Full Characterization of the Depth Overhead for Quantum Circuit Compilation with Arbitrary Qubit Connectivity Constraint

Pei Yuan and Shengyu Zhang

Tencent Quantum Laboratory, Tencent, Shenzhen, Guangdong 518057, China

In some physical implementations of quantum computers, 2-qubit operations can be applied only on certain pairs of qubits. Compilation of a quantum circuit into one compliant to such qubit connectivity constraint results in an increase of circuit depth. Various compilation algorithms were studied, yet what this depth overhead is remains elusive. In this paper, we fully characterize the depth overhead by the *routing number* of the underlying constraint graph, a graph-theoretic measure which has been studied for 3 decades. We also give reduction algorithms between different graphs, which allow compilation for one graph to be transferred to one for another. These results, when combined with existing routing algorithms, give asymptotically optimal compilation for all commonly seen connectivity graphs in quantum computing.

1 Introduction

Quantum computation has demonstrated a substantial advantage over its classical counterpart in solving significant problems such as integer factorization [1], search [2] and a wide variety of others, by structurally designed algorithms [3] as well as variational algorithms [4]. Quantum hardware technologies have seen considerable advancements in recent years [5–14], enabling the execution of quantum algorithms. A crucial aspect of this execution involves the compilation of quantum algorithms into quantum circuits, typically composed of 1- and 2-qubit gates.

Despite the theoretically proven advantages, the practical implementation of quantum algorithms and quantum circuits faces numerous challenges, one of which is the qubit connectivity constraint. On certain hardware platforms such as superconducting [5, 7, 8], quantum dots [9–13] and cold atoms [15–17], 2-qubit gates can only act on certain pairs of qubits, while operations on two distant qubits are usually achieved by a sequence of SWAP operations [18–25]. This qubit connectivity constraint can be naturally modeled by a connected *constraint graph* $G = (V, E)$, where the vertex set V represents the qubits, and the edge set E specifies the pairs of qubits on which 2-qubit gates can act. A circuit respecting the G constraint is termed *G-compliant* in this paper.

Various constraint graphs exist on real hardware, including path [5, 26], bilinear chains [5, 6], 2D-grids [7, 8], brick walls [5], cycle-grids [27] and trees [5]. Future connectivity patterns may even broaden this variety. The wide diversity of the constraint graphs calls for systematic studies on the compilation overhead due to the connectivity constraint. In this paper, we focus on circuit depth, which typically corresponds to the circuit's execution time. In the NISQ era [28], the

Pei Yuan: peiyuan@tencent.com
Shengyu Zhang: shengyzhang@tencent.com

execution time is particularly relevant as most variational quantum algorithms require executing the same circuit thousands of times and use the sample average to estimate the expectation of an observable on the circuit's final state.

Several questions arise about the depth overhead. On a given constraint graph G, what is the smallest depth overhead, as a measure of the graph, that a compiler can possibly achieve? How do we actually compile a circuit achieving this minimal depth increase? When designing a quantum chip, how should we lay out the qubits to ensure their connectivity facilitates a small circuit depth overhead? In this paper, we address these questions by fully characterizing the depth overhead for *any* given graph, with an explicit compilation algorithm given and the optimality shown.

A widely adopted generic method compiling a quantum circuit under qubit connectivity constraint is to insert SWAP gates into the original circuit to bring (the states in) two distant qubits together, apply the two-qubit operations, and then move them back [18–25]. This method is also the focus of this paper.

Before diving into details, let us specify the measure for depth overhead. Take a common universal set of 1-qubit and 2-qubit gates [1]. Given an n-qubit circuit C with depth $d(C)$ assuming all-to-all qubit connectivity, we need to compile it to a circuit C' with depth $d(C')$ that respects the constraint graph G. We use the ratio of $d(C')/d(C)$ as the overhead measure for circuit instance C, aiming to compile any C with a small overhead. Formally, we study the following measure doh (standing for *depth overhead*)

$$\mathsf{doh}(G) := \max_{C} \min_{\substack{C' \sim C: \\ G\text{-compliant}}} \frac{d(C')}{d(C)}. \tag{1}$$

Here the minimum is over all G-compliant circuits C' equivalent to C and obtained from C by inserting SWAP gates, and the maximum is over all n-qubit circuits C. Namely, we hope to find a good compiling algorithm $C \to C'$ such that the depth increase ratio $d(C')/d(C)$ is controlled for any possible input circuit C [2].

While there are a few compilation algorithms working for a few specific graph constraints [18–25], the studies fall short in two aspects. Firstly, the proposed routing algorithms for the specific graphs were not always optimal. For instance, Ref. [29] proposed a QAOA circuit that is hardware-compliant under a grid constraint. The method was to first find a long path in the grid and then to use the known SWAP routing algorithm on the path. For a grid of size $\sqrt{n} \times \sqrt{n}$, this results in a $O(n)$ depth overhead. Later we will show a superior routing algorithm with $O(\sqrt{n})$ depth overhead, and show that it is optimal. Secondly, the compilation algorithms so far are *ad hoc* for different specific graphs, and no systematic studies on general graphs have been conducted. This paper is the first to provide a unified, provably optimal algorithm for *all* graphs.

Main results. In this work, we present a unified algorithm for qubit routing for any given constraint graph G, with the depth overhead fully characterized by the routing number $\mathsf{rt}(G)$, a well-studied graph theoretic measure with a long history. We demonstrate that for all connected graphs G,

$$\mathsf{doh}(G) = \Theta(\mathsf{rt}(G)). \tag{2}$$

[1] The 1-qubit gates are not restricted by the qubit connectivity. Common choices for the 2-qubit gates include CNOT, CZ, iSWAP, etc. Note that a SWAP gate can be easily realized by three CNOT gates.

[2] Here we use ratio $d(C')/d(C)$ rather than difference $d(C') - d(C)$ because $d(C')$ increases linearly with $d(C)$ for a generic circuit C, while the difference $d(C') - d(C)$ can be arbitrarily large. (Though it is also possible to consider multiple layers of C together in compression, this semantic compression does not have much advantage for a *generic* depth-d circuit C, as each layer can be arbitrary and different layers do not admit a good compression. Even for the rare cases admitting significant semantic compression, the computational complexity of finding such a good compression is high.)

Specifically, on any graph G, we provide an algorithm to compile an arbitrarily given circuit C (with no connectivity constraint) into another circuit C' with the depth increase ratio bounded by $O(\mathsf{rt}(G))$ from above. Furthermore, we show that this is the best possible outcome—one cannot compile a generic C with a depth increase factor asymptotically better than $O(\mathsf{rt}(G))$.

As a graph theoretic measure that has been studied for about three decades, the routing number $\mathsf{rt}(G)$ has been pinned down for many specific graphs G such as paths, grids, trees, complete bipartite graphs, hypercubes, etc. [30–35]. By combining our algorithm with these known routing methods, we immediately obtain compilation algorithms for quantum circuits under these graph constraints.

Additionally, we introduce a reduction algorithm between different graphs, enabling us to derive efficient routing algorithms for new constraint graphs from existing algorithms on some basic graphs. To demonstrate this, we construct compilers for IBM's brick-wall graphs [5] and Rigetti's cycle-grid graphs [27] by reducing SWAP networks on those graphs to the ones on the 2D-grid.

Previous work. The result in Eq. (2) bears resemblance to the canonical work on lower bounding quantum circuit size complexity by the geodesic distance on the unitaries manifold [36, 37]. However, our work diverges in two significant ways: (1) we provide matching upper and lower bounds, and (2) our approach is more operational as it presents an explicit algorithm to compile a given arbitrary circuit C to a G-compliant circuit C' with depth $d(C') = O(d(C) \cdot \mathsf{rt}(G))$.

Ref. [38] explores the qubit connectivity constraint for general and special classes of unitaries, such as those for quantum state preparation (QSP), by exploiting techniques in earlier work [39] and carefully arranging qubits such that most two-qubit gates act on nearby qubits. The paper shows that, somewhat surprisingly, the qubit connectivity constraint does not significantly increase circuit complexity for these classes of unitaries either in the worst or a generic case. For example, it gives a parametrized QSP circuit of depth $O(2^n/n)$ to generate a given arbitrary n-qubit quantum state, while even circuits *without* the qubit connectivity constraint require the depth of the same order of depth [40]. Though this might suggest that the qubit connectivity constraint does not increase circuit complexity, it is important to note that the worst-case or generic unitary matrices have exponentially high complexity. Thus Ref. [38] merely demonstrates that the connectivity constraint does not exacerbate these already complex cases. However, our primary concern in practice lies with efficient (e.g. polynomial depth) circuits—After all, these are the ones to be used in the future. Results in Ref. [38] do not provide insight into how the connectivity constraint affects these *efficient* circuits, and particularly do not rule out the possibility that shallow circuits significantly suffer from the constraint. This work shows that, fortunately, this is not the case: If a circuit C with all-to-all connections has depth d, then it can always be compiled to a G-compliant circuit C' with depth at most $O(d \cdot \mathsf{rt}(G)) = O(dn)$. In particular, if $d(C)$ is a polynomial function of n, so is the depth of C'.

Ref. [14] considered routing for partial permutations, in which one only needs to map $k < n$ vertices x_i to k other vertices y_i, and the rest $n - k$ vertices can be mapped arbitrarily. When the circuit depth is concerned with, the paper gave a reduction ("Greedy Depth Mapper") from a partial routing protocol to circuit compilation. Their algorithm is essentially the same as ours in Lemma 2. However, this compilation can be very loose as explained after Lemma 2. They also gave a number of partial routing protocols, which may complement our result: They gave efficient routing methods, which may be utilized by our reduction to obtain efficient compilation methods (though one should also be careful about the difference between partial and full permutation).

Organization. The rest of this paper is organized as follows. In Section 2, we introduce notations and review some previous results. In Section 3, we show the full characterization of the

depth overhead. Then we demonstrate a reduction of routing numbers between different graphs and construct routing algorithms for practice qubit connectivity constraints in Section 4. Finally, we discuss our results in Section 5.

2 Preliminaries

Notation. Let $[n] := \{1, 2, \ldots, n\}$. Let $|S|$ denote the size of set S. In this paper we study general undirected graphs $G = (V, E)$, where V is the set of vertices and E is the set of edges. We denote the number of vertices by n, and sometimes identify the vertex set V with the set $[n]$. For a vertex $u \in V$ and a subset $S \subseteq V$, the neighbor of u inside S is $N_S(u) = \{v \in S : (u, v) \in E\}$. We drop the subscript S and just write $N(u)$ if $S = V$. For a subset $S \subseteq V$, the *induced subgraph* of G on S is

$$G|_S = (S, E') \text{ with } E' = \{(u, v) \in E : u \in S, v \in S\}. \tag{3}$$

The *diameter* of a graph G is the largest distance of two vertices, denoted by

$$diam(G) := \max_{u,v \in V} d(u, v), \tag{4}$$

where $d(u, v)$ is the distance of vertices u and v on graph G (i.e. the number of edges on the shortest path between u and v). A *matching* in an undirected graph $G = (V, E)$ is a vertex-disjoint subset of edges $M \subseteq E$.

Quantum circuit compilation and depth overhead. The qubit connectivity constraint can be modeled by a connected graph $G = (V, E)$. A two-qubit gate can be applied to a pair of qubits $i, j \in V$ if and only if $(i, j) \in E$. We refer to G as the *constraint graph* and a circuit satisfying the above constraint as a *G-compliant circuit*. When G is the complete graph K_n, we say that the circuit has all-to-all connectivity.

The quantum circuit compilation problem studied in this paper is as follows: Given an n-qubit input quantum circuit C consisting of 1- and 2-qubit gates with all-to-all connectivity, and a constraint graph G with n vertices (identified with the n qubits), construct a G-compliant circuit C' equivalent to C by adding SWAP gates. Here two circuits are equivalent if they implement the same unitary operation. To measure the quality of the hardware-compliant circuit, we introduce the following concept of *depth overhead*:

$$\mathsf{doh}(G, C) := \min_{C'} d(C')/d(C), \tag{5}$$

where $d(C)$ and $d(C')$ denote the depth of circuits C and C', respectively, and the minimum is over all C' that satisfy the above compilation requirement. The *depth overhead* of a constraint graph G is then defined as

$$\mathsf{doh}(G) := \max_{C} \mathsf{doh}(C, G). \tag{6}$$

where the maximum is taken over all n-qubit circuits C with all-to-all qubit connectivity.

Permutations. The set of all permutations on set $[n]$ is denoted by S_n. A permutation $\pi \in S_n$ is a *transposition* if $\pi = (a_1, a_2)(a_3, a_4) \cdots (a_{2k-1}, a_{2k})$ with distinct $a_1, a_2, \ldots, a_{2k}$, where (a_{2i-1}, a_{2i}) means to exchange a_{2i-1} and a_{2i}. A permutation π is a transposition if and only if it satisfies $\pi^2 = id$, the identity permutation. It is well known that any permutation can be written as the composition of two transpositions.

Lemma 1 ([41]). *Any $\pi \in S_n$ can be decomposed as $\pi = \sigma_1 \circ \sigma_2$, where $\sigma_1, \sigma_2 \in S_n$ are transpositions.*

Routing number $\mathsf{rt}(G)$**.** The routing number is defined by the following game [30]. Given a connected graph $G = (V, E)$ with n vertices, we place one pebble at each vertex $v \in V$, and move the pebbles in rounds. In each round i, we are allowed to select a matching $M_i \subseteq E$ and swap the two pebbles at u and v for all edges $(u, v) \in M_i$. For any permutation $\pi \in S_n$, $\mathsf{rt}(G, \pi)$ is the *minimum* number of rounds in which one can move all pebbles from their initial positions v to the destinations $\pi(v)$. For any graph G, the routing number is defined as

$$\mathsf{rt}(G) = \max_{\pi \in S_n} \mathsf{rt}(G, \pi). \tag{7}$$

3 Full characterization of depth overhead

In this section, we present a protocol for constructing a G-compliant circuit with the depth overhead of at most $O(\mathsf{rt}(G))$ for any given circuit C in Section 3.1. Then we demonstrate that for any G, there exists a circuit C with depth overhead at least $\Omega(\mathsf{rt}(G))$ in Section 3.2, thereby offering a complete characterization of the depth overhead of G.

Computing the depth overhead for a given C and G turns out to be NP-hard; see Appendix A for the proof. However, the asymptotic behavior of $\mathsf{doh}(G)$ can be fully characterized by a graph measure called the *routing number*, denoted by $\mathsf{rt}(G)$, of a graph G, as defined in Eq. (7).

3.1 Hardware-compliant circuit construction and depth overhead upper bound

In this section, we first present a compiling algorithm by the maximum matching. Second, we give a graph partition algorithm. Third, based on the graph partition algorithm, we present the general compiling algorithm and the depth overhead upper bound.

Before diving into detailed constructions, let us first compare the measures $\mathsf{doh}(G)$ and $\mathsf{rt}(G)$, and explain why the upper bound $\mathsf{doh}(G) = O(\mathsf{rt}(G))$ does not immediately follow from their definitions, despite the apparent similarities. For easier comparison, let us formulate $\mathsf{doh}(G)$ in a language of games similar to that for the routing number $\mathsf{rt}(G)$ (Eq. (7)): we compile circuit C layer by layer, and for each layer, suppose the two-qubit gates are on pairs $(u_1, v_1), \ldots, (u_k, v_k)$, then we need to use SWAP gates to move u_i and v_i next to each other, apply the gate, and move them back. This formulation highlights immediate similarities between $\mathsf{doh}(G)$ and $\mathsf{rt}(G)$: (1) Both can be viewed as games in rounds, with each round consisting of SWAP operations. (2) Both measures represent the minimum number of rounds.

However, also note that there are some *key differences* between the two measures: (1) In circuit compilation, u_i and v_i need to be moved to *next* to each other, while in graph routing, i needs to be moved to $\pi(i)$. Note that given a routing algorithm to move each i to $\pi(i)$ for any permutation π, it is still hard to move u_i and v_i next to each other because we cannot simply find a neighbor v_i' of v_i and let $\pi(u_i) = v_i'$; for example, if all v_i's have degree 1 and are all connected to a common "port" node v to reach the rest of the graph, then all v_i' equal to v, making the map $\pi(u_i) = v$ not a permutation. We will need to handle this type of bottleneck issue in designing the protocol for $\mathsf{doh}(G)$. (2) In circuit compilation, it suffices that u_i and v_i are next to each other at *some* time step $t_i \leq \mathsf{doh}(G)$ (different pairs (u_i, v_i) may have different t_i), while in graph routing, all i need to be moved to $\pi(i)$ exactly at time $\mathsf{rt}(G, \pi)$. This gives us some freedom to design the $\mathsf{doh}(G)$ protocol.

One basic property that will be used later is a linear upper bound of $\mathsf{rt}(G)$ [30]:

$$\mathsf{rt}(G) \leq 3n. \tag{8}$$

A routing protocol induces a SWAP circuit in a natural way: For any permutation π on vertices in graph G and any routing protocol in the definition of $\mathsf{rt}(G, \pi)$, if two pebbles at two vertices i

and j are swapped, then we apply a swap operation on qubits i and j in the SWAP circuit. Then the following unitary transformation U_π

$$|x_1 \cdots x_n\rangle \xrightarrow{U_\pi} |x_{\pi(1)} \cdots x_{\pi(n)}\rangle, \quad \forall x_i \in \{0,1\}, i \in [n], \tag{9}$$

can be realized by a $\mathsf{rt}(G,\pi)$-depth circuit consisting of swap operations.

Before we give the general compiling algorithm, we first give a lemma which can compile circuits for graphs with a large matching.

Lemma 2. *For any connected graph G with the maximum matching size ν, we can construct G-compliant circuits with the depth overhead at most $O(\mathsf{rt}(G) \cdot n/\nu)$. That is, $\mathsf{doh}(G) = O(\mathsf{rt}(G) \cdot n/\nu)$.*

Proof. Fix a given n-qubit circuit C. For each layer C_i with at least one two-qubit gate, suppose that the 2-qubit gates are $C_{i1}, \ldots, C_{ik}$ on pairs $\{(u_1, v_1), \ldots, (u_k, v_k)\}$ of qubits, respectively, where $1 \le k \le n/2$ and the $2k$ vertices $u_1, v_1, \ \ldots \ , u_k, v_k$ are all distinct. Let $\{(x_1, y_1), \ \ldots \ , (x_\nu, y_\nu)\}$ be a maximum matching of G. We can compile this layer of C_i to a G-compliant circuit in depth $O(\mathsf{rt}(G))$ as follows:

1. Apply all single-qubit gates in C_i.
2. Pick any permutation π that permutes u_j to x_j and v_j to y_j, for all $j \in [\nu]$. Run the circuit U_π (in Eq. (9)) of depth at most $\mathsf{rt}(G)$.
3. Apply the 2-qubit gates C_{ij} on (x_j, y_j), for all $j \in [\nu]$;
4. Run $U_\pi^\dagger$, the reverse process of Step 2.

If $k \le \nu$, then this implements C_i already in depth $2 \cdot \mathsf{rt}(G) + 2$. If $k > \nu$, then this implements the first ν two-qubit gates among k ones. Repeat the last three steps in the above procedure until all 2-qubit gates are handled, which needs $\lceil k/\nu \rceil$ iterations. Each iteration needs at most $2 \cdot \mathsf{rt}(G) + 1$ depth, thus the overall depth overhead is $1 + (2\mathsf{rt}(G) + 1) \cdot \lceil k/\nu \rceil = O(\mathsf{rt}(G) \cdot n/\nu)$. □

Note that Lemma 2 can already give compiling algorithms with depth overhead at most $O(\mathsf{rt}(G))$ for some specific graphs G, including 1D-chain, 2D-grid, IBM's brick wall or Rigetti's bilinear cycle, binary tree, etc, all of which have a matching of size $\Theta(n)$. But for graphs with a small matching size (an extreme example is the star graph which has the maximum matching size $\nu = 1$), the bound of $O(\mathsf{rt}(G) \cdot n/\nu)$ is very loose.

Lemma 2 has a clear intuition that the existence of a large matching facilitates moving the pebble around. Actually, it is even tempting to conjecture that this dependence of $O(n/\nu)$ is inevitable since a bottleneck in a graph does make simultaneous pebble moving inefficient due to traffic congestion. However, this bottleneck also affects $\mathsf{rt}(G)$ protocols and should be inherently characterized in the $\mathsf{rt}(G)$ measure. What we need to do is to technically relate the difficulty in the two measures and construct a reduction from one to the other. Next, we give details on how to remove the $O(n/\nu)$ factor in Lemma 2.

To improve it to the optimal bound of $O(\mathsf{rt}(G))$, one idea is to partition the constraint graph into vertex-disjoint connected subgraphs, each having $O(1)$ diameter and containing at most $O(\mathsf{rt}(G))$ vertices. We aim to move each pair of qubits (u_i, v_i) on which a two-qubit gate acts to one of these subgraphs (different pairs may go to different subgraphs). If this can be achieved, then we can implement two-qubit gates within one subgraph efficiently. Indeed, since each subgraph has diameter $O(1)$, it takes $O(1)$ SWAP gates to implement one gate and since the subgraph has size $\mathsf{rt}(G)$, all the gates inside this subgraph can be done by $O(\mathsf{rt}(G))$ SWAP gates, which

takes at most $O(\mathsf{rt}(G))$ rounds. Also note that the routings in different subgraphs can be carried out in parallel. Thus the overall overhead is $O(\mathsf{rt}(G))$.

The challenge is that it is not always possible to achieve such a good partition of the constraint graph. We present a good graph partition algorithm in Lemma 4, for which we will first show the following bottleneck lemma.

Lemma 3 (bottleneck)**.** *For a connected graph $G = (V, E)$, suppose that there exist vertex sets $V_1, V_2 \subseteq V$ such that*

1. *$V_1 \cap V_2 = \emptyset$;*
2. *for any $u \in V_1$, $N(u) \subseteq V_2$, where $N(u) := \{w \in V : (w, u) \in E\}$.*

Then the routing number of G satisfies

$$\mathsf{rt}(G) = \Omega\left(\frac{|V_1|}{|V_2|}\right). \tag{10}$$

Proof. Let us label the pebbles in V_1 as $1, 2, \ldots, |V_1|$. Let $s := \lfloor |V_1|/2 \rfloor$. Define a permutation $\pi := (1, 1+s)(2, 2+s)\cdots(s, 2s)$. Note that if we move the pebble at vertex i to the vertex $i + s$, the pebble must go through V_2 since $N(i) \subseteq V_2$ by assumption. Since at most $|V_2|$ pebbles in V_1 can go through V_2 in one round, moving s pebbles needs at least $s/|V_2|$ rounds. Therefore,

$$\mathsf{rt}(G) \geq \mathsf{rt}(G, \pi) \geq s/|V_2| = \Omega(|V_1|/|V_2|).$$

□

Next, we present the graph partition algorithm in Lemma 4, which outputs two families of vertex sets $\mathcal{W}$ and $\mathcal{W}'$.

Lemma 4. *There exists an algorithm which, on any n-vertex connected graph $G = (V, E)$, outputs two families of vertex sets*

$$\mathcal{W} = \{W_1, \ \ldots, \ W_s : W_i \subseteq V, \ \forall i \in [s]\}, \tag{11}$$
$$\mathcal{W}' = \{W'_1, \ \ldots, \ W'_t : W'_j \subseteq V, \ \forall j \in [t]\}, \tag{12}$$

for some $s, t \in [n]$, satisfying the following properties.

1. *(disjointness) For any distinct $i, i' \in [s]$ and distinct $j, j' \in [t]$, we have $W_i \cap W_{i'} = \emptyset$ and $W'_j \cap W'_{j'} = \emptyset$;*
2. *(coverage) $|\bigcup_{i\in[s]} W_i| + |\bigcup_{j\in[t]} W'_j| \geq n/2$;*
3. *(size bound) For any $i \in [s]$ and $j \in [t]$, we have $2 \leq |W_i| = O(\mathsf{rt}(G))$ and $2 \leq |W'_j| = O(\mathsf{rt}(G))$;*
4. *(diameter bound) For all $i \in [s]$ and $j \in [t]$, the induced subgraphs $G_i = G|_{W_i}$ and $G'_j = G|_{W'_j}$ all have diameter at most 2.*

The algorithm runs in time $O(n^3)$.

Proof. The family $\mathcal{W}$ is constructed as follows. First find a maximal matching

$$\mathcal{M} = \{(w_1, w_2),\ (w_3, w_4),\ \ldots,\ (w_{2s-1}, w_{2s})\} \tag{13}$$

of G. Put

$$\mathcal{W} = \{W_i : i \in [s]\},\ \text{where } W_i = \{w_{2i-1}, w_{2i}\} \subseteq V,\ \forall i \in [s]. \tag{14}$$

Then $\mathcal{W}$ satisfies the properties 1, 4 and 3. Indeed, each $G|_{W_i}$ is essentially an edge and thus the two nodes are connected, different $G|_{W_i}$'s are disjoint as $\mathcal{M}$ is a matching, and $|W_i| = 2 = O(\mathsf{rt}(G))$.

The family $\mathcal{W}'$ is constructed as follows. Define vertex set $T = V - \cup_{i\in[s]} W_i$, the vertices not in the maximal matching $\mathcal{M}$. Note that T is an independent set, i.e. any two vertices $a, b \in T$ are not connected; otherwise, we could have added the edge (a, b) in $\mathcal{M}$ to form a larger matching, contradicting $\mathcal{M}$ being maximal. Define set

$$S := \Big\{ w \in \bigcup_{i\in[s]} W_i : N_T(w) \neq \emptyset \Big\} \tag{15}$$

to contain those vertices with a connection to T. The family $\mathcal{W}'$ is constructed by Algorithm 1.

Algorithm 1: Construction of $\mathcal{W}'$

Input: Connected graph $G = (V, E)$, vertex sets $T, S \subseteq V$.
Output: A family $\mathcal{W}'$ of sets.
Initialize $N_w^0 := \emptyset$, $\forall w \in S$.
Initialize $A_{0,p} := \emptyset$, $\forall p \in [|T|]$.
$T_1 := T$, $S_1 := S$, $k_1 := |S_1|$.
for $p = 1$ to $|T|$ **do**
  for $i = 1$ to $|S|$ **do**
    $A_{i,p} := \emptyset$.
    if there are at least 1 and at most $\lceil |T_p|/k_p \rceil$ neighbors of w_i in $T_p - \bigcup_{r=1}^{i-1} A_{r,p}$ **then**
      Let $A_{i,p}$ contain all these neighbors.
    else if there are more than $\lceil |T_p|/k_p \rceil$ neighbors of w_i in $T_p - \bigcup_{r=1}^{i-1} A_{r,p}$ **then**
      Let $A_{i,p}$ contain arbitrary $\lceil |T_p|/k_p \rceil$ many of these neighbors.
    end if
    Let $N_{w_i}^p := N_{w_i}^{p-1} \cup A_{i,p}$.
  end for
  if $|\bigcup_{i=1}^{|S|} N_{w_i}^p| \geq |T|/2$ **then**
    return $\mathcal{W}' := \{N_{w_i}^p \cup \{w_i\} : |N_{w_i}^p| \geq 1,\ i \in [|S|]\}$ and end the whole program.
  end if
  Set $T_{p+1} := T_p \backslash \bigcup_{r=1}^{|S|} A_{r,p}$.
  Set $S_{p+1} := \{w_i : |A_{i,p}| = \lceil |T_p|/k_p \rceil, i \in [|S|]\}$.
  Set $k_{p+1} := |S_{p+1}|$.
end for

In the algorithm, T_p, $A_{i,p}$, S_p, k_p and $N_{w_i}^p$ are defined as follows. The set T_p contains those vertices in T not selected in the first $p-1$ iterations. In the p-th iteration, the set $A_{i,p}$ denotes the neighbor set within $T_p - \bigcup_{r=1}^{i-1} A_{r,p}$ of vertex $w_i \in S$ with cardinality bounded by $\lceil |T_p|/k_p \rceil$. The set S_p contains all vertices $w_i \in S$ that have exactly $\lceil |T_{p-1}|/k_{p-1} \rceil$ neighbors being chosen in the $(p-1)$-th iteration. The number k_p denotes the size of

vertex set S_p. The set $N^p_{w_i}$ contains all neighbors of vertex w_i chosen in the first p iterations.

We will first show that $\mathcal{W}'$ satisfies its corresponding properties in 1 (disjointness) and 4 (diameter). For the disjointness property, we note that actually all $A_{i,p}$'s, for different i and different p, are pairwise disjoint. Indeed, in each outer iteration $p+1$, the new set T_{p+1} removes all sets $A_{r,p}$ in iteration p (line 17), thus any set $A_{r',p+1}$ selected from T_{p+1} is disjoint from all sets $A_{r,p'}$ from previous iterations $p' \le p$. Now we check the sets $A_{i,p}$ for different i in the same iteration p. Note that when we consider neighbors of w_i, we ignore those in previous inner iterations by only considering $T_p - \bigcup_{r=1}^{i-1} A_{r,p}$ (line 7 and 9), thus the new $A_{i,p}$ are disjoint from $A_{r,p}$ for all $r < i$. Now that all $A_{i,p}$'s are pairwise disjoint, and all w_i's are distinct, the sets $N^p_{w_i} \cup \{w_i\}$ in the definition of $\mathcal{W}'$ are pairwise disjoint as well. This shows the property of disjointness.

As shown in lines 8 and 10, all vertices in $A_{i,p}$ are connected with vertices w_i, which implies that all vertices in $N^p_{w_i}$ (line 12) are connected with w_i. Therefore, the subgraph induced by $N^p_{w_i} \cup \{w_i\}$ has diameter at most 2.

With the above, we can next show that the algorithm ends and returns $\mathcal{W}'$ in line 15, i.e. the condition in line 14 is satisfied at some iteration $p = \ell$, in at most $|T|/2 \le n/2$ iterations. We will show this by arguing that in each outer iteration p, each vertex $v \in T_p$ has at least one neighbor in S_p. Once we show this, we know that at least one $A_{i,p}$ is nonempty, for which $N^p_{w_i}$ has size strictly larger than that of $N^{p-1}_{w_i}$ (line 12): All $A_{i,p}$'s are disjoint as shown above, thus $A_{i,p} \cap N^{p-1}_{w_i} = \emptyset$ and thus $|N^p_{w_i}| = |N^{p-1}_{w_i}| + |A_{i,p}| > |N^{p-1}_{w_i}|$. Therefore the set $\bigcup_{i=1}^{|S|} N^p_{w_i}$ strictly increases its size as p grows. Thus the condition in line 14 is met and the algorithm ends after at most $|T|/2$ outer iterations.

Now we argue by induction that in each outer iteration p, any vertex $v \in T_p$ has at least one neighbor in S_p, and all neighbors of v are in S_p. Namely, we have

$$\emptyset \neq N(v) \subseteq S_p, \tag{16}$$

for all $p \in [|T|]$ and all $v \in T_p$. This is true for $p = 1$ as v does not have neighbors in $T_1 = T$, thus all its neighbors are in $W = \{w_1, w_2, \ldots, w_{2s}\}$. Furthermore, all its neighbors are actually in $S \subseteq W$, as S exactly contains those $u \in W$ that have neighbors in T, i.e. $W - S$ does not have any edge to T. Therefore, v has at least one neighbor and all v's neighbors are in $S_1 = S$. For the induction step, let us assume $\emptyset \neq N_{S_p}(v)$ for each $v \in T_p$, and consider iteration $p+1$. For each $v \in T_{p+1} \subseteq T_p$, its neighbors $w_{i_1}, \ldots, w_{i_k}$ are all in S_p and $k \ge 1$ by inductive hypothesis. But this v is selected in line 17 to be in T_{p+1}. This happens must because it is not in $A_{r,p}$ for any $r \in |S|$, including $A_{i_j,p}$. As each w_{i_j} has at least one neighbor $v \in T_p$, we know that the condition in line 9 is satisfied, i.e. w_{i_j} has more than $\lceil |T_p|/k_p \rceil$ neighbors in $T_p - \bigcup_{r=0}^{i_j-1} A_{r,p}$, but it then happens in line 10 that v is not chosen into $A_{i_j,p}$. This means that $|A_{i_j,p}| = \lceil |T_p|/k_p \rceil$ and thus $w_{i_j} \in S_{p+1}$ (line 18). Therefore, v's neighbors w_{i_j} are all in S_{p+1}, completing the inductive step.

We can also show that the algorithm never runs into the situation of $S_{p+1} = \emptyset$ in line 18 (and k_{p+1} is always nonzero in the next line, justifying it being denominator in lines 7-10). Indeed, if $S_{p+1} = \emptyset$, it means that all $w_i \in S_p$ has $|A_{i,p}| \le \lceil |T_p|/k_p \rceil - 1$ and line 8 is executed. But then $\bigcup_{i=1}^{|S|} N^p_{w_i}$ is the entire T, and thus the condition in line 14 is satisfied and algorithm returns $\mathcal{W}'$ in line 15 before line 18.

Next we show that $\mathcal{W}'$ satisfies property 3, namely $2 \le |W'_j| = O(\mathsf{rt}(G))$. Assume that Algorithm 1 stops in the ℓ-th iteration of the outer loop. Since $N^\ell_{w_i} \cup \{w_i\}$ in $\mathcal{W}'$ satisfies $|N^\ell_{w_i}| \ge 1$ (line 7 and 9), each set in $\mathcal{W}'$ has size at least 2. We next prove that it has size at most $O(\mathsf{rt}(G))$.

Suppose $S_\ell := \{w_{i_1}, \ldots, w_{i_{k_\ell}}\}$. Consider the process of obtaining $S_1, \ldots, S_\ell$ in the algorithm. In each outer iteration p, the algorithm checks each $w \in S_p$ and selects as many neighbors as possible, but up to $\lceil |T_p|/k_p \rceil$, to form $A_{i,p}$. If there are more than $\lceil |T_p|/k_p \rceil$ neighbors, then it continues to collect these neighbors in the next outer iteration. S_p contains those $w_i \in S_{p-1}$ with $|A_{i,p-1}| = \lceil |T_{p-1}|/k_{p-1} \rceil$. So for any $w_i \in S_\ell$, the size of $A_{i,p}$, for $p = 1, 2, \ldots, \ell - 1$, is $\lceil |T_1|/k_1 \rceil, \lceil |T_2|/k_2 \rceil, \ldots, \lceil |T_{\ell-1}|/k_{\ell-1} \rceil$, respectively. And the size of the corresponding set $N_{w_i}^{\ell-1}$ has

$$|N_{w_i}^{\ell-1}| = \Big| \bigcup_{r \in [\ell-1]} A_{i,r} \Big| = \sum_{r=1}^{\ell-1} |A_{i,r}| = \sum_{r=1}^{\ell-1} \lceil |T_r|/k_r \rceil. \tag{17}$$

where the second equality uses the fact that all $A_{i,r}$'s are disjoint.

Define $C := \bigcup_{w \in S_\ell} N_w^{\ell-1}$, the union of the corresponding sets of vertices in S_ℓ. Based on Eq. (17) and the fact that all these $N_w^{\ell-1}$'s are disjoint, we have

$$|C| = \sum_{w \in S_\ell} |N_w^{\ell-1}| = k_\ell \cdot \sum_{r=1}^{\ell-1} \lceil |T_r|/k_r \rceil. \tag{18}$$

Since Algorithm 1 did not stop in the $(\ell-1)$-th step, we have $\left| \bigcup_{w \in S} N_w^{\ell-1} \right| < |T|/2$. Then

$$\begin{aligned}
|C| =& \Big| \bigcup_{w \in S_\ell} N_w^{\ell-1} \Big| \leq \Big| \bigcup_{w \in S} N_w^{\ell-1} \Big| < |T|/2, \\
|T_\ell| =& \Big| T_{\ell-1} - \bigcup_{r=0}^{|S|} A_{r,\ell-1} \Big| = \cdots = \Big| T - \bigcup_{p=1}^{\ell-1} \bigcup_{r=0}^{|S|} A_{r,p} \Big| = \Big| T - \bigcup_{r=0}^{|S|} N_{w_r}^{\ell-1} \Big| \\
\geq& |T| - |T|/2 = |T|/2,
\end{aligned}$$

Therefore $|T_\ell| \geq |T|/2 > |C|$. Since $T_\ell \subseteq T$, $S_\ell \subseteq S$, it follows that $T_\ell \cap S_\ell = \emptyset$. We have showed that $N(u) \subseteq S_\ell$ for all $u \in T_\ell$ by Eq. (16), and thus can apply Lemma 3 to obtain that $\mathsf{rt}(G) = \Omega(\lceil |T_\ell|/|S_\ell| \rceil) = \Omega(\lceil |T_\ell|/k_\ell \rceil)$. Combined with Eq. (18), we have

$$\mathsf{rt}(G) = \Omega(\lceil |T_\ell|/k_\ell \rceil) \geq \Omega(\lceil |C|/k_\ell \rceil) = \Omega\left(k_\ell \cdot \sum_{r=1}^{\ell-1} \lceil |T_r|/k_r \rceil / k_\ell \right) = \Omega\left(\sum_{r=1}^{\ell-1} \lceil |T_r|/k_r \rceil \right).$$

Therefore, the routing number $\mathsf{rt}(G)$ satisfies

$$\begin{aligned}
\mathsf{rt}(G) =& \max \left\{ \Omega\left(\sum_{r=1}^{\ell-1} \lceil |T_r|/k_r \rceil \right), \Omega\left(\lceil |T_\ell|/k_\ell \rceil \right) \right\} \\
=& \Omega\left(\sum_{r=1}^{\ell-1} \lceil |T_r|/k_r \rceil + \lceil |T_\ell|/k_\ell \rceil \right) \\
=& \Omega\left(\sum_{r=1}^{\ell} \lceil |T_r|/k_r \rceil \right).
\end{aligned}$$

For arbitrary set $N_{w_i}^{\ell} \cup \{w_i\}$ in $\mathcal{W}'$,

$$\begin{aligned}
&|N_{w_i}^{\ell} \cup \{w_i\}| \\
=&|N_{w_i}^{\ell-1} \cup A_{i,\ell}| + |\{w_i\}| = |N_{w_i}^{\ell-1}| + |A_{i,\ell}| + |\{w_i\}|, \\
\leq& \sum_{r=1}^{\ell} \lceil |T_r|/k_r \rceil + 1, && \text{(Eq. (17))} \\
=& O(\mathsf{rt}(G)). && (\mathsf{rt}(G) = \Omega(\sum_{r=1}^{\ell} \lceil |T_r|/k_r \rceil))
\end{aligned}$$

Therefore, $\mathcal{W}'$ satisfies property 3.

Recall that $T = V \setminus \bigcup_{i\in[s]} W_i$ and $\mathcal{W} = \{W_i : i \in [s]\}$. In the definition of $\mathcal{W}' := \{N_{w_i}^{p} \cup \{w_i\} : |N_{w_i}^{p}| \geq 1, i \in [|S|]\}$ (line 15), suppose there are t many $i \in [|S|]$ satisfying $|N_{w_i}^{p}| \geq 1$, and denote the sets $N_{w_i}^{p} \cup \{w_i\}$ as $W_1', \ldots, W_t'$. The condition in line 14 of Algorithm 1 implies that

$$|\bigcup_{i\in[s]} W_i| + |\bigcup_{j\in[t]} W_j'| \geq n - |T| + |T|/2 = n - |T|/2 \geq n/2, \tag{19}$$

showing property 2 in the theorem.

Finally, we analyze the complexity. Finding a maximal matching can be easily done in time $O(n^3)$ by repeatedly adding an edge (u, v) into the matching set M and removing u and v from the vertex set. In the i-th inner loop, lines 7-12 can be realized in time $O(|T|)$. Since there are $|S|$ inner loops, then lines 5-13 can be realized in time $O(|S| \cdot |T|)$. In the p-th outer loop, the updates in lines 17-19 can be completed in $O(|T| + |S|)$ time. Since there are $|T|$ outer loops, the total time of Algorithm 1 is

$$|T| \cdot (O(|S| \cdot |T|) + O(|S| + |T|)) = O(|T|^2|S|) = O(n^3).$$

The total time for constructing $\mathcal{W}$ and $\mathcal{W}'$ is $O(n^3) + O(n^3) = O(n^3)$.

□

With this result, we can state and prove the main compilation algorithm next, from which it will also be clear why we need those properties of the two families.

Theorem 5. *For any connected graph G, we have* $\mathsf{doh}(G) = O(\mathsf{rt}(G))$.

Proof. For any graph $G = (V, E)$, we use Lemma 4 to find two families of sets $\mathcal{W} = \{W_i : \forall i \in [s]\}$ and $\mathcal{W}' = \{W_j' : \forall j \in [t]\}$ satisfying the properties in the lemma. Since $|\bigcup_{i\in[s]} W_i| + |\bigcup_{j\in[t]} W_j'| \geq n/2$, at least one of $|\bigcup_{i\in[s]} W_i|$ and $|\bigcup_{j\in[t]} W_j'|$ is of size at least $n/4$. Without loss of generality, we assume that $|\bigcup_{i\in[s]} W_i| \geq n/4$. Consider any one layer of a given circuit. Suppose it has k ($k \leq \lfloor n/2 \rfloor$) 2-qubit gates, which act on pairs $(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)$ of qubits. Note that qubits i_r and j_r are generally not adjacent on G. We call these the original gates, to distinguish from the compiled ones that are G-compliant, which will be referred to as *G-gates*.

We now show how to compile one layer of 2-qubit original gates to a G-compliant circuit of depth at most $O(\mathsf{rt}(G))$.

1. Take a permutation π that moves the first $\gamma_1 = \lfloor |W_1|/2 \rfloor$ pairs $(i_1, j_1), \ldots, (i_{\gamma_1}, j_{\gamma_1})$ to inside W_1, and the next $\gamma_2 = \lfloor |W_2|/2 \rfloor$ pairs $(i_{\gamma_1+1}, j_{\gamma_1+1}), \ldots, (i_{\gamma_1+\gamma_2}, j_{\gamma_1+\gamma_2})$ to inside W_2, and so on, until we move $\gamma_s = \lfloor |W_s|/2 \rfloor$ pairs to W_s. Implementing this permutation needs at most $\mathsf{rt}(G)$ rounds (see Eq. (9)). Now the problem of implementing the $\sum_s \gamma_s$ original gates U_{i_r,j_r} is reduced to implementing the corresponding gates $U_{p,q}$ with p and q in some W_i.

2. Inside W_i for each $i \in [s]$, we can implement all γ_i 2-qubit gates $U_{p,q}$ in depth at most $3\gamma_i$. Indeed, for each gate $U_{p,q}$ on qubits (p, q) in W_i, since $diam(G_i) \leq 2$, either $(p, q) \in E$ or there is another vertex $r \in W_i$ connecting p and q. In the former case, we can apply the gate directly. In the latter case, we can swap p and r, apply the gate on (r, q), and swap p and r back. In any case, we can implement one gate by at most three G-gates, thus $3\gamma_i$ G-gates suffice to implement all γ_i 2-qubit gates in W_i. Note that each $|W_i| = O(\mathsf{rt}(G))$ by Lemma 4, thus the depth is $O(\mathsf{rt}(G))$ by Eq. (8).

3. By repeating the above two steps $\lceil k/(\sum_{i=1}^{s} \gamma_i)\rceil$ iterations, we can implement all of the original 2-qubit gates. Since $k \leq n/2$, $\gamma_i = \lfloor |W_i|/2 \rfloor$, and $\sum_i |W_i| \geq n/4$, the number of iterations is at most $O(1)$.

4. Move all qubits to their original positions, which takes at most $\mathsf{rt}(G)$ rounds.

Putting everything together, the depth overhead for compiling one layer of the original circuit is at most $O(\mathsf{rt}(G))$. Applying this to all layers completes the proof. □

Remark. Though the above theorem is only on the depth overhead, from the proof we can see that actually our construction from an $\mathsf{rt}(G)$ protocol to a $\mathsf{doh}(G)$ protocol can be done very efficiently. Since there are efficient routing protocols for many commonly seen specific graphs G, our construction enables us to obtain efficient compilation for quantum circuits. To be more specific, given any qubit connectivity graph G, we only need $O(n^3)$ classical pre-processing time for finding $\mathcal{W}$ and $\mathcal{W}'$ in Lemma 4, which needs to be computed only once and can be used for compiling any circuit. For each quantum circuit, the compilation time is merely $O(n)$, because it is easily verified that all steps in the above proof just need to identify some permutation π and run the given routing algorithms in $\mathsf{rt}(G)$.

3.2 Depth overhead lower bound

The following theorem gives a lower bound of the depth overhead, which matches the upper bound in Theorem 5.

Theorem 6. *For any connected graph G, we have* $\mathsf{doh}(G) = \Omega(\mathsf{rt}(G))$.

Proof. Suppose $\mathsf{rt}(G) = \mathsf{rt}(G, \pi^*)$, namely π^* is the hardest permutation in the definition of $\mathsf{rt}(G)$. By Lemma 1, π^* can be decomposed into two compositions. Thus there is an unconstrained SWAP circuit C^* of depth 2 realizing π^*. By definition of $\mathsf{doh}(G)$, we have

$$\mathsf{doh}(G) = \max_{C} \min_{\substack{C' \sim C: \\ G\text{-compliant}}} \frac{d(C')}{d(C)} \geq \min_{\substack{C' \sim C: \\ G\text{-compliant}}} \frac{d(C')}{d(C^*)} = \min_{\substack{C' \sim C: \\ G\text{-compliant}}} \frac{d(C')}{2}.$$

Take a C' achieving the minimum in the above ratio. Since C^* is a SWAP circuit and C' is obtained from C by inserting SWAP gates, C' is also a SWAP circuit. Therefore by the correspondence in Eq. (24), we obtain a routing algorithm A^* of $d(C')$ rounds which can realize π^*. Since this is one particular routing algorithm realizing π^*, we have

$$\mathsf{rt}(G) = \mathsf{rt}(G, \pi^*) \leq \text{ the number of rounds in } A^* = d(C') = 2 \cdot \mathsf{doh}(G).$$

Therefore $\mathsf{doh}(G) \geq \mathsf{rt}(G)/2 = \Omega(\mathsf{rt}(G))$. □

Table 1: A summary for the routing numbers of different graphs.

Graph G	Number of vertices	Upper bound of $\mathsf{rt}(G)$	Reference
Tree T_n	n	$3n$	[30]
		$\lfloor 3n/2 \rfloor + O(\log(n))$	[42]
Complete d-ary tree T_n^d	n	$n + o(n)$	[42]
Complete bipartite graph $K_{s,t}$ $(s > t)$	$s + t$	$\lfloor 3s/2t \rfloor + 7$	[33]
2-dimensional grid $Grid_{n_1,n_2}$ $(n_1 \leq n_2)$	$n_1 n_2$	$2n_1 + n_2$	[30]
Path P_n	n	n	[30]
Hypercube Q^n	2^n	$2n - 2$	[30]

Combining Theorems 5 and 6, we see that the depth overhead of a connectivity graph G is fully characterized by its routing number, i.e., $\mathsf{doh}(G) = \Theta(\mathsf{rt}(G))$. This result shows that our algorithm in Theorem 5 is asymptotically optimal. The characterization also gives quantitative guidance for the qubit layout and connectivity design of quantum processors when the depth overhead is considered.

4 Routing number for many common graphs and reduction between graphs

In this section, we demonstrate a reduction of routing numbers between different graphs and construct routing algorithms for cycle-grids and brick walls.

Many routing algorithms for different specific graphs have been widely investigated, see Table 1 for a summary. These algorithms, combined with our general algorithm in Theorem 5, give optimal routing algorithms for many existing connectivity graphs such as paths, bilinear chains [5, 6], 2-dimensional grids [7, 8], and trees [5]. This improves some of previous SWAP algorithms. For example, Ref. [29] proposed a circuit ansatz for QAOA, but to make it hardware-compliant for grid constraints, their algorithm has a depth overhead of $O(n)$ on a $\sqrt{n} \times \sqrt{n}$ 2D-grid. Using Theorem 5 and the result for 2D-grids in [30], we easily achieve a depth overhead of $O(\sqrt{n})$, quadratically improving the previous one and being the best possible.

While the routing number for the above graphs has been well studied, it has not been studied for graphs that are less commonly seen in graph theory but typical in quantum computing, such as IBM's brick walls and Rigetti's cycle-grids. Lemma 2 can be used to solve some graphs, and here we provide another method based on reduction, which can give good routing algorithms for more graphs.

We will in particular need one result for the Path graph (i.e. 1D-chain).

Lemma 7 ([30]). *Let P_n denote a path with n vertices, then $\mathsf{rt}(P_n) = n$.*

Now we prove Theorem 8, which gives a reduction between routing numbers of two graphs.

Theorem 8. *Let $G = (V, E)$ and $G' = (V, E')$ be two connected graphs with the same vertex set V. Suppose that $E' - E := \{e : e \in E' \text{ and } e \notin E\}$ can be partitioned into $\bigcup_{i=1}^{c} E'_i$ such that the following two conditions hold.*

1. *$E'_i \cap E'_j = \emptyset$ for arbitrary distinct $i, j \in [c]$.*
2. *$\mathsf{rt}(G, \pi_i) \leq c'$ for each $i \in [c]$, where $\pi_i = \circ_{(u,v)\in E'_i}(u, v)$ exchanges the two ends of each edge in E'_i.*

Then $\mathsf{rt}(G) \leq (1 + cc') \cdot \mathsf{rt}(G')$.

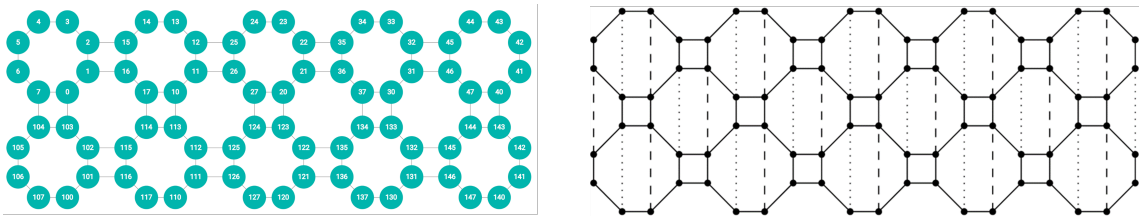

Figure 1: Left: The layout of 80 qubits in Rigetti's Aspen-M chip series. Right: A cycle-grid graph and a grid constructed from it. The cycle-grid graph consists of vertices and solid edges. A grid of size 4×20 is constructed by adding dotted and dashed edges.

Proof. Any permutation π on graph G' can be realized by a routing algorithm of at most $\mathsf{rt}(G')$ rounds. In one round, we swap two pebbles at vertices u, v if $(u, v) \in M$ for some matching $M \subseteq E'$ of G'. Now we demonstrate how to implement this round on the graph G. We partition edges in M into at most $c + 1$ sets:

$$M'_0 := \{(u, v) : \forall (u, v) \in M \cap E\}, \tag{20}$$

$$M'_i := \{(s, t) : \forall (s, t) \in M \cap E'_i\}, \quad i = 1, 2, \ldots, c. \tag{21}$$

It takes one round to swap pebbles on u and v for all $(u, v) \in M'_0$ as all these (u, v) are also edges in G. For each $i \in [c]$, the edges in M'_i form a permutation π_i, which can be realized by a routing algorithm of c' rounds by the second condition. Therefore, one round of routing on graph G' can be implemented by a routing of $(1 + cc')$ rounds on graph G. Repeating this for all $\mathsf{rt}(G')$ rounds gives $\mathsf{rt}(G) \leq (1 + cc') \cdot \mathsf{rt}(G')$. □

One simple scenario in which the second condition in Theorem 8 holds is that there are vertex-disjoint paths P_{uv} connecting u and v in G of length at most c' for all $(u, v) \in E'_i$, for each $i \in [c]$. Indeed, in this case, the routing algorithm in $rt(G, \pi_i)$ can be done by simply following these paths P_{uv}, which are vertex-disjoint and thus enable parallel routing. Next, we utilize this fact to design routing algorithms for the cycle-grids and brick walls.

Rigetti's Aspen-M chip series has the qubit connectivity graph as in Fig. 1, which is a 2×5 grid with each (super)node being a cycle of length 8, and adjacent (super)nodes connected by two edges. The grid sizes and the cycle length can vary. The routing and circuit compilation for such graphs can be reduced to those for grids by inserting edges. Specifically, we add two vertical edges to each cycle, which results in a grid of size 4×20, as shown in Fig. 1 (right). The newly added edge set is partitioned into two disjoint edge sets, the dotted edge set E_1 and the dashed edge set E_2. For each $i \in [2]$, all edges $(u, v) \in E_i$ have paths of length 3 between u and v in the cycle-grid graph, and all paths are vertex-disjoint. Thus, we can apply Theorem 8 with parameters $c = 2$ and $c' = 3$, obtaining routing algorithms and circuit compilation from those on the 2D grid of size 4×20.

We can also apply this reduction result on brick walls. For integers $n_1, n_2 \geq 1$, $b_1 \geq 2$, $b_2 \geq 3$, $b_1 < b_2$ and b_2 odd, the (n_1, n_2, b_1, b_2)-brick wall graph contains n_1 layers of n_2 "bricks", with each brick being a rectangle containing b_1 vertices on each "vertical" edge and b_2 vertices on each "horizontal" edge. In IBM's brick wall chips [5], $b_1 = 3$ and $b_2 = 5$.

The reduction of brick walls to grids is more complicated because the reduction changes the vertex set. Yet a reduction in the same spirit can still be achieved as follows.

Theorem 9. *For an (n_1, n_2, b_1, b_2)-brick wall $\texttt{Brickwall}^{b_1,b_2}_{n_1,n_2}$, we have*

$$\mathsf{rt}(\texttt{Brickwall}^{b_1,b_2}_{n_1,n_2}) = O((b_1 + b_2)(n_1 + b_2 n_2)). \tag{22}$$

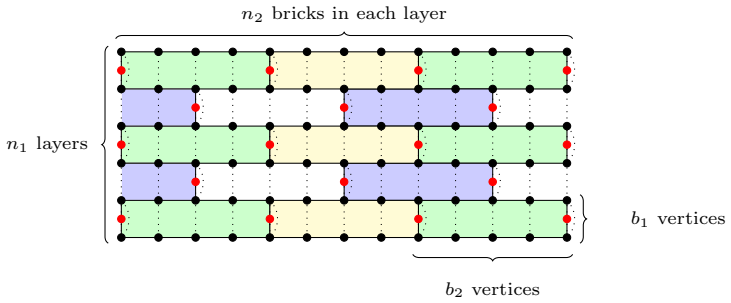

Figure 2: An (n_1, n_2, b_1, b_2)-brick wall graph and a grid constructed from it. The (n_1, n_2, b_1, b_2)-brick wall graph consists of black and red vertices and solid edges. A grid is constructed by (i) removing the vertices in the middle of each vertical edge (the red vertices), and (ii) adding dotted edges. Bricks are divided into 4 groups, indicated by the green, white, yellow and blue colors, where the bricks of the same color are vertex-disjoint.

Proof. For ease of presentation, we color the vertices in $\texttt{Brickwall}^{b_1,b_2}_{n_1,n_2}$ black and red as follows. Recall that each brick is a rectangle containing b_1 vertices on each vertical edge and b_2 vertices on each horizontal edge. Color the $2b_2$ vertices on the horizontal edges black, and the rest $2(b_1-2)$ vertices red; see Fig. 2. We will first show that any permutation σ on *black* vertices has

$$\mathsf{rt}(\texttt{Brickwall}^{b_1,b_2}_{n_1,n_2}, \sigma) = O((b_1+b_2)(n_1+b_2n_2)). \tag{23}$$

We will do this by reducing the routing problem to that on a 2D-grid. To do so, we first remove the b_1-2 red vertices in each vertical edge, and then add edges to connect the vertically aligned two vertices in each brick—the curved and straight dotted edges in Fig. 2. In this way, the graph $\texttt{Brickwall}^{b_1,b_2}_{n_1,n_2}$ is transformed to $\texttt{Grid}_{n_1+1,n_2b_2-n_2+1}$, a grid of size $(n_1+1)\times(n_2b_2-n_2+1)$. And note that all black vertices are in this grid, thus σ is also a permutation on vertices of $\texttt{Grid}_{n_1+1,n_2b_2-n_2+1}$. Now to implement σ in $\texttt{Brickwall}^{b_1,b_2}_{n_1,n_2}$, we simulate the routing protocol $\mathcal{P}$ in $\mathsf{rt}(\texttt{Grid}_{n_1+1,n_2b_2-n_2+1}, \sigma)$. In each round of $\mathcal{P}$, there are swaps of pebbles on two vertices (i,j) for some disjoint edges (i,j) in the grid graph. If (i,j) is an horizontal edge, it can be directly carried out in the brick wall graph as well, because (i,j) is also an edge in the brick wall graph. Therefore it suffices to efficiently swap pebbles on (i,j) for dotted edges (i,j).

For better parallelization, we partition the bricks into four disjoint groups, indicated by the four colors in Fig. 2: Use two colors for the odd layers and the other two colors for the even layers; inside each layer, use the two colors alternately. Note that the bricks in the same color do not overlap on vertices or edges, thus the routing can be made parallel for different bricks of the same color. Swapping pebbles on (i,j) for any number of dotted edges inside each brick in $\texttt{Brickwall}^{b_1,b_2}_{n_1,n_2}$ is a routing problem inside the brick, which is $C_{2b_2+2b_1-4}$, a cycle of length $2b_2+2b_1-4$. The routing number for this cycle is $\mathsf{rt}(C_{2b_2+2b_1-4}) = O(b_1+b_2)$ by path result in Lemma 7, and note that routing for all the bricks of the same color can be done in parallel. Thus overall one round of the pebble swapping on $\texttt{Grid}_{n_1+1,n_2b_2-n_2+1}$ can be implemented in $O((b_1+b_2)(n_1+b_2n_2))$ rounds on $\texttt{Brickwall}^{b_1,b_2}_{n_1,n_2}$, proving Eq. (23).

Now let us consider a general permutation on vertices of $\texttt{Brickwall}^{b_1,b_2}_{n_1,n_2}$. First, according to Lemma 1, any permutation can be decomposed as two transpositions, and we can implement them one by one. Now consider a transposition $\pi = (a_1,a_2)(a_3,a_4)\cdots(a_{2k-1},a_{2k})$ on $\texttt{Brickwall}^{b_1,b_2}_{n_1,n_2}$, where $a_1, a_2, \ldots, a_{2k}$ are all distinct. This transposition has three parts, according to the color of the two vertices in each pair. More precisely, we decom-

pose $\pi = \pi_1 \circ \pi_2 \circ \pi_3$, where π_1, π_2 and π_3 consist of all the transpositions (a_{2i-1}, a_{2i}) in π where a_{2i-1} and a_{2i} are both black, both red, and one black and one red, respectively. We discuss the implementation of them one by one.

- Permutation π_1: Since π_1 are on black vertices only, we can use the protocol as discussed above to achieve $\mathsf{rt}(\texttt{Brickwall}^{b_1,b_2}_{n_1,n_2}, \pi_1) = O((b_1+b_2)(n_1+b_2n_2))$.

- Permutation π_2: Consider all bricks of the green color, and perform the following in parallel. For each brick b, collect all vertices $a_{b1}, \ldots, a_{bj}$ that are both in the left side of this brick and appear in π_2. Swap (pebbles on) all these j vertices with j distinct black vertices $c_{b1}, \ldots, c_{bj}$ in the upper side in the same brick—this can be conducted because of the assumption that $b_2 \geq b_1$. We also do this for the other three colors one by one, then all red vertices are moved to black ones (except for the ones on the right boundary of the green and blue bricks, which can be easily handled later). This moving takes $O(b_1 + b_2)$ rounds. The problem reduces to the first case π_1, which can be solved by $O((b_1+b_2)(n_1+b_2n_2))$ rounds. Note that in this process, the pebbles originally on $c_{b1}, \ldots, c_{bj}$ remain still in $a_{b1}, \ldots, a_{bj}$ (because these pebbles are now on red vertices but π_1 only swaps black vertices). We then swap pebbles on $a_{b1}, \ldots, a_{bj}$ and $c_{b1}, \ldots, c_{bj}$ back. The overall cost is $O((b_1+b_2)(n_1+b_2n_2))$ rounds. (The red ones on the right boundary of the green and blue bricks can then be handled similarly with at most the same cost.)

- Permutation π_3: For transpositions (a_{2i-1}, a_{2i}), assume without loss of generality that a_{2i-1} is black and a_{2i} is red. Let p_{2i-1} and p_{2i} denote the pebbles at vertices a_{2i-1} and a_{2i} respectively. First, we permute each pebble p_{2i-1} from a_{2i-1} to a black vertex b_{2i-1} within the same brick of red vertices a_{2i}. (If a_{2i-1} and a_{2i} are already in the same brick then we do not need to move it). This is possible because $b_2 \geq b_1$. This permutation among black vertices only can be implemented in $O((b_1+b_2)(n_1+b_2n_2))$ rounds. Second, we exchange pebbles p_{2i} at red vertices a_{2i} and pebbles p_{2i-1} at black vertices in the same brick. This can be implemented in $O(b_1+b_2)$ rounds. Third, we apply the inverse permutation of the first step in $O((b_1+b_2)(n_1+b_2n_2))$ rounds. The total rounds of π_3 is $2 \cdot O((b_1+b_2)(n_1+b_2n_2)) + O(b_1+b_2) = O((b_1+b_2)(n_1+b_2n_2))$.

Putting all steps together, the permutation $\pi = \pi_1 \circ \pi_2 \circ \pi_3$ on brick wall $\texttt{Brickwall}^{b_1,b_2}_{n_1,n_2}$ can be implemented in $O((b_1+b_2)(n_1+b_2n_2))$ rounds. □

5 Discussion

We have fully characterized the depth overhead when compiling a quantum circuit with all-to-all qubit connections to one under a connectivity graph constraint for any graph, by a well-studied graph measure called routing number. We also developed compiling algorithms to achieve the asymptotically optimal depth overhead. This enables us to utilize existing routing algorithms for various specific graphs such as paths, grids and trees, to construct hardware-compliant compilers. Constraint graphs like IBM's brick walls and Rigetti's cycle-grids can also be handled via some simple reductions. The characterization of the depth overhead can also be utilized in the design of quantum processors when a small depth overhead is crucially needed.

References

[1] Peter W Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". SIAM review **41**, 303–332 (1999).

[2] Lov K Grover. “A fast quantum mechanical algorithm for database search”. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. Pages 212–219. (1996).

[3] Stephen Jordan. “Quantum algorithm zoo”. https://quantumalgorithmzoo. org/.

[4] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, et al. “Variational quantum algorithms”. Nature Reviews Physics **3**, 625–644 (2021).

[5] IBM Quantum. “IBM Quantum”. https://quantumexperience.ng.bluemix.net /qx/devices.

[6] Yangsen Ye, Zi-Yong Ge, Yulin Wu, Shiyu Wang, Ming Gong, Yu-Ran Zhang, et al. “Propagation and localization of collective excitations on a 24-qubit superconducting processor”. Physical Review Letters **123**, 050502 (2019).

[7] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, et al. “Quantum supremacy using a programmable superconducting processor”. Nature **574**, 505–510 (2019).

[8] Ming Gong, Shiyu Wang, Chen Zha, Ming-Cheng Chen, He-Liang Huang, Yulin Wu, et al. “Quantum walks on a programmable two-dimensional 62-qubit superconducting processor”. Science **372**, 948–952 (2021).

[9] M Ciorga, AS Sachrajda, Pawel Hawrylak, C Gould, Piotr Zawadzki, S Jullian, Y Feng, and Zbigniew Wasilewski. “Addition spectrum of a lateral dot from coulomb and spin-blockade spectroscopy”. Physical Review B **61**, R16315 (2000).

[10] JM Elzerman, R Hanson, JS Greidanus, LH Willems Van Beveren, S De Franceschi, LMK Vandersypen, S Tarucha, and LP Kouwenhoven. “Few-electron quantum dot circuit with integrated charge read out”. Physical Review B **67**, 161308 (2003).

[11] JR Petta, AC Johnson, CM Marcus, MP Hanson, and AC Gossard. “Manipulation of a single charge in a double quantum dot”. Physical Review Letters **93**, 186802 (2004).

[12] D Schröer, AD Greentree, L Gaudreau, K Eberl, LCL Hollenberg, JP Kotthaus, and S Ludwig. “Electrostatically defined serial triple quantum dot charged with few electrons”. Physical Review B **76**, 075306 (2007).

[13] DM Zajac, TM Hazard, Xiao Mi, E Nielsen, and Jason R Petta. “Scalable gate architecture for a one-dimensional array of semiconductor spin qubits”. Physical Review Applied **6**, 054013 (2016).

[14] Andrew M Childs, Eddie Schoute, and Cem M Unsal. “Circuit transformations for quantum architectures”. In 14th Conference on the Theory of Quantum Computation, Communication and Cryptography. Pages 3:1–3:24. (2019).

[15] Immanuel Bloch. “Quantum coherence and entanglement with ultracold atoms in optical lattices”. Nature **453**, 1016–1022 (2008).

[16] Iulia Buluta, Sahel Ashhab, and Franco Nori. “Natural and artificial atoms for quantum computation”. Reports on Progress in Physics **74**, 104401 (2011).

[17] Hannes Bernien, Sylvain Schwartz, Alexander Keesling, Harry Levine, Ahmed Omran, Hannes Pichler, et al. “Probing many-body dynamics on a 51-atom quantum simulator”. Nature **551**, 579–584 (2017).

[18] Kyle Booth, Minh Do, J Beck, Eleanor Rieffel, Davide Venturelli, and Jeremy Frank. “Comparing and integrating constraint programming and temporal planning for quantum circuit compilation”. In Proceedings of the International Conference on Automated Planning and Scheduling. Pages 366–374. (2018).

[19] Dmitri Maslov, Sean M. Falconer, and Michele Mosca. "Quantum circuit placement". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **27**, 752–763 (2008).

[20] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. "Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures". In Proceedings of the 50th annual design automation conference. Pages 1–6. (2013).

[21] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. "Qubit placement to minimize communication overhead in 2d quantum architectures". In 2014 19th Asia and South Pacific Design Automation Conference. Pages 495–500. (2014).

[22] Abhoy Kole, Kamalika Datta, and Indranil Sengupta. "A heuristic for linear nearest neighbor realization of quantum circuits by swap gate insertion using n-gate lookahead". IEEE journal on emerging and selected topics in circuits and systems **6**, 62–72 (2016).

[23] Chia-Chun Lin, Susmita Sur-Kolay, and Niraj K. Jha. "Paqcs: Physical design-aware fault-tolerant quantum circuit synthesis". IEEE Transactions on Very Large Scale Integration (VLSI) Systems **23**, 1221–1234 (2015).

[24] Ritu Ranjan Shrivastwa, Kamalika Datta, and Indranil Sengupta. "Fast qubit placement in 2d architecture using nearest neighbor realization". In 2015 IEEE International Symposium on Nanoelectronic and Information Systems. Pages 95–100. (2015).

[25] Gushu Li, Yufei Ding, and Yuan Xie. "Tackling the qubit mapping problem for nisq-era quantum devices". In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. Pages 1001–1014. (2019).

[26] Julian Kelly, Rami Barends, Austin G Fowler, Anthony Megrant, Evan Jeffrey, Theodore C White, et al. "State preservation by repetitive error detection in a superconducting quantum circuit". Nature **519**, 66–69 (2015).

[27] Rigetti. "Rigetti". http://docs.rigetti.com/en/latest/qpu.html.

[28] John Preskill. "Quantum computing in the NISQ era and beyond". Quantum **2**, 79 (2018).

[29] Matthew P Harrigan, Kevin J Sung, Matthew Neeley, et al. "Quantum approximate optimization of non-planar graph problems on a planar superconducting processor". Nature Physics **17**, 332–336 (2021).

[30] Noga Alon, F. R. K. Chung, and R. L. Graham. "Routing permutations on graphs via matchings". SIAM Journal on Discrete Mathematics **7**, 513–530 (1994).

[31] Louxin Zhang. "Optimal bounds for matching routing on trees". SIAM Journal on Discrete Mathematics **12**, 64–77 (1999).

[32] Indranil Banerjee and Dana Richards. "New results on routing via matchings on graphs". In Fundamentals of Computation Theory. Pages 69–81. (2017).

[33] Wei-Tian Li, Linyuan Lu, and Yiting Yang. "Routing numbers of cycles, complete bipartite graphs, and hypercubes". SIAM Journal on Discrete Mathematics **24**, 1482–1494 (2010).

[34] Rajko Nenadov. "Routing permutations on spectral expanders via matchings". Combinatorica Pages 1–6 (2023).

[35] Indranil Banerjee, Dana Richards, and Igor Shinkar. "Sorting networks on restricted topologies". In SOFSEM 2019: Theory and Practice of Computer Science. Pages 54–66. (2019).

[36] MA Nielsen. "A geometric approach to quantum circuit lower bounds". Quantum Information & Computation **6**, 213–262 (2006).

[37] M.A. Nielsen, M.R. Dowling, M. Gu, and A.C. Doherty. “Quantum computation as geometry”. Science **311**, 1133 (2006).

[38] Pei Yuan, Jonathan Allcock, and Shengyu Zhang. “Does qubit connectivity impact quantum circuit complexity?”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **43**, 520–533 (2024).

[39] Xiaoming Sun, Guojing Tian, Shuai Yang, Pei Yuan, and Shengyu Zhang. “Asymptotically optimal circuit depth for quantum state preparation and general unitary synthesis”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **42**, 3301–3314 (2023).

[40] Martin Plesch and Caslav Brukner. “Quantum-state preparation with universal gate decompositions”. Physical Review A **83**, 032302 (2011).

[41] Charles C Pinter. “A book of abstract algebra: second edition”. Courier Corporation. (2019).

[42] Alan Roberts, Antonis Symvonis, and Louxin Zhang. “Routing on trees via matchings”. In Algorithms and Data Structures. Pages 251–262. (1995).

[43] Sanjeev Arora and Boaz Barak. “Computational complexity: a modern approach”. Cambridge University Press. (2009).

A Hardness of computing depth overhead

In this section, we show the hardness of computing the depth overhead of a given circuit C and a constraint graph G.

First, we present a few initial concepts that will be utilized in the subsequent proof.

Quantum gates and circuits. A SWAP gate implements the unitary which swaps the basis $|01\rangle$ and $|10\rangle$ and keeps $|00\rangle$ and $|11\rangle$ unchanged. A SWAP gate can be implemented by three CNOT gates.

Throughout this paper, we consider quantum circuits on n qubits. Qubit connectivity constraint is specified by an undirected graph $G = (V, E)$, where V is the set of n vertices, often identified with the n qubits, and E is the set of edges, specifying pairs of qubits that two-qubit gates can act on. A circuit without any qubit connectivity constraint is called an *unconstrained circuit*. A circuit made of SWAP gates only is called a *SWAP circuit*. A SWAP circuit C implements a permutation on qubits in the sense that $C|x_1x_2\dots x_n\rangle = |x_{\pi(1)}x_{\pi(2)}\dots x_{\pi(n)}\rangle$ for some qubit permutation $\pi \in S_n$.

For any qubit connectivity graph G and any permutation π of vertices, there is a natural one-one correspondence between the following two sets:

$$\{G\text{-compliant SWAP circuits } C \text{ of depth } d \text{ realizing } \pi\} \leftrightarrow \{\text{routing algorithm } A \text{ of } d \text{ rounds on } G \text{ realizing } \pi\}. \quad (24)$$

Indeed, each layer of a SWAP circuit C consists of SWAP gates on distinct qubits, which 1-1 correspond to one round of a routing algorithm consisting of swapping pebbles. Note that C and A implement the same permutation, and are compliant with the same graph G.

Two quantum circuits C and C' are equivalent if they realize the same unitary operation. In this paper, we are interested in a special *SWAP equivalence*, denoted by $C' \sim C$, in which C' is obtained from C by inserting SWAP gates and applying C's gates on permuted qubits. More specifically, suppose that $C = \prod_{i=1}^{d} L_i$ where L_i is the i-th layer, made of one- and two-qubit gates on distinct qubits. A circuit C' satisfies $C' \sim C$ if $C' = P(\pi)^\dagger \prod_{i=1}^{d} L'_i$, where L'_i is a circuit similar to L_i, but possibly with SWAP circuits inserted between the gates in L_i: If a gate U in L_i is on qubits p and q, then in L'_i it should be U applied to qubits $\sigma(p)$ and $\sigma(q)$, where σ is the permutation on qubits realized by the inserted SWAP gates in circuit C' from beginning to right before U; π is the permutation of qubits by *all* SWAP gates in circuit $\prod_{i=1}^{d} L'_i$, and $P_\pi^\dagger : |x_{\pi(1)}x_{\pi(2)}\dots x_{\pi(n)}\rangle \mapsto |x_1x_2\dots x_n\rangle$ is just to permute the qubits back to their original positions.

Languages and decision problems. A *language* L is a set of strings of finite lengths over a finite alphabet Σ. The corresponding *decision problem* D_L is to decide whether an input instance x belongs to the language L. A language L (or its decision problem D_L) is in NP if there is a polynomial-time Turing machine M such that the following two statements are equivalent for any input x: (1) $x \in L$, and (2) there is a certificate string y of length polynomial in that of x such that M accepts (x, y) [43]. A language L or its decision problem D_L is *NP-complete* if L is in NP and any other NP problem can be reduced to L in polynomial time. If an NP-complete language L reduces to an NP language L' in polynomial time, then L' is also NP-complete.

It is known that computing the routing number $\mathsf{rt}(G, \pi)$ of a given connected graph G and a permutation π is hard, as the following lemma states.

Lemma 10 (Routing deciding problem, [32])**.** *Deciding whether* $\mathsf{rt}(G, \pi) \leq k$ *on a given graph* G*, vertex permutation* π*, and any integer* $k \geq 3$ *is NP-complete.*

Second, using this result, we can show the hardness of computing the depth overhead.

Theorem 11 (Depth overhead deciding problem)**.** *Deciding whether* $\mathsf{doh}(G, C) \leq \alpha$ *on a given n-node graph* G*, a quantum circuit* C*, and a rational number* α *(with description length polynomial in* n*) is NP-complete.*

Proof. We first show that the hardness of deciding whether $\mathsf{doh}(G, C) \leq \alpha$ is no more than that of NP, or more precisely, the language $\{(G, C, \alpha) : \mathsf{doh}(G, C) \leq \alpha\}$ belongs to NP. Given a Yes input instance, i.e. a tuple (G, C, α) where $\mathsf{doh}(G, C) \leq \alpha$, we can use as a certificate a minimum-depth G-compliant circuit C' which is equivalent to C and obtained from C by inserting SWAP gates only. By definition of $\mathsf{doh}(G, C)$, the depth of C' is $d(C') = \mathsf{doh}(G, C) \cdot d(C)$, where $d(C)$ is the depth of C. The verification algorithm needs to check that (1) $d(C')/d(C) \leq \alpha$, and (2) $C' \sim C$. The first is trivially done in polynomial time, and the second is also easy as one can keep track of all permutations and check whether for each gate U in C and the corresponding gate U in C', whether the latter applies on the π-permuted qubits with π being the qubit permutation from the beginning of C' up to gate U.

Next, for any given graph $G = (V, E)$, permutation π and integer $k \geq 3$ of the routing deciding problem $\mathsf{rt}(G, \pi) \leq k$, we will efficiently construct a circuit C_π under no graph constraint, and define a rational number α, such that $\mathsf{rt}(G, \pi) \leq k$ if and only if $\mathsf{doh}(G, C_\pi) \leq \alpha$.

Circuit C_π and rational number α are determined as follows.

- For any permutation π of routing deciding problem $\mathsf{rt}(G, \pi) \leq k$, compute π^2.
- If $\pi^2 = id$, then π is a transposition and can be written as

  $$\pi = (a_1, a_2)(a_3, a_4) \cdots (a_{2m-1}, a_{2m})$$

  for some $m \leq n/2$ and distinct $a_1, a_2, \cdots, a_{2m} \in V$.

  - Define an unconstricted circuit $C_\pi = \prod_{i=1}^{m} \mathrm{SWAP}(a_{2i-1}, a_{2i})$, which consists of m SWAP gates. Since all a_i's are distinct, the depth of C_π is clearly $d(C_\pi) = 1$.
  - Let $\alpha = k$.
- If $\pi^2 \neq id$, then π is not a transposition. According to Lemma 1, π can be decomposed as two transpositions $\pi = \sigma_1 \circ \sigma_2$. As discussed above, we can construct two SWAP circuits of depth 1 for σ_1 and σ_2.

  - Combining these SWAP circuits gives an unconstrained SWAP circuit C_π with depth $d(C_\pi) = 2$.
  - Let $\alpha = k/2$.

Observe that in either case, the circuit C_π realizes permutation π in the natural sense that $C_\pi |x_1 x_2 \ldots x_n\rangle = |x_{\pi(1)} x_{\pi(2)} \ldots x_{\pi(n)}\rangle$, and that $\alpha \cdot d(C_\pi) = k$. Furthermore, the above algorithm mapping (G, π, k) to (G, C_π, α) can clearly be implemented in polynomial time. Now we will show that

$$\mathsf{rt}(G, \pi) \leq k \text{ if and only if } \mathsf{doh}(G, C_\pi) \leq \alpha.$$

First, we claim that the following equality holds for this particular C_π.

$$\min \left\{ d(C'_\pi) : C'_\pi \sim C_\pi, \text{ and } C'_\pi \text{ is } G\text{-compliant} \right\} = \mathsf{rt}(G, \pi). \tag{25}$$

- "$\geq$": Take any C'_π that achieves the minimum of the left-hand side. Since C_π realizes π and $C'_\pi \sim C_\pi$, we have that C'_π realizes π as well. Also note that C_π is a SWAP circuit, and C'_π is obtained from C_π by inserting SWAP gates, therefore C'_π is also a SWAP circuit. By the correspondence in Eq. (24), C'_π induces a $d(C'_\pi)$-round routing algorithm realizing π, thus $\mathsf{rt}(G,\pi)$ as the minimum number of rounds of such protocols is at most $d(C'_\pi)$.
- "$\leq$": Take any routing algorithm A with $\mathsf{rt}(G,\pi)$ rounds. Again by the correspondence in Eq. (24), A induces a $\mathsf{rt}(G,\pi)$-depth SWAP circuit $C(A)$ realizing π. Any $C'_\pi \sim C_\pi$ only inserts SWAP gates to C_π, thus is also a SWAP circuit. Therefore, the set in the left-hand side contains all G-compliant SWAP circuits realizing π, including $C(A)$. Thus the minimum depth $d(C'_\pi)$ is at most the depth of $C(A)$, which is $\mathsf{rt}(G,\pi)$.

Now with Eq. (25), we can see that

$$
\begin{aligned}
&\mathsf{doh}(G,C) \leq \alpha \\
\Leftrightarrow\ &\min\left\{d(C'_\pi) : C'_\pi \sim C_\pi,\ \text{and } C'_\pi \text{ is } G\text{-compliant}\right\}/d(C_\pi) \leq \alpha && \text{(by definition of } \mathsf{doh}(G,C)) \\
\Leftrightarrow\ &\frac{\mathsf{rt}(G,\pi)}{d(C_\pi)} \leq \alpha && \text{(by Eq. (25))} \\
\Leftrightarrow\ &\mathsf{rt}(G,\pi) \leq k && \text{(by definition of } \alpha)
\end{aligned}
$$

This completes the proof. □