structured two-dimensional lattice. Each qubit has up to 4 neighbors. For instance, IBM Melbourne architecture has a 2×N grid like topology, as shown in Fig. 3. Maslov [9] did not provide a manual solution for the 2D architecture. But the paper predicts a lower bound depth of $3n + O(1)$. Using our algorithm, we found a generalizable pattern for arbitrary size QFT. Our generalized solution has $3n+O(1)$ depth which matches the lower bound depth provided by Maslov [9]. The asymptotic term is only at the constant component. Hence our solution is not only confirmed to be optimal for small inputs but also for arbitrary size inputs. Our solution is presented in Section 6.

Time-optimal solutions, even for small size circuit, could be very useful. If an optimal solution for a logical circuit has recurring pattern, we can obtain the optimal solution for small-size inputs, and use that to deduce the generalized solution. The solutions to the QFT program on two types of architectures demonstrate the effectiveness of our algorithm in discovering such patterns for circuit families. For finding a solution to QFT-6 on LNN, our mapper takes less than 1 second. For finding a solution to QFT-8 on 2D architecture, it takes less than 30 seconds. In the following, we will describe our model and search algorithm in details.

4 TIME-OPTIMAL MAPPING FRAMEWORK

In this section, we define a time-optimal model for the qubit mapping problem. We first define the search space. Then we present a guided search framework. We prove the optimality of the framework in Section 5 and Appendix A.

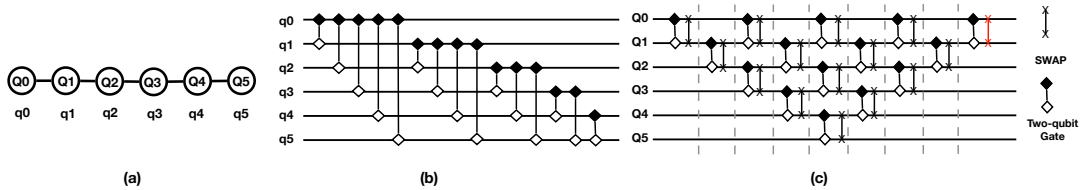

Figure 2: Adapting QFT logical circuit to LNN architecture: (a) 6-qubit LNN and initial mapping, (b) logical qft-6 circuit where each line represents a logical qubit q_i, and (c) physical qft-6 circuit where each line represents a physical qubit Q_i. The last swap gate in red in (c) is added for showing the symmetric pattern. Our solver does not have the swap in its returned solution.

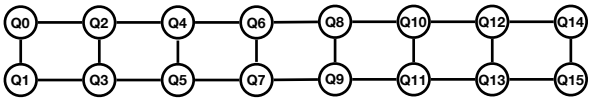

Figure 3: 2×N Qubit Coupling Graph

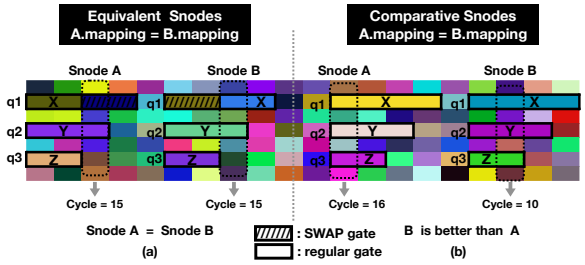

Figure 5: Filtering example (a) Two equivalent search nodes. Both A and B will finish the same number of original gates. They also have achieved the same mapping assuming all active swaps immediately take effect. But the order of the gates is different. (b) The two state nodes will have to take different number of cycles to achieve the same state. But B is faster, A can be safely pruned without affecting optimality.

4.1 Search Space

We discover that any valid execution of a circuit can be partitioned into a sequence of states. We refer to *cycle* as a unit of time. A circuit is decomposed with respect to its state at each cycle. A *state* of the circuit represents a qubit mapping and the specific busy/idle status of each qubit. The status of a qubit is represented as which gate it is executing if the qubit is busy, or which gate it has just completed if the qubit is idle. An example is shown in Fig. 4 (a).

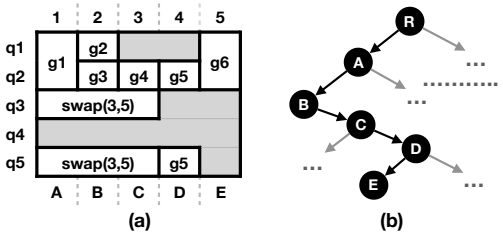

Figure 4: (a) Cycle-by-cycle partition of a 5-cycle circuit, (b) the search path representing the execution in (a). R is the root node. E is a terminal node.

We define our search graph. Each node in the search graph represents a state of the circuit (which also includes a cycle number).

A node expands into one or multiple children nodes. Each child node represents a possible state the circuit will be in at the next cycle. A child node's cycle number is its parent's cycle number + 1. The executing gate(s) of a child node in its own cycle must satisfy the constraints of the qubit coupling and gate dependence.

By enumerating all possible combinations of gates and swaps in next cycle, a node can determine its complete set of child node(s). The circuit makes progress when a gate in the original circuit executes. The circuit updates its qubit mapping when an inserted SWAP finishes its execution.

Root Node and Terminal Node. We let the *root node* be the initial state of the circuit at cycle 0 where all qubits are idle and no gate in the original circuit has been scheduled.

We say a node is *terminal*, if all logical qubits have completed their gates in the original circuit, and the last gate of all just finished at the cycle of this node.

The goal is to find a terminal node such that its path to the root node is the shortest. There is only one root node but there could be more than one terminal node. An optimal solution corresponds to an optimal terminal node. The path from the root node to a terminal node represents a transformed circuit.

Fig. 4 (b) shows a valid path from the root node to a terminal node. It corresponds to the example circuit in Fig. 7 and the execution in Fig. 4 (a). The node E is a terminal node, and R is the root node.

4.2 Guided Search Framework

Since there is a finite number of gates and physical qubits, there is only a finite number of combinations of gates and swaps that can be executed simultaneously at any given moment. Thus each node in the search graph has only a finite number of child nodes. An exhaustive breadth-first search (BFS) algorithm is guaranteed to find the optimal terminal node in finite time. However, BFS search is not realistic as it is brute-force and search space is large even when the input is small.

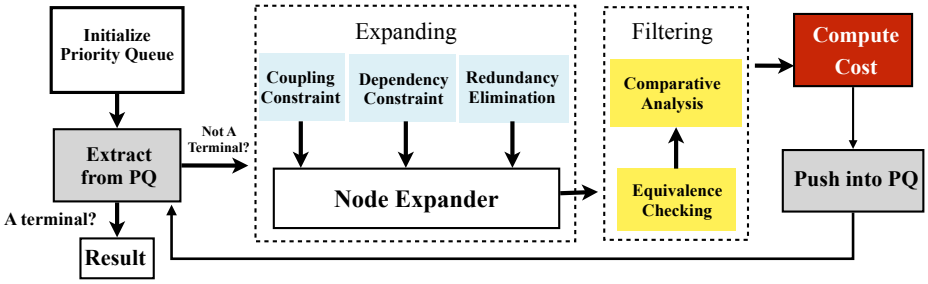

Figure 6: Our Optimal Search Framework

Overview of Our Framework. We present a guided search framework. It uses a priority queue to keep track of the nodes that need to be expanded. We show our framework in Fig. 6. It first initializes the priority queue to be empty and then inserts the root node into the priority queue.

Next it extracts a node from the priority queue, expands it, and push new nodes into the queue. The three steps repeat until a terminal node is for the first time popped out of the priority queue. It is based on A* search. Each node is associated with a priority (cost). We set the priority (cost) function to be admissible to ensure the the A* search returns an optimal solution. The detailed definition of the priority function and the proof for optimality is shown in Section 5 and Appendix A.

Using A* guarantees optimality. We also ensure efficiency by space pruning techniques. It is often possible that the circuit reaches the same state using different combinations of gates and swaps. If the only difference between two nodes is their timestamps, then the slower one of the two nodes can be dismissed without affecting optimality. We prune these nodes in the Expanding component and the Filtering component in Fig. 6.

Expander. The node expander is responsible for expanding the node extracted from the priority queue. After expansion, the children nodes of the expanded nodes will be collected. We apply several restrictions on top of these children nodes. Below are the criteria that we want these children nodes to meet:

- **Coupling** One node needs to satisfy the current qubit coupling constraints. Thus we first filter out the nodes that cannot satisfy coupling constraint.
- **Dependency** The gates scheduled in a child node should also have their dependency resolved. This means all parent gates of any newly added active gate should be finished prior to the child node's cycle.
- **Redundancy Check** We check two kinds of redundancy. First we check that whether at least one active gate of the child node depend on some gate in the parent node's active gates (except that the active gate is the first gate on a qubit). This criteria is based on the fact that any non-depending gate in the child node could have been placed earlier in the parent's sibling nodes.
  Second we check if there are cyclic swaps, which means identical swaps applied consecutively to the same two qubits. It does nothing but cancel out the previous swap effect.

Filter. We have a filtering pass on the expanded nodes that meet the criteria we listed above. In this filtering pass, every expanded node has its hash value calculated based on its current qubit mapping (assuming all active swaps have taken effect). For each node, we look at its hash value and compare it to all the previous nodes (in the priority queue) that have the same hash value. When comparing one expanded node with all the previous nodes that have the same hash value, we check for equivalence and relative goodness.

- **Equivalence Check** We check if this expanded node is equivalent to any previous node. Equivalence means the two nodes will have finished the same set of gates assuming all active gates have taken effect. The active gates on same qubits will finish in the same exact cycle. The current cycle must be the same. If we find this equivalence, we filter out this expanded node. An example is shown in Fig. 5 (a).
- **Comparative Analysis** If there's no equivalence found, we then switch to comparative analysis between the expanded node and the previous node with same hash value. We look at the projected finish cycle on each qubit (if the qubit is busy). If, for all qubits, the expanded node has fewer finished gates but longer projected finishing time, we know that this expanded node is less desirable than this previous node. We then filter out this expanded node as itself and its sub-tree will not lead to a better solution than one previous node. An example is shown in Fig. 5 (b).

5 OPTIMALITY GUARANTEE

5.1 Admissible Cost Function

We assign each search node v in the search graph a cost $f(v)$. The cost $f(v)$ consists of two parts $g(v), h(v)$ such that

$$f(v) = g(v) + h(v) \tag{1}$$

$g(v)$ represents the length of the path from the root node to v, which is the number of cycles already executed. $h(v)$ is a heuristic cost function as a lower bound on the number of cycles from v to any terminal node.

The circuit now contains two parts with respect to a search node v: (1) the part that has been scheduled due to v and v's ancestor nodes, and (2) the remaining circuit that hasn't been scheduled. The cost $g(v)$ is for part (1), and cost $h(v)$ is for part (2). An example is shown in Fig. 7.

The cost for part (1) is trivial as a search node contains a time stamp. The cost $h(v)$ for part (2) is more complex.

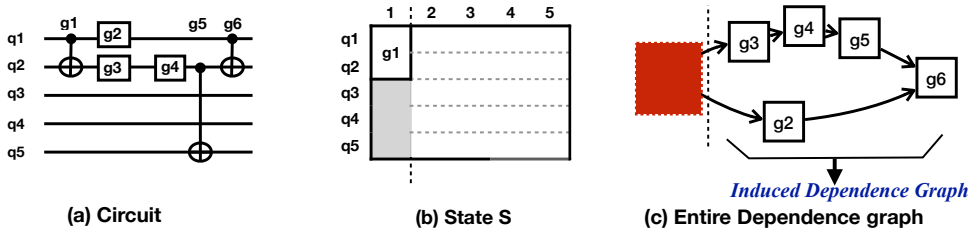

Figure 7: Two components of a circuit with respect to a search node: (a) logical circuit, (b) a search node S for cycle 1 indicating already scheduled gates by cycle 1, and (c) the dependence graph, where the part after the dashed line is remaining circuit to be executed.

Our heuristic function $h(v)$ is defined with respect to the dependency graph G_{rem} and the qubit mapping π_{rem} of the remaining circuit. If the node v contain any active swap that hasn't been completed, we assume swap has taken effect for calculating π_{rem}. As it is not difficult to obtain remaining dependence graph from search node v and v's ancestor nodes, we omit the details here.

Let $G_{rem} = (V_{rem}, E_{rem})$ be the dependency graph of the gates in the remaining circuit. Thus V_{rem} consists of the gates that haven't been scheduled or have been scheduled but executed in part, and G_{rem} is a directed acyclic graph. We define the heuristic function $h(v)$ via induction on G_{rem}.

Let $g_1, g_2, \cdots, g_n$ be a topological ordering on the vertices (gates) in G_{rem}. For each gate g,let $len(g)$ be the number of cycles that g needs to execute (if $len(g)$ is partially executed, $len(g)$ is equal to the length of the unexecuted part). We will define $t_{min}(g)$, which is a lower bound on the time when g begins to execute.

Base case: If g_i is a single-qubit gate that has no predecessors in G_{rem}, then $t_{min}(g_i) = 0$, meaning, g_i may begin to execute immediately.

Inductive case: Suppose gate g_i depends on gate $h_1, h_2, \cdots$, then g_i can only begin after these gates have finished. First let $u = \max_i t_{min}(h_i) + len(h_i)$. We must have $t_{min}(g_i) \geq u$. If g_i is a single-qubit gate, then we simply take $t_{min}(g_i) = u$.

If g_i involves two qubits, then we also have to consider the delay caused by inserting SWAP gates. Suppose that gate g_i involves qubits q_a and q_b. Let H_a be the set of gates (in G_{rem}) that involve q_a and are direct or indirect predecessors of g_i. Similarly we may define the set H_b. Let T_a be the sum of the number of cycles needed by the gates in H_a. Then $u - T_a$ represents the "slack" space that may be used by qubit q_a to perform SWAPs. Similarly we define T_b.

Now, in the current qubit mapping π_{rem}, let $d(a, b)$ be the shortest distance between qubit q_a, q_b. Then we need at least $d(a, b) - 1$ SWAPs in total on qubit q_a with some other qubit, and on q_b with some other qubit. We are only concerned with the delay on q_a and q_b but not the delay on the qubit which they have SWAP with. Suppose we place r SWAPs on q_a and s SWAPs on q_b, then the delay on q_a is $\max\{r \cdot len(SWAP) - (u - T_a), 0\}$, and the delay on q_b is $\max\{s \cdot len(SWAP) - (u - T_b), 0\}$. We have $r + s \geq d(a, b) - 1$. We let $r + s = d(a, b) - 1$ as it will not increase the cost, and enumerate all possible combinations of (r, s) with $r = 0 \ldots d(a, b) - 1$. We take the pair (r, s) that minimizes the larger of the two delays. Let u' be the minimized delay, then we take $t_{min}(g_i) = u + u'$.

Definition 5.1.1. *(Heuristic cost function $h(v)$) Having computed $t_{min}(g)$ for each gate g, we define the heuristic cost $h(v)$ to be*

$$h(v) = \max_{g \in V_{rem}} t_{min}(g) + len(g)$$

Cost Calculation Example. An example of calculating the cost function with respect to a CNOT gate is shown in Fig. 8. The state node F we focus on is in Fig. 8 (a), where it has already scheduled the $g1$ and started $SWAP\ Q_4, Q_5$. We assume each single original gate in the circuit takes 1 cycle and each swap takes 3 cycles.

The remaining dependency graph of the circuit is shown in Fig. 8 (c). Since g_1 is completed, s_{45} has executed in part, G_{rem} consists of the remaining gates and part of s_{45}. First we set $t_{min}(g_2) = t_{min}(g_3) = 0$ as they have no predecessors. $t_{min}(s_{45})$ is also 0. Then $t_{min}(g_4) = 1$ as it depends on the single-cycle gate g_3.

The qubit mapping π_{rem} after gate s_{45} is shown in Fig. 8 (b). g_5 is not immediately executable, as q_2 and q_5 are not adjacent to each other. The shortest path between them is $Q_2 \rightarrow Q_3 \rightarrow Q_4$ with a distance of $d = 2$. Now at least $d - 1 = 1$ total SWAP needs to be inserted in total on q_2 with some other qubit, and on q_5 with some other qubit.

Suppose g_5 could be executed immediately, then it would have got start time as $u = 2$. On the part of qubit q_2, gate g_5 depends on g_3 and g_4, $2 - 2 = 0$, so there's no slack space on q_2. On the part of qubit q_5 the predecessor is part of s_{45}, which is 2 cycles. Therefore, the slack space on qubit q_5 is 0 cycle too. Hence $t_{min}(g_5) = 2 + 3 = 5$, as inserting SWAP on either q_2 or q_5 introduces a delay of 3.

Finally, $t_{min}(g_6) = 6$ since q_1, q_2 are adjacent, and the cost for search node F is 8.

Another example is for the search node A in Fig. 8 (e) where g_1 and $swap(3, 5)$ is scheduled in cycle 1. The induced qubit mapping is in Fig. 8 (f). Since each CNOT in remaining circuit can be immediately executed, the cost of A is the critical path of the remaining circuit plus 1, which is 5. Therefore search node A is better than search node F as its cost is lower.

Common Fallacy. It is tempting to assume that, since two qubits can move towards each other by performing SWAPs simultaneously, the optimal position for two qubits to meet is always at exactly the middle of the shortest path between them. Hence one may simply need to insert dummy gates of length $(d-1) \cdot len(SWAP)/2$. This is not true as it is oblivious to the slack in the original circuit that can potentially absorb SWAP overhead. An example is demonstrated in Fig. 9. Here we assume that, in the initial mapping,

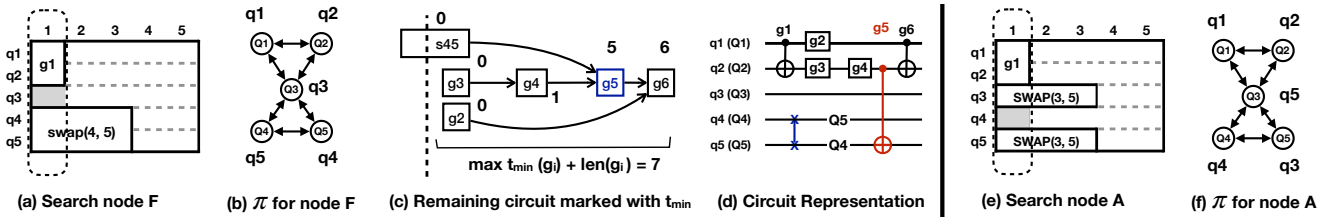

Figure 8: Cost calculation (a) Search node F, (b) qubit mapping for remaining circuit induced by F, (c) remaining dependency graph marked with t_{min}, and (d) circuit representation; (e) search node A, (f) qubit mapping for remaining circuit induced by A.

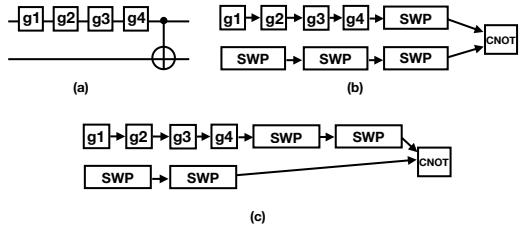

Figure 9: An example circuit, demonstrating a common fallacy in reasoning about the heuristic function. (a) Logical circuit, assuming that the distance between the two qubits is 5. (b) The dependency graph of the circuit, if we choose to place 1 SWAP on the first qubit, and 3 SWAPs on the second. (c) The dependency graph of the circuit, if we choose to insert two swaps on each qubit. Assuming each SWAP takes 2 cycles and each original gate takes 1 cycle in this example.

the distance between the two qubits is 5, so at least 4 SWAPs are needed. Suppose we let the two qubits meet in the middle. Assume a SWAP takes 2 cycles. Then we need to insert a 4-cycle delay for each qubit. The length of the critical path is 8 cycles. However, suppose we let the first qubit do 1 SWAP, and let the second qubit do 3 SWAPs. Then the length of the critical path becomes 6 cycles. This demonstrates the necessity of trying all possibilities of splitting delay to two participating qubits when calculating the cost.

5.2 Optimality

Our heuristic cost function $h(v)$, in effect, computes a lower bound on the time needed by the remaining circuit through two different ways, and takes the larger of the two bounds. The first way considers the immediate predecessors of each gate. The second way considers the indirect predecessors and potential SWAPs. We prove rigorously it is a lower bound time of the remaining circuit in Appendix through Lemma A.1. We relegate the details to the Appendix.

Lemma 5.1. *The cost function defined in Eq. 1, $f(v) = g(v) + h(v)$ is admissible.*

Proof. We've shown in Lemma A.1 that the heuristic function $h(v)$ never overestimates the execution time of the remaining circuit. And also g(v) is the number of cycles already executed till node v. Therefore the cost function $f(P)$ is admissible. □

Theorem 5.2. *The A-star search algorithm we proposed here is complete and optimal.*

Proof. It has been shown that A-star search will find an optimal solution if the problem satisfies the following conditions:

(1) Each node in the search graph only branches into a finite number of child nodes. This is true, because the set of gates that can possibly execute at any given moment is finite.
(2) Each transition increases the cost of the path. This is true for our problem, because each gate takes at least one cycle to execute, and so increases the cost by at least 1.
(3) The heuristic cost function is admissible. This is proven in the lemma above.

Thus our algorithm satisfies all conditions for optimality. □

5.3 Initial Mapping

Our optimal mapper works in two different modes: (1) It finds an optimal solution given an input initial mapping, and (2) It finds both optimal initial mapping and the transformed circuit.

For (1), it is trivial. Our previous technical description already addressed how to find the solution after initial mapping is given. For (2), we start with a random initial mapping. Then we allow at most d consecutive cycles of pure swaps before any original gate is scheduled, which represents a search for initial mapping. In the meantime, we modified our cost function such that the pure swap cycles at the beginning are not counted, as if the circuit starts at some initial mapping resulted from these consecutive pure swap cycles. The parameter d is defined as the maximum of the longest path (without going through any node twice) between any two qubits in the physical architecture. It is because one mapping layout could be transformed into another mapping layout with at most the number of cycles equivalent to the maximum longest path length in the graph, assuming swaps on disjoint qubits run in parallel.

For the pure swap cycles, our hash filter is also applied, ensuring that no unique initial mapping will appear twice in the priority queue. It is because one initial mapping can be achieved through different ways of swap combinations.

.

6 ANALYSIS

6.1 Exact Analysis

When the number of qubits is small, our algorithm can find (a set of) exact optimal solution(s). This is helpful in the applications

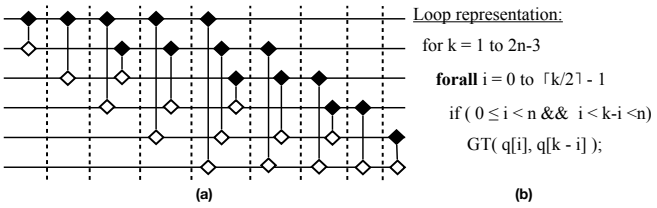

Figure 10: (a) QFT-6 circuit rearranged to exploit parallelism; (b) Affine loop representation of re-arranged n-qubit circuit.

where optimal solution has a recurring pattern. We use a two-step approach. We first find optimal solution(s) for small inputs. Then we generalize the solution(s) to larger inputs. We demonstrate it using QFT program in Section 6.1.1. We also show the optimal results for small logical reversible circuits in Section 6.1.2.

6.1.1 Optimal QFT Mapping. Recall that QFT has a regular structure, with n qubits and $n(n-1)/2$ generic two-qubit gates (Section 3). QFT has an all-to-all qubit interaction pattern. Every two qubits need to interact. The QFT circuit with 6 qubits is shown in Fig. 2.

We present a different representation of the QFT circuit in Fig. 10 (a) to facilitate discussion. As gates that operate on non-intersecting qubits commute, a QFT program can run in linear depth if the underlying architecture is fully connected. Fig. 10 (a) is equivalent to the circuit in Fig. 2 (b) except that it is organized into parallel layers such that each layer consists of concurrent two-qubit gates, and the affine loop representation is shown in Fig. 10 (b).

LNN Architecture. Recall that in LNN, the physical qubits are arranged on a (conceptual) straight line, and each qubit may only interact with the qubit on its left or right. For QFT, the logical qubits are initially mapped according to its natural order such that logical qubit q_0 is mapped to the leftmost physical qubit and q_5 is mapped to the rightmost physical qubit as shown in Fig. 11 (step 0).

The LNN architecture is often considered as a good approximation to what a scalable quantum architecture may be. If a circuit can be adapted well to LNN, it typically can be adapted to other architectures represented by a bounded degree graph [9].

Our search algorithm finds an optimal solution[2] for QFT with 6 qubits, visualized in Fig. 11. We next show that the discovered pattern for 6-qubit QFT can be generalized to n-qubit QFT.

The qubit mapping changes every two consecutive steps (from step 0). In every two consecutive cycles, a set of GT gate(s) first operate on some qubits, then a set of swaps on exactly the same qubits. At the end of these steps, the layout of the logical qubits are reversed such that q_5 is placed at the left end of the LNN, and q_0 is placed at the right end of the LNN. Each qubit first shifts to left until it hits the left end of the LNN, and then to right until it reaches its destination.

To generalize this, we assume a sequence of logical qubits on the chain at a cycle m where m is even and $m/2$ is also even, and

[2]There might be multiple optimal solutions, but not all of them have a recurring pattern. However, our algorithm can be adapted to find all optimal solutions such that the solution with recurring pattern can be easily obtained. On the other hand, even some solutions do not have a strict recurring pattern, but the pattern can be inferred by performing some transformation based on gate commutativity or cancel-able swaps. We demonstrate this in Appendix B.

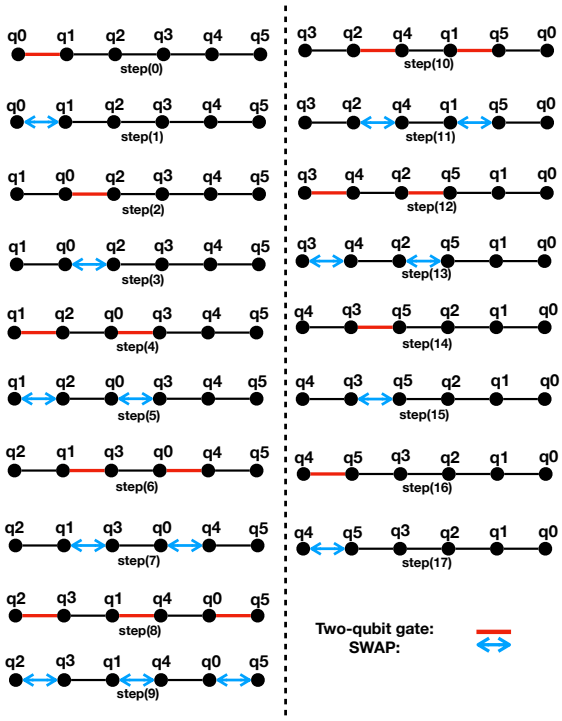

Figure 11: QFT-6 on LNN. Step (17) is not necessary. We are adding it to show the pattern.

we let $i = m/2$, the qubit placement from left to right is

$$q_{\frac{i}{2}}, q_{\frac{i}{2}+1}, q_{\frac{i}{2}-1}, q_{\frac{i}{2}+2}, ..., q_i, q_0, q_{i+1}, q_{i+2}, q_{i+3}, q_{i+4}, ..., q_{n-1}$$

At the step m, it is a parallel computation stage of two-qubit gates. We have GT(q_0, q_{i+1}), GT(q_1, q_i), ..., GT($q_{\frac{i}{2}}, q_{\frac{i}{2}+1}$) scheduled. Then at the step $m+1$, we have SWAP(q_0, q_{i+1}), SWAP(q_1, q_i), ..., SWAP($q_{\frac{i}{2}}, q_{\frac{i}{2}+1}$). For each pair of qubits, their subscript adds up to $i+1$, which is $m/2+1$.

At the step $m+2$ where m is even and now $(m+2)/2$ is odd, we still let $i = m/2$. The qubit placement from left to right is:

$$q_{\frac{i}{2}+1}, q_{\frac{i}{2}}, q_{\frac{i}{2}+2}, q_{\frac{i}{2}-1}, ..., q_{i+1}, q_0, q_{i+2}, q_{i+3}, q_{i+4}, ..., q_{n-1}$$

At step $m+2$, the operations are GT(q_0, q_{i+2}), GT(q_1, q_{i+1}), ..., GT($q_{\frac{i}{2}}, q_{\frac{i}{2}+2}$). At step $m+3$, it performs parallel swaps: SWAP(q_0, q_{i+2}), SWAP(q_1, q_{i+1}), ..., SWAP($q_{\frac{i}{2}}, q_{\frac{i}{2}+2}$). For each pair of qubits, their subscript adds up to $i+2$, which is $(m+2)/2+1$ or $\lfloor (m+3)/2 \rfloor + 1$.

Now the pattern is clear from Fig. 11. It repeats the two above sets of operations, and each pair of qubits that perform GT or swap their subscripts sum up to $\lfloor m/2 \rfloor + 1$ if m is the iteration number. The generalized strategy can be applied to QFT with arbitrary number of qubits. We describe it using an affine loop in Fig. 13 (a).

Note that the swap gate(s) in the last step is not necessary. But we add it because it fits into the pattern, and also that after the last swap, the physical connectivity graph of the logical qubits is isomorphic to the one right before step 0.

Our generalized solution for QFT on LNN is the same as that found by Maslov [9]. However, that solution is found manually and the author proved the solution is at most a constant factor away from the optimal solution. Our qubit mapper at least confirmed that this solution is optimal for small input size of QFT.

2D Architecture. Now we describe our generalized solution for QFT on $2 \times N$ architecture (where $N = n/2$). Maslov [9] does not report a generalized solution for this, but predicts a lower bound of $3n+O(1)$ circuit depth. Our solution is $3n+O(1)$, which is the same (asymptotically at the constant component). This is discovered for the first time as far as we can tell for QFT on a 2D architecture.

We visualize the solution of QFT-8 on a 2x4 architecture in Fig. 12. We denote the physical qubit on the j-th column and the i-th row as $Q_{i,j}$. Initial placement is a column major order such that $q_{2j+i} \rightarrow Q_{i,j}$ as shown in step (1) of Fig. 12.

With *column major* order, it takes 17 cycles, and the pattern is generalizable. We also tried the row major order, it takes 21 cycles, and the pattern is not generalizable. We only present the result of column-major initial qubit placement here. It is worth mentioning that we also tried the version which does not allow concurrent swap and computation gates to run. It takes 19 cycles and is generizable too. The generalized solution is shown later in this section.

With some analysis, it can be shown that the $2 \times N$ solution is a non-trivial extended version of $1 \times n$. If we look at the qubit layout at steps (2), (5), (8), (11), (14), and (17) in Fig. 12, placement of pairs of qubits on $(Q_{0,i}, Q_{1,i})$ resembles placement of qubits on Q_i in LNN as the circuit makes progresses.

We let every three steps starting from step (2) in Fig. 12 form one iteration (iteration is indexed from 0). Let iteration number be i. If i is even, at the first step of iteration i and at the top row of the qubit layout, the placement of qubits is as follows:

$$q_{2(\frac{i}{2})}, q_{2(\frac{i}{2}+1)}, q_{2(\frac{i}{2}-1)}, q_{2(\frac{i}{2}+2)}, .., q_0, q_{2(i+1)}, q_{2(i+2)}, .., q_{2(N-1)}$$

We present it on purpose such that the subscript is a multiply of 2 and a number (the number need not to be integer).

The placement of logical qubits on the bottom of row is similar except that q's subscript increases by 1.

At the first step of the iteration i, swap and GT are performed simultaneously such that GT is on top row, and swap on the bottom row. GT is on even-index qubits, and swap on odd-index qubits.

For GTs, they are on $\{q_{2(\frac{i}{2})}, q_{2(\frac{i}{2}+1)}\}$, $\{q_{2(\frac{i}{2}-1)}, q_{2(\frac{i}{2}+2)}\}$, ..., $\{q_0, q_{2(i+1)}\}$. And for swaps, they are on $\{q_{2(\frac{i}{2})+1}, q_{2(\frac{i}{2}+1)+1}\}$, $\{q_{2(\frac{i}{2}-1)+1}, q_{2(\frac{i}{2}+2)+1}\}$, ..., $\{q_{2*0+1}, q_{2(i+1)+1}\}$.

At the second step of iteration i, the placement of logical qubits on bottom row changes to the following:

$$q_{i+3}, q_{i+1}, q_{i+5}, q_{i-1}, .., q_{2(i+1)+1}, q_1, q_{2(i+2)+1}, .., q_{2(N-1)+1}$$

GT gates are performed between two qubits at the same column on $\{ q_i, q_{i+3} \}, \{ q_{i+2}, q_{i+1}\}$, ..., $\{q_0, q_{2i+3} \}$.

In the third step of the iteration i, swaps and GT are performed in parallel on top row and bottom row separately. This time, top row performs swaps and bottom row performs GT. Swaps are on $\{q_i, q_{i+2}\}, \{q_{i-2}, q_{i+4}\}, ...\{q_0, q_{2*(i+1)}\}$. GTs are on $\{q_{i+3}, q_{i+1}\}$, $\{q_{i+5}, q_{i-1}\}, ...\{q_{2i+3}, q_1\}$.

With these steps, the even iteration completes. It will go to an odd iteration $i + 1$. Before operations at iteration $i + 1$ start, the qubit layout at the top row now becomes

$$q_{i+2}, q_i, q_{i+4}, q_{i-2}, .., q_{2(i+1)}, q_0, q_{2(i+2)}, .., q_{2(N-1)}$$

The steps in the even iteration complete a part of GTs such for which the subscript summation of each pair of qubits is $2i + 2$, and all GTs such that the subscript summation $2i + 3$, and a part of GTs for which each pair's subscripts sum to $2i + 4$.

The odd iterations are similar. It is just that the pairs of qubits should start from the second column from the left end of the grid when doing GT or SWAP for the step 1 and step 3 in the iteration, which we will not discuss in details here.

By repeating this pattern, all GT gates can be performed with respect to the loop in Fig. 10 (b). It is just that one parallel iteration in that loop might be split into two parts to be distributed into an even iteration and an odd iteration.

Asymptotically, every two parallel layers in Fig. 10 (b) take 3 cycles using the transformation pattern in Fig. 12. The total number of layers is $2n - 3$ when the number of qubits is n. Our generalized solution then takes $3n + O(1)$ cycles.

Our generalized solution is presented in Fig. 13 (b).

A constrained optimal solution for QFT on 2×N arch. In some scenarios, it does not allow concurrent swaps and two-qubit gates. Hence, we added the constraint that either GT gates or swap gates in each cycle (but not both). Under this constraint, we solve for an optimal solution. And we visualize such a solution of QFT-8 on a 2x4 architecture in Fig. 14.

Initial placement is column major as shown in Fig. 14 step (1).

Now a more elegant pattern shows and the depth is still $3n+O(1)$. We still form every three steps as one iteration (starting from step 1). We let iteration index be i, starting from 0. For each iteration, the first step only perform GTs and all GTs applied to qubits on the same column. The number of GT performed increases by 1 at each iteration until about half way of the iterations and then decreases by 1 at each iteration. The second step of each iteration only performs swap gates. All swaps only happen to qubits on the same row. The third step of each iteration performs GT gates only.

It is worth noting that the layout of these logical qubits form a graph that is isomorphic to that at the beginning (as if the layout is mirrored, similar to the case in LNN). This is a nice property of the structured QFT transformation methods we discovered.

We show the pseudocode of this solution in Fig. 13 (c).

6.1.2 Building Block Circuits. We show results for the building block circuits that include adders, modular function, and various counters in Table 1. These are also the benchmarks used in the work by Wille *et al.* [20] for gate-optimal qubit mapping. Our mapper finds time-optimal solution very fast, usually in less than one second. The quantum architecture is IBM's QX2. Note that for this set of benchmarks, both initial mapping and transformed circuit are determined optimally, while for QFT experiments in Section 6.1.1, we tried different initial mappings (row major and column major) just for discovering the patterns.

In this table, *ideal cycle* refers to how many cycles the original circuit would take on an ideal architecture where every two qubits are connected; *optimal cycle* is the number of cycles we found for the target architecture; *mapper overhead* is the time the mapper

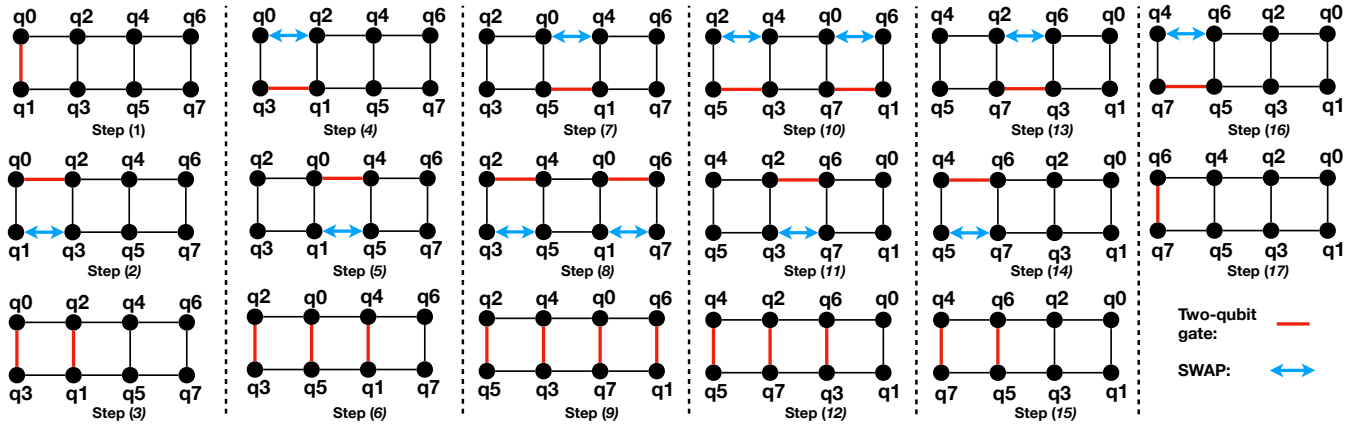

Figure 12: Optimal scheme for QFT-8 with 2×4 qubits. Each sub-figure represents a step of the execution, which takes one cycle. It also represents the state of the circuit at each cycle. There are in total 17 steps, and thus 17 cycles.

```
for m = 0; m+=2; m < 4n - 6 do
  k = (m/2)+1;
  for i= 0; i++; i < (k - i) do
    if i < n && k - i < n, then
      GT( q[i], q[k - I] );
  for i = 0; i++; i < (k - i) do
    if i < n && k - i < n, then
      SWAP( q[i], q[k - I] );
```

(a)

```
for i = 0; i++; 2*i+2 < 2n - 1 do
  for j = 0; j += 2; j < i do
    if (2i - j) < n && j < n, then GT( q [j], q [2i-j] );
    if (2i -j + 1) < n && j+1 <n, then SWAP( q[j+1], q[2i - j + 1] );
  for j = 0; j += 2; j < 2*i+1, do
    if (2i+1-j) <n && j < n, then GT( q[j], q[2i+1-j] );
  for j = 1; j += 2; j <= i , do
    if (2i+2-j) <n, then GT( q[j], q [2i + 2 - j] );
    if (2i+1-j) <n, then SWAP( q[j-1], q[2i+1-j] );
```

(b)

```
for i = 0; i++; i <= n-2 do
  for j = 0; j++; j < i, do
    if j < n && (2i-j) < n, then SWAP( q[j], q[2i-j] );
  for j = 0; j++; j < i do
    if j < n && (2i-j) < n, then GT( q[j], q[2i-j] );
  for j = 0; j++; j < i+1 do
    if j < n && (2i+1-j) < n, then GT( q[j], q[2i+1-j] );
```

(c)

Figure 13: Generalized solution for optimal schemes of QFT: (a) n-qubit QFT on LNN; (b) n-qubit QFT on 2×N architecture where $N = n/2$; (c) n-qubit QFT on 2×N architecture where swaps and CNOT cannot be mixed in one cycle.

takes to find the optimal solution. We implemented the mapper using C++ and it was running on Intel Xeon E5-2620v2 CPU.

6.1.3 Comparison with OLSQ. In Table 2 we show our results compared against OLSQ's depth-optimal results [18] on the benchmarks used in that paper. We correctly find the same optimal depths, but are able to do so around 9 to 1500 times faster depending on the benchmark. For this experiment, swaps were treated as having a latency of 3 cycles, and all other gates as having a latency of 1 cycle. When using our program we first tried to find an initial mapping that could satisfy all CNOTs in the circuit without swaps – if that failed, then we reran our mapper program with pure initial swaps allowed, and added together the times in our overhead column of Table 2. Our output circuit includes both a selected initial mapping and inserted swaps using the approach defined in Section 5.3.

6.2 Approximate Analysis

We relax our model to solve for large benchmarks. We aim to find a good solution within reasonable amount of time while not sacrificing the search quality too much.

We approximate it in the following ways. When an original gate is ready to execute with respect to dependence and coupling constraints, we immediately schedule it. Thus we eliminate the expanded nodes which do not schedule all ready original gates implied by their parent node (the state node).

We also reduce the number of expanded nodes by not allowing swaps that cause the executable gates on the CNOT frontier not executable. By executable we meant the coupling and dependence constraints are resolved. We rank the expanded nodes and only push the top-k into the priority queue. When the priority queue size reaches a threshold q, we cut it by a fixed number v through deleting the nodes that made the least progress in the circuit. If we need a tie breaker, we just rank them by the cost function. We choose the parameters k, q, and v as 10, 2000, and 1000.

We handle initial mapping on-the-fly in a greedy manner. Before calculating the cost of a node, we look at the qubits in each of its CNOT gates that are executable with respect to dependence constraints: if one or both of its qubits are not yet mapped, then we pick an assignment that minimizes their physical distance. If at the end of the program there are any qubits that were never

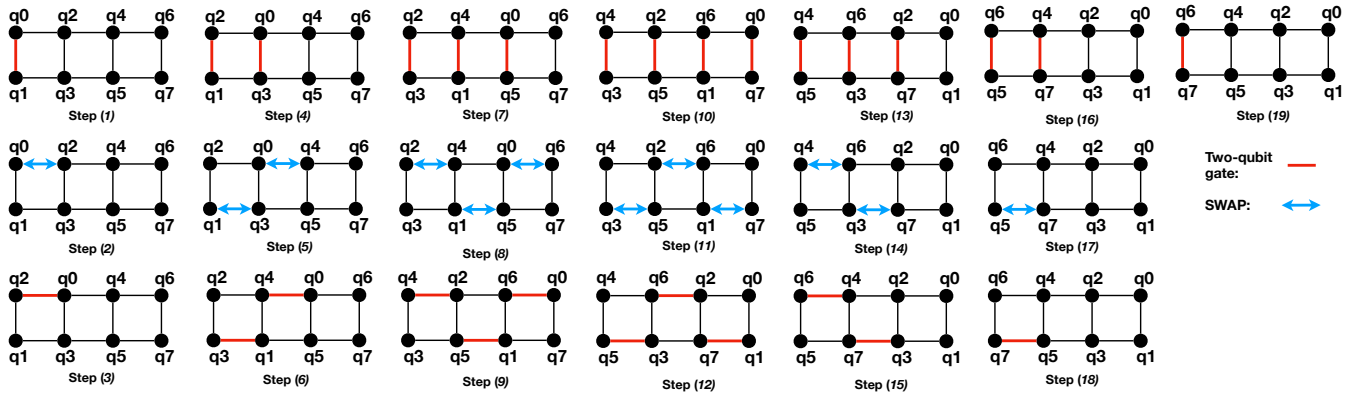

Figure 14: An alternative optimal scheme for QFT-8 with 2×4 qubits. Each sub-figure represents a step of the execution, which takes one cycle. It also represents the state of the circuit at each cycle. There are in total 19 steps, and thus 19 cycles.

Table 1: Summary of optimal analysis on Wille's [20] benchmarks for IBM QX2 architecture, with swap latency of 6 cycles and CX latency of 2 cycles; n denotes the number of qubits; Mapper Overhead, measured in seconds, is how long it took to generate the mapping.

Name	n	Gate Count	Ideal Cycle	Optimal Cycle	Mapper Overhead (s)
3_17_13	3	36	39	39	0.012
4gt11_82	5	27	38	40	0.044
4gt11_84	5	18	19	19	0.011
4gt13_92	5	66	64	64	0.014
4mod5-v0_19	5	35	37	45	0.075
4mod5-v0_20	5	20	21	27	0.052
4mod5-v1_22	5	21	22	28	0.053
4mod5-v1_24	5	36	36	42	0.085
alu-v0_27	5	36	35	40	0.043
alu-v1_28	5	37	37	42	0.029
alu-v1_29	5	37	36	41	0.052
alu-v2_33	5	37	36	41	0.036
alu-v3_34	5	52	53	59	0.314
alu-v3_35	5	37	37	42	0.038
alu-v4_37	5	37	37	42	0.038
ex-1_166	3	19	21	21	0.013
ham3_102	3	20	24	24	0.013
miller_11	3	50	52	52	0.016
mod5d1_63	5	22	24	34	0.076
mod5mils_65	5	35	37	46	0.115
qft_4	4	6	10	16	0.035
rd32-v0_66	4	34	36	41	0.045
rd32-v1_68	4	36	36	41	0.042

mapped (due to never being used in CNOT gates), then we assign them arbitrarily.

This method scales better than the optimal search method. It is non-optimal, but in practice it performs well.

6.2.1 Experiment Results. We evaluate our non-optimal mapper with benchmarks selected from RevLib [21], IBM Qiskit [1], and ScaffCC [6]. We also provide the ideal time of the circuit, which assumes an all-to-all qubit connection in the hardware. The information of these benchmarks is provided in Table 3.

Table 2: Comparison of our results against OLSQ's depth-optimal results; We let each gate take 1 cycle as the setup of OLSQ [18]. Mapper overhead is measured in seconds. OLSQ is using a different CPU for qubit mapper implementation which is Intel Xeon E5-2699v3.

Name	Arch	Ideal Cycle	OLSQ		Ours	
			Cycle	Overhead	Cycle	Overhead
4gt13_92	ibmqx2	38	38	145.74	38	0.01
4mod5-v1_22	grid2by3	12	20	90.20	20	0.64
4mod5-v1_22	grid2by4	12	20	151.28	20	17.35
4mod5-v1_22	ibmqx2	12	15	21.60	15	0.03
adder	grid2by3	11	11	10.95	11	0.03
adder	grid2by4	11	11	13.45	11	0.01
adder	ibmqx2	11	15	39.71	15	0.06
mod5mils_65	ibmqx2	21	24	87.76	24	0.05
or	ibmqx2	8	8	3.55	8	0.01
qaoa5	ibmqx2	14	14	10.41	14	0.01
queko_05_0	aspen-4	5	5	68.89	5	0.01
queko_10_3	aspen-4	10	10	592.91	10	1.02
queko_15_1	aspen-4	15	15	4912.35	15	26.70

We compare our work with two best known qubit mapping solutions [23] (denoted as *Zulehner*) and the Sabre qubit mapper from [8] (denoted as *Sabre*). It is worth mentioning that our approach can take any gate latency as input parameters and generate transformed circuits based on the input. To make evaluation results as close to real machines as possible. We use the results from the study by [4], where different types of quantum architecture are investigated, and the study reveals that two-qubit gate usually takes twice as much time as single-qubit gate. Hence we let single-qubit gate take 1 cycle and two-qubit CNOT gate take 2 cycles in our experiments. We also let a SWAP take 6 cycles as the IBM architecture uses bidirectional link and 3 CNOTs to implement one SWAP. The time is reported as the total number of executed cycles. We use IBM's 20-qubit Q20 Tokyo architecture [8] as the underlying quantum hardware.

Our approach is scalable up to hundreds of thousands of gates. Results are shown in Table 3. The results are not optimal, but still show significant advantages over the state-of-the-art qubit mappers. It reduces the execution time of the transformed quantum circuits. Speedup ranges from 0.99X to 1.36X and the average is 1.21X. The average speedup of our scheme over Sabre is 1.23X and the average speedup of our scheme over Zulehner is 1.18X.

Table 3: Summary results of approximate analysis on large benchmarks; n denotes the number of qubits; Circuit time is calculated in the unit of cycles.

Benchmark				Cycle		
Name	n	Gate Count	Ideal Cycle	Sabre	Zulehner	**Ours**
cm82a_208	8	650	571	752	1011	759
rd53_251	8	1291	1203	1961	1956	1779
urf2_277	8	20112	19698	40533	36500	31090
urf1_278	9	54766	53256	105984	95763	83226
hwb8_113	9	69380	64758	119930	115767	93357
urf1_149	9	184864	172518	335230	303697	264752
qft_10	10	200	97	226	193	181
rd73_252	10	5321	4829	9194	8431	7267
sqn_258	10	10223	9176	18055	16552	13845
z4_268	11	3073	2756	5250	5117	4271
life_238	11	22445	20867	39340	37944	33366
9symml	11	34881	32084	63339	56413	48606
sqrt8_260	12	3009	2779	5645	4831	4457
cycle10_2	12	6050	5662	10972	10659	9605
rd84_253	12	13658	12176	24860	23357	18225
adr4_197	13	3439	3088	5732	6005	4704
root_255	13	17159	14799	29511	27269	23841
dist_223	13	38046	32968	66791	62879	54905
cm42a_207	14	1776	1574	2473	2857	2186
pm1_249	14	1776	1574	2591	2857	2186
cm85a_209	14	11414	10630	19540	18393	16204
square_root	15	7630	6367	12374	11922	9311
ham15_107	15	8763	8092	15388	13767	12341
dc2_222	15	9462	8759	16947	15266	12945
inc_237	16	10619	9790	18250	17610	14804
mlp4_245	16	18852	17258	31836	30285	27214

7 RELATED WORK

Early studies on qubit mapping problem focus on linear nearest neighbor architectures, that is when qubits are placed in a one dimensional grid, and one qubit has at most two neighbors. In this type of architecture, Shafei *et al.* [13] have modeled the qubit mapping problem as constraint solving problem and use satisfiability (SAT) solvers to solve for qubit mapping. It works well when the number of qubits is small and the search space is small. Maslov [9] has obtained and proved optimal qubit mapping for the quantum fourier transform (QFT) algorithm for LNN.

As quantum computers with more complex connectivity structure are built, a larger number of studies investigate the qubit mapping problem. However, most of these studies [8, 16, 17, 20, 22, 23] focus on minimizing the number of inserted swap gates and enhancing the parallelism among the swaps, but not the time of the entire circuit. Zulehner *et al.* [23] proposes a systematic A* algorithm for optimizing the number of swap gates for a fixed layer of CNOT gates that need to run concurrently. It pre-processes the circuit by partitioning the circuit into different layers, and solve the mapping problem layer by layer. Li *et al.* [8] use a frontier to keep track of the CNOT gates that cannot be scheduled on the fly and formulates a multi-objective function for ranking different SWAP insertion strategies. Li *et al.* [8] has briefly discussed the trade-off between the inserted SWAP number and the depth of the circuit, but not systematically addressed the time-optimal problem. Siraichi *et al.* [16] notes the similarity between the swap insertion problem and the subgraph isomorphism problem, which is essentially fitting a program qubit interaction graph into the physical qubit coupling graph. But they do not provide optimal solutions. The studies [10, 19] observed the variability of qubit error rates in IBM quantum computer and develop variability-aware qubit mapping strategies.

The study that is most relevant to ours is OLSQ by Tan *et al.* [18]. OLSQ is a constrained based solver. It solves for the time coordinate of each gate (including swap gate) and the qubit mapping at every time coordinate, and the objective function is the total depth. It is optimal with respect to a depth upper-bound. It tests different upper bounds of the circuit depth until it finds a solution. If the preset depth upper-bound is smaller than the actual optimal depth, it will not return a solution. They start from the the longest weighted path T in the DAG, since a circuit runs at least T cycles. If it does not return a solution with depth $\leq$ T, it changes the upper bound to T+1. If with T+1, it is still unsatisfiable, it goes to T+2, T+3, and so on until a feasible solution is found. While their method can find optimal solutions, it may suffer from scalability issues when the optimal circuit time is not close to T. Therefore they geometrically increase T each time by $(1 + \epsilon)$x until an optimal solution is found. Our model explicitly solves for an optimal solution and does not impose any constraints.

The study by Childs *et al.* [3] aims to minimize the depth. But it minimizes the depth of inserted swaps. Each set of co-running swaps is modeled as a graph matching (as no two parallel swaps share a qubit), then it tries to find a minimal sequence of matchings to achieve the desired permutation. Their theoretical model does not consider the parallelism between inserted swaps and original gates. Lao *et al.* [7] considers the parallelism between inserted swaps ad original gates, but their approach is not theoretically optimal.

8 CONCLUSION

The physical layout of contemporary quantum devices imposes limitations for mapping a high level quantum program to the hardware. It is critical to develop an efficient qubit mapper. Most existing studies aim to reduce the gate count but are oblivious to the time of the transformed circuit. This paper presents a time-optimal qubit mapping model scheme. Experiment results show that our proposed solution generates hardware-compliant circuits with minimal circuit time with much less overhead compared with state-of-the-art qubit mapping approaches.

ACKNOWLEDGEMENT

We thank Ali Javadi-Abhari for being our shepherd and the anonymous reviewers for their constructive comments. This work is supported by grants from Rutgers Research Council and NSF-CCF-1628401. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

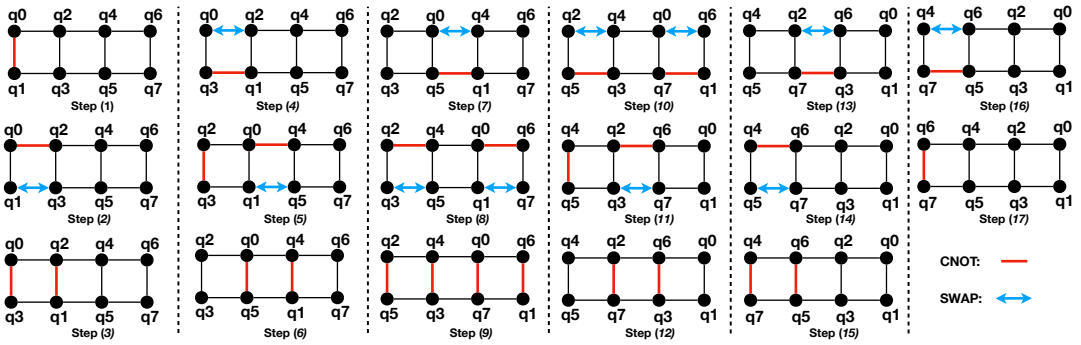

Figure 15: One possible solution for QFT-8 on 2×4 allowing two-qubit gate and swap on one cycle.

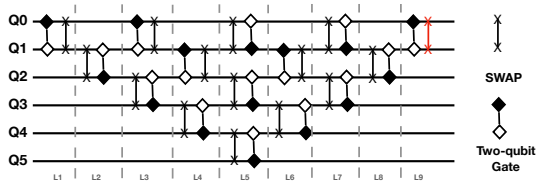

Figure 16: One possible solution for QFT-6 on LNN.

A PROOF OF OPTIMALITY

Lemma A.1. *For each state node P, the heuristic cost function $h(P)$ is a lower bound of the length of all paths from P to a terminal node in the search graph.*

Proof. Let $G_{rem} = (V_{rem}, E_{rem})$ be the dependency graph of the remaining circuit. For each gate $g \in V_{rem}$, let $t^*_{min}(g)$ be the length of the actual shortest path from P to any state node where gate g has already been scheduled to execute. In the next few paragraphs we will prove that $t_{min}(g) \le t^*_{min}(g)$ for every gate g. Let X be any terminal node in the search graph, and let t^* be the distance between P and X. Since each gate must finish execution at state X, for each gate g we have $t^* \ge t^*_{min}(g) + len(g)$. Together, for each gate g we have

$$t_{min}(g) + len(g) \le t^*_{min}(g) + len(g) \le t^*$$

Thus $h(P) = \max_g t_{min}(g) + len(g) \le t^*$. This shows that $h(P)$ is a lower bound on the length from P to any terminal node.

Base case: If g is a single-qubit gate that has no predecessors, then $t^*_{min}(g) = 0$, since it can be scheduled to execute immediately. Note that our definition of G_{rem} includes gates which are only partially executed, so gates which have no predecessors do not need to wait current gates to finish.

Inductive case: Suppose gate g depends on a number of other gates. If g is a two-qubit gate then we also have to consider potential SWAPs before g, even if g has no apparent predecessor in G_{rem}. We will give two lower bounds on $t^*_{min}(g)$ and show that $t_{min}(g)$ is equal to the larger of the two bounds.

The first bound is derived from the immediate predecessors of g. If $h_1, h_2, \cdots$ are the immediate predecessors of g in G_{rem}, then g cannot be scheduled until all these gates have finished. Thus $t^*_{min}(g) \ge t^*_{min}(h_i) + len(h_i)$ for each gate h_i. This gives the first bound

$$t^*_{min}(g) \ge \max_i t^*_{min}(h_i) + len(h_i)$$

The second bound comes from gates which operate on the some same qubit as g. Suppose g involves two qubits q_a, q_b. Let H_a be the set of all gates on q_a which are direct or indirect predecessors of g. Since they all operate on the same qubit, they must be executed in the order they appear in the circuit. Suppose this ordering is $h_1, h_2, \cdots, h_n$. We have

$$t^*_{min}(h_2) \ge t^*_{min}(h_1) + len(h_1)$$
$$t^*_{min}(h_3) \ge t^*_{min}(h_2) + len(h_2) \ge t^*_{min}(h_1) + len(h_1) + len(h_2)$$
$$\vdots$$

By induction we have $t^*_{min}(g) \ge t^*_{min}(h_1) + \sum_i len(h_i) \ge \sum_i len(h_i)$. Now suppose we place r SWAPs before gate g on qubit q_a. They also cannot execute simultaneously with any of the gates in H_a. Thus

$$t^*_{min}(g) \ge r \cdot len(SWAP) + \sum_{h \in H_a} len(h) = u_1$$

Similarly we may define the set H_b, and if we place s SWAPs on q_b before g we have

$$t^*_{min}(g) \ge s \cdot len(SWAP) + \sum_{h \in H_b} len(h) = u_2$$

Let $d(a, b)$ be the distance between qubit q_a, q_b in the qubit mapping π_{rem}. Then at least $d(a, b) - 1$ SWAPs are needed before gate g, so $r + s \ge d(a, b) - 1$. Since u_1, u_2 increases linearly with r, s, we fix $r + s = d(a, b) - 1$ to minimize them. We choose the pair (r, s) such that $\max\{u_1, u_2\}$ is minimized. We take the minimized $\max\{u_1, u_2\}$ to be the second bound.

In the above we have derived two lower bounds on $t^*_{min}(g)$. If we compare them with the definition of $t_{min}(g)$, we see that $t_{min}(g)$ is exactly equal to the larger of the two bounds. Therefore $t_{min}(g) \le t^*_{min}(g)$. This finishes the proof. □

B MULTIPLE OPTIMAL SOLUTIONS

In certain benchmarks, multiple depth-optimal solutions exist. Our tool can be easily tuned to find all of them. Our algorithm waits until the first optimal solution is found. Typically in A*, one terminate the algorithm as long as the first solution is found. But it is not necessary. We then record the depth of the first optimal solution,

and continue to run the extract-expand-push process as shown in Fig. 6 and report more solutions. We stop reporting solutions whenever an extracted node from the queue suggests a solution with a larger depth than the optimal one. At this time, our algorithm has found all solutions.

We need multiple optimal solutions because not all optimal solutions for small circuits have a recurring pattern. Hence, it is necessary to generate all optimal solutions and discover the one with recurring pattern to generalize to larger circuits. For instance, for QFT-8 on 2×4 architecture (without allowing CNOT and swap at the same cycle) only one solution among the eight optimal solutions shows the pattern in Fig. 14.

Another issue that arises when we try to manually generalize a solution is that the solution circuit might need slight transformation. It is possible that an optimal solution returned by our tool has cancelable swap gates (which usually involves multiple qubits and cannot be automatically found like cyclic swaps). However, it is easy to discover them when visualizing the small solution circuit.

Further, certain gates can be scheduled earlier or later without affecting the overall mapping or depth. We show an example in Fig. 15. As can be seen in step (5), the two-qubit gate for $\{q_2, q_3\}$ can be moved to step (6) without affecting the overall depth. Similarly, in step (11), the two-qubit gate or $\{q_4, q_5\}$ can be moved to step (12). This transformation is inferred from step (3) and (9). At this point, we are doing the generalization/inference of recurring patterns manually, but this could potentially done automatically. We leave it as our future work.

Last but not least, if a swap is followed by a two-qubit gate, the two-qubit can be moved in front of the swap by reversing the control and target, and the transformed circuit is equivalent. Similarly when a two-qubit is followed by a swap. We show a solution for QFT-6 on LNN in Fig. 16, where if in layers L2 to L8, the order of swap and two-qubit gate can be swapped to be consistent with L1 and L9, the entire solution is the same as we show in Fig. 2.

REFERENCES

[1] Héctor Abraham, AduOffei, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Gadi Alexandrowics, Eli Arbel, Abraham Asfaw, Carlos Azaustre, AzizNgoueya, Panagiotis Barkoutsos, George Barron, Luciano Bello, Yael Ben-Haim, Daniel Bevenius, Lev S. Bishop, Sorin Bolos, Samuel Bosch, Sergey Bravyi, David Bucher, Artemiy Burov, Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Ginés Carrascal, Adrian Chen, Chun-Fu Chen, Richard Chen, Jerry M. Chow, Christian Claus, Christian Clauss, Abigail J. Cross, Andrew W. Cross, Simon Cross, Juan Cruz-Benito, Chris Culver, Antonio D. Córcoles-Gonzales, Sean Dague, Tareq El Dandachi, Matthieu Dartiailh, DavideFrr, Abdón Rodríguez Davila, Anton Dekusar, Delton Ding, Jun Doi, Eric Drechsler, Drew, Eugene Dumitrescu, Karel Dumon, Ivan Duran, Kareem EL-Safty, Eric Eastman, Pieter Eendebak, Daniel Egger, Mark Everitt, Paco Martín Fernández, Axel Hernández Ferrera, Albert Frisch, Andreas Fuhrer, MELVIN GEORGE, Julien Gacon, Gadi, Borja Godoy Gago, Claudio Gambella, Jay M. Gambetta, Adhisha Gammanpila, Luis Garcia, Shelly Garion, Austin Gilliam, Juan Gomez-Mosquera, Salvador de la Puente González, Jesse Gorzinski, Ian Gould, Donny Greenberg, Dmitry Grinko, Wen Guan, John A. Gunnels, Mikael Haglund, Isabel Haide, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Stefan Hillmich, Hiroshi Horii, Connor Howington, Shaohan Hu, Wei Hu, Haruki Imai, Takashi Imamichi, Kazuaki Ishizaki, Raban Iten, Toshinari Itoko, JamesSeaward, Ali Javadi, Ali Javadi-Abhari, Jessica, Kiran Johns, Tal Kachmann, Naoki Kanazawa, Kang-Bae, Anton Karazeev, Paul Kassebaum, Spencer King, Knabberjoe, Arseny Kovyrshin, Rajiv Krishnakumar, Vivek Krishnan, Kevin Krsulich, Gawel Kus, Ryan LaRose, Raphaël Lambert, Joe Latone, Scott Lawrence, Dennis Liu, Peng Liu, Yunho Maeng, Aleksei Malyshev, Jakub Marecek, Manoel Marques, Dolph Mathews, Atsushi Matsuo, Douglas T. McClure, Cameron McGarry, David McKay, Dan McPherson, Srujan Meesala, Martin Mevissen, Antonio Mezzacapo, Rohit Midha, Zlatko Minev, Abby Mitchell, Nikolaj Moll, Michael Duane Mooring, Renier Morales, Niall Moran, MrF, Prakash Murali, Jan Müggenburg, David Nadlinger, Ken Nakanishi, Giacomo Nannicini, Paul Nation, Edwin Navarro, Yehuda Naveh, Scott Wyman Neagle, Patrick Neuweiler, Pradeep Niroula, Hassi Norlen, Lee James O'Riordan, Oluwatobi Ogunbayo, Pauline Ollitrault, Steven Oud, Dan Padilha, Hanhee Paik, Simone Perriello, Anna Phan, Francesco Piro, Marco Pistoia, Alejandro Pozas-iKerstjens, Viktor Prutyanov, Daniel Puzzuoli, Jesús Pérez, Quintiii, Rudy Raymond, Rafael Martín-Cuevas Redondo, Max Reuter, Julia Rice, Diego M. Rodríguez, RohithKarur, Max Rossmannek, Mingi Ryu, Tharmashastha SAPV, SamFerracin, Martin Sandberg, Hayk Sargsyan, Ninad Sathaye, Bruno Schmitt, Chris Schnabel, Zachary Schoenfeld, Travis L. Scholten, Eddie Schoute, Joachim Schwarm, Ismael Faro Sertage, Kanav Setia, Nathan Shammah, Yunong Shi, Adenilton Silva, Andrea Simonetto, Nick Singstock, Yukio Siraichi, Iskandar Sitdikov, Seyon Sivarajah, Magnus Berg Sletfjerding, John A. Smolin, Mathias Soeken, Igor Olegovich Sokolov, SooluThomas, Dominik Steenken, Matt Stypulkoski, Jack Suen, Shaojun Sun, Kevin J. Sung, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Soolu Thomas, Mathieu Tillet, Maddy Tod, Enrique de la Torre, Kenso Trabing, Matthew Treinish, TrishaPe, Wes Turner, Yotam Vaknin, Carmen Recio Valcarce, Francois Varchon, Almudena Carrera Vazquez, Desiree Vogt-Lee, Christophe Vuillot, James Weaver, Rafal Wieczorek, Jonathan A. Wildstrom, Robert Wille, Erick Winston, Jack J. Woehr, Stefan Woerner, Ryan Woo, Christopher J. Wood, Ryan Wood, Stephen Wood, Steve Wood, James Wootton, Daniyar Yeralin, Richard Young, Jessie Yu, Christopher Zachow, Laura Zdanski, Christa Zoufal, Zoufalc, a matsuo, adekusar drl, azulehner, bcamorrison, brandhsn, chlorophyll zz, dan1pal, dime10, drholmie, elfrocampeador, faisaldebouni, fanizzamarco, gadial, gruu, jliu45, kanejess, klinvill, kurarrr, lerongil, ma5x, merav aharoni, michelle4654, ordmoj, sethmerkel, strickroman, sumitpuri, tigerjack, toural, vvilpas, welien, willhbang, yang.luh, yelojakit, and yotamvakninibm. 2019. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2562110

[2] Mehdi Bozzo-Rey and Robert Loredo. 2018. Introduction to the IBM Q Experience and Quantum Computing. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering* (Markham, Ontario, Canada) *(CASCON '18)*. IBM Corp., USA, 410–412.

[3] Andrew M Childs, Eddie Schoute, and Cem M Unsal. 2019. Circuit transformations for quantum architectures. *arXiv preprint arXiv:1902.09102* (2019).

[4] Haowei Deng, Yu Zhang, and Quanxi Li. 2020. CODAR: A Contextual Duration-Aware Qubit Mapping for Various NISQ Devices. *arXiv preprint arXiv:2002.10915* (2020).

[5] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96)*. ACM, New York, NY, USA, 212–219. https://doi.org/10.1145/237814.237866

[6] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. 2014. ScaffCC: a framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*. ACM, 1.

[7] Lingling Lao, Hans van Someren, Imran Ashraf, and Carmen G. Almudever. 2020. Timing and resource-aware mapping of quantum circuits to superconducting processors. arXiv:1908.04226 [quant-ph]

[8] Gushu Li, Yufei Ding, and Yuan Xie. 2019. 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*. ACM, 1001–1014.

[9] Dmitri Maslov. 2007. Linear depth stabilizer and quantum Fourier transformation circuits with no auxiliary qubits in finite-neighbor quantum architectures. *Physical Review A* 76, 5 (Nov 2007). https://doi.org/10.1103/physreva.76.052310

[10] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. 2019. Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. ACM, New York, NY, USA, 1015–1029. https://doi.org/10.1145/3297858.3304075

[11] Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information.

[12] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. 2014. A variational eigenvalue solver on a photonic quantum processor. In *Nature Communications*, Vol. 5. 4213. https://doi.org/10.1145/237814.237866

[13] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. 2014. Qubit placement to minimize communication overhead in 2D quantum architectures. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 495–500.

[14] Peter W Shor. 1994. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 124–134.

[15] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (Oct. 1997), 1484–1509. https://doi.org/10.1137/S0097539795293172

[16] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintão Pereira. 2019. Qubit Allocation as a Combination of Subgraph Isomorphism and Token Swapping. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 120 (Oct. 2019), 29 pages. https://doi.org/10.1145/3360546

[17] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintão Pereira. 2018. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. ACM, 113–125.

[18] Bochen Tan and Jason Cong. 2020. Optimal Layout Synthesis for Quantum Computing. In *Proceedings of the 39th International Conference on Computer-Aided Design* (Virtual Event, USA) *(ICCAD '20)*. Association for Computing Machinery, New York, NY, USA, Article 137, 9 pages. https://doi.org/10.1145/3400302.3415620

[19] Swamit S. Tannu and Moinuddin K. Qureshi. 2019. Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. ACM, New York, NY, USA, 987–999. https://doi.org/10.1145/3297858.3304007

[20] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. 2019. 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*. ACM, 142.

[21] Robert Wille, Daniel Große, Lisa Teuber, Gerhard W Dueck, and Rolf Drechsler. 2008. RevLib: An online resource for reversible functions and reversible circuits. In *38th International Symposium on Multiple Valued Logic (ismvl 2008)*. IEEE, 220–225.

[22] Alwin Zulehner, Stefan Gasser, and Robert Wille. 2017. Exact Global Reordering for Nearest Neighbor Quantum Circuits Using A∗. In *International Conference on Reversible Computation*. Springer, 185–201.

[23] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2018. Efficient mapping of quantum circuits to the IBM QX architectures. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1135–1138.