IEEE Transactions on Quantum Engineering

Received 28 August 2024; revised 20 December 2024; accepted 6 January 2025; date of publication 8 January 2025; date of current version 5 February 2025.

Digital Object Identifier 10.1109/TQE.2025.3527399

Benchmarking Quantum Circuit Transformation With QKNOB Circuits

SANJIANG LI[1], XIANGZHEN ZHOU[2], AND YUAN FENG[3]

[1]Centre for Quantum Software and Information, Faculty of Engineering and Information Technology, University of Technology Sydney, Ultimo, NSW 2007, Australia
[2]Nanjing Tech University, Nanjing 210037, China
[3]Department of Computer Science and Technology, Tsinghua University, Beijing 100190, China

Corresponding author: Sanjiang Li (e-mail: sanjiang.li@uts.edu.au).

This work was supported in part by the Australian Research Council under Grant DP220102059, in part by the National Natural Science Foundation of China under Grant 12071271, Grant 12471437, and Grant 92465202, in part by the Young Scientists Fund of the Natural Science Foundation of Jiangsu Province under Grant BK20240536, and in part by the Innovation Program for Quantum Science and Technology under Grant 2021ZD0302901.

ABSTRACT Current superconducting quantum devices impose strict connectivity constraints on quantum circuit execution, necessitating circuit transformation before executing quantum circuits on physical hardware. Numerous quantum circuit transformation (QCT) algorithms have been proposed. To enable faithful evaluation of state-of-the-art QCT algorithms, this article introduces qubit mapping benchmark with known near-optimality (QKNOB), a novel benchmark construction method for QCT. `QKNOB` circuits have built-in transformations with near-optimal (close to the theoretical optimum) SWAP count and depth overhead. `QKNOB` provides general and unbiased evaluation of QCT algorithms. Using `QKNOB`, we demonstrate that `SABRE`, the default Qiskit compiler, consistently achieves the best performance on the 53-qubit IBM Q Rochester and Google Sycamore devices for both SWAP count and depth objectives. Our results also reveal significant performance gaps relative to the near-optimal transformation costs of `QKNOB`. Our construction algorithm and benchmarks are open-source.

INDEX TERMS Architecture, hardware/software co-design, performance optimization, placement, routing.

I. INTRODUCTION

Superconducting noisy intermediate-scale quantum devices impose strict connectivity constraints on quantum circuit execution. This necessitates a crucial compilation step known as quantum circuit transformation (QCT), also referred to as transpilation, qubit mapping, or layout synthesis. QCT adapts ideal circuits for execution on physical quantum devices by ensuring that two-qubit gate (like CNOT or cz) can only be performed between neighbouring qubits.

QCT has become a significant research focus in quantum computing [1], [2], [3], [4], [5], electronic design automation [6], [7], [8], [9], [10], [11], [12], [13], [14], and computer architecture [15], [16], [17], [18], [19]. This article refers to this procedure as, largely interchangeably, qubit mapping, and (quantum) circuit transformation.

Over the past several years, numerous QCT algorithms have been proposed for mapping ideal circuits to physical quantum devices [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28]. These algorithms typically involve constructing an initial mapping and inserting SWAP gates as needed to ensure that all two-qubit gates comply with the device's connectivity constraints. The cost of a transformation is usually measured by the *SWAP count* (number of inserted SWAP gates) or the depth overhead (increase in circuit depth). Determining whether a transformation exists with a SWAP count or depth overhead below a given threshold has been proven to be nondeterministic polynomial time (NP)-complete [20], [29]. Consequently, exact algorithms [3], [5], [21], [30] are often computationally intractable for circuits with more than approximately 10 qubits. Therefore, most QCT algorithms are heuristic.

Evaluating the performance of QCT algorithms is challenging, as their effectiveness depends on the target hardware architecture, input circuit structure, and optimization objectives. Benchmarking offers a systematic approach to comparing QCT algorithms through standardized circuits and evaluation metrics [29], [31], [32], [33], [34]. These benchmarks enable controlled performance assessments, ensuring fair comparisons and reproducible results, while also facilitating the identification of algorithmic strengths and weaknesses to guide future development.

While reversible circuit benchmarks like RevLib[1] and quantum circuits from QASMBench [31] and MQTBench [32] are valuable for assessing scalability and practical applicability, they lack known optimal transformation costs, hindering the evaluation of how closely a given transformation approximates the theoretical optimum. To address this issue, the QUEKO benchmark [29] was introduced, featuring circuits with zero-cost optimal transformations. However, QUEKO circuits may not be generalizable to typical QCT scenarios, potentially leading to skewed evaluation, particularly for algorithms relying on subgraph isomorphism for initial mappings (e.g., [23], [24]). These limitations underscore the need for more versatile and representative benchmarking frameworks.

To address these challenges, this article introduces qubit mapping benchmark with known near-optimality (QKNOB), a novel framework for evaluating QCT algorithms. The construction of QKNOB circuits is based on theoretical guarantees provided by Theorems 1 and 2. Theorem 1 demonstrates that any circuit transformation can be represented as a "partition-and-permute" process, where the circuit is partitioned into subcircuits and transformed via an initial mapping followed by a sequence of permutations. The transformation cost is determined by the number of SWAP gates required to implement these permutations. QKNOB circuits are constructed by reversing this process: starting with subcircuits derived from subgraph-guided selections, we link them with restricted permutations. This construction, guaranteed by Theorem 2, ensures that each QKNOB circuit has a built-in transformation cost determined by the number of SWAP gates required to realize the permutations. To ensure near-optimality, the construction method restricts the types of permutations used, such as those achievable with up to two consecutive SWAPs (for SWAP optimality) or parallel SWAP operations (for depth optimality). This restriction enables the generation of benchmarking circuits that are both representative of realistic quantum computing scenarios and adaptable to the connectivity constraints of various quantum devices.

For SWAP count and depth optimality evaluation, we generated QKNOB circuits for three representative quantum devices: IBM Q Tokyo (20 qubits), IBM Q Rochester (53 qubits), and Google Sycamore (53 qubits). The construction process carefully selects interaction graphs of subcircuits, subgraph embeddings, and permutations to match the unique connectivity constraints of these devices, as will be explained shortly. Using QKNOB alongside QUEKO circuits, we evaluated five state-of-the-art QCT algorithms. Among these algorithms, SABRE [15], the default Qiskit compiler, consistently demonstrated the best performance across both objectives and all three devices. Unlike QUEKO, QKNOB benchmarks provided more faithful and unbiased evaluation. In addition, our evaluation revealed significant gaps between the built-in transformation costs of QKNOB circuits and the performance of even the best QCT algorithms.

[1]https://www.revlib.org/

The rest of this article is organized as follows. Section II introduces the necessary background on quantum circuits, graphs, and permutations, while Section III discusses QCT process and algorithms. The theoretical foundations and design principles behind QKNOB are presented in Sections IV and V. In Section VI, we evaluate state-of-the-art QCT algorithms on both QKNOB and QUEKO benchmark sets, providing comparative insights. Section VII addresses QKNOB's scalability and limitations. Finally, Section VIII concludes this article. Proofs for the lemmas and theorems are included in the Appendix.

II. PRELIMINARIES

This section covers relevant background on quantum circuits, subgraph isomorphism, and permutations, essential for describing and formalizing the construction methods used in this work.

A. QUANTUM CIRCUITS

Quantum circuits are the standard model for describing quantum algorithms. Like classical combinational circuits, a quantum circuit consists of a sequence of quantum gates acting on qubits (quantum bits). A general state of qubit q has the form $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ with α_0, α_1 being complex values and $|\alpha_0|^2 + |\alpha_1|^2 = 1$.

Quantum gates are unitary transformations. An n-qubit gate is represented as a $2^n \times 2^n$ unitary matrix. The following one-qubit gates are often used:

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

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}.$$

CNOT (also called cx) and cz are two-qubit gates. For any computational basis state $|i\rangle|j\rangle$, CNOT and cz map $|i\rangle|j\rangle$ to, respectively, $|i\rangle|i \oplus j\rangle$ and $(-1)^{i \cdot j}|i\rangle|j\rangle$, where $\oplus$ denotes exclusive-or and $\cdot$ denotes Boolean conjunction.

One-qubit gates and CNOT are sufficient to implement an arbitrary quantum gate. In addition, any quantum gate can be approximated with arbitrary precision using only H, S, T, and CNOT gates. In particular, the two-qubit SWAP gate, which maps $|i\rangle|j\rangle$ to $|j\rangle|i\rangle$, can be implemented by three CNOT gates, i.e., $\text{SWAP}(p, q) = \text{CNOT}(p, q)\text{CNOT}(q, p)\text{CNOT}(p, q)$.

While different quantum devices may support different universal sets of quantum gates, in practice, the two-qubit gate in such a universal set is CNOT or cz. As the actual functionality of a one-qubit gate plays no role in QCT, in the rest of this article, we denote a one-qubit gate simply by the qubit it acts on. For example, an H gate on q_i is simply denoted by $\langle q_i \rangle$. Analogously, we write a CNOT or cz gate with control qubit q_i and target qubit q_j simply as $\langle q_i, q_j \rangle$.

A circuit C is usually represented as a sequence of gates $g_0, g_1, \ldots, g_{M-1}$, but this sequence does not necessarily reflect execution order. Gates acting on distinct qubits may be

IEEE Transactions on Quantum Engineering

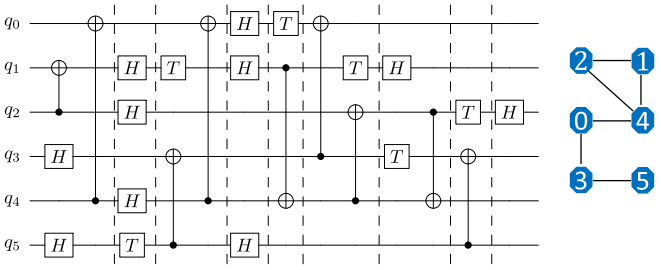


FIGURE 1. (Left) Quantum circuit and (right) its interaction graph.

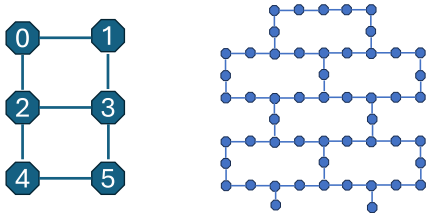


FIGURE 2. (Left) AG of *Grid*(3, 2) and (right) IBM Q Rochester.

executed in parallel. Naturally, we partition C into layers, scheduling each gate as early as possible. The *depth* of a circuit is the number of its layers.

For example, the circuit in Fig. 1 has a depth of 9, with its first layer containing gates $\langle 5\rangle$, $\langle 2, 1\rangle$, $\langle 4, 0\rangle$, $\langle 3\rangle$.

B. GRAPHS AND SUBGRAPH ISOMORPHISM

Graphs naturally arise in QCT. A quantum device is represented as an undirected architecture graph (AG) $G = (V, E)$, where vertices in V are physical qubits and edges in E represent allowed two-qubit interactions. That is, $(p, q) \in E$ if and only if a two-qubit gate can be applied to qubits p and q. Since G is undirected, $(p, q) \in E$ if and only if $(q, p) \in E$. Fig. 2 shows the AG of $Grid(3, 2)$ (an artificial device) and IBM Q Rochester. Each quantum circuit also induces an *interaction graph*.

Definition 1 (Interaction Graph): Let C be a quantum circuit on qubit set Q. The interaction graph (Q, E) of C is defined as: for all $p, q \in Q$, $(p, q) \in E$ if and only if $\langle p, q\rangle$ or $\langle q, p\rangle$ is in C.

The well-known subgraph isomorphism problem plays a critical role in QCT (see Section III). Specifically, if a subgraph isomorphism f exists that embeds the circuit's interaction graph into the device's AG, the circuit can be executed directly using f, eliminating the need for SWAP insertions.

Definition 2 (Subgraph Isomorphism): Given two graphs $G_i = (V_i, E_i)$ $(i = 1, 2)$, we say G_1 is a subgraph of G_2 if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. G_1 is *embeddable* into G_2 if there is an injective mapping $f : V_1 \to V_2$ such that $(f(p), f(q)) \in E_2$ for any edge $(p, q) \in E_1$. In this case, f is called a *subgraph isomorphism* or an *embedding*.

Subgraph isomorphisms can be found or disproved by algorithms like VF2 [35]. A subgraph isomorphism can also be quickly disproved by identifying a property that G_1 has but G_2 lacks (e.g., the presence of a 3-cycle). A 3-cycle in a graph G is a path (v_0, v_1, v_2, v_3) of length 3 in G such that $v_0 = v_3$ while v_0, v_1, v_2 are pairwise different.

Consider the circuit C in Fig. 1 (left). Its interaction graph contains a 3-cycle (2,1,4,2) [see Fig. 1 (right)]. Because $Grid(3, 2)$ has no 3-cycles, the interaction graph cannot be embedded in $Grid(3, 2)$.

C. PERMUTATIONS AND SWAP CIRCUITS

In this article, we refer to the circuit before transformation as a logical circuit and the transformed circuit as a physical circuit. Similarly, qubits in a logical (physical) circuit are referred to as logical (physical) qubits. Note that the term "logical" used in QCT should not be confused with its use in error correction.

Our benchmark circuit construction relies heavily on SWAP circuits (i.e., circuits consisting of SWAP gates) and permutations of physical qubits on the AG. Permutations, implemented via SWAP circuits, modify the mappings from logical to physical qubits; this modification through permutation is essential for constructing circuits with known transformation costs.

Let $G = (V, E)$ be an undirected graph. Assume $V = [n] \triangleq \{0, 1, \ldots, n-1\}$. A *permutation* $\pi : V \to V$ is a bijection on V. For example, the identity mapping id_V is a permutation; a SWAP operation on an edge (i, j) of G induces a permutation $\pi_{i,j}$ which maps i to j and j to i, leaving other vertices unchanged. Permutations can be composed to form more complicated permutations.

Formally, a permutation π is implementable by a sequence of c SWAPs $\pi_{p_1,q_1}, \ldots, \pi_{p_c,q_c}$ if $\pi = \pi_{p_c,q_c} \circ \cdots \circ \pi_{p_1,q_1}$, where (p_i, q_i) is an edge in G for $1 \le i \le c$. Any permutation π on a connected graph G can be implemented by a sequence of SWAPs on edges of G. We denote by $\|\pi\|$ the minimum number of SWAPs required to implement π and call this the *SWAP cost* of (implementing) π.

Since $V = [n]$, we represent each permutation π on V as the vector $(\pi(0), \pi(1), \ldots, \pi(n-1))$. For example, $\pi = (3, 0, 2, 1, 4, 5)$ denotes the permutation on $V = [6]$ that maps 0 to 3, 1 to 0, 3 to 1 and leaves the other vertices unchanged. We can implement π by first swapping 0 and 1 and then swapping 1 and 3. That is, $\pi = \pi_{1,3} \circ \pi_{0,1}$. Note that permutation composition is not commutative. For example, $\pi_{0,1} \circ \pi_{1,3} = (1, 3, 2, 0, 4, 5) \neq \pi$.

Regarding QCT, for each edge (i, j) in an AG $G = (V, E)$, the permutation $\pi_{i,j}$ corresponds to the SWAP gate SWAP(i, j).

Next, we explain how a permutation on G can be used to modify a logical-to-physical qubit mapping using SWAP gates. Suppose C is a quantum circuit on qubit set Q. For convenience, we assume $Q \subseteq V$. Let $\sigma : Q \to V$ be the current (logical to physical) mapping. Applying a SWAP gate SWAP(i, j) transforms σ into a new mapping $\sigma' = \pi_{i,j} \circ \sigma$, where $\sigma'(p) = j$ if $\sigma(p) = i$, $\sigma'(p) = i$ if $\sigma(p) = j$, and $\sigma'(p) = \sigma(p)$ otherwise. In general, for a SWAP circuit $S \triangleq (\text{SWAP}(p_1, q_1), \ldots, \text{SWAP}(p_c, q_c))$, S transforms σ into

$\sigma' \triangleq \pi_{p_c,q_c} \circ \cdots \circ \pi_{p_1,q_1} \circ \sigma$. In this case, we say that $\pi \triangleq \pi_{p_c,q_c} \circ \cdots \circ \pi_{p_1,q_1}$ is implemented by S. Therefore, any permutation π on G can be implemented by a SWAP circuit to modify a logical-to-physical mapping. We denote by $[\![\pi]\!]$ a SWAP circuit that implements π with a minimal number of SWAP gates.

The following lemma formalizes how the concatenation of SWAP circuits implements the composition of their corresponding permutations on the AG.

Lemma 1: Let π_1, π_2 be two permutations on a graph $G = (V, E)$. Suppose S_i is a SWAP circuit that implements π_i for $i = 1, 2$. Then, $S_1 + S_2$ implements $\pi_2 \circ \pi_1$, where "+" denotes circuit concatenation.

When constructing QKNOB circuits, we often permute subgraphs and circuits. These procedures formalizes how subgraphs and circuits are rearranged during circuit construction, ensuring that the relationships between qubits and connectivity constraints are accurately represented.

Given a graph G, a permuted graph is obtained by rearranging its nodes.

Definition 3 (Permuted Subgraph): Let $G = (V, E)$ be an undirected graph and $G_1 = (V_1, E_1)$ a subgraph of G. The permutation of G_1 under a permutation π on V, denoted $\pi(G_1)$, is the graph $(\pi(V_1), \pi(E_1))$, where $\pi(V_1) \triangleq \{\pi(v) \mid v \in V_1\}$ and $\pi(E_1) \triangleq \{(\pi(v), \pi(v')) \mid (v, v') \in E_1\}$.

Analogously, a permuted circuit is obtained by rearranging its qubits.

Definition 4 (Permuted Circuit): For a circuit $C \triangleq (g_1, \ldots, g_M)$ on qubits in V, the permutation of C under a permutation π on V is $\pi(C) \triangleq (\pi(g_1), \ldots, \pi(g_M))$, where $\pi(g)$ is the same gate as g but operates on

1) qubit $\pi(q_i)$ if g is a one-qubit gate on q_i,
2) qubits $\pi(q_i)$ and $\pi(q_j)$ if g is a two-qubit gate on q_i, q_j.

The following lemma establishes the reversibility of permutations on circuits and their distributivity over circuit concatenation, which are key properties used in the construction and analysis of QKNOB circuits.

Lemma 2: For a permutation π and circuits C, C_1, C_2, we have $\pi^{-1}(\pi(C)) = C$ and $\pi(C_1 + C_2) = \pi(C_1) + \pi(C_2)$.

III. QUANTUM CIRCUIT TRANSFORMATION

QCT is crucial for compiling quantum algorithms to execute on quantum devices with limited connectivity. This section outlines the fundamental steps involved in QCT, including mapping transformations and gate executions, and provides a theoretical framework for understanding the cost and structure of these transformations, focusing on core QCT components and their relevance to QKNOB circuit construction.

A. OVERVIEW OF QCT

When designing a quantum algorithm using the quantum circuit model, the designer often does not consider a specific quantum device. Consequently, the resulting ideal quantum circuit may contain two-qubit gates acting on arbitrary qubit pairs, which may violate hardware connectivity constraints. Therefore, QCT is necessary.

Let $\mathbb{AG} \triangleq (\mathbb{V}, \mathbb{E})$ be the AG of a given quantum device. Device-supported one-qubit gates can be executed directly on $\mathbb{AG}$. A device-supported two-qubit gate is directly executable on $\mathbb{AG}$ if its two qubits are adjacent in $\mathbb{AG}$. A quantum circuit C is referred to as an $\mathbb{AG}$-circuit if all its two-qubit gates are directly executable given the device's connectivity constraints.

If C is not an $\mathbb{AG}$-circuit, transformation is necessary for execution. Let Q be the qubit set of C, assuming w.l.o.g. that $Q \subseteq \mathbb{V}$. To execute C on $\mathbb{AG}$, we first construct an *initial mapping* $\sigma_1 : Q \to \mathbb{V}$. If all gates in C are executable under σ_1 (i.e., for every two-qubit gate $\langle u, v \rangle$ in C, $(\sigma_1(u), \sigma_1(v)) \in \mathbb{E}$), we say that C is executable on $\mathbb{AG}$ under σ_1. This occurs if and only if $\sigma_1(C)$ is an $\mathbb{AG}$-circuit.

If C is not executable under σ_1, QCT alternates between two key procedures.

1) *Mapping transformation:* Modify the current mapping by inserting SWAP gates.
2) *Gate execution:* Remove gates from the logical circuit that become executable under the updated mapping.

The process repeats until all gates in C have been removed.

B. TRANSFORMATION BY PARTITIONING AND PERMUTING

A straightforward, though not necessarily optimal, approach to QCT (see [23], [24]) involves partitioning the input circuit C into nonempty subcircuits $C_1, \ldots, C_s$, such that each C_i is executable under a specific injective mapping $\sigma_i : Q_i \to \mathbb{V}$, where $Q_i \subseteq Q$ is the set of qubits in C_i. Each transformed subcircuit, denoted $\widetilde{C}_i \triangleq \sigma_i(C_i)$, becomes an $\mathbb{AG}$-circuit. Since $C_i = \sigma_i^{-1}(\widetilde{C}_i)$ for $1 \le i \le s$, we have

$$C = C_1 + \cdots + C_s = \sigma_1^{-1}(\widetilde{C}_1) + \cdots + \sigma_s^{-1}(\widetilde{C}_s) \quad (1)$$

where each σ_i^{-1} is defined on $\sigma_i(Q) \subseteq \mathbb{V}$.

The transformation and execution proceed as follows.

1) Initial Mapping Application: Apply σ_1 to C. This transforms in particular the first subcircuit into the $\mathbb{AG}$-circuit $\widetilde{C}_1$. We remove all gates in C_1 from C (since they are all executable), setting $PC_1 = \widetilde{C}_1$ for the current physical circuit and $LC_1 = C_2 + \cdots + C_s$ for the remaining logical circuit. We then append to PC_1 the SWAP circuit $[\![\sigma_1^{-1}]\!]$, which implements σ_1^{-1} with the minimal number of SWAPs. This does not change LC_1 but updates PC_1 to $\widetilde{C}_1 + [\![\sigma_1^{-1}]\!]$ and reverts the current mapping σ_1 to the identity mapping $id_{\mathbb{V}}$.

2) Subsequent Subcircuits: Suppose the current mapping is $id_{\mathbb{V}}$ and the logical and physical circuits are $LC_{i-1} = C_i + \cdots + C_s$ and PC_{i-1} for some $1 < i < s$. We append a SWAP circuit $[\![\sigma_i]\!]$ to PC_{i-1} immediately after $[\![\sigma_{i-1}^{-1}]\!]$. This changes the current mapping to σ_i and transforms the ith subcircuit C_i to $\widetilde{C}_i$. We append $\widetilde{C}_i$ to PC_{i-1}, obtaining

$$PC_i = \widetilde{C}_1 + [\![\sigma_1^{-1}]\!] + [\![\sigma_2]\!] + \cdots + \widetilde{C}_{i-1} + [\![\sigma_{i-1}^{-1}]\!] + [\![\sigma_i]\!] + \widetilde{C}_i.$$

IEEE Transactions on Quantum Engineering

The logical circuit is updated to $LC_i = C_{i+1} + \cdots + C_s$. Next, we append a SWAP circuit $[\![\sigma_i^{-1}]\!]$ to PC_i. This does not change LC_i but updates PC_i to $PC_i + [\![\sigma_i^{-1}]\!]$ and reverts the current mapping σ_i to $id_{\mathbb{V}}$.

3) Last Subcircuit: In the final step ($i = s - 1$), $LC_i = C_s$, and the current mapping is $id_{\mathbb{V}}$. We append a SWAP circuit $[\![\sigma_s]\!]$ to PC_{s-1}. This changes the current mapping to σ_s and transforms the last subcircuit C_s to $\widetilde{C}_s$. We remove all gates in C_s from LC_{s-1}, resulting an empty logical circuit. Accordingly, the final physical circuit is

$$PC_s = \widetilde{C}_1 + [\![\sigma_1^{-1}]\!] + [\![\sigma_2]\!] + \cdots + \widetilde{C}_{s-1} + [\![\sigma_{s-1}^{-1}]\!] + [\![\sigma_s]\!] + \widetilde{C}_s.$$

By Lemma 1, two consecutive SWAP circuits can be replaced with a single circuit. In particular, we can replace $[\![\sigma_i^{-1}]\!] + [\![\sigma_{i+1}]\!]$ with $[\![\sigma_{i+1} \circ \sigma_i^{-1}]\!]$. Thus, the final physical circuit can be written as

$$PC = \widetilde{C}_1 + [\![\sigma_2 \circ \sigma_1^{-1}]\!] + \cdots + \widetilde{C}_{s-1} + [\![\sigma_s \circ \sigma_{s-1}^{-1}]\!] + \widetilde{C}_s. \tag{2}$$

The transformation ensures equivalence between PC and C, up to an initial mapping σ_1 and a final mapping σ_s.

In summary, the logical circuit C has been transformed into the physical circuit PC by 1) applying the initial mapping σ_1 on C and 2) inserting SWAP circuit $S_i \triangleq [\![\sigma_{i+1} \circ \sigma_i^{-1}]\!]$ between C_i and C_{i+1} for $1 \le i \le s-1$. As the initial and final mappings incur no cost, the total transformation cost is the number of SWAPs used in the SWAP circuits $[\![\sigma_{i+1} \circ \sigma_i^{-1}]\!]$ for $1 \le i < s$, i.e., the number of SWAPs in $S_1 + \cdots + S_{s-1}$.

Key theoretical result: The above process provides a specific circuit transformation that can be represented as a sequence of subcircuits and permutations. Moreover, any arbitrary transformation of C into a physical circuit PC on $\mathbb{AG}$ can be achieved in the same manner. Specifically, let σ_1 be the initial mapping, and let $S_1, \ldots, S_{s-1}$ be the sequence of SWAP circuits inserted into C. We can partition the circuit C into subcircuits $C_1, \ldots, C_s$ as in (1) and represent PC as in (2), where σ_i ($1 \le i \le s$) are permutations such that $S_i = [\![\sigma_{i+1} \circ \sigma_i^{-1}]\!]$.

This leads to the following theorem.

Theorem 1: Let C be a logical circuit and $\mathbb{AG}$ the AG of a quantum device. For any transformation of C on $\mathbb{AG}$ with cost c, there exist a partition of C into $1 \le s \le c+1$ nonempty subcircuits $C_1, \ldots, C_s$ and s permutations σ_i ($1 \le i \le s$) on $\mathbb{AG}$ such that $\sum_{i=2}^{s} \|\sigma_i \circ \sigma_{i-1}^{-1}\| = c$, and for each $1 \le i \le s$, $\widetilde{C}_i \triangleq \sigma_i(C_i)$ is an $\mathbb{AG}$-circuit. Moreover, the transformed circuit has the form shown in (2).

The circuit transformation form described in Theorem 1 and shown in (2) is called the partition-and-permute transformation.

C. CIRCUIT TRANSFORMATION EXAMPLE

This section provides an example demonstrating how initial mapping and SWAP insertions resolve connectivity constraints and how the transformation can be expressed in the partition-and-permute form described in Theorem 1.

Let $\mathbb{AG}$ be the $Grid(3, 2)$ AG (see Fig. 2, left); and consider the logical circuit shown in Fig. 1. Suppose our target is to minimize the SWAP count. Because one-qubit gates can be executed directly, we remove them, leaving the logical circuit

$$C = [\langle 2,1\rangle, \langle 4,0\rangle, \langle 4,0\rangle, \langle 5,3\rangle, \langle 1,4\rangle, \langle 4,2\rangle, \langle 2,4\rangle, \langle 3,0\rangle, \langle 5,3\rangle].$$

Because C's interaction graph (see Fig. 1) contains a 3-cycle (2,1,4,2), it cannot be embedded into $\mathbb{AG}$; thus this circuit is not executable under any initial mapping. We next show that it can be transformed into an executable circuit with one SWAP.

First, we partition C into two subcircuits such that each subcircuit's interaction graph is embeddable in $\mathbb{AG}$. For example, let $C_1 = [\langle 2,1\rangle, \langle 4,0\rangle, \langle 4,0\rangle, \langle 5,3\rangle, \langle 1,4\rangle]$, and $C_2 = [\langle 4,2\rangle, \langle 2,4\rangle, \langle 3,0\rangle, \langle 5,3\rangle]$. Using a subgraph isomorphism algorithm, we find a mapping $\sigma_1 = (5, 1, 0, 4, 3, 2)$ that transforms C_1 into an $\mathbb{AG}$-circuit.

Note that σ_1 permutes the whole circuit as

$$\sigma_1(C) = [\langle 0,1\rangle, \langle 3,5\rangle, \langle 3,5\rangle, \langle 2,4\rangle, \langle 1,3\rangle, \langle 3,0\rangle, \langle 0,3\rangle, \langle 4,5\rangle, \langle 2,4\rangle].$$

Because all gates in $\sigma_1(C_1)$ act on neighbouring physical qubits in $\mathbb{AG}$, they are executable and therefore removed from $\sigma_1(C)$. The remaining circuit is $\sigma_1(C_2) = [\langle 3,0\rangle, \langle 0,3\rangle, \langle 4,5\rangle, \langle 2,4\rangle]$. Because $\langle 2,4\rangle$ and $\langle 4,5\rangle$ correspond to edges in $\mathbb{AG}$, we only need bring the logical qubit mapped to physical qubit 0 next to the logical qubit mapped to physical qubit 3 (or vice versa). This can be achieved by inserting the SWAP gate SWAP(0, 1), which implements the permutation $\pi_{0,1}$. Note that $\pi_{0,1}^{-1} = \pi_{0,1}$. Because $\pi_{0,1}(\sigma_1(C_2)) = [\langle 3,1\rangle, \langle 1,3\rangle, \langle 4,5\rangle, \langle 2,4\rangle]$ is an $\mathbb{AG}$-circuit, it can be executed directly.

In conclusion, to transform C, we apply an initial mapping σ_1 and then insert SWAP(0, 1). Because the SWAP cost is 1, the transformation is optimal.

Let $\sigma_2 = \pi_{0,1} \circ \sigma_1$. Then, the circuit C_i is executable under σ_i for $i = 1, 2$. By definition, $\sigma_2 \circ \sigma_1^{-1} = (\pi_{0,1} \circ \sigma_1) \circ \sigma_1^{-1} = \pi_{0,1}$. Consequently, the set $S_1 = [\![\sigma_2 \circ \sigma_1^{-1}]\!]$ consists of a single SWAP(0, 1). This construction explicitly relates the above transformation to the form described in Theorem 1.

D. CIRCUIT TRANSFORMATION IMPLEMENTATIONS

Many QCT algorithms have been developed. Some aim to find transformations with the optimal cost [3], [4], [5], [17], [20], [21], [22], [30], but these are applicable only to circuits with approximately ten or fewer qubits. Most algorithms employ heuristic search techniques [1], [2], [11], [13], [14], [15], [23], [24]. These heuristic algorithms can be further classified by their optimization objective. Some aim to maximize fidelity or minimize the error rate [6], [7], [17], [18]; many aim to reduce the SWAP count [11], [13], [14], [15], [24]; and others aim to reduce the output circuit's depth [2], [12], [19], [25] to respect the limited qubit coherence time.

Several QCT algorithms have used subgraph isomorphism. The BMT algorithm [23] combines subgraph isomorphism with token swapping and is closely related to the general transformation pattern as specified in (1). Based on exhaustive search, BMT consumes significant time and memory, making it unsuitable for circuits with 20 or more qubits. Using the subgraph isomorphism algorithm VF2 [35], FiDLS [24] selects an initial mapping that brings the input circuit's interaction graph closer to the AG. When circuits are directly executable, FiDLS can often find an embedding and transform the circuit without inserting any SWAPs. This is especially true for QUEKO circuits [29]. A recent work [28] also employs subgraph isomorphism in a divide-and-shuttle atom approach for qubit mapping on neutral-atom quantum devices.

Other works exploit commutation rules [8], [10], synthesis [9], gate optimizations [16], or remote CNOT [26] for circuit transformation. These techniques can be combined with most QCT algorithms. In this article, we do not consider these extensions.

Our evaluation focuses on five state-of-the-art QCT algorithms: t|ket⟩, SABRE, SAHS, MCTS, and the Qiskit transpiler Stochasticswap.

t|ket⟩ is a powerful quantum circuit compiler tool proposed by Quantinuum. First described in [2], the t|ket⟩ router attempts to select the SWAP operation which maximally reduces the interaction graph's diameter for the current layer. SABRE [15] adopts a three-fold strategy and a heuristic function that models the fitness of a mapping with the two-qubit gates in the first several layers of the circuit. Specifically, SABRE starts from a random initial mapping and transforms the input circuit C using the heuristic function; it then uses the final mapping as the initial mapping and transforms the reverse of C; and finally, it uses the final mapping of the second transformation as the initial mapping and transforms C again. This approach incorporates information about the entire circuit. The SAHS algorithm [13] first selects an initial mapping that best fits the input circuit C using the simulated annealing method and then, during routing, simulates the search process one step further, selecting the SWAP with the best subsequent SWAP to apply. The Monte Carlo Tree Search (MCTS) method for QCT, denoted MCTS, was first introduced in [11] for SWAP count optimization and extended in [12] for depth optimization. The core idea is to explore the search space in a balanced way. On average, MCTS-based QCT algorithms can search deeper and find better solutions.

IV. QKNOB CIRCUITS

This section describes our method for constructing QKNOB benchmarking circuits. These circuits are designed to address limitations in existing benchmarks [29] by providing instances with known transformation costs and varying complexities. Our procedure for constructing QKNOB circuits is the *reverse* of the partition-and-permute circuit transformation procedure described in Theorem 1 of Section III.

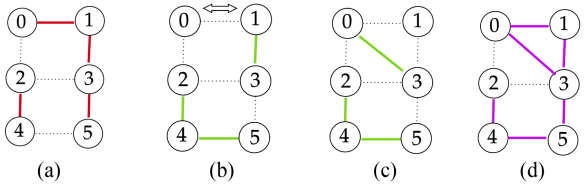


FIGURE 3. Strong glink $G_1 \overset{\pi_2}{\Rightarrow} G_2$. (a) Subgraph G_1. (b) Subgraph G_2 (c) Graph $\pi_2(G_2)$ obtained by permuting G_2 with $\pi_2 = \pi_{0,1}$. (d) Union of G_1 and $\pi_2(G_2)$.

A. QKNOB CIRCUITS WITH TWO SUBCIRCUITS

To illustrate the construction, consider the task of creating a benchmarking circuit with an optimal SWAP cost 1. If a circuit's interaction graph is not embeddable in $\mathbb{AG}$, its transformation cost is at least 1. We randomly generate subgraphs G_1, G_2 of $\mathbb{AG}$, and permute G_2 by the permutation $\pi_{i,j}$ for a randomly selected edge (i, j) of $\mathbb{AG}$.

We then generate $\mathbb{AG}$-circuits $\widetilde{C}_1$ and $\widetilde{C}_2$ with interaction graphs G_1 and G_2, respectively. Permuting $\widetilde{C}_2$ with $\pi_{i,j}$ yields a new circuit $\widetilde{C}_1 + \pi_{i,j}(\widetilde{C}_2)$, with an optimal cost of at most 1. To increase the generality of the generated circuit, we apply another permutation π_1 to $\widetilde{C}_1 + \pi_{i,j}(\widetilde{C}_2)$. The resulting circuit, $\pi_1(\widetilde{C}_1 + \pi_{i,j}(\widetilde{C}_2))$, is a QKNOB circuit with an optimal SWAP cost of at most 1. Equality holds when the union of G_1 and $\pi_{i,j}(G_2)$ is not embeddable in $\mathbb{AG}$.

The general case involves the following notion of a subgraph link (glink), which connects the interaction graphs of two subcircuits.

Definition 5 (Glink): Let $\mathbb{AG} = (\mathbb{V}, \mathbb{E})$ be an AG and $G_i = (V_i, E_i)$ $(i = 1, 2)$ its subgraphs. Suppose π is a permutation on $\mathbb{V}$. Define the graph $G' = (V', E')$, where

1) $V' = V_1 \cup \pi(V_2)$
2) $E' = E_1 \cup \pi(E_2)$, that is, for $u, v \in V'$, $(u, v) \in E'$ if and only if $(u, v) \in E_1$ or $(\pi^{-1}(u), \pi^{-1}(v)) \in E_2$.

We call $\langle G_1, \pi, G_2 \rangle$ a *subgraph link* (*glink* for short), denoted by $G_1 \overset{\pi}{\rightarrow} G_2$. When G' is not embeddable in $\mathbb{AG}$, we call $\langle G_1, \pi, G_2 \rangle$ a *strong glink*, denoted by $G_1 \overset{\pi}{\Rightarrow} G_2$.

Fig. 3 shows an example of strong glinks.

Starting from a glink $G_1 \overset{\pi_2}{\rightarrow} G_2$, we construct a circuit as follows: first, randomly generate $\mathbb{AG}$-circuits $\widetilde{C}_i$ for $i = 1, 2$ such that the $\mathbb{AG}$-subgraph G_i is the interaction graph of $\widetilde{C}_i$; second, apply a randomly generated permutation π_2 to $\widetilde{C}_2$ and concatenate it with $\widetilde{C}_1$; finally, apply another permutation π_1 to $\widetilde{C}_1 + \pi_2(\widetilde{C}_2)$. Let $C = \pi_1(\widetilde{C}_1) + (\pi_1 \circ \pi_2)(\widetilde{C}_2)$. Then, C is a QKNOB circuit constructed from the glink $G_1 \overset{\pi_2}{\rightarrow} G_2$.

We show that C has a built-in transformation. The following lemma guarantees that, before applying π_1, the circuit can be transformed into an $\mathbb{AG}$-circuit by inserting any SWAP circuit that implements π_2^{-1} between $\widetilde{C}_1$ and $\pi_2(\widetilde{C}_2)$.

Lemma 3: Let $G_1 \overset{\pi_2}{\rightarrow} G_2$ be a glink. Suppose $\widetilde{C}_i$ $(i = 1, 2)$ is a circuit whose interaction graph is G_i. Then, $\widetilde{C}_1 + \pi_2(\widetilde{C}_2)$ can be transformed into an $\mathbb{AG}$-executable circuit by

IEEE Transactions on Quantum Engineering

inserting between $\widetilde{C}_1$ and $\pi_2(\widetilde{C}_2)$ any SWAP circuit that implements π_2^{-1}.

Furthermore, because C is obtained by applying π_1 on $\widetilde{C}_1 + \pi_2(\widetilde{C}_2)$, it can be transformed by first applying the initial mapping $\sigma_1 \triangleq \pi_1{}^{-1}$ and then inserting any SWAP circuit implementing π_2^{-1} between $\widetilde{C}_1$ and $\pi_2(\widetilde{C}_2)$.

As a concrete example, we show that the circuit in Fig. 1 (and Section III-C) can be reconstructed as a QKNOB circuit.

Circuit in Fig. 1 as a QKNOB *Circuit:* Let $V_1 \triangleq \{0, 1, 2, 3, 4, 5\}$, $E_1 \triangleq \{(0, 1), (1, 3), (2, 4), (3, 5)\}$ and $V_2 \triangleq \{1, 2, 3, 4, 5\}$, $E_2 \triangleq \{(1, 3), (2, 4), (4, 5)\}$. Then, $G_i = (V_i, E_i)$ $(i = 1, 2)$ are two subgraphs of $\mathbb{AG} = Grid(3, 2)$; see Fig. 3 for illustration.

Let $\pi_2 \triangleq \pi_{0,1}$ be the permutation induced by swapping 0 and 1in $\mathbb{AG}$. Permuting G_2 with π_2, and letting G' be the union of G_1 and $\pi_2(G_2)$, G' has edges (0, 1), (1, 3), (2, 4), (3, 5), (0, 3), and (4,5) [see Fig. 3(d)]. Because G' contains a 3-cycle (0,1,3,0), it cannot be embedded in $\mathbb{AG}$. That is, we have a strong glink $G_1 \overset{\pi_2}{\Longrightarrow} G_2$.

Starting from G_1 and G_2, we construct two circuits, $\widetilde{C}_1$ and $\widetilde{C}_2$, whose interaction graphs are G_1 and G_2, respectively. For instance, let

$$\widetilde{C}_1 = [\langle 2\rangle, \langle 0, 1\rangle, \langle 1\rangle, \langle 3, 5\rangle, \langle 3\rangle, \langle 1\rangle, \langle 3, 5\rangle, \langle 4\rangle, \langle 2\rangle, \langle 2, 4\rangle, \langle 1\rangle, \langle 5\rangle, \langle 1, 3\rangle]$$

$$\widetilde{C}_2 = [\langle 0\rangle, \langle 1\rangle, \langle 2\rangle, \langle 3, 1\rangle, \langle 5\rangle, \langle 1, 3\rangle, \langle 1\rangle, \langle 1\rangle, \langle 4, 5\rangle, \langle 4\rangle, \langle 0\rangle, \langle 2, 4\rangle].$$

Then, $\widetilde{C}_1$ and $\widetilde{C}_2$ are two $\mathbb{AG}$-circuits.

Permuting $\widetilde{C}_2$ with $\pi_2 = \pi_{0,1}$ yields

$$\pi_2(\widetilde{C}_2) = [\langle 1\rangle, \langle 0\rangle, \langle 2\rangle, \langle 3, 0\rangle, \langle 5\rangle, \langle 0, 3\rangle, \langle 0\rangle, \langle 0\rangle, \langle 4, 5\rangle, \langle 4\rangle, \langle 1\rangle, \langle 2, 4\rangle].$$

Finally, we generate an arbitrary permutation π_1 and apply it to $\widetilde{C}_1 + \pi_2(\widetilde{C}_2)$. For example, let $\pi_1 = (2, 1, 5, 4, 3, 0)$. We have a QKNOB circuit

$$C \triangleq \pi_1\left(\widetilde{C}_1 + \pi_2(\widetilde{C}_2)\right) = \pi_1(\widetilde{C}_1) + (\pi_1 \circ \pi_2)(\widetilde{C}_2).$$

Let $C_1 = \pi_1(\widetilde{C}_1)$ and $C_2 = (\pi_1 \circ \pi_2)(\widetilde{C}_2)$. Then, $C = C_1 + C_2$.

The QKNOB circuit C is exactly the logical circuit shown in Fig. 1 and studied in Section III-C. As shown in Section III-C, starting from the initial mapping $\sigma_1 \triangleq \pi_1^{-1} = (5, 1, 0, 4, 3, 2)$, all gates in C_1 are executable and removed. After inserting a SWAP$(0, 1)$, the mapping $\sigma_1 = \pi_1^{-1}$ evolves to $\sigma_2 \triangleq (\pi_1 \circ \pi_2)^{-1} = \pi_2^{-1} \circ \pi_1^{-1} = \pi_2^{-1} \circ \sigma_1 = (5, 0, 1, 4, 3, 2)$, which transforms $\langle 4, 2\rangle$ to $\langle 3, 1\rangle$, $\langle 3, 0\rangle$ to $\langle 4, 5\rangle$, and $\langle 5, 3\rangle$ to $\langle 2, 4\rangle$. All gates in C_2 are now executable.

B. GENERAL QKNOB CIRCUIT CONSTRUCTION

To construct general QKNOB circuits, we generate multiple subcircuits and connect them using glinks. This process relies on the notion of a glink chain, which is a sequence of glinks defining the connections between subcircuits and determining the cost of the built-in transformation for the resulting circuit.

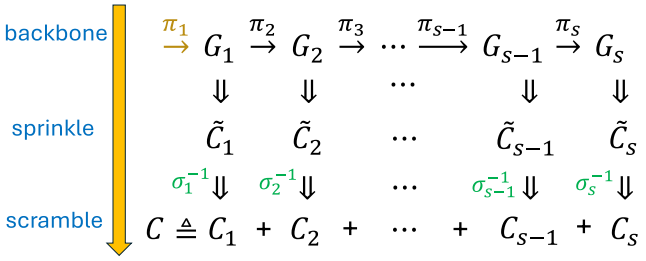


FIGURE 4. Construction of a QKNOB circuit C on an $\mathbb{AG}$, where π_i, G_i, $\widetilde{C}_i$, C_i represent, respectively, a permutation on $\mathbb{AG}$, a subgraph of $\mathbb{AG}$, an $\mathbb{AG}$-circuit, and a subcircuit of C. Additionally, $\sigma_i^{-1} = \pi_1 \circ \cdots \circ \pi_i$. The built-in transformation of C has initial mapping $\sigma_1 = \pi_1^{-1}$ and sequentially inserts SWAP circuits $[\![\pi_i^{-1}]\!]$ $(1 \le i \le s)$, where $[\![\pi_i^{-1}]\!]$ denotes any optimal SWAP circuit implementing π_i^{-1}.

Definition 6 (Glink chain): A glink chain is a sequence

$$G_1, \pi_2, G_2, \pi_3, \ldots, G_{s-1}, \pi_s, G_s$$

such that $G_i \xrightarrow{\pi_{i+1}} G_{i+1}$ for each $1 \le i < s$. A glink chain is *strong* if all its glinks are strong.

Given a glink chain $\langle G_1, \pi_2, G_2, \ldots, \pi_s, G_s\rangle$ and a permutation π_1, we generate a QKNOB circuit C using a three-phase construction process similar to QUEKO [29] (see Fig. 4) as follows.

1) For each i, generate a random circuit $\widetilde{C}_i$ such that its interaction graph is G_i.

2) Concatenate these circuits backward with the permutations

$$C = \pi_1\left(\widetilde{C}_1 + \pi_2\left(\widetilde{C}_2 + \pi_3\left(\cdots \pi_{s-1}(\widetilde{C}_{s-1} + \pi_s(\widetilde{C}_s))\cdots\right)\right)\right). \tag{3}$$

Note that each $\widetilde{C}_i$ constructed above is an $\mathbb{AG}$-circuit.

The following theorem guarantees that C has a built-in transformation with cost $\sum_{i=2}^{s} \|\pi_i\|$, which provides an upper bound for C's optimal transformation cost.

Theorem 2: Suppose C is a QKNOB circuit as in (3). Then, C can be transformed into an $\mathbb{AG}$-executable circuit by

1) taking π_1^{-1} as the initial mapping;
2) inserting, from left to right, a SWAP circuit S_i that implements π_{i+1}^{-1} after executing and removing gates in $C_i \triangleq (\pi_1 \circ \cdots \circ \pi_i)(\widetilde{C}_i)$ from C for $1 \le i < s$.

This built-in transformation has cost of $\sum_{i=2}^{s} \|\pi_i\|$.

The built-in transformation in Theorem 2 corresponds precisely to the partition-and-permute form stated in Theorem 1. Indeed, letting $\sigma_1 \triangleq \pi_1^{-1}$ and $\sigma_{i+1} \triangleq \pi_{i+1}^{-1} \circ \sigma_i$ for $1 \le i < s - 1$ (cf., Fig. 4), we see that $\pi_{i+1}^{-1} = \sigma_{i+1} \circ \sigma_i^{-1}$, $\widetilde{C}_i = \sigma_i(C_i)$, and these C_i, σ_i satisfy the conditions given in Theorem 1.

There is a potential issue with using arbitrary permutation to scramble an $\mathbb{AG}$-circuit: computing the exact value of $\|\pi\|$ can be difficult. In our benchmark construction for SWAP count optimality, we require each of $\pi_2, \ldots, \pi_s$ to be derived from a small number of SWAPs. The first permutation, π_1, can still be arbitrary because it does not contribute to the transformation cost. To ensure that the transformation cost is close to the optimal one, we further require that the union of G_i and $\pi_{i+1}(G_{i+1})$ is not embeddable in $\mathbb{AG}$ for each $1 \leq i < s$. This implies that at least one SWAP must be inserted between two consecutive subcircuits C_i and C_{i+1}. With these restrictions, the QKNOB circuit C's built-in SWAP cost is close to the optimal cost, although the exact cost generally remains difficult to compute.

While the above discussion focuses on benchmarks for evaluating SWAP count optimality, our approach can also generate benchmarks for evaluating depth optimality. For this purpose, we replace the permutations used above (except the first, which can still be arbitrary) with permutations implementable by a set of SWAP gates that do not share any qubits; that is, a SWAP circuit of depth 1. These SWAPs are often called *parallel* SWAPs.

V. BENCHMARK DESIGN

This section describes the detailed procedure for generating QKNOB benchmarking circuits for $\mathbb{AG} = (\mathbb{V}, \mathbb{E})$, the architecture graph of a quantum device.

A. QKNOB CONSTRUCTION ALGORITHM

We require the following subroutines (the parameters used there will be explained in detail in Section V-B).

1) Randomly generate a subgraph G of $\mathbb{AG}$ with the specified subgraph size (either "small" or "large").
2) Given a subgraph G of $\mathbb{AG}$, and a given qubit gate ratio ρ_{qbg} (specifying the ratio between the numbers of one- and two-qubit gates), randomly generate an $\mathbb{AG}$-circuit ($\widetilde{C}$) that respects ρ_{qbg} and has G as its interaction graph.
3) Given two subgraphs G_1 and G_2 of $\mathbb{AG}$, randomly select a permutation π of a given type such that $G_1 \overset{\pi}{\Rightarrow} G_2$ (i.e., $\langle G_1, \pi, G_2 \rangle$ is a strong glink).

When the optimization objective is "SWAP count," we have two permutation types, "opt1" and "opt2" to be introduced in Section V-B). When the optimization objective is "depth," the permutation is implemented by parallel SWAPs.

Using these subroutines, we can generate benchmarking circuits of the form shown in (3), whose built-in transformation costs are near-optimal. The pseudocode is described in Algorithm 1.

From Theorem 1, any transformation of an input circuit can be presented as in (3). This suggests that our benchmarks are general and representative.

By Theorem 2, each circuit constructed using Algorithm 1 has a built-in transformation. When the optimization objective is "SWAP count," we implement each permutation (other than π_1) using one or two SWAPs. Because we use strong glinks, at least one SWAP must be inserted between any two consecutive subcircuits. This implies that the built-in cost should be close to the optimal cost and thus provides us a known near-optimal transformation cost. When the optimization objective is "depth," the situation is more complex. While each SWAP gate is implemented with three consecutive CNOTs, the depth increase from inserting a depth-1 SWAP circuit between C_i and C_{i+1} can be significantly greater than 3. Indeed, a single SWAP insertion can double the circuit depth [25]! To generate benchmarking circuits with a small depth ratio, we modify each $\widetilde{C}_i$ so that there are few or no free qubits in its last layer. This can be achieved by, for example, rearranging the gates in $\widetilde{C}_i$ or randomly inserting one-qubit gates and some two-qubit gates that have already appeared.

Algorithm 1: QKNOB Circuits Construction.

Require: An architecture graph $\mathbb{AG}$, optimization type opt_{type}, target cost c, permutation type $perm_{\text{type}}$, subgraph size *graph_size*, and qubit gate ratio ρ_{qbg}
Ensure: A QKNOB circuit C

```
Randomly generate a permutation π_1
Randomly generate a subgraph G_1 of 𝔸𝔾 respecting subgraph size
Randomly generate an 𝔸𝔾-circuit C̃_1 with interaction graph G_1 respecting qubit gate ratio
glinkChain ← G_1
cost ← 0
ℓ ← 1
while cost < c do
    Randomly generate a strong glink G_ℓ ⟹^{π_{ℓ+1}} G_{ℓ+1} starting from the last subgraph G_ℓ of glinkChain that respects permutation type and subgraph size
    Extend glinkChain with G_ℓ ⟹^{π_{ℓ+1}} G_{ℓ+1}
    Randomly generate an 𝔸𝔾-circuit C̃_{ℓ+1} with interaction graph G_{ℓ+1} respecting qubit gate ratio
    if opt_type = 'SWAP count' then
        cost ← cost + ||π_{ℓ+1}||
    else if opt_type = 'depth' then
        cost ← cost + 1
    end if
    ℓ ← ℓ + 1
end while
C ← π_1 (C̃_1 + π_2(C̃_2 + π_3(··· π_ℓ(C̃_ℓ + π_{ℓ+1}(C̃_{ℓ+1}))···)))
```

B. DIMENSIONS AND PARAMETERS OF QKNOB

Our design has the following dimensions.

Optimization Objective (opt_{type}): The target can be minimizing the SWAP cost or the depth cost. SWAP cost counts the number of inserted SWAPs, whereas depth cost is the difference between the output and input circuit depths.

Target Transformation Cost (c): If the optimization objective is SWAP cost, we select c from $\{0, 1, 2, 3,$

IEEE Transactions on Quantum Engineering

TABLE 1. (Top) QKNOB and (bottom) QUEKO Benchmark Sets

	benchmark set name	$perm_{\text{type}}$	$graph\_size$	ρ_{qbg}	#circuit
QKNOB	20Q_gate_tokyo	opt1 or opt2	large	1.5	100 ($\times 2$)
	53Q_gate_Sycamore	opt1 or opt2	small or large	1.5	100 ($\times 4$)
	53Q_gate_Rochester	opt1 or opt2	small or large	1.5	100 ($\times 4$)
	20Q_depth_tokyo	parallel	large	1.5 or 2.55	60 ($\times 2$)
	53Q_depth_Sycamore	parallel	small or large	1.5 or 2.55	60 ($\times 4$)
	53Q_depth_Rochester	parallel	small or large	1.5 or 2.55	60 ($\times 4$)
QUEKO	20Q_bss_tokyo	-	-	2.55	90
	16Q_bntf_Aspen-4	-	-	1.5	90
	54Q_bntf_Sycamore	-	-	2.55	90

4, 5, 10, 15, 20, 25}; if the objective is depth cost, we select the cost (in swap layers) from {1, 2, 3, 4, 5, 10}. We focus on near-term feasible circuits [29] with depths usually no more than 80. Because the circuit depth can increase by three or more when adding a layer or a sequence of SWAPs, considering SWAP cost below 25 and SWAP layers below 10 is reasonable.

Permutation Type ($perm_{type}$): For SWAP count optimization, permutations (excluding the first, which does not contribute to transformation cost) are implemented by either a single SWAP or two consecutive SWAPs. If the permutation type is "opt1," each permutation is implemented by a single swap; if it is "opt2," each permutation is randomly implemented by either a single SWAP or two consecutive SWAPs. For depth optimization, we implement each permutation (except the first) by a set of parallel SWAPs (i.e., a depth-1 SWAP circuit).

AG: Our benchmark construction method can be applied on any near-term superconducting quantum devices. In this article, we consider three representative quantum devices: IBM Q Tokyo (20 qubits), IBM Q Rochester (53 qubits), and Google's Sycamore (53 qubits), also considered in [12], [24], and [29]. Benchmarks designed for Rochester may not be near-optimal for Sycamore, and vice versa, because 1) some Sycamore subgraphs are not embeddable in Rochester and 2) a strong glink for Rochester might not be a strong glink for Sycamore.

Subgraph Size (graph_size): Subgraphs of the AG are used for generating glinks. Larger subgraphs result in more gates in the subcircuits (C_i in Algorithm 1) of the benchmarking circuit. For the 20-qubit IBM Q Tokyo, the subgraphs in the generated glinks have approximately five edges on average. For the two 53-qubit devices, we offer two choices: small subgraphs (approximately eight edges) or large subgraphs (approximately 16 edges).

Qubit Gate Ratio (ρ_{qbg}) for evaluating depth optimality: For depth optimality, the number (and distribution) of one-qubit gates in the input circuit can significantly affect the transformed circuit's depth. To reflect this, we introduce the "qubit gate ratio" parameter, $\rho_{\text{qbg}} = M_1/M_2$, in QKNOB construction, where M_1 and M_2 denote the number of one- and two-qubit gates, respectively. Following [29], we consider two ratios: the "QSE" ratio ($\rho_{\text{qbg}} = 2.55$) based on the random circuit used in Google's quantum supremacy experiment [36], and the "TFL" ratio ($\rho_{\text{qbg}} = 1.5$) based on the Toffoli circuit.

For each legal combination of these dimensions, we randomly generate 10 circuits. In total, we have six sets of QKNOB circuits, as shown at the top of Table 1. For example, the circuit set "53Q_gate_Rochester" includes 100 circuits for each parameter pair (permutation type, graph size), which can be used to evaluate the SWAP count optimality of QCT algorithms on IBM Q Rochester.

VI. EXPERIMENTS AND EVALUATION

This section evaluates the effectiveness of QKNOB circuits for benchmarking QCT algorithms. We compare on QKNOB and QUEKO benchmarks the performance of five state-of-the-art QCT algorithms: the transpiler of $t|ket\rangle$ [2] from Quantinuum, SABRE [15], SAHS [13], MCTS [12], and the Qiskit transpiler Stochasticswap, focusing on SWAP count and depth optimality. Our experiments also highlight general trends across different architectures and optimization objectives. All our experiments were run on a laptop with i7-11800 CPU, 32 GB memory and RTX 3060 GPU.

A. DETAILS OF THE COMPARED QCT ALGORITHMS

Qiskit provides multiple choices for both initial mapping and routing procedures.[2] To facilitate comparison with QUEKO benchmarks, we choose DenseLayout and Stochasticswap as the Qiskit transpiler to compare. SABRE was initially described in [15] and has recently been assembled in Qiskit. In this article, we choose this Qiskit implementation of SABRE and select the advanced "lookahead" heuristics. The version of SAHS we use is from GitHub.[3] The original MCTS algorithm [11] targeted SWAP count and was modified in [12] to address depth optimization.[4] We call these two versions MCTS-size and MCTS-depth, respectively. In our evaluation of SWAP count optimality, we use MCTS-size; otherwise, we use MCTS-depth. The initial mappings used for MCTS are obtained by the Simulated Annealing method in SAHS [13]. When evaluating $t|ket\rangle$ on QUEKO, Tan and Cong [29] selected GraphPlacement for initial mapping, which might have led to favorable results for $t|ket\rangle$, as GraphPlacement uses subgraph isomorphism and can find optimal solutions for some QUEKO circuits. To facilitate comparison with [29], in our evaluation, we also choose

[2]The version of Qiskit used in our evaluation is 0.33.0.
[3]https://github.com/BensonZhou1991/circuittransform
[4]https://github.com/BensonZhou1991/MCTS-New

to use GraphPlacement.[5] In addition, we disable all optimization passes for fair comparisons.

As MCTS, SABRE, and Stochasticswap are random algorithms, for each circuit, we ran these algorithms five times and recorded the best value. This could improve the performance by 10%–20%.

B. MEASURING SWAP COUNT AND DEPTH OPTIMALITIES

This work focuses on two key metrics for evaluating QCT: SWAP count optimality and depth optimality.

SWAP Count Optimality is measured using the CNOT gate ratio ("*cx ratio*"), which is the ratio of the number of CNOT gates in the transformed (output) circuit to the number of CNOT gates in the original (input) circuit. Formally, the CNOT gate ratio is defined as

$$\rho_{\text{cx}} = \frac{\text{number of CNOT gates in the output circuit}}{\text{number of CNOT gates in the input circuit}}. \quad (4)$$

Here, we assume that each inserted SWAP gate is decomposed into three consecutive CNOT gates in the output circuit.

A smaller value of ρ_{cx} indicates a transformation requires fewer SWAP insertions. Note that $\rho_{\text{cx}} \geq 1$, and equality occurs when no SWAP is inserted during the transformation.

Depth Optimality is measured using the *depth ratio* (denoted as ρ_{depth}), which is the ratio of the circuit depth after transformation to the depth of the original circuit. Formally, he depth ratio is defined as

$$\rho_{\text{depth}} = \frac{\text{depth of the output circuit}}{\text{depth of the input circuit}}. \quad (5)$$

Similarly, the smaller the value of ρ_{depth}, the closer the transformation is to optimal in terms of depth. Note that $\rho_{\text{depth}} \geq 1$, with equality achieved when the transformed circuit has the same depth as the original.

These two metrics collectively provide a comprehensive evaluation of QCT, capturing both gate-level efficiency and overall circuit execution complexity. We also use $\rho_{\text{cx}} - 1$ and $\rho_{\text{depth}} - 1$ to measure the gate overhead per CNOT gate and the depth overhead per layer, respectively, relative to the input circuit.

C. SUMMARY OF EVALUATION RESULTS

For SWAP count optimality, we consider three QKNOB benchmark sets: "53Q_gate_Sycamore," "53Q_gate_Rochester," "20Q_gate_Tokyo," and three QUEKO sets "54Q_bntf_Sycamore," "20Q_bss_Tokyo," and "16Q_bntf_Aspen-4." As no "20Q_bntf_Tokyo" benchmark set is provided in the GitHub website of QUEKO,[6] we replace it with "20Q_bss_Tokyo," which are benchmarks with depth from 100 to 900 for scaling study. We also run benchmarks in the "16Q_bntf_Aspen-4" on the 20-qubit IBM Q Tokyo. In addition, when running on benchmarks in

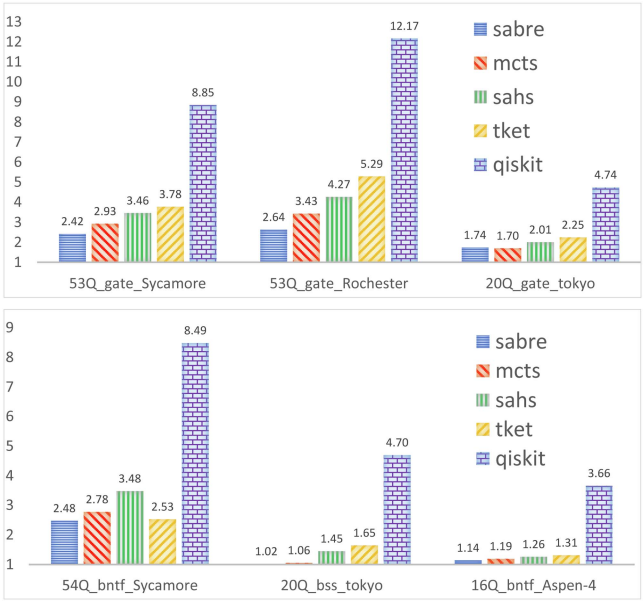


FIGURE 5. (Top) SWAP count optimality performance of QCT algorithms on the QKNOB and (bottom) QUEKO benchmark sets, where the *y*-axes denote the average cx ratios ρ_{cx} [cf., (4)]. Lower values are better.

"54Q_bntf_Sycamore," we use the ideal AG of Sycamore where the bad node, as well as its connections, is restored. The evaluation results are summarized in Fig. 5, from which we can see the following.

1) Qiskit's Stochasticswap performs significantly worse across all six benchmark sets.

2) SABRE performs the best on all but one benchmark sets; its average cx ratio [cf., (4)] on the QKNOB benchmark set "20Q_gate_Tokyo" is only slightly worse than that of MCTS-size (1.74 versus 1.70). Its performance on the two 53-qubit QKNOB benchmark sets is conspicuously (> 17%) better than the other algorithms.

For depth optimality, we evaluate, in addition to the three QUEKO sets, QKNOB sets "53Q_depth_Sycamore," "53Q_depth_Rochester," and "20Q_depth_Tokyo." The evaluation results are summarized in Fig. 6, from which we can see the following.

1) Qiskit's Stochasticswap performs clearly worst on all but "54Q_bntf_Sycamore"; its performance on "54Q_bntf_Sycamore" is the second worst.

2) SABRE performs best on all benchmark sets; its performance on the two 53-qubit QKNOB benchmark sets is at least ≥ 10% better than that of any other algorithm.

When comparing performances on the QKNOB benchmarks across different architectures, Figs. 5 and 6 reveal the following general trends:

$$\text{SABRE} \prec \text{MCTS} \prec \text{SAHS} \prec \text{t|ket}\rangle \prec \texttt{StochasticSWAP}$$

where ≺ denotes "better than." Note that SABRE≺ MCTS is violated only on "20Q_gate_Tokyo," where the cx ratios of

[5]The version of t|ket⟩ used for our evaluation is 0.17.0.
[6]https://github.com/tbcdebug/QUEKO-benchmark

IEEE Transactions on Quantum Engineering

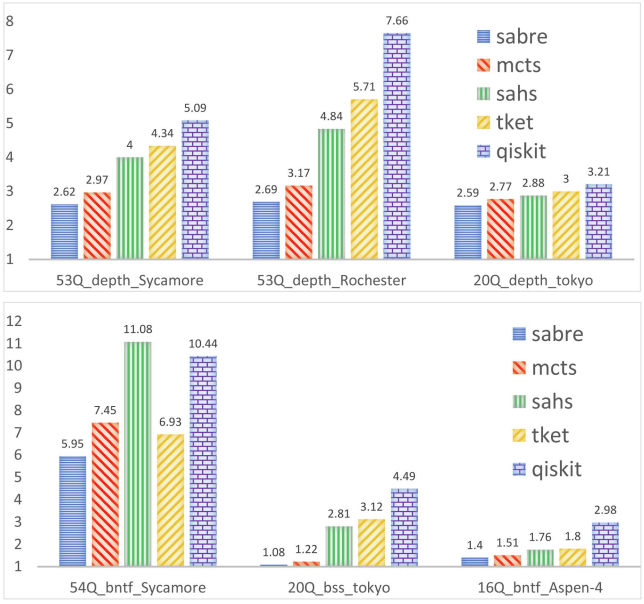


FIGURE 6. **(Top) Depth optimality performance of QCT algorithms on the QKNOB and (bottom) QUEKO benchmark sets, where the *y*-axes denote the average depth ratios ρ_{depth} [cf., (5)]. Lower values are better.**

SABRE and MCTS are 1.74 and 1.70, respectively. Furthermore, the cx/depth ratios on Sycamore are generally lower than those on Rochester, reflecting Sycamore's greater qubit connectivity.

Remark 1: SABRE is faster than all tested algorithms except Stochasticswap. To investigate whether repeated executions of Stochasticswap could improve results, we compared the algorithms on the ten "53Q_depth_Rochester_large_opt_10_2.55" circuits. The results are presented below, where "Ratio" denotes the depth ratio; MCTS denotes MCTS-depth; and SSx5, SSx100, SEx5, and SEx100 indicate repeating Stochasticswap or SABRE 5 or 100 times, respectively.

Alg.	SAHS	MCTS	t|ket⟩	SSx5	SSx100	SEx5	SEx100
Ratio	6.80	4.03	7.89	8.07	7.16	4.11	**3.35**
Time (s)	356	2755	977	76	1150	159	2690

D. COMPARING QKNOB WITH QUEKO

As pointed out in the introduction, QUEKO circuits are designed with optimal transformations that incur zero costs. While this simplifies benchmark construction, it limits QUEKO's ability to faithfully evaluate QCT algorithms that rely on subgraph isomorphism for initial mapping. For instance, FiDLS [24] achieves optimal transformations on all QUEKO "16Q_bntf_Aspen-4" and "20Q_bss_Tokyo" circuits, yet its performance on QKNOB "20Q_gate_Tokyo" is comparable to that of t|ket⟩, far from being optimal.

Fig. 5 (bottom) provides further insights. Both SABRE and MCTS-size exhibit near-optimal performance on QUEKO benchmark set "20Q_bss_Tokyo" (1.02 versus 1.06), where the optimality score is 1. This near-optimal performance is unexpected, as neither algorithm uses subgraph isomorphism for initial mapping. This suggests that QUEKO fails to reveal these QCT algorithms' true performance. In contrast, on QKNOB "20Q_gate_Tokyo," the cx ratios for SABRE and MCTS-size are 1.74 and 1.70, respectively. A similar phenomenon is also observed in Fig. 6 (bottom): SABRE and MCTS-depth achieve near-optimal depth ratios (1.08 versus 1.22) on QUEKO "20Q_bss_Tokyo" but exhibit much higher depth ratios on QKNOB "20Q_depth_Tokyo" (2.59 versus 2.77).

Another notable observation is t|ket⟩'s inconsistent performance on the QUEKO and QKNOB benchmarks. As shown in Figs. 5 and 6, t|ket⟩ significantly outperforms MCTS and SAHS on QUEKO "54Q_bntf_Sycamore" but performs worse than these algorithms on the four 53Q QKNOB benchmark sets. This discrepancy likely stems from t|ket⟩'s GraphPlacement mapping pass, which can find optimal initial mappings for certain QUEKO benchmarks.

In summary, QKNOB addresses the limitations of QUEKO and demonstrates its effectiveness in faithfully evaluating QCT algorithms. This capability stems from QKNOB's construction methodology (cf., Algorithm 1) and the theoretical properties established in Theorems 1 and 2.

E. BENCHMARK INFORMATION

This section provides a detailed examination of the QKNOB benchmarks used in our evaluation. Due to space limitation, we narrow our discussion to the Rochester benchmark sets for depth optimality and analyse how the depths of the benchmarking circuits and the depth ratios associated with their built-in transformations (henceforth "built-in depth ratios") vary with target depth cost.

In Fig. 7(a) and (b), we examine four sets of QKNOB "53Q_depth_Rochester" circuits, characterized by their graph size ("large" or "small") and qubit gate ratio ("TFL," corresponding to $\rho_{\text{qbg}} = 1.5$, or "QSE," corresponding to $\rho_{\text{qbg}} = 2.55$). The target SWAP depth costs are within the values 1 to 5. For clarity, data for circuits with a target SWAP depth cost of 10—which have average depths between 40 and 80—are omitted from the plots.

Fig. 7(a) demonstrates that the average depths of these circuits scale linearly with the target depth cost. Circuits with a "large" graph size and a "QSE" qubit gate ratio exhibit larger depths compared to their counterparts with a "small' graph size and a qubit gate ratio. This trend highlights the impact of graph size and qubit gate ratio on circuit depth.

Fig. 7(b) compares the built-in depth ratios across these circuits. It shows that "TFL" circuits generally have greater built-in depth ratios than their "QSE" counterparts. This discrepancy is partially attributable to "QSE" circuits containing more one-qubit gates, which increase the overall depth. In addition, the figure reveals that the depth ratios of QKNOB circuits scale effectively (at most linearly) with increasing target SWAP depth costs.

These observations validate the design of QKNOB benchmarks, demonstrating their scalability and ability to represent

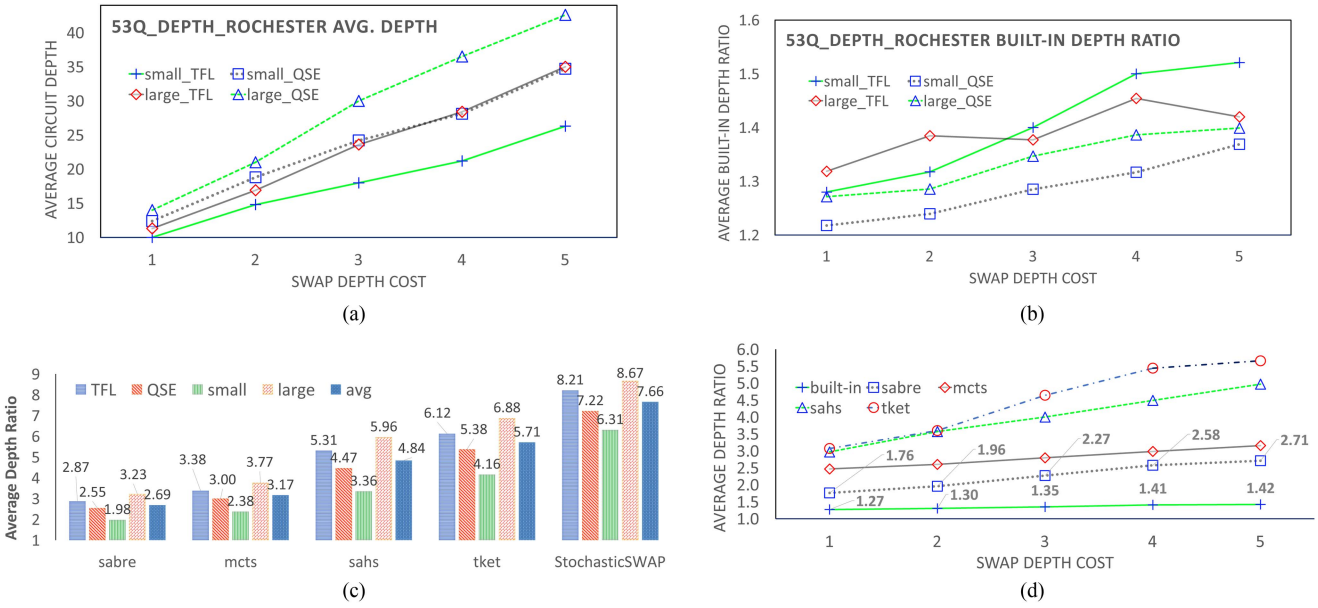


FIGURE 7. Information and evaluation results for QKNOB "53Q_depth_Rochester" circuits on IBM Q Rochester. (a) Average circuit depth. (b) Average built-in depth ratios. (c) Performance comparison among four subsets ("TFL," "QSE," "small," "large") and the full set ("avg"). (d) Evaluation results. In (a), (b), and (d), the *x*-axis denotes the target SWAP depth cost, and each point represents the average value over ten QKNOB circuits with the same parameters.

diverse circuit properties, which are essential for evaluating depth optimality in QCT algorithms.

F. FACTORS THAT AFFECT OPTIMALITY

The design of QKNOB circuits highlights three critical factors that influence QCT algorithms's SWAP count (or depth) optimality: target cost, permutation type (or qubit gate ratio), and subgraph size.

For large devices like Rochester, the subgraphs used in glinks are classified as either "small" (approximately 8 edges) or "large" (approximately 16 edges). Circuits with "small" subgraphs naturally contain fewer gates. In addition, QKNOB circuits with the "TFL" qubit gate ratio have fewer one-qubit gates but a comparable number of two-qubit gates compared to those with the "QSE" ratio.

Fig. 7(c) reveals the following two key observations.

1) *Significant effect of qubit gate ratio:* The qubit gate ratio has a noticeable impact on QCT algorithms, often exceeding a 10% difference.
2) *Greater impact of subgraph size:* The subgraph size has an even larger effect, with QCT algorithms achieving significantly lower depth ratios (often more than 50% lower) on circuits with small subgraphs. This is partly because smaller subgraphs lead to fewer total gates and reduced circuit depth. *Circuits with large subgraphs are inherently more challenging to transform.* For instance, SABRE achieves an average depth ratio of 3.23 on circuits with large subgraphs, which is 63% higher than its average depth ratio of 1.98 on circuits with small subgraphs.

Fig. 7(d) shows that the performance of QCT algorithms scales well with increasing target depth costs in the construction of QKNOB circuits.

VII. FURTHER DISCUSSION

1) Scalability: The proposed QKNOB benchmark construction method operates in time linear to the number of gates. However, it requires a subgraph isomorphism check to determine whether a glink is strong (line 8 of Algorithm 1), which is not polynomial in the number of qubits. This limitation is manageable for two reasons. First, near-term quantum devices typically have no more than several thousand qubits, for which subgraph isomorphism can be checked efficiently. Second, the construction method often only needs to disprove a subgraph isomorphism, which is computationally easier than proving it. For example, it takes about 200 seconds to generate a 900-qubit QKNOB circuit on Grid(30, 30) with permutation type "opt2," graph_size 450, qubit gate ratio 1.5, and target transformation cost 10.

2) Generality and Theoretical Guarantees: The construction method described in Section IV is general. Theorems 1 and 2 guarantee that for any circuit C and any transformation of C, C can be reconstructed as a QKNOB circuit with the given transformation as its built-in transformation. Consequently, every QKNOB circuit has a built-in transformation and an associated known cost, which serves as an upper bound for the optimal transformation cost.

3) Addressing the Gap Between Built-In and Optimal Costs: Using arbitrary permutations in (3) makes it difficult to estimate the *gap* between the built-in and optimal costs. For QKNOB circuits designed to benchmark SWAP count

IEEE Transactions on Quantum Engineering

optimality, permutations generated by one or two consecutive SWAPs help to narrow the gap. In contrast, constructing benchmarking circuits for depth optimality (cf., the paragraph before Section V-B) is less theoretically rigorous. Despite this limitation, the constructed QKNOB circuits exhibit relatively small built-in depth ratios. For instance, the "53Q_depth_Rochester" circuits with a target SWAP layer cost of 5 achieves an average built-in depth ratio of 1.42 [cf., Fig. 7(d)], translating to an average depth overhead of 0.42—only about a quarter of that incurred by SABRE ($2.71-1=1.71$)!

4) Limitations of the Framework: The QKNOB framework does not account for commutation rules, synthesis optimisations, gate absorption, or remote CNOTs. Consequently, evaluating QCT algorithms employing such techniques [8], [9], [10], [16], [26] or mitigating crosstalk errors [10] may introduce bias.

5) Adapting to Hardware Characteristics: Real quantum devices exhibit spatial and temporal variability in qubit reliability and connectivity strengths. The QKNOB construction method can be adapted to reflect these characteristics by leveraging calibration data to select subgraphs comprising highly reliable qubits and edges. Similarly, permutations for generating strong glinks can prioritize highly reliable edges. This approach effectively reverses the variability-aware qubit mapping strategy in [17] and [18], tailoring benchmarks to specific hardware characteristics.

VIII. CONCLUSION

In this work, we introduced QKNOB, a novel benchmarking framework for QCT that features circuits with built-in transformations and near-optimal costs. Unlike existing benchmarks, such as QUEKO, QKNOB reflects the general QCT process, enabling the generation of representative and versatile benchmarking circuits. Researchers can also use our open-source benchmark construction algorithm to customize benchmarks for specific devices or algorithm evaluations.

Through detailed experiments, we demonstrated QKNOB's fairness and comprehensiveness in evaluating QCT algorithms, including those leveraging subgraph isomorphism (e.g., $\natural\text{ket}\rangle$). Notably, SABRE emerged as the best-performing algorithm on large devices, highlighting its potential for scaling to future quantum hardware. However, our results also reveal a significant gap between the performance of state-of-the-art QCT algorithms and the built-in transformations of QKNOB circuits. For instance, on medium-depth circuits, even the best-performing algorithms incur more than four times the overhead of the built-in transformations [cf., Fig. 7(d)]. This underscores the need for further innovation to narrow this gap.

In summary, QKNOB provides a robust and scalable framework for benchmarking QCT algorithms, advancing the evaluation of both existing and emerging algorithms. While it highlights the strengths of current approaches, it also identifies critical areas for improvement. Future research could focus on developing more optimal QCT algorithms, incorporating hardware-aware adaptations, and leveraging QKNOB to explore new strategies for enhancing real-world performance.

CODE AND BENCHMARK AVAILABILITY

Our algorithm (implemented in Python 3) and benchmarks are available as open-source under the MIT license at https://github.com/ebony72/quekno.

APPENDIX
PROOFS

Proof of Lemma 1: Let $S_1=(\text{SWAP}(p_1,q_1),\ldots,\text{SWAP}(p_c,q_c))$ and $S_2=(\text{SWAP}(p_{c+1},q_{c+1}),\ldots,\text{SWAP}(p_{c+d},q_{c+d}))$, where $c,d\geq 1$ and each (p_j,q_j) is an edge in G. By definition, we have $\pi_1=\pi_{p_c,q_c}\circ\cdots\circ\pi_{p_1,q_1}$ and $\pi_2=\pi_{p_{c+d},q_{c+d}}\circ\cdots\circ\pi_{p_{c+1},q_{c+1}}$. It is now clear that $\pi_2\circ\pi_1=\pi_{p_{c+d},q_{c+d}}\circ\cdots\circ\pi_{p_{c+1},q_{c+1}}\circ\pi_{p_c,q_c}\circ\cdots\circ\pi_{p_1,q_1}$ is implemented by $S_1+S_2=(\text{SWAP}(p_1,q_1),\ldots,\text{SWAP}(p_c,q_c),\text{SWAP}(p_{c+1},q_{c+1}),\ldots,\text{SWAP}(p_{c+d},q_{c+d}))$. □

Proof of Lemma 2: This follows directly from Definition 4. □

Proof of Theorem 1: By assumption, C has a transformation with cost c. That is, we can transform C into an executable circuit using an initial (logical to physical) mapping σ_1 and c SWAP gates $\text{SWAP}(p_1,q_1),\ldots,\text{SWAP}(p_c,q_c)$. Let C_1 be C's subcircuit containing all gates executable under σ_1. Due to optimality, C_1 cannot be empty. Removing all gates in C_1 and inserting the SWAP gates one by one until some gates in the remainder of C are executable under the current mapping. Let S_1 denote this SWAP circuit and π_2 denote the inverse of the permutation implemented by S_1. Then, $\sigma_2\triangleq\pi_2^{-1}\circ\sigma_1$ is the current (logical to physical) mapping. Continuing the above procedure until all gates in C are executed, we partition C into $1\leq s\leq c+1$ nonempty subcircuits $C_1,\ldots,C_s$ and partition the c SWAP gates into $s-1$ nonempty SWAP circuits $S_1,\ldots,S_{s-1}$ such that S_i is inserted between C_i and C_{i+1}. Let π_{i+1} be the inverse of the permutation implemented by S_i and let $\sigma_{i+1}\triangleq\pi_{i+1}^{-1}\circ\sigma_i$. It can be proved inductively that C_i is executable under σ_i for $1\leq i\leq s$. Clearly, C has the form as shown in (1). Because $\pi_{i+1}^{-1}=\sigma_{i+1}\circ\sigma_i^{-1}$ and $S_i=[\![\pi_{i+1}^{-1}]\!]$ for each $1\leq i\leq s-1$, the transformed circuit has the form shown in (2). □

Proof of Lemma 3: Taking the identity mapping $id_{\mathbb{V}}$ as the initial mapping, gates in $\widetilde{C}_1$ can be executed and removed from C directly. The current mapping remains to be $id_{\mathbb{V}}$. If we insert a SWAP circuit that implements π^{-1}, the current logical to physical mapping becomes π^{-1}, which can execute $\pi(\widetilde{C}_2)$ because $\pi^{-1}(\pi(\widetilde{C}_2))=\widetilde{C}_2$ is an $\mathbb{AG}$-circuit. □

Proof of Theorem 2: For each $1\leq i\leq s$, let σ_i denote the permutation $(\pi_1\circ\cdots\circ\pi_i)^{-1}$. Then, C is represented in the form shown in (1). Following the analysis given in Section III or directly by Lemma 3, we can show the correctness of the transformation. Because the cost of this transformation is $c\triangleq$

$\sum_{i=2}^{s} \|\pi_i^{-1}\| = \sum_{i=2}^{s} \|\pi_i\|$, we know that the optimal cost is no greater than c. □

ACKNOWLEDGMENT

Tha authors thank the anonymous reviewers for their constructive feedback, which has significantly enhanced the presentation of this work. The authors also thank Ky Dan Nguyen for his assistance in reimplementing the code (available at https://github.com/dnngky/quekno-rx) and for generating a 900-qubit QKNOB circuit.

REFERENCES

[1] A. M. Childs, E. Schoute, and C. M. Unsal, "Circuit transformations for quantum architectures," in *Proc. 14th Conf. Theory Quantum Comput. Commun. Cryptography*, 2019, pp. 3:1–3:24, doi: 10.4230/LIPIcs.TQC.2019.3.

[2] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, "On the qubit routing problem," in *Proc. 14th Conf. Theory Quantum Comput. Commun. Cryptography*, 2019, pp. 5:1–5:32, doi: 10.4230/LIPIcs.TQC.2019.5.

[3] G. Nannicini, L. S. Bishop, O. Günlük, and P. Jurcevic, "Optimal qubit assignment and routing via integer programming," *ACM Trans. Quantum Comput.*, vol. 4, no. 1, Oct. 2022, Art. no. 7, doi: 10.1145/3544563.

[4] M. Saeedi, R. Wille, and R. Drechsler, "Synthesis of quantum circuits for linear nearest neighbor architectures," *Quantum Inf. Process.*, vol. 10, no. 3, pp. 355–377, 2011, doi: 10.1007/s11128-010-0201-2.

[5] D. Venturelli, M. Do, E. Rieffel, and J. Frank, "Compiling quantum circuits to realistic hardware architectures using temporal planners," *Quantum Sci. Technol.*, vol. 3, 2018, Art. no. 025004, doi 10.1088/2058-9565/aaa331.

[6] A. Ash-Saki, M. Alam, and S. Ghosh, "QURE: Qubit re-allocation in noisy intermediate-scale quantum computers," in *Proc. 56th Annu. Des. Automat. Conf.*, ACM, 2019, Art. no. 141, doi: 10.1145/3316781.3317888.

[7] H. Deng, Y. Zhang, and Q. Li, "Codar: A contextual duration-aware qubit mapping for various NISQ devices," in *Proc. 57th ACM/IEEE Des. Automat. Conf.*, 2020, pp. 1–6, doi: 10.1109/DAC18072.2020.9218561.

[8] T. Itoko, R. Raymond, T. Imamichi, A. Matsuo, and A. W. Cross, "Quantum circuit compilers using gate commutation rules," in *Proc. 24th Asia South Pacific Des. Autom. Conf.*, 2019, pp. 191–196, doi: 10.1145/3287624.3287701.

[9] B. Tan and J. Cong, "Optimal qubit mapping with simultaneous gate absorption," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, 2021, pp. 1–8, doi: 10.1109/ICCAD51958.2021.9643554.

[10] L. Xie, J. Zhai, and W. Zheng, "Mitigating crosstalk in quantum computers through commutativity-based instruction reordering," in *Proc. 58th ACM/IEEE Des. Automat. Conf.*, 2021, pp. 445–450, doi: 10.1109/DAC18074.2021.9586145.

[11] X. Zhou, Y. Feng, and S. Li, "A Monte Carlo tree search framework for quantum circuit transformation," in *Proc. 57th IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2020, pp. 1–7, doi: 10.1145/3400302.3415621.

[12] X. Zhou, Y. Feng, and S. Li, "Quantum circuit transformation: A Monte Carlo tree search framework," *ACM Trans. Des. Autom. Electr. Syst.*, vol. 27, no. 6, pp. 59:1–59:27, 2022, doi: 10.1145/3514239.

[13] X. Zhou, S. Li, and Y. Feng, "Quantum circuit transformation based on simulated annealing and heuristic search," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4683–4694, Dec. 2020, doi: 10.1109/TCAD.2020.2969647.

[14] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 7, pp. 1226–1236, 2018, doi: 10.23919/DATE.2018.8342181.

[15] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for NISQ-era quantum devices," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 1001–1014, doi: 10.1145/3297858.3304023.

[16] J. Liu, P. Li, and H. Zhou, "Not all swaps have the same cost: A case for optimization-aware qubit routing," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2022, pp. 709–725, doi: 10.1109/HPCA53966.2022.00058.

[17] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 1015–1029, doi: 10.1145/3297858.3304075.

[18] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for NISQ-era quantum computers," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 987–999, doi: 10.1145/3297858.3304007.

[19] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen, and E. Z. Zhang, "Time-optimal qubit mapping," in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2021, pp. 360–374, doi: 10.1145/3445814.3446706.

[20] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proc. 2018 Int. Symp. Code Gener. Optim.*, 2018, pp. 113–125, doi: 10.1145/3168822.

[21] K. E. Booth, M. Do, J. C. Beck, E. Rieffel, D. Venturelli, and J. Frank, "Comparing and integrating constraint programming and temporal planning for quantum circuit compilation," in *Proc. 28th Int. Conf. Autom. Plan. Scheduling*, 2018, pp. 366–374, doi: 10.1609/icaps.v28i1.13920.

[22] A. A. de Almeida, G. W. Dueck, and A. C. da Silva, "Finding optimal qubit permutations for IBM's quantum computer architectures," in *Proc. 32nd Symp. Integr. Circuits Syst. Des.*, 2019, Art. no. 13, doi: 10.1145/3338852.3339829.

[23] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation as a combination of subgraph isomorphism and token swapping," *Proc. ACM Program. Lang.*, pp. 1–29, 2019, doi: 10.1145/3360546.

[24] S. Li, X. Zhou, and Y. Feng, "Qubit mapping based on subgraph isomorphism and filtered depth-limited search," *IEEE Trans. Comput.*, vol. 70, no. 11, pp. 1777–1788, Nov. 2021, doi: 10.1109/TC.2020.3023247.

[25] S. Li, K. D. Nguyen, Z. Clare, and Y. Feng, "Single-qubit gates matter for optimising quantum circuit depth in qubit mapping," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, IEEE, 2023, pp. 1–9, doi: 10.1109/ICCAD57390.2023.10323863.

[26] P. Niemann, L. Mueller, and R. Drechsler, "Combining SWAPs and remote CNOT gates for quantum circuit transformation," in *Proc. 24th Euromicro Conf. Digit. Syst. Des.*, 2021, pp. 495–501, doi: 10.1109/DSD53832.2021.00080.

[27] M. Bandic, C. G. Almudéver, and S. Feld, "Interaction graph-based characterization of quantum benchmarks for improving quantum circuit mapping techniques," *Quantum Mach. Intell.*, vol. 5, no. 2, pp. 1–30, 2023, doi: 10.1007/s42484-023-00124-1.

[28] Y. Huang, D. Gao, S. Ying, and S. Li, "Dasatom: A divide-and-shuttle atom approach to quantum circuit transformation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2024, doi: 10.1109/TCAD.2025.3532818.

[29] B. Tan and J. Cong, "Optimality study of existing quantum computing layout synthesis tools," *IEEE Trans. Comput.*, vol. 70, no. 9, pp. 1363–1373, Sep. 2021, doi: 10.1109/TC.2020.3009140.

[30] B. Tan and J. Cong, "Optimal layout synthesis for quantum computing," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, San Diego, CA, USA, 2020, pp. 137:1–137:9, doi: 10.1145/3400302.3415620.

[31] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "QASMBench: A low-level quantum benchmark suite for NISQ evaluation and simulation," *ACM Trans. Quantum Comput.*, vol. 4, no. 2, Feb. 2023, Art. no. 10, doi: 10.1145/3550488.

[32] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT bench: Benchmarking software and design automation tools for quantum computing," *Quantum*, vol. 7, 2023, Art. no. 1062, doi: 10.22331/q-2023-07-20-1062.

[33] K. Chen et al., "VeriQBench: A benchmark for multiple types of quantum circuits," 2022, *arXiv:2206.10880*, doi: 10.48550/arXiv.2206.10880.

[34] P. D. Nation et al., "Benchmarking the performance of quantum computing software," 2024, *arXiv:2409.08844*, doi: 10.48550/arXiv.2409.08844.

IEEE Transactions on Quantum Engineering

[35] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1367–1372, Oct. 2004, doi: 10.1109/TPAMI.2004.75.

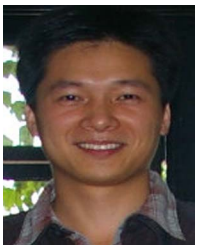
[36] F. Arute et al., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, pp. 505–510, 2019, doi: 10.1038/s41586-019-1666-5.

Sanjiang Li received the B.Sc. in mathematics from Shaanxi Normal University, Xi'an, China, in 1996, and the Ph.D. degree in mathematics from Sichuan University, Chengdu, China, in 2001.

He is a Professor in Centre for Quantum Software and Information, University of Technology Sydney, Ultimo, NSW, Australia. His research interests include knowledge representation, artificial intelligence, and quantum circuit compilation.

Xiangzhen Zhou received the B.E. degree in engineering from Nanjing Normal University, Nanjing, China, in 2010, and the Ph.D. degree in engineering from Southeast University, Nanjing.

From 2018 to 2020, he was a Visiting Student with the Centre for Quantum Software and Information, University of Technology Sydney, Ultimo, NSW, Australia. He is currently a Lecturer with Nanjing Tech University, Nanjing. His research interests include quantum computing and quantum circuit optimization.



Yuan Feng received the B.S. degree in applied mathematics and Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 1999 and 2004, respectively.

He is currently a Professor with the Department of Computer Science and Technology, Tsinghua University. His research interests include quantum programming theory, quantum information and quantum computation, and probabilistic systems.