Qubit Mapping Based on Subgraph Isomorphism and Filtered Depth-Limited Search

Sanjiang Li, Xiangzhen Zhou, Yuan Feng

Abstract—Mapping logical quantum circuits to Noisy Intermediate-Scale Quantum (NISQ) devices is a challenging problem which has attracted rapidly increasing interests from both quantum and classical computing communities. This paper proposes an efficient method by (i) selecting an initial mapping that takes into consideration the similarity between the architecture graph of the given NISQ device and a graph induced by the input logical circuit; and (ii) searching, in a filtered and depth-limited way, a most useful SWAP combination that makes executable as many as possible two-qubit gates in the logical circuit. The proposed circuit transformation algorithm can significantly decrease the number of auxiliary two-qubit gates required to be added to the logical circuit, especially when it has a large number of two-qubit gates. For an extensive benchmark set of 131 circuits and IBM's current premium Q system, viz., IBM Q Tokyo, our algorithm needs, in average, 0.4346 extra two-qubit gates per input two-qubit gate, while the corresponding figures for three state-of-the-art algorithms are 0.6047, 0.8154, and 1.0067 respectively.

Index Terms—NISQ, quantum circuit transformation, qubit mapping, subgraph isomorphism, heuristic search.

1 Introduction

Since Shor's exciting quantum algorithms for solving integer factorization and discrete logarithm [1], many quantum algorithms have been proposed that could offer an exponential speed-up when compared with best classical algorithms. These include in particular the HHL algorithm for solving systems of linear equations [2] and other machine learning algorithms derived from HHL (cf. [3] for a summary). Typically, the implementation of these algorithms requires quantum computers with millions of qubits which are perhaps still not available in the next two decades. On the other hand, IBM, Intel and Google have all announced their quantum devices with around 50-70 qubits recently. The Noisy Intermediate-Scale Quantum (NISQ) era seems coming in reality. Despite that quantum error correction is not yet available in the near future, quantum supremacy is recently demonstrated in Google's 53-qubit quantum processor Sycamore [4].

There is yet another gap between theoretical research on quantum algorithms and their implementation on realistic quantum devices. When designing quantum algorithms, typically, the quantum circuit model allows multi-qubit gates to act on any set of qubits without restriction. This is, however, not the case in realistic NISQ devices, which have "limited number of qubits, limited connectivity between qubits, restricted (hardware-specific) gate alphabets, and limited circuit depth due to noise" [5]. In the superconducting devices of IBM, Google, and Rigetti, only single and special two-qubit gates (like CNOT or CZ) are supported. Even worse, these two-qubit gates can only be implemented between neighbouring qubits. For example, Fig. 1 shows the architecture graph of IBM's current premium quantum system IBM Q Tokyo (also known as IBM Q20), which supports elementary single-qubit gates and two-qubit CNOT gates and a CNOT gate can be implemented only between qubits which are connected by an (undirected) edge. In order to use these NISQ devices, the desired quantum functionality in an ideal quantum circuit should be *transformed* or *mapped* so that the underlying coupling constraints imposed by these quantum devices are satisfied.

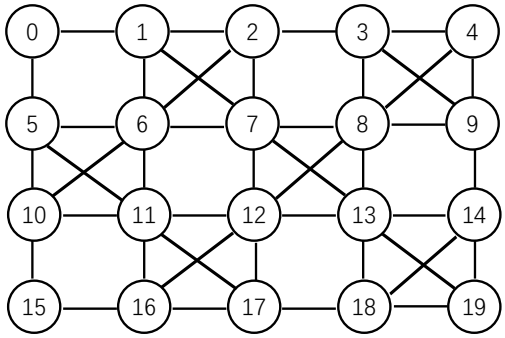

Fig. 1. The architecture graphs for IBM Q Tokyo.

More precisely, in order to implement an ideal quantum circuit on an NISQ device like IBM Q Tokyo, we need to address two issues. The first is to decompose the desired functionality (arbitrary quantum gates) into elementary operations that can be directly applied on the NISQ device. This issue has already been properly addressed in several works [6], [7], [8]. The second issue, known as *qubit mapping* or *circuit transformation*, is to map or route the qubits in the ideal quantum circuit to qubits of the quantum device so that the coupling constraints imposed by the quantum device are satisfied and thus the two-qubit gates in the ideal quantum circuit are executable.

- *Sanjiang Li and Yuan Feng are with Centre for Quantum Software and Information (QSI), Faculty of Engineering and Information Technology, University of Technology Sydney, NSW 2007, Australia. E-mail: {sanjiang.li, yuan.feng}@uts.edu.au*
- *Xiangzhen Zhou is with State Key Lab of Millimeter Waves, Southeast University, Nanjing 211189, China and Centre for Quantum Software and Information, University of Technology Sydney, NSW 2007, Australia.*

In the past several years, the qubit mapping problem has attracted rapidly increasing interests from both classical and quantum computing communities, see, e.g., [9], [10], [11], [12], [13], [14], [15], [16], [17], [18] and references therein. Given an arbitrary quantum circuit and an NISQ device, the task of qubit mapping is to construct automatically a quantum circuit with the same functionality which can be immediately implemented in the NISQ device. For ease of presentation, we call an arbitrary quantum circuit a *logical* circuit and call a circuit that is implementable in the NISQ device a *physical* circuit. Similarly, we call qubits in a logical (physical) circuit logical (physical) qubits. We assume that gates in the input logical circuit are already decomposed into elementary gates supported by the NISQ device. In particular, each gate in the logical circuit involves at most two qubits. Naturally, we also assume that the number of logical qubits is not larger than the number of physical qubits.

It is not difficult to find such a solution for an input logical circuit. Indeed, we can start with an arbitrary initial mapping and then execute one two-qubit gate per round (note single-qubit gates can be executed directly) by inserting several SWAP operations to transform the current mapping to a mapping that can execute the current two-qubit gate. The challenge lies in that if we can find a solution with minimal overhead in terms of the number of auxiliary SWAP gates added. This is crucial for the success of quantum computing as a large number of extra two-qubit gates will significantly accumulate the error of the output physical circuit.

Finding an optimal solution for the qubit mapping problem is often very difficult. Indeed, it is NP-complete [11] to decide if an input logical circuit can be transformed into an equivalent physical circuit using up to k SWAP gates for a fixed integer $k > 0$. Several previous works use off-the-shelf tools like dynamic programming [11], SAT solvers [10], temporal planners [12], Integer Linear Programming (ILP) [19], satisfiability modulo theory (SMT) solvers [13]. In worst cases, all these approaches take time exponential in the number of qubits.

Many other works devise specialised heuristic search algorithms for solving the qubit mapping problem. For example, Zulehner, Paler and Wille [14] partition the input logical circuit into layers and introduce an A^* search algorithm. When combined with a lookahead scheme and a dedicated method for selecting the initial mapping, their algorithm performs much better than IBM's own solution. However, this A^* algorithm also takes time exponential in the number of qubits. Li, Ding and Xie [15] propose a search algorithm based on reverse traversal, which is polynomial in the number of qubits and works very well for small circuits with less than a hundred two-qubit gates. Their algorithm, called SABRE, outperforms the A^*-approach [14] with exponential speedup and comparable or better results on various benchmarks. Childs, Schoute and Unsal [16] also propose efficient methods that attempt to minimise the circuit depth or size overhead and have worst-case time complexity polynomial in the sizes of the input circuit and the architecture graph. To this end, they decompose the problem into two subproblems: qubit movement and qubit placement. The first subproblem concerns how to transform the current mapping to a selected next mapping by imposing SWAP gates on edges in the architecture graph, while the second subproblem gives method to compute the next mapping. In another work, Cowtan et al. [17] describe a solution implemented in the platform-independent compiler t|ket⟩. They also partition the input circuit into layers and then select the SWAP which can maximally reduce the diameter of the subgraph composed of all pairs of qubits in the current layer. We address their algorithm as the Cambridge algorithm henceforth. In [18], we designed a new qubit mapping algorithm, called SAHS in this paper, which uses simulated annealing for constructing an initial mapping and searches the next mapping by using a heuristic function that reflects the variable influence of gates in different layers. Empirical results showed that SAHS outperforms both SABRE and the Cambridge algorithm by a large margin. Initial mappings of SAHS are, however, computed non-deterministically by simulated annealing, which is sometimes unstable and runs slowly when the circuit size is large [18].

In this paper, we propose a new search algorithm based on subgraph isomorphism and filtered depth-limited search. The idea is to construct a graph G from the input circuit which is isomorphic to a subgraph of $\mathcal{AG}$, the architecture graph of the given NISQ device, and select any embedding from G to $\mathcal{AG}$ as the initial mapping. Starting from this initial mapping, we then, step by step, construct the physical circuit while removing executable gates from the logical circuit. If the current mapping can execute some gates in the front layer of the logical circuit, we remove them from the logical circuit and properly append them to the current physical circuit; if there are no executable gates in the front layer, then we need to insert SWAP gates and obtain a new mapping so that some two-qubit gates in the front layer can be executed. To select a good next mapping, we tend to exhaustively search all possible combinations of SWAP operations such that the number of executable two-qubit gates per SWAP is maximised. As selecting the best SWAP combination is expensive, we set up a fixed limit $k > 0$ and only consider combinations of at most k SWAPs. The search process could be further sped up if we 'filter' those SWAPs which do not interact with gates in the front layers of the circuit. Such filters are designed and used in our algorithm. While less efficient when compared with SABRE, our algorithm is, if neglecting not considering the time for computing the initial mapping, still polynomial in all relevant parameters and can significantly reduce the number of SWAPs required to transform the input logical circuit. Indeed, empirical evaluation shows that our algorithm can often reduce by half the number of SWAPs required when compared with SABRE, if the input logical circuit has hundreds or more two-qubit gates. Similar empirical evaluations also show that our algorithm is significantly better than the Cambridge algorithm developed by Cowtan et al. [17] and our SAHS algorithm [18] in terms of the size of the output circuits, when no postmapping optimisation is used.

The remainder of this paper is organised as follows. In Section 2, we recall some background of quantum computation and quantum circuits, and describe and analyse our algorithm in Section 3. Detailed empirical evaluation is reported in Section 4. The last section concludes the paper

with discussions on directions for future research.

2 BACKGROUNDS

In this section, after a brief introduction of quantum gates and quantum circuits, we describe the dependency graph associated to a logical circuit and show how to partition the logical circuit into layers by using the dependency graph.

2.1 Quantum Gates and Quantum Circuits

Qubit is the counterpart of bit in quantum computation. While a 'classical' bit can only be in one of two states, viz., 0 and 1, a qubit can be in the superposition state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ of the two basis states, $|0\rangle$ and $|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ are probability amplitudes satisfying $|\alpha|^2 + |\beta|^2 = 1$. For example, $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ are two superposition states. The success of quantum computation partially lies in ingenious use of quantum superposition.

Quantum computation is realised by applying quantum gates on qubits. Complex, multi-qubit gates can be decomposed into elementary single or two-qubit gates. In fact, any quantum gate can be approximated to arbitrary accuracy using a fixed set of single-qubit gates and CNOT gates [20]. Fig. 2 illustrates three very useful gates: Hadamard gate H, CNOT gate and SWAP gate. Hadamard gate is a single-qubit gate which can evenly mix the basis states to produce a superposed one. Precisely, H maps $|0\rangle$ to $|+\rangle$ and $|1\rangle$ to $|-\rangle$. CNOT and SWAP are both two-qubit gates, i.e., they operate on two qubits. A CNOT gate flips the target qubit (indicated graphically with $\oplus$) if and only if the control qubit (indicated graphically with a black dot $\bullet$) is in state $|1\rangle$, while a SWAP gate exchanges the states of the two qubits operated. Precisely, CNOT maps $|a\rangle|b\rangle$ to $|a\rangle|a \oplus b\rangle$ and SWAP maps $|a\rangle|b\rangle$ to $|b\rangle|a\rangle$ for $a, b \in \{0, 1\}$. Most NISQ devices do not support SWAP gates directly and, if this is the case, we may implement a SWAP gate by three CNOT gates (see Fig. 2 (right)).

Quantum circuits are the most commonly used model to describe quantum algorithms, which consist of input qubits, quantum gates, measurements and classical registers [21]. As only input qubits and quantum gates are relevant in the qubit mapping problem, in this paper, we represent a quantum circuit simply as a pair (Q, C), where Q is the set of involved qubits and C a sequence of quantum gates.

2.2 Dependency Graph and Front Layer

Two-qubit gates in a logical circuit $LC = (Q, C)$ are not independent in general. We say a two-qubit gate g_1 *depends* on another two-qubit gate g_2 if the latter must be executed before the former. This happens when g_2 is in front of g_1 in C and they share a common qubit, or when g_1 depends on a two-qubit gate g_3 which depends on g_2. For clarity, we say g_1 *directly depends* on g_2 if g_2 is in front of g_1 in C and they share a common point and there are no other gates between them which share the same common point.

For a logical circuit $LC = (Q, C)$, we construct a directed acyclic graph (DAG), called the *dependency graph*, to characterise the direct dependency between two-qubit gates in LC [15], [16]. Each node of the dependency graph represents a two-qubit gate and each directed edge the direct dependency relationship from one two-qubit gate to another. The front layer of LC, denoted $\mathcal{F}(LC)$ or $\mathcal{L}_0(LC)$, consists of all two-qubit gates in LC which have no parents in the dependency graph. The second layer $\mathcal{L}_1(LC)$ is then the front layer of the circuit obtained from LC by deleting all gates in $\mathcal{F}(LC)$. Analogously, we can define the k-th layer $\mathcal{L}_k(LC)$ of LC for all $k \geq 0$.

Example 1. *Consider the logical circuit $LC = (Q, C)$ shown in Fig. 3 (left), where*

$$Q = \{q_0, q_1, q_2, q_3\},$$
$$C = (g_0 \equiv \langle q_2, q_0\rangle, g_1 \equiv \langle q_3, q_2\rangle, g_2 \equiv \langle q_0, q_3\rangle, g_3 \equiv \langle q_0, q_2\rangle,$$
$$g_4 \equiv \langle q_3, q_2\rangle, g_5 \equiv \langle q_0, q_3\rangle, g_6 \equiv \langle q_3, q_1\rangle),$$

where $\langle q_2, q_0\rangle$, for example, denotes the CNOT *gate in the logical circuit with q_2 being the control qubit and q_0 the target.*

For this circuit, we have $\mathcal{F}(LC) = \{g_0\}$, $\mathcal{L}_1(LC) = \{g_1\}$, $\mathcal{L}_2(LC) = \{g_2\}$, and $\mathcal{L}_3(LC) = \{g_3\}$, and so on. From the dependency graph (showing in Fig. 3 (right)), we can see that, for example, gate g_2 can be executed only after g_0 and g_1.

3 THE PROPOSED APPROACH

The main objective of qubit mapping is to transform an input logical circuit to a physical one with minimal size or depth so that the constraints imposed by the NISQ device are satisfied. To simplify the discussion, we only consider the connectivity constraints for two-qubit gates as specified by the architecture graph. This means that single-qubit gates have no effect in the circuit transformation process. Furthermore, we make the following assumptions:[1]

1) The NISQ device supports all single-qubit gates and CNOT gates;
2) The architecture graph of the NISQ device, $\mathcal{AG}$, is an undirected graph;
3) CNOT gates are the only two-qubit gates in the input logical circuit.

From now on and as in Example 1, we write a CNOT gate simply as a pair $\langle q, q'\rangle$, where q is the control qubit and q' is the target qubit. We call the CNOT gate $\langle q', q\rangle$ the *inverse* of $\langle q, q'\rangle$.

Let $\mathcal{AG} = (V, E)$ be the undirected architecture graph of the NISQ device we are given, where V is the set of physical qubits and E the set of edges along which CNOT gates can be performed. Recall that an edge e in an undirected graph is an unordered pair of the two endnodes p, q of e. In the following, we write e simply by $\{p, q\}$, i.e., the set of its two endnodes.

Let $LC = (Q, C)$ be a logical circuit with $|Q| \leq |V|$. Suppose C consists of only CNOT gates after removing all single-qubit gates. We need to construct a physical circuit $PC = (V, C^p)$ which contains only CNOT gates, is functionally equivalent to LC after adding back all single-qubit gates accordingly, and respects the connectivity constraints imposed by $\mathcal{AG}$, i.e., for any CNOT gate $\langle q, q'\rangle$ in C^p, we have $\{q, q'\} \in E$.

1. The occurrences of CNOT gates may be replaced by CZ gates when, e.g., a Rigetti device is used.

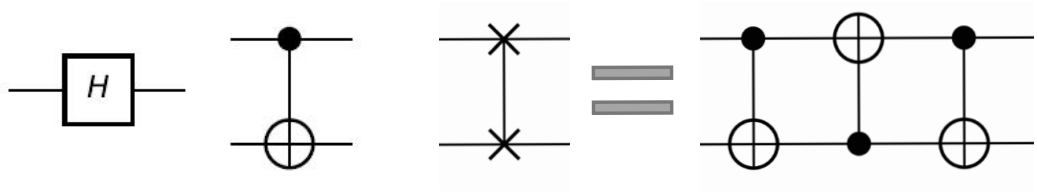

Fig. 2. Hadamard, CNOT and SWAP gate (from left to right).

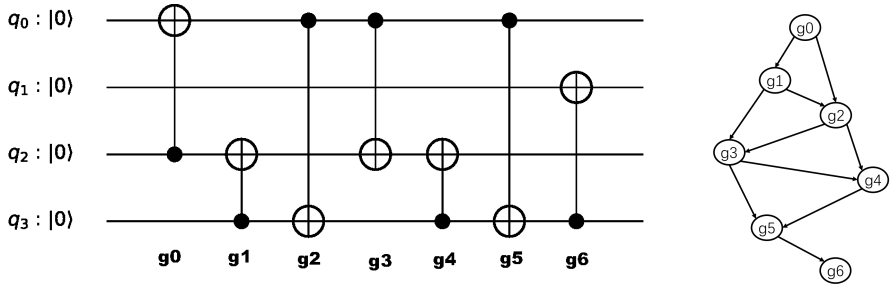

Fig. 3. A logical quantum circuit with only CNOT gates (left) and its dependency graph (right).

It is easy to find a physical circuit that satisfies the above conditions, but the real challenge is to find one with *minimal* size or depth, which is NP-hard in general [11]. In this paper, we modify the input logical circuit stepwise by inserting auxiliary SWAP operations (each implemented with three CNOT gates as in Fig. 2 (right)) until the logical circuit is transformed into a physical circuit that can be executed on the NISQ device. To evaluate the effectiveness of qubit mapping algorithms, we use the sizes of the output circuits, i.e., the total number of its two-qubit gates.

3.1 Qubit Mapping

In each step, qubits in the logical circuit are mapped or allocated to physical qubits in the NISQ device. Mathematically, a (partial) qubit mapping is a (partial) function τ from Q to V such that $\tau(q) = \tau(q')$ if and only if $q = q'$ for any $q, q' \in Q$. We say a partial qubit mapping is *complete* if it is defined for every q in Q. A physical qubit v is *occupied* if τ maps some logical qubit q to v. If otherwise, we say v is unoccupied. The mapping may change in consecutive steps of the transformation which is determined by the inserted auxiliary SWAP operations.

Given a logical circuit LC and a mapping τ, a CNOT gate $g = \langle q, q' \rangle$ in LC is said to be *satisfied* by τ, or τ satisfies g, if $\{\tau(q), \tau(q')\}$ is an edge in $\mathcal{AG}$. Furthermore, g is *executable* by τ if it appears in the front layer of LC and τ satisfies g. If this is the case, we remove g from LC and append a CNOT gate $\tau(g) := \langle \tau(q), \tau(q') \rangle$ to the end of the physical circuit. This process is called the *execution* of g.

For two physical qubits v, v' in $\mathcal{AG}$, we write $\mathsf{dist}_{\mathcal{AG}}(v, v')$ for the distance (i.e., the length of a shortest path) from v to v' in $\mathcal{AG}$.

For any mapping τ and any two-qubit gate $g = \langle q, q' \rangle$, the *physical distance* between the two qubits q, q' in g under mapping τ, written $\mathsf{dist}_{ph}(q, q', \tau)$, is defined as

- the distance between $\tau(q)$ and $\tau(q')$ in $\mathcal{AG}$, i.e., $\mathsf{dist}_{\mathcal{AG}}(\tau(q), \tau(q'))$, if both $\tau(q)$ and $\tau(q')$ are defined,
- the shortest distance between $\tau(q)$ or $\tau(q')$ to all unoccupied physical qubits if only one of $\tau(q)$ and $\tau(q')$ is defined, and
- the shortest distance between two unoccupied qubits if neither $\tau(q)$ nor $\tau(q')$ is defined.

Apparently, a gate $g = \langle q, q' \rangle$ in the front layer is executable by τ (after possible extension if τ is incomplete) iff the physical distance between q and q' under τ is 1.

Example 2 (Example 1 cont'd)**.** *Consider the logical circuit LC shown in Fig. 3. Let $\tau_1 : Q \to V$ be the mapping specified by $\tau_1(q_0) = v_2$, $\tau_1(q_1) = v_0$, $\tau_1(q_2) = v_{10}$, $\tau_1(q_3) = v_6$, see Figure 4 (left). Then, because $\tau_1(q_2) = v_{10}$ and $\tau_1(q_0) = v_2$ and $\mathsf{dist}_{\mathcal{AG}}(v_2, v_{10}) = 2$, we can see that $g_0 \equiv \langle q_2, q_0 \rangle$ is not executable by τ_1 in IBM Q Tokyo. However, for the mapping τ_3 shown in Figure 4 (right), g_0 is executable by τ_3 and, indeed, every gate in LC is satisfied by τ_3.*

3.2 Initial Mapping

An initial mapping can be constructed step by step or selected arbitrarily or computed from a dedicated subroutine. Zulehner et al. [14] tested both arbitrary initial mappings and initial mappings evolved from an empty mapping. Their experimental evaluation shows that, in general, the latter approach has better performance. In [15], Li et al. proposed to use an initial mapping that takes the whole input circuit into consideration. Starting from a randomly generated mapping τ_0, they first take this mapping as the initial mapping and apply it on the input circuit (Q, C), the obtained final mapping τ_1 is then used as the initial mapping and applied to the inverse circuit[2] (Q, C^{inv}), and lastly, the obtained final mapping τ_f is selected as the initial mapping for their main algorithm SABRE. Their approach is demonstrated as consistently better than the A^* search algorithm in [14]. In [18], we proposed SAHS, which uses

2. Note the "inverse" here has a different meaning as the "inverse" CNOT gate defined in page 3.

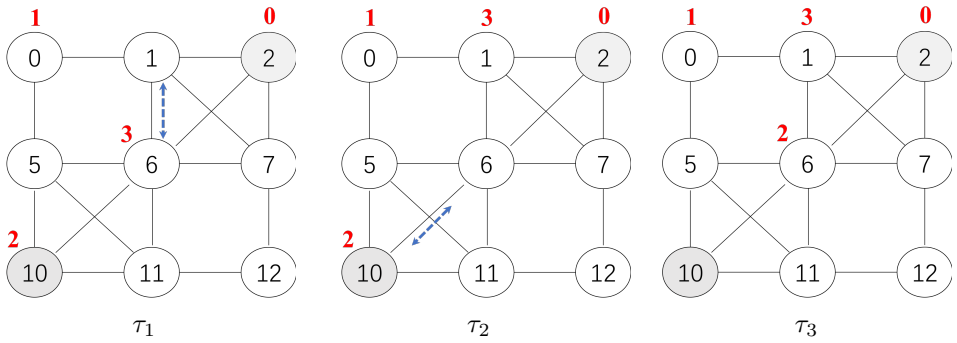

Fig. 4. Qubit mappings $\tau_i : \{q_0, q_1, q_2, q_3\} \to V$ for $i = 1, 2, 3$, where τ_2 is obtained from τ_1 by SWAP(1,6) and τ_3 is obtained from τ_2 by SWAP(6,10), where SWAP$(1, 6)$ is a shorthand for SWAP(v_1, v_6).

simulated annealing to search for the best initial mapping that fits well with the input logical circuit and empirical evaluation there shows that, when compared with the naive mapping that sends q_i to v_i, SAHS works significantly better with the initial mapping obtained by simulated annealing.

In this section we show how to obtain a good initial mapping by matching a particular graph induced by the input circuit with the architectural graph.

Suppose $LC = (Q, C)$ is the input logical circuit. We first construct an undirected graph $\mathcal{G}_{\text{circ}}(C) = (Q, E_{\text{circ}})$ on Q, where $\{q, q'\}$ is an edge in E_{circ} if either $\langle q, q' \rangle$ or its inverse $\langle q', q \rangle$ is in C. If $\mathcal{G}_{\text{circ}}(C)$ happens to be isomorphic to a subgraph of the architecture graph $\mathcal{AG}$, then the qubit mapping problem is solved by constructing an (arbitrary) isomorphic embedding τ from $\mathcal{G}_{\text{circ}}(C)$ to $\mathcal{AG}$. For NISQ devices, which have up to several thousands qubits, this can be solved by, for example, the VF2 algorithm [22]. If $\mathcal{G}_{\text{circ}}(C)$ is not isomorphic to a subgraph of $\mathcal{AG}$, then we may select a maximal sub-circuit C_{top} of C such that

(i) C_{top} is a front section of $C = \{g_i \in C \mid 1 \leq i \leq n\}$ w.r.t. the dependency graph of C, i.e., a two-qubit gate g_i is in C_{top} only if all two-qubit gates g_j on which g_i depends are in C_{top};
(ii) the graph $\mathcal{G}_{\text{circ}}(C_{\text{top}})$ is isomorphic to a subgraph of $\mathcal{AG}$; and
(iii) the graph $\mathcal{G}_{\text{circ}}(C_{\text{top}} \cup \{g_{i^*}\})$ is not isomorphic to a subgraph of $\mathcal{AG}$ for any g_{i^*} that is in the front layer of $C \setminus C_{\text{top}}$.

We call $\mathcal{G}_{\text{circ}}(C_{\text{top}})$ a *top subgraph* (*topgraph* for short) of $\mathcal{G}_{\text{circ}}(C)$. Let τ_{top} be an isomorphic embedding from $\mathcal{G}_{\text{circ}}(C_{\text{top}})$ to $\mathcal{AG}$. We select τ_{top} as the initial mapping, which satisfies all gates in C_{top}. Note that τ_{top} might not be a complete mapping from Q to V.

Example 3 (Example 1 cont'd). *For the circuit LC in Example 1, we have $Q = \{q_0, q_1, q_2, q_3\}$ and $E_{\text{circ}} = \{\{q_0, q_2\}, \{q_0, q_3\}, \{q_2, q_3\}, \{q_1, q_3\}\}$. Clearly, $\mathcal{G}_{\text{circ}}(C)$ is isomorphic to a subgraph of $\mathcal{AG}$ and such an isomorphism is specified by the qubit mapping τ_3 in Fig. 4 (right).*

Note that τ_{top} often does not take the whole circuit into consideration. We propose another method for constructing the initial mapping that considers the whole circuit. For a logical circuit $LC = (Q, C)$, we introduce a weight function ω which assigns a weight on each edge of E_{circ} (the edge set of the undirected graph $\mathcal{G}_{\text{circ}}(C)$ defined above) such that $\omega(\{q, q'\})$ is the number of gates g_i in C with $g_i = \langle q, q' \rangle$ or $g_i = \langle q', q \rangle$. Let $E^w_{\text{circ}} = \{e_1, e_2, ..., e_n\}$ where $\omega(e_1) \geq \ldots \geq \omega(e_n)$. We then construct a subgraph $\mathcal{G}^* = (Q, E^*)$ of $\mathcal{G}_{\text{circ}}(C)$ which is isomorphic to a subgraph of $\mathcal{AG}$ as follows. We start by letting $E^* = \{e_1\}$ and then consider the next edge e_2. In general, suppose we have decided if e_i should be put into E^* or not for all $i < k$ for some $k \leq n$ and the current subgraph $\mathcal{G}^*$ is isomorphic to some subgraph of $\mathcal{AG}$. We consider e_{i+1}. If putting e_{i+1} into E^* will make $\mathcal{G}^*$ non-isomorphic to any subgraph of $\mathcal{AG}$, we skip this edge; otherwise, we put e_{i+1} into E^* and update $\mathcal{G}^*$, which is still isomorphic to some subgraph of $\mathcal{AG}$. If $i + 1 < n$, we continue to consider e_{i+2} till there is no edge left in E^w_{circ}. In this way, we obtain a subgraph $\mathcal{G}^*$ of $\mathcal{G}_{\text{circ}}(C)$ that is isomorphic to some subgraph of $\mathcal{AG}$. The sum of weights of edges in $\mathcal{G}^*$, though not necessary the largest, is sufficiently large among all subgraphs of $\mathcal{G}_{\text{circ}}(C)$ that are isomorphic to some subgraph of $\mathcal{AG}$. Using the VF2 algorithm, we can find an embedding τ_{wgt} which embeds $\mathcal{G}^*$ into $\mathcal{AG}$. Again, we note that τ_{wgt} might be a partial mapping from Q to V.

In the following, we call τ_{top} the topgraph initial mapping and call τ_{wgt} the weighted graph initial mapping of LC. Besides these two initial mappings, we also introduce a method for evolving an initial mapping from the empty mapping. Similar idea was used by Zulehner et al. [14], while we extend a partial mapping only when necessary, i.e., when the thus extended mapping can execute a two-qubit gate in the current front layer or it can reduce the *minimum* physical distance (cf. Section 3.1) between qubits in a two-qubit gate in the current front layer. This mapping extension technique is also used when τ_{top} or τ_{wgt} is incomplete.

3.3 Fixed-Depth Heuristic Search

In most search-based algorithms for the qubit mapping problem, a heuristic function is used to select an action (i.e., a SWAP or a sequence of SWAPs) which can maximally reduce the sum or the minimum of the distances between the two qubits in the CNOT gates of the front layer and, sometimes, the lookahead layer.

For each edge $e = \{v, v'\}$ in $\mathcal{AG}$ there is an associated SWAP operation, written SWAP(e), which swaps the states on v and v'. More precisely, suppose τ is the current mapping and $\tau(q) = v$, $\tau(q') = v'$. Then SWAP(e) transforms τ into a new mapping τ' such that $\tau'(q) = v'$, $\tau'(q') = v$, and

$\tau'(q^*) = \tau(q^*)$ for $q^* \notin \{q, q'\}$. In case if $\tau(q)$ ($\tau(q')$) is not defined and $\tau(q') = v'$ ($\tau(q) = v$), then $\tau'(q')$ ($\tau'(q)$) is not defined and $\tau'(q) = v'$ ($\tau'(q') = v$). If both are undefined, then $\tau' = \tau$. We often write $\text{SWAP}(e) \circ \tau$ for τ'.

Example 4 (Example 1 cont'd)**.** *For the three qubit mappings in Fig. 4, we have* $\tau_2 = \text{SWAP}(1,6) \circ \tau_1$ *and* $\tau_3 = \text{SWAP}(6,10) \circ \tau_2$.

In this section, we propose a new heuristic function which measures how *efficient* the mapping can execute gates in the logical circuit. For convenience, we say a CNOT gate (in or not in the front layer) is executable by a mapping τ if itself and all CNOT gates it depends on are satisfiable by τ.

Starting with a selected initial mapping τ^0, we write $s^0 = (\tau^0, PC^0, LC^0)$ for the initial state of the search process, where LC^0 is obtained by removing all CNOT gates $\langle q, q' \rangle$ that are executable by τ^0 from LC, and PC^0 is obtained by adding the corresponding CNOT gates $\langle \tau^0(q), \tau^0(q') \rangle$ in an empty physical circuit. Step by step, we select an action a from $\mathcal{S}$, the set of sequences of SWAPs on $\mathcal{AG}$ and enforce all SWAPs in a one by one to get the next mapping (and the next state) till there are no gates left in the logical circuit.

Suppose s^i is the current state with $s^i = (\tau^i, PC^i, LC^i)$ and all gates that are executable by τ^i are already removed from LC^i. For a sequence $a = (\text{SWAP}_1, \text{SWAP}_2, ..., \text{SWAP}_\ell)$ of SWAPs on $\mathcal{AG}$, we define a value function

$$\mathsf{val}(\tau^i, a) = \frac{\text{number of gates executable by } \tau'}{\mathsf{len}(a) \times 3}, \quad (1)$$

where τ' is the mapping obtained by enforcing SWAPs in action a one by one on τ^i and $\mathsf{len}(a) = \ell$ is the number of SWAPs in a. Recall each SWAP is implemented by three CNOT gates (see Fig. 2 (right)).

Our action set consists of all sequences of SWAPs on $\mathcal{AG}$ and we select any one with the maximal value, i.e., we select a^* from

$$\arg\max_{a \in \mathcal{S}} \ \mathsf{val}(\tau^i, a). \quad (2)$$

After selecting a^*, we enforce on τ^i SWAPs in a^* one by one and obtain the next mapping τ^{i+1}. Then we remove all gates that are executable by τ^{i+1} from LC^i and write LC^{i+1} for the resulted logical circuit. In the meanwhile, we append to PC^i three CNOT gates (as in Fig. 2 (right)) for each SWAP in a^*, and a CNOT gate $\langle \tau^{i+1}(q), \tau^{i+1}(q') \rangle$ for each CNOT gate $\langle q, q' \rangle$ removed from LC^i. In this way, we obtain PC^{i+1} and the next state $s^{i+1} = (\tau^{i+1}, PC^{i+1}, LC^{i+1})$.

Apparently, considering all sequences of SWAPs is inefficient. In practice, we propose to consider actions with up to k SWAPs for some fixed $k \geq 1$. In particular, for IBM Q Tokyo, we select $k = 3$, which reflects a good compromise between efficiency and effectiveness.

Example 5 (Example 1 cont'd)**.** *Suppose* τ_1 *is the qubit mapping which maps* q_0, q_1, q_2, q_3 *to, respectively,* v_2, v_0, v_{10}, v_6*, see Fig. 4 (left). As the front layer contains only* $g_0 = \langle q_2, q_0 \rangle$*, which is not executable by* τ_1*, there are no gates in* LC *that can be executed by* τ_1*. Examining all sequences of up to 3* SWAPs*, the four best actions are as follows:*

- $a_1 = (\text{SWAP}(1,6), \text{SWAP}(6,10))$*, which can execute all 7 gates in* LC*;*
- $a_2 = (\text{SWAP}(5,6), \text{SWAP}(2,6))$*, which can execute all 7 gates in* LC*;*
- $a_3 = (\text{SWAP}(6,7), \text{SWAP}(6,10))$*, which can execute all but the last gate in* LC*;*
- $a_4 = (\text{SWAP}(6,11), \text{SWAP}(2,6))$*, which can execute all but the last gate in* LC*.*

The fifth best action contains 3 SWAPs*. Thus* a_1 *and* a_2 *are the optimal actions, with the optimal value* $\mathsf{val}(\tau_1, a_1) = \mathsf{val}(\tau_1, a_2) = 7/6$.

3.3.1 Heuristics used in related works

Now it is a good time to compare our heuristic function with those used in the related works.

Zulehner et al. [14] select the action that results in a mapping which can execute all gates in the front layer and the lookahead layer. The action consists of a sequence of SWAPs and is selected by using A^* search and the following heuristics:

$$h(\tau^i) = \sum \Big\{ 3 \times (\mathsf{dist}_{\mathcal{AG}}(\tau^i(q), \tau^i(q')) - 1) \mid \langle q, q' \rangle \in \mathcal{L}_0(LC^i) \cup \mathcal{L}_1(LC^i) \Big\},$$

where, for any two-qubit gate $\langle q, q' \rangle$, $\mathsf{dist}_{\mathcal{AG}}(\tau^i(q), \tau^i(q'))$ is the distance from $\tau^i(q)$ to $\tau^i(q')$ in $\mathcal{AG}$. The heuristic cost is not admissible and thus an optimal action is not guaranteed. Moreover, the worst-case time complexity of this A^* search algorithm is exponential in the number of logical qubits.

Childs et al. [16] select the action which can maximally reduce the total distance between qubits in the CNOT gates in the current front layer, i.e.,

$$R(\tau^i) = \sum \Big\{ \mathsf{dist}_{\mathcal{AG}}(\tau^i(q), \tau^i(q')) \mid \langle q, q' \rangle \in \mathcal{L}_0(LC^i) \Big\}. \quad (3)$$

Their algorithm is polynomial in all relevant parameters but its performance is not directly compared with the A^* algorithm in [14].

To overcome the inefficiency of the A^* search algorithm, several researchers (see, e.g., [15], [17]) propose to select a *single* SWAP each time. Their methods are more efficient than the A^* approach when processing logical circuits with more than 15 qubits. In [15], Li et al. design a heuristic cost function that can reduce the sum of distances between the two qubits in each two-qubit gate in the front (and the lookahead) layers. Analogously, Cowtan et al. [17] use a heuristic cost function that can reduce the diameter of the subgraph composed of all qubits in the two-qubit gates of the front layer. In the evaluation section, we will see that the efficiency of these algorithms is achieved at the cost of the quality of the output physical circuit.

In the SAHS algorithm [18], we introduce a heuristic function that supports weight parameters to reflect the variable influence of gates in different layers. In each step, instead of selecting the action with the minimal cost to apply, the SAHS algorithm selects the SWAP which has the best consecutive SWAP to apply.

3.4 Optimisation and Fallback

Considering all sequences of up to k SWAPs is still not very efficient for devices with a medium to large architecture graph. Let τ be the current mapping and $\mathcal{L}_i$ the current i-th $(0 \leq i \leq k)$ layer. Write Q_i for the set of logical qubits in $\mathcal{L}_i$. It's natural not to consider SWAPs that do not interact with gates in the front layers. This idea was used for edges in the front layer in, e.g., [14], [15], [18].

For an edge $e = \{v, v'\}$, if neither $\tau^{-1}(v)$ nor $\tau^{-1}(v')$ is in Q_0, then swapping v and v' does not reduce the minimum distance between qubits in a two-qubit gate in the current front layer, viz. $\mathcal{L}_0$. Therefore, it is reasonable to introduce the following filter for selecting actions $a = (e_1, e_2, ..., e_\ell)$ with at most k SWAPs:

1. Q_0-filter: We say $a = (e_1, e_2, ..., e_\ell)$ is a Q_0-plausible action if, for any edge $e_j = \{v, v'\}$ of a, we have either $\tau_{j-1}^{-1}(v)$ or $\tau_{j-1}^{-1}(v')$ is in Q_0, where $\tau_0 \equiv \tau$ and τ_j is obtained from τ_{j-1} by enforcing $\text{SWAP}(e_j)$ for $1 \leq j \leq \ell$.

Similarly, we could look ahead and introduce Q_i-filter for $0 < i < k$.

2. Q_i-filter: We say $a = (e_1, e_2, ..., e_\ell)$ is a Q_i-plausible action if, for any edge $e_j = \{v, v'\}$ of a with $j > i$, we have either $\tau_{j-1}^{-1}(v)$ or $\tau_{j-1}^{-1}(v')$ is in Q_i, where $\tau_0 \equiv \tau$ and τ_j is obtained from τ_{j-1} by enforcing $\text{SWAP}(e_j)$ for $1 \leq j \leq \ell$.

The above Q_i-filter could be weakened by requiring that either $\tau_{i-1}^{-1}(v)$ or $\tau_{i-1}^{-1}(v')$ is in (a subset of) $Q_0 \cup Q_1 \cup \cdots \cup Q_i$. In our evaluation, we used Q_0 and (weakened) Q_1 filters and the results are very promising (see Section 4).

It should be stressed that, sometimes, Q_0-filter may 'filter' out optimal actions.

Example 6 (Example 1 cont'd)**.** *Note that $Q_0 = \{0, 2\}$ and $\tau_1(q_0) = v_2$, $\tau_1(q_2) = v_{10}$. Each a_i for $1 \leq i \leq 4$ in Example 5 is not Q_0-plausible. Thus our algorithm with Q_0-filter cannot find an optimal action. In fact, the following Q_0-plausible action is selected by our algorithm*

- $a_5 = (\text{SWAP}(2, 7), \text{SWAP}(1, 6), \text{SWAP}(6, 10))$.

This action (see Fig. 4) can execute all 7 gates in LC and has $\mathit{val}(\tau_1, a_5) = 7/9$.

Another type of filters is also introduced in our algorithm. Let τ be the current mapping and $\mathcal{L}_i$ the i-th layer of the current logical circuit. Recall the notion of physical distance defined in Sec. 3.4 and assume that $\gamma \in [0, 1]$ is a discount factor and $s \geq 0$. For two mappings τ_1 and τ_2, we say τ_1 is s-better than τ_2 if $\hat{R}_s(\tau_1) < \hat{R}_s(\tau_2)$, where for $i = 1, 2$ we have

$$\hat{R}_s(\tau_i) = \sum_{\ell=0}^{s} \gamma^\ell \times \left(\sum \{ \mathsf{dist}_{ph}(q, q', \tau_i) \mid \langle q, q' \rangle \in \mathcal{L}_\ell \} \right). \quad (4)$$

Intuitively, τ_1 being s-better than τ_2 implies that it is easier to execute all gates in the first s-level if we starts from τ_1 instead of τ_2.

Let τ be the current mapping. We define the D_s-filter as follows.

3. D_s-filter: We say $a = (e_1, e_2, ..., e_\ell)$ is a D_s-plausible action if τ_{j-1} is *not* s-better than τ_j for any $1 \leq j \leq \ell$, where $\tau_0 \equiv \tau$ and τ_j is obtained from τ_{j-1} by enforcing $\text{SWAP}(e_j)$ for $1 \leq j \leq \ell$.

Note that if $\gamma = 0$ then D_s-filter is the same as D_0-filter.

Fallback

For a prefixed positive integer k, it is possible that, in some cases, no sequence of SWAPs with length $\leq k$ can lead to a mapping which can execute any CNOT gate in the current front layer. If this is the case, we use the following natural fallback:

Fallback: Select any SWAP that can reduce $FB(\tau)$, the minimum distance between qubits in a two-qubit gate in the current front layer, which is formally defined as follows:

$$FB(\tau) = \min \left\{ \mathsf{dist}_{\mathcal{AG}}(\tau(q), \tau(q')) \mid \langle q, q' \rangle \in \mathcal{L}_0 \right\}. \quad (5)$$

It is worth noting that, for our experiments on IBM Q Tokyo and an extensive set of logical circuits, the fallback is rarely activated.

3.5 Complexity Analysis

In the following we give a rough estimation of the complexity of the search process of our algorithm.

The construction of the topgraph and weighted graph initial mappings requires finding an isomorphic graph embedding. In general, it is NP-hard to check if a graph is isomorphic to a subgraph of another but there are efficient algorithms, say the VF2 algorithm [22], which can quickly solve this problem and output an embedding if the answer is yes for graphs with several thousands nodes. Therefore, for the purpose of qubit mapping with NISQ devices, we don't take into our analysis the time of computing an initial mapping. Experiments on various architecture graphs with up to 361 nodes and circuits with up to 50 qubits and 15,000 CNOT gates show that the time consumption is acceptable. For detailed discussion please see Sec. **??**.

Suppose $LC = (Q, C)$ is a logical circuit and $\mathcal{AG} = (V, E)$ is the architecture graph of a NISQ device. Write $|Q|$, $|V|$, and $|E|$ be, respectively, the cardinalities of Q, V, and E. Let $diam$ be the diameter of $\mathcal{AG}$ and m the number of CNOT gates in C.

We have the following simple observations:

- The dependency graph of LC can be computed in time linear in m.
- For any mapping τ and any logical circuit LC, we can identify (and remove from LC as well as from its dependency graph) in time linear in m the set of gates in LC executable by τ.

We first consider the ideal case when fallback is never activated during the search process. As described in Section 3.3, starting with a selected initial mapping τ^0, step by step, we select an action a consisting of up to k SWAPs on $\mathcal{AG}$ and enforce all SWAPs in a one by one to get the next mapping till there are no gates left in the logical circuit. Suppose $s^i = (\tau^i, PC^i, LC^i)$ is the current search state. As there are at most $O(|E|^k)$ actions with up to k SWAPs, we can generate at most $O(|E|^k)$ different mappings from τ^i. To

select from these mappings the one which can execute the most gates in LC^i, we need time $O(|E|^k \cdot m)$ (cf. the second observation above). Because each step removes at least one CNOT from LC, in at most $O(|E|^k \cdot m^2)$ time, we can execute all gates in LC. We note that this is an overestimated upper bound. If any Q-filter is used, there are at most $n^k \cdot deg^k$ actions with up to k SWAPs, where deg is the maximum degree of $\mathcal{AG}$. For IBM Q Tokyo, the maximum degree is 6, and for most grid-like AGs, the maximum degree is 4.

Now, suppose fallback is activated. Since each activation of the fallback reduces by (at least) one the minimum distance between qubits in a CNOT gate in the current front layer (i.e., $FB(\tau)$ in Eq. 5), the whole search process activates the fallback procedure at most $m \times diam$ times. Note that each activation (see Eq. 5) needs to compute the shortest distance between the control and target qubits in a CNOT gate in the front layer of the current logical circuit and there are at most $|Q|/2$ CNOT gates in the front layer. Using Dijkstra's algorithm with lists, $FB(\tau)$ can be computed in time $O(|Q| \cdot |V|^2)$.[3] Thus the total fallback on-cost is at most $O(|Q| \cdot |V|^2 \cdot m \cdot diam)$.

Therefore, the overall time complexity of the search process is $O(|E|^k \cdot m^2 + |Q| \cdot |V|^2 \cdot m \cdot diam)$. As $|Q| \leq |V| \leq |E|+1$ and $diam$ is usually very small when compared with m, the overall time complexity is bounded by $O(|E|^k \cdot m^2)$ if $k \geq 3$. In practice, this could be significantly reduced if we use Q-filters as the base in $|E|^k$ can be significantly reduced.

As for the space complexity, in each state s, we maintain, besides the logical and physical circuits, the dependency graph of the current logical circuit and the set of plausible actions with up to k SWAPs. Thus the space complexity of the algorithm is bounded by $O(|E|^k + m)$.

4 Evaluation

In this section, we compare our approach with the SABRE algorithm of Li et al. [15], the Cambridge algorithm of [17], and our qubit transformation algorithm SAHS based on simulated annealing and heuristic search [18], which are the state-of-the-art algorithms for the qubit mapping problem on IBM Q Tokyo (see Fig. 1). Although we focus on a particular NISQ device in the evaluation, our approach is applicable to any undirected architecture graph, including for example Rigetti 16Q Aspen-4[4]. We use Python as our programming language and IBM Qiskit [23] as auxiliary environment.[5] All experiments are conducted in a MacBook Pro with 3.1 GHz Intel Core i5 processor and 8GB memory.

As for benchmark circuits, we consider all publicly available circuits evaluated in [15] or [17]. Note that only CNOT gates are concerned in our comparison. For each individual circuit, we extract all its CNOT gates and use the thus reduced circuit as the input of our qubit mapping algorithm. We then compare the involved algorithms in terms of the number of auxiliary CNOT gates required. One may also use the following relative measure R_{CNOT}, first introduced in [17], to compare different algorithms on a particular circuit:

$$R_{\text{CNOT}} = \frac{\#\text{CNOT in the output physical circuit}}{\#\text{CNOT in the input logical circuit}}.$$

In order to compare algorithms evaluated over different benchmark sets of circuits, for any benchmark set $\mathcal{B}$ of quantum circuits, we define the following CNOT index, written $I^{\mathcal{B}}_{\text{CNOT}}$, of an algorithm relative to $\mathcal{B}$ as the fraction of the total number of CNOT gates in all output physical circuits over the total number of CNOT gates in all input logical circuits from $\mathcal{B}$, i.e.,

$$I^{\mathcal{B}}_{\text{CNOT}} = \frac{\sum_{LC \in \mathcal{B}} \#\text{CNOT in the output physical circuit of } LC}{\sum_{LC \in \mathcal{B}} \#\text{CNOT in } LC}.$$

For convenience, we write $\mathcal{B}_s$ and $\mathcal{B}_c$ for the benchmark sets of circuits used in [15] and [17] respectively. Note that $\mathcal{B}_c$ contains 131 circuits and includes all the 23 circuits in $\mathcal{B}_s$. For more precise comparison, we decompose $\mathcal{B}_c$ into three categories according to the number of CNOTs these benchmark circuits contain: small (0-99), medium (100-999) and large (≥ 1000).

For all experiments reported here, we fix the search depth k as 3 and use Q_0-filter for the first SWAP and Q_1-filter for all the other SWAPs for filtering actions $a = (e_1, ..., e_\ell)$ with at most 3 SWAPs. In addition, we also adopt the D_0-filter to exclude more less significant actions.

4.1 Comparison Among Different Initial Mappings

We first compare the two subgraph isomorphism related initial mappings (viz., the topgraph initial mapping τ_{top} and the weighted graph initial mapping τ_{wgt}) introduced in Section 3 with the empty mapping and the naive initial mapping (which maps q_i to v_i for each q_i in Q). Table 3 summarises the results for all (small, medium, large) circuits in $\mathcal{B}_c$, while we give the detailed results in Tables 7-9 in Section A. For each of these circuits and each initial mapping, the transformation can be completed within about 500 seconds by using our algorithm.

From Table 3, we can see that the two isomorphism subgraph related initial mappings, τ_{top} and τ_{wgt} are significantly better than the empty initial mapping and the naive initial mapping for small and medium circuits, but the difference is not significant when large circuits are evaluated. This is not a surprise as the search heuristics plays a dominant role if the circuit has a large size. Note that if a logical circuit can be transformed into a physical circuit with zero overhead, our algorithm, when using either the topgraph or weighted graph initial mapping, will very likely detect this.

Since the topgraph initial mapping is slightly better than the other three initial mappings, in the following, when compared with other algorithms, we always use the topgraph initial mapping.

4.2 Comparison with SABRE, Cambridge, and SAHS

We then compare our algorithm with SABRE [15] on the small benchmark set $\mathcal{B}_s$ of circuits used in [15]. We use the topgraph initial mapping τ_{top}. The results are reported in Table 2, in which the 'Comparison' column shows the improvement of our algorithm over SABRE in terms of the

3. In practice, we precompute the distance between every two nodes of $\mathcal{AG}$ and store it in a table.

4. Note that this device supports CZ instead of CNOT.

5. Code available at https://github.com/BensonZhou1991/Qubit-Mapping-Subgraph-Isomorphism

benchmarks	#circ.	topgr.i.m.	wghtgr.i.m.	empty i.m.	naive i.m.	SAHS	Cambridge
small	63	1.2521	1.3175	1.5041	1.8760	1.2619	1.5103
medium	39	1.2886	1.3034	1.4074	1.5727	1.3101	1.6854
large	29	1.4280	1.4366	1.4504	1.4422	1.6151	1.8211
all	131	1.4231	1.4324	1.4497	1.4486	1.6047	1.8154

TABLE 1
Summary of the $I^{\mathcal{B}}_{\text{CNOT}}$-index of our algorithm with four different initial mapping constructing methods and SAHS and Cambridge

numbers of auxiliary CNOT gates added. Specifically, let n_{sabre} and n_{ours} be the numbers of CNOT gates added by SABRE and by ours, respectively. Then the improvement ratio is calculated as $(n_{sabre} - n_{ours})/n_{sabre}$. From Table 2 we can see that only two circuits have negative improvement against SABRE. For all circuits with more than 300 CNOT gates, the improvement is at least 48%. In terms of the CNOT index, we have successfully decreased the index $I^{\mathcal{B}_s}_{\text{CNOT}}$ from $1 + 50874/50534 = 2.0067$ to $1 + 20790/50534 = 1.4114$.

We further compare our algorithm with the Cambridge algorithm of [17] and our SAHS algorithm [18] on the large benchmark set $\mathcal{B}_c$, which contains 131 circuits. A summary of the results in terms of the CNOT index is presented in Table 3. As for detailed results, see Tables 7-9 in Section A. It is worth stressing that the results of the Cambridge algorithm is obtained without using postmapping optimisations. Precisely, we have removed the following codes from their algorithm:

- 'Transform.OptimisePhaseGadgets().apply(tkcirc)' and
- 'Transform.OptimisePostRouting().apply(outcirc)'.

Thus the results of their algorithm given in Tables 7-9 are different from those reported in [17, Table VI]. In this sense, Tables 7-9 provide a fair comparison as we do not do postmapping optimisations either.

From Table 3 we can see that, for all 131 circuits in $\mathcal{B}_c$, the $I^{\mathcal{B}_c}_{\text{CNOT}}$ index of our algorithm is $1+141252/333811 = 1.4231$, while the indices for SAHS and Cambridge are, respectively, 1.6047 and 1.8154. This shows that our algorithm can in average generate significantly better results than Cambridge and SAHS. This is particularly true for large circuits which contain 1000 or more CNOT gates. For small circuits with less than 100 CNOT gates, our algorithm is slightly better than SAHS, 2 points (1.2886 vs. 1.3101) better than SAHS when medium circuits with 100-999 CNOT gates are considered. Both algorithms are significantly better than Cambridge in all three categories.

In the above experiments, we used Q_0-filter for the first SWAP and Q_1-filter (see Section 3.4) for all the other SWAPs when filtering actions with up to 3 SWAPs. If we use Q_0-filter for all SWAPs, then the index becomes 1.4774, which is about 5 points inferior to the index 1.4231 reported in Table 3. In addition, if we weaken Q_1-filter by using the qubits in the front layer and the lookahead layer, then the index could be further improved to **1.3801** from 1.4231. However, this is achieved at the cost of slower search process: for six large circuits (e.g., 'mlp4_245' with 16 qubits and 8232 CNOT gates), the compete process requires 900-1600 seconds.

4.3 Detailed Empirical Results

In the following, we present the detailed empirical results in Tables 7-9.

5 Further Discussion

From the above evaluation, we can see that our algorithm has significant better performance on IBM Q Tokyo than state-of-the-art algorithms. In this section, we give a more detailed discussion on the effectiveness, extensibility and time efficiency of our algorithm. In particular, we report more experiments on three larger architecture graphs.

5.1 Extension

Our filtered depth-limited approach actually can adopt heuristic value functions other than Eq. 1. Indeed, we have implemented another value function that based on the function specified in Eq. 4. The new heuristic value function is obtained by replacing "number of gates executable by τ'" in Eq. 1 with $\hat{R}_s(\tau^i) - \hat{R}_s(\tau')$, i.e., we have

$$\widehat{\mathsf{val}}_s(\tau^i, a) = \frac{\hat{R}_s(\tau^i) - \hat{R}_s(\tau')}{\mathsf{len}(a) \times 3}, \tag{6}$$

where τ' is the mapping obtained by enforcing SWAPs in action a one by one on τ^i and $\mathsf{len}(a) = \ell$ is the number of SWAPs in a. Using this value function, we then select any action with the maximal value as our next action. The algorithm adopting this value function has the same computational complexity as the one discussed before while its performance is similar to that of the Cambridge algorithm on IBM Q Tokyo and the benchmark set $\mathcal{B}_c$.

5.2 Effectiveness

While it works very good in average and especially on large circuits, our algorithm performs bad on several small or medium circuits. For example, the circuit 'alu-v0_27' contains 5 qubits and 17 CNOT gates. To execute it on IBM Q Tokyo, our QCT algorithm needs to insert 3 (2, resp.) swaps if the topograph (wgtgraph, resp.) initial mapping is used, while both Cambridge and SABRE only require 1 swap.

This bad performance is perhaps due to three reasons. First, during the search process, our algorithm works in a greedy way and always try to find the best action. This, however, often leads to a series of local optimal transformations, which gives no guarantee to the optimality of the global transformation. Second, there are many different mappings which can embed a given graph into $\mathcal{AG}$. Our algorithm selects an arbitrary one. Selecting a good embedding may further improve the performance of our algorithm. Third, our selection of the initial mapping is also greedy (cf. Sec. 3.2). For example, when constructing the topgraph initial mapping, we select a maximal subcircuit C_{top} of C whose corresponding graph $\mathcal{G}_{\mathsf{circ}}(C_{\mathsf{top}})$ is isomorphic to a subgraph of $\mathcal{AG}$. The following example shows that, however, this is not always a good choice.

Circuit Name	qubit no.	input gate	input CNOT	SABRE added	topgraph added	topgraph time (s)	Comp.
4mod5-v1_22	5	21	11	0	0	0.01	1
mod5mils_65	5	35	16	0	0	0.01	1
alu-v0_27	5	36	17	3	9	0.07	3
decod24-v2 43	4	52	22	0	0	0.01	1
4gt13_92	5	66	30	0	0	0.01	1
ising_model_10	10	480	90	0	0	0.01	1
ising_model_13	13	633	120	0	0	0.01	1
ising_model_16	16	786	150	0	0	0.01	1
qft_10	10	200	90	54	39	0.01	0.72
qft_16	16	512	240	186	153	124	0.82
rd84_142	15	343	154	105	72	8.72	0.69
adr4_197	13	3439	1498	1614	630	37.4	0.39
radd_250	13	3213	1405	1275	555	52.8	0.44
z4_268	11	3073	1343	1365	630	27.5	0.46
sym6_145	7	3888	1701	1272	513	4.8	0.40
misex1_241	15	4813	2100	1521	786	40.9	0.52
rd73_252	10	5321	2319	2133	1095	23.6	0.51
cycle10_2_110	12	6050	2648	2622	1194	23.7	0.46
square_root_7	15	7630	3089	2598	1338	446.9	0.52
sqn_258	10	10223	4459	4344	1578	39.9	0.36
rd84_253	12	13658	5960	6147	2352	111.8	0.38
co14_215	15	17936	7840	8982	4257	194.6	0.47
sym9_193	11	34881	15232	16653	5589	161.1	0.34
sum	-	117289	50534	50874	20790	-	**0.41**

TABLE 2
Comparison of our algorithm with SABRE in [15] on IBM Q Tokyo. The numbers in the last column indicate the ratio of our added CNOT gates (with the topgraph initial mapping) against that of SABRE.

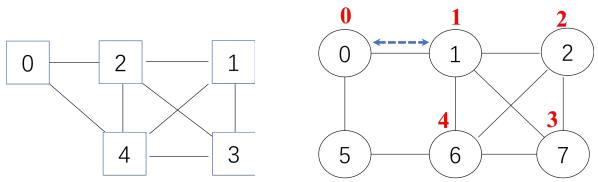

Fig. 5. The circuit graph of 'alu-v0_27' (left) and a mapping (right).

Example 7. *Consider the circuit 'alu-v0_27', which contains 5 qubits and the 17* CNOT *gates in*

$$C = \Big(\langle 3,4\rangle, \langle 2,1\rangle, \langle 1,3\rangle, \langle 2,1\rangle, \langle 3,2\rangle, \langle 3,1\rangle, \langle 2,1\rangle, \langle 3,2\rangle, \langle 1,3\rangle, \langle 2,0\rangle, \langle 0,4\rangle, \langle 2,0\rangle, \langle 4,2\rangle, \langle 4,0\rangle, \langle 2,0\rangle, \langle 4,2\rangle, \langle 0,4\rangle \Big).$$

The circuit graph (Fig. 5 (left)) is not embeddable in $\mathcal{AG}$, the architecture graph of IBM Q Tokyo. Let C_{top} be the sub-circuit of the first 10 gates in C. Then the circuit graph of C_{top} is the topgraph of C and can be embedded in $\mathcal{AG}$. Let τ' be such an embedding. After removing gates in C_{top}, the rest gates (involving only qubit 0, 2, and 4) cannot be solved with one swap from τ'. However, letting τ be the mapping as showing in the right of Fig. 5, we can see that τ solves only the first 9 gates and, after swapping 0 and 1, the rest 7 gates can also be solved.

5.3 Scalability and More on Time Complexity

In Sec. 3.5 we have seen that the search process is polynomial in all relevant parameters. As we go deeper in the search tree, our algorithm is considerably slower than SABRE and Cambridge. The significant decrease of the CNOT index shows that this is, however, worthwhile.

In contrast with the search process, the initial mapping construction process relies on the efficiency of subgraph isomorphism algorithms, which have time complexity exponential in the number of qubits in the circuit. We argue that this is not a serious problem for the following reasons. First, for the QCT problem in the NISQ era, only graphs with up to 1000 nodes are involved. Second, the architecture graphs usually have very simple and regular (e.g., grid-like) topologies, which can be exploited to design customised efficient subgraph isomorphism algorithms. Last but not least, good approximate solutions can also do the job well. This is partially evidenced by the results summarised in Table 3, where it shows that, for large circuits, the transformation results with an empty mapping are only slightly inferior to that using the selected topgraph initial mappings.

To further evaluate the effectiveness and efficiency of our approach, we have added experiments on IBM Q Rochester (53 qubits), Google's Sycamore (53 qubits), and an artificial 19×19 grid architecture graph, called Q19×19, which has 361 nodes. As circuits in the benchmark set $\mathcal{B}_c$ have only up to 16 qubits, we also tested a benchmark of circuits with large number of qubits. The benchmark was used in xx for xx, containing 19 circuits with 20-50 qubits and up to 15000 CNOT gates. The running time of our algorithm on these larger AGs does not get worse. See Table xx.

benchmarks	#circ.	topgr.i.m.	wghtgr.i.m.	empty i.m.	naive i.m.	SAHS	Cambridge
smallx	63	1.2521	1.3175	1.5041	1.8760	1.2619	1.5103
medium	39	1.2886	1.3034	1.4074	1.5727	1.3101	1.6854
large	29	1.4280	1.4366	1.4504	1.4422	1.6151	1.8211
all	131	1.4231	1.4324	1.4497	1.4486	1.6047	1.8154

TABLE 3
Summary of the $I^{\mathcal{B}}_{\mathsf{CNOT}}$-index of our algorithm with four different initial mapping constructing methods and SAHS and Cambridge

Circuit name	qubit no.	input CNOT	topgr. added	wgtgr. added	empty added	naive added	SAHS added	Cambridge added
ex1_226	6	5	0	0	3	18	0	0
graycode6_47	6	5	0	0	0	9	0	0
xor5_254	6	5	0	0	3	18	0	0
4gt11_84	4	9	0	0	9	12	0	0
ex-1_166	3	9	0	0	6	6	0	0
4mod5-v0_20	5	10	0	0	9	9	0	9
4mod5-v1_22	5	11	0	0	9	12	0	9
ham3_102	3	11	0	0	9	6	0	0
mod5d1_63	5	13	0	0	3	12	0	0
4gt11_83	5	14	0	0	0	18	0	12
4mod5-v0_19	5	16	0	0	0	18	0	9
4mod5-v1_24	5	16	0	0	15	21	0	12
mod5mils_65	5	16	0	0	15	21	0	9
rd32-v0_66	4	16	0	0	18	18	0	0
rd32-v1_68	4	16	0	0	18	18	0	0
3_17_13	3	17	0	0	9	9	0	0
alu-v0_27	5	17	9	6	12	18	6	3
alu-v1_29	5	17	9	6	12	24	6	3
alu-v2_33	5	17	9	6	12	18	6	9
4gt11_82	5	18	3	3	3	27	3	12
alu-v1_28	5	18	9	6	21	18	6	3
alu-v3_35	5	18	9	6	12	24	6	3
alu-v4_37	5	18	9	6	12	18	6	3
decod24-v2_43	4	22	0	0	18	15	0	0
decod24-v0_38	4	23	0	0	27	15	0	0
miller_11	3	23	0	0	6	9	0	0
alu-v3_34	5	24	9	6	27	27	6	3
mod5d2_64	5	25	18	9	18	36	12	12
4gt13_92	5	30	0	0	30	42	0	18
4gt13-v1_93	5	30	0	0	33	27	0	18
4mod5-bdd_287	7	31	6	18	9	36	6	15
4mod5-v0_18	5	31	9	12	30	36	9	9
4mod5-v1_23	5	32	9	21	30	36	9	12
decod24-bdd_294	6	32	15	27	24	24	15	21
one-two-three-v2_100	5	32	9	9	18	30	9	9
one-two-three-v3_101	5	32	15	18	24	36	6	15
rd32_270	5	36	18	24	39	30	12	18
4gt5_75	5	38	9	12	24	51	15	15
alu-bdd_288	7	38	24	15	39	36	24	45
alu-v0_26	5	38	12	9	30	36	9	21
decod24-v1_41	5	38	3	21	21	45	15	18
4gt5_76	5	46	21	36	48	48	15	27
4gt13_91	5	49	6	6	15	45	15	6
alu-v4_36	5	51	6	15	30	30	15	36
4gt13_90	5	53	9	9	18	48	27	9
4gt5_77	5	58	9	18	18	45	9	36
one-two-three-v1_99	5	59	24	33	42	48	12	39
rd53_138	8	60	30	42	30	42	27	39
decod24-v3_45	5	64	15	24	36	72	15	39
one-two-three-v0_98	5	65	18	27	48	63	24	27
4gt10-v1_81	5	66	15	18	36	60	27	33
aj-e11_165	5	69	33	36	42	33	18	24
4mod7-v0_94	5	72	12	27	33	51	12	39
4mod7-v1_96	5	72	21	21	48	45	18	42
alu-v2_32	5	72	15	45	69	45	15	39
mod10_176	5	78	15	24	63	66	24	36
4gt4-v0_80	6	79	15	39	48	69	24	78
cnt3-5_179	16	85	3	30	72	87	15	87
4gt12-v0_88	6	86	21	15	69	66	21	21
ising_model_10	10	90	0	0	18	27	0	0
qft_10	10	90	45	33	57	96	36	57
sys6-v0_111	10	98	54	81	63	60	45	111
4_49_16	5	99	18	30	42	48	36	69
sum		2428	618	849	1602	2133	636	1239
I-index			1.255	1.350	1.660	1.879	1.262	1.510

Circuit name	qubit no.	input CNOT	topgr. added	wgtgr. added	empty added	naive added	SAHS added	Cambridge added

TABLE 4: Comparison on IBM Q Tokyo with small circuits

Circuit name	qubit no.	input CNOT	topgr. added	wgtgr. added	empty added	naive added	SAHS added	Cambridge added
4gt12-v1_89	6	100	57	18	30	81	24	93
0410184_169	14	104	6	27	75	93	12	75
4gt4-v0_79	6	105	12	12	30	87	12	96
hwb4_49	5	107	36	42	54	63	33	45
mod10_171	5	108	39	27	27	60	24	60
4gt4-v0_78	6	109	15	15	33	93	15	99
4gt12-v0_87	6	112	6	6	24	69	6	123
4gt4-v0_72	6	113	45	39	51	93	42	90
4gt12-v0_86	6	116	9	9	27	75	9	123
4gt4-v1_74	6	119	39	27	75	93	78	114
ising_model_13	13	120	0	0	36	45	0	0
mini-alu_167	5	126	30	33	69	87	33	75
one-two-three-v0_97	5	128	42	78	72	90	66	66
rd53_135	7	134	60	84	99	111	54	48
decod24-enable_126	6	149	66	63	72	84	87	81
ham7_104	7	149	48	42	75	51	81	102
ising_model_16	16	150	0	0	48	48	0	0
mod8-10_178	6	152	69	33	69	87	21	162
rd84_142	15	154	84	126	87	108	102	198
ex3_229	6	175	24	81	87	102	18	174
4gt4-v0_73	6	179	99	42	117	120	42	177
mod8-10_177	6	196	123	78	72	87	39	135
alu-v2_31	5	198	78	90	60	99	54	63
rd53_131	7	200	63	78	93	81	90	87
C17_204	7	205	111	84	144	147	96	114
alu-v2_30	6	223	60	54	87	93	45	105
mod5adder_127	6	239	84	81	93	114	51	87
qft_16	16	240	189	135	204	231	135	195
rd53_133	7	256	60	174	138	150	105	159
majority_239	7	267	66	105	87	213	84	123
ex2_227	7	275	78	108	90	126	96	270
cm82a_208	8	283	117	105	102	225	84	222
sf_274	6	336	30	36	63	180	24	381
sf_276	6	336	36	36	102	159	24	384
con1_216	9	415	273	153	210	195	192	375
rd53_130	7	448	267	207	168	222	171	390
f2_232	8	525	336	126	192	312	213	225
rd53_251	8	564	201	195	225	240	204	309
hwb5_53	6	598	207	204	195	237	174	210
sum		8513	3165	2853	3582	4851	2640	5835
I-index			1.372	1.335	1.421	1.570	1.310	1.685

TABLE 5: Comparison on IBM Q Tokyo with medium circuits

Circuit name	qubit no.	input CNOT	topgr. added	wgtgr. added	empty added	naive added	SAHS added	Cambridge added
z4_268	11	1343	525	468	609	600	546	1671
radd_250	13	1405	633	567	555	549	669	1647
adr4_197	13	1498	681	741	642	807	711	1146
sym6_145	7	1701	540	585	540	765	744	2139
misex1_241	15	2100	621	726	858	924	921	1263
rd73_252	10	2319	1062	852	1227	1074	1065	2115
cycle10_2_110	12	2648	1125	1290	1320	1236	1038	2424
hwb6_56	7	2952	1077	1026	1098	1011	1104	1719
square_root_7	15	3089	1263	1374	1242	1470	1353	1326
sqn_258	10	4459	1467	1716	1638	1986	1953	3192
cm85a_209	14	4986	2073	2091	2289	2397	2337	4173
rd84_253	12	5960	2952	2841	3174	3009	3198	5286
root_255	13	7493	2928	3099	3468	3399	3525	5601
co14_215	15	7840	3975	4563	4629	4437	4356	7752
mlp4_245	16	8232	4275	4146	4173	4104	4116	6462
sym9_148	10	9408	1947	2166	1992	2388	2172	6438
urf2_277	8	10066	6285	6267	6135	6045	5934	8205
hwb7_59	8	10681	3684	3846	3696	3588	4602	6378
max46_240	10	11844	4410	4308	4560	4530	5289	9681
clip_206	14	14772	6762	6834	6963	6405	6843	12624
9symml_195	11	15232	5481	6462	5373	5682	6036	11454
sym9_193	11	15232	5481	6462	5373	5682	6123	11454
dist_223	13	16624	7470	6582	7107	7254	6936	12834
sao2_257	14	16864	7596	6756	8361	6792	7827	11742
urf5_280	9	23764	11988	11679	12060	11802	13065	20436

Circuit name	qubit no.	input CNOT	topgr. added	wgtgr. added	empty added	naive added	SAHS added	Cambridge added
urf1_278	9	26692	13872	13809	14190	14022	15678	24600
sym10_262	12	28084	11490	10623	11520	10635	11697	20115
hwb8_113	9	30372	11295	11382	11769	11394	59977	35376
urf2_152	8	35210	18342	18342	18018	18489	18780	25857
sum		322870	141300	141603	144579	142476	198595	265110
I-index			1.438	1.439	1.448	1.441	1.615	1.821

TABLE 6: Comparison on IBM Q Tokyo with large circuits

6 Conclusion

We have proposed a new algorithm for qubit mapping which can significantly reduce the extra two-qubit gates required per two-qubit gate in the input circuit. Our algorithm is based on subgraph isomorphism and filtered depth-limited search. If the input circuit can be executed directly, the proposed approach can always detect this. It seems that this nice property is not enjoyed by any other approach.

From our experimental results, we can see that, when the circuit has less than 1000 two-qubit gates, our subgraph isomorphism induced initial mappings are much better than empty mappings and naive mappings that assign the i-th qubit in the logical circuit to the i-th qubit in the quantum device. For large circuits, our results show that initial mappings are not very important.

A weighted graph like ours (see Section 3.2) is also introduced in a recent work [24], where Lin, Anschuetz, and Harrow exploit spectral graph theory to qubit mapping. The performance of their algorithm is in general not better than the A^* approach of [14]. They also suggested to use their "spectral mapper to provide an initial mapping" while its effectiveness needs further investigation.

It seems that we are still quite far from devising algorithms that could output circuits with nearly minimal overheads. Future work will investigate along the following directions:

- Although our approach basically can be applied on any NISQ device with an undirected architecture graph, it is not certain if the actual efficacy highly depends on the size or 'compactness' of the architecture graph. Further work is required to evaluate our approach on more general NISQ devices with a large number of qubits and various topologies.
- Minimizing depth or latency and circuit error is also important for qubit mapping. Although the number of CNOT gates in our output circuit is already smaller than the depth of the output circuit (only CNOT gates are counted) of some compared algorithm [17] (see [18] for a more detailed analysis), it will be nice if we can adapt our approach to address other or multiple optimisation objectives.
- More on heuristic search and filters. Could we do better by designing new filters?
- In our approach, the initial mapping is obtained by constructing a subgraph isomorphism. We certainly could do this in each step, after those executable gates are removed from the logical circuit. But the clear obstacle is how to transform the current mapping to a selected next mapping and how to select a good subgraph isomorphism.
- Machine learning and deep learning algorithms may be designed to quickly select the best action in Eq. 2.

References

[1] P. W. Shor, "Polynominal time algorithms for discrete logarithms and factoring on a quantum computer," in *Algorithmic Number Theory, First International Symposium, ANTS-I, Ithaca, NY, USA, May 6-9, 1994, Proceedings*, ser. Lecture Notes in Computer Science, L. M. Adleman and M. A. Huang, Eds., vol. 877. Springer, 1994, p. 289. [Online]. Available: https://doi.org/10.1007/3-540-58691-1_68

[2] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Physical Review Letters*, vol. 103, no. 15, p. 150502, 2009.

[3] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, pp. 195 EP –, 09 2017. [Online]. Available: https://doi.org/10.1038/nature23474

[4] A. K. B. R. e. a. Arute, F., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, p. 505–510, 2019. [Online]. Available: https://doi.org/10.1038/nature23474

[5] S. Khatri, R. LaRose, A. Poremba, L. Cincio, A. T. Sornborger, and P. J. Coles, "Quantum-assisted quantum compiling," *Quantum*, vol. 3, p. 140, 2019.

[6] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 32, no. 6, pp. 818–830, 2013. [Online]. Available: https://doi.org/10.1109/TCAD.2013.2244643

[7] K. Matsumoto and K. Amano, "Representation of quantum circuits with clifford and $\pi/8$ gates," *arXiv:0806.3834*, 2008.

[8] R. Wille, M. Soeken, C. Otterstedt, and R. Drechsler, "Improving the mapping of reversible circuits to quantum circuits using multiple target lines," in *18th Asia and South Pacific Design Automation Conference, ASP-DAC 2013, Yokohama, Japan, January 22-25, 2013*. IEEE, 2013, pp. 145–150. [Online]. Available: https://doi.org/10.1109/ASPDAC.2013.6509587

[9] D. Maslov, S. M. Falconer, and M. Mosca, "Quantum circuit placement," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 27, no. 4, pp. 752–763, 2008. [Online]. Available: https://doi.org/10.1109/TCAD.2008.917562

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

[11] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. ACM, 2018, pp. 113–125.

[12] D. Venturelli, M. Do, E. Rieffel, and J. Frank, "Compiling quantum circuits to realistic hardware architectures using temporal planners," *Quantum Science and Technology*, vol. 3, no. 2, p. 025004, Feb 2018. [Online]. Available: https://doi.org/10.1088%2F2058-9565%2Faaa331

[13] 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 *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, I. Bahar, M. Herlihy, E. Witchel, and A. R. Lebeck, Eds. ACM, 2019, pp. 1015–1029. [Online]. Available: https://doi.org/10.1145/3297858.3304075

[14] A. Zulehner, A. Paler, and R. Wille, "Efficient mapping of quantum circuits to the IBM QX architectures," in *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*. IEEE, 2018, pp. 1135–1138. [Online]. Available: https://doi.org/10.23919/DATE.2018.8342181

[15] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 1001–1014.

[16] A. M. Childs, E. Schoute, and C. M. Unsal, "Circuit transformations for quantum architectures," in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2019, June 3-5, 2019, University of Maryland, College Park, Maryland, USA.*, ser. LIPIcs, W. van Dam and L. Mancinska, Eds., vol. 135. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, pp. 3:1–3:24. [Online]. Available: https://doi.org/10.4230/LIPIcs.TQC.2019.3

[17] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, "On the qubit routing problem," in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2019, June 3-5, 2019, University of Maryland, College Park, Maryland, USA.*, ser. LIPIcs, W. van Dam and L. Mancinska, Eds., vol. 135. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, pp. 5:1–5:32. [Online]. Available: https://doi.org/10.4230/LIPIcs.TQC.2019.5

[18] X. Zhou, S. Li, and Y. Feng, "Quantum circuit transformation based on simulated annealing and heuristic search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020, doi: 10.1109/TCAD.2020.2969647, arXiv:1908.08853.
[19] A. A. de Almeida, G. W. Dueck, and A. C. da Silva, "Finding optimal qubit permutations for IBM's quantum computer architectures," in *Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design*. ACM, 2019, p. 13.
[20] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Phys. Rev. A*, vol. 52, pp. 3457–3467, Nov 1995. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.52.3457
[21] R. Van Meter, *Quantum Networking*. John Wiley & Sons, 2014.
[22] 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, 2004. [Online]. Available: https://doi.org/10.1109/TPAMI.2004.75
[23] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C. Chen *et al.*, "Qiskit: An open-source framework for quantum computing," *Accessed on: March 16, 2019*, 2019.
[24] J. X. Lin, E. R. Anschuetz, and A. W. Harrow, "Using spectral graph theory to map qubits onto connectivity-limited devices," *arXiv:1910.11489*, 2019.
[25] W. van Dam and L. Mancinska, Eds., *14th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2019, June 3-5, 2019, University of Maryland, College Park, Maryland, USA*, ser. LIPIcs, vol. 135. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. [Online]. Available: http://www.dagstuhl.de/dagpub/978-3-95977-112-2

APPENDIX
DETAILED EMPIRICAL RESULTS

Circuit name	qubit no.	input CNOT	topgr. added	wgtgr. added	empty added	naive added	SAHS added	Cambridge added
ex1_226	6	5	0	0	3	18	0	0
graycode6_47	6	5	0	0	0	9	0	0
xor5_254	6	5	0	0	3	18	0	0
4gt11_84	4	9	0	0	9	12	0	0
ex-1_166	3	9	0	0	6	6	0	0
4mod5-v0_20	5	10	0	0	9	9	0	9
4mod5-v1_22	5	11	0	0	9	12	0	9
ham3_102	3	11	0	0	9	6	0	0
mod5d1_63	5	13	0	0	3	12	0	0
4gt11_83	5	14	0	0	0	18	0	12
4mod5-v0_19	5	16	0	0	0	18	0	9
4mod5-v1_24	5	16	0	0	15	21	0	12
mod5mils_65	5	16	0	0	15	21	0	9
rd32-v0_66	4	16	0	0	18	18	0	0
rd32-v1_68	4	16	0	0	18	18	0	0
3_17_13	3	17	0	0	9	9	0	0
alu-v0_27	5	17	9	6	12	18	6	3
alu-v1_29	5	17	9	6	12	24	6	3
alu-v2_33	5	17	9	6	12	18	6	9
4gt11_82	5	18	3	3	3	27	3	12
alu-v1_28	5	18	9	6	21	18	6	3
alu-v3_35	5	18	9	6	12	24	6	3
alu-v4_37	5	18	9	6	12	18	6	3
decod24-v2_43	4	22	0	0	18	15	0	0
decod24-v0_38	4	23	0	0	27	15	0	0
miller_11	3	23	0	0	6	9	0	0
alu-v3_34	5	24	9	6	27	27	6	3
mod5d2_64	5	25	18	9	18	36	12	12
4gt13_92	5	30	0	0	30	42	0	18
4gt13-v1_93	5	30	0	0	33	27	0	18
4mod5-bdd_287	7	31	6	18	9	36	6	15
4mod5-v0_18	5	31	9	12	30	36	9	9
4mod5-v1_23	5	32	9	21	30	36	9	12
decod24-bdd_294	6	32	15	27	24	24	15	21
one-two-three-v2_100	5	32	9	9	18	30	9	9
one-two-three-v3_101	5	32	15	18	24	36	6	15
rd32_270	5	36	18	24	39	30	12	18
4gt5_75	5	38	9	12	24	51	15	15
alu-bdd_288	7	38	24	15	39	36	24	45
alu-v0_26	5	38	12	9	30	36	9	21
decod24-v1_41	5	38	3	21	21	45	15	18
4gt5_76	5	46	21	36	48	48	15	27
4gt13_91	5	49	6	6	15	45	15	6
alu-v4_36	5	51	6	15	30	30	15	36
4gt13_90	5	53	9	9	18	48	27	9
4gt5_77	5	58	9	18	18	45	9	36
one-two-three-v1_99	5	59	24	33	42	48	12	39
rd53_138	8	60	30	42	30	42	27	39
decod24-v3_45	5	64	15	24	36	72	15	39
one-two-three-v0_98	5	65	18	27	48	63	24	27
4gt10-v1_81	5	66	15	18	36	60	27	33
aj-e11_165	5	69	33	36	42	33	18	24
4mod7-v0_94	5	72	12	27	33	51	12	39
4mod7-v1_96	5	72	21	21	48	45	18	42
alu-v2_32	5	72	15	45	69	45	15	39
mod10_176	5	78	15	24	63	66	24	36
4gt4-v0_80	6	79	15	39	48	69	24	78
cnt3-5_179	16	85	3	30	72	87	15	87
4gt12-v0_88	6	86	21	15	69	66	21	21
ising_model_10	10	90	0	0	18	27	0	0
qft_10	10	90	45	33	57	96	36	57
sys6-v0_111	10	98	54	81	63	60	45	111
4_49_16	5	99	18	30	42	48	36	69
sum		2428	618	849	1602	2133	636	1239
I-index			1.255	1.350	1.660	1.879	1.262	1.510

TABLE 7: Comparison on IBM Q Tokyo with small circuits

Circuit name	qubit no.	input CNOT	topgr. added	wgtgr. added	empty added	naive added	SAHS added	Cambridge added
4gt12-v1_89	6	100	57	18	30	81	24	93
0410184_169	14	104	6	27	75	93	12	75
4gt4-v0_79	6	105	12	12	30	87	12	96
hwb4_49	5	107	36	42	54	63	33	45
mod10_171	5	108	39	27	27	60	24	60
4gt4-v0_78	6	109	15	15	33	93	15	99

Circuit name	qubit no.	input CNOT	topgr. added	wgtgr. added	empty added	naive added	SAHS added	Cambridge added
4gt12-v0_87	6	112	6	6	24	69	6	123
4gt4-v0_72	6	113	45	39	51	93	42	90
4gt12-v0_86	6	116	9	9	27	75	9	123
4gt4-v1_74	6	119	39	27	75	93	78	114
ising_model_13	13	120	0	0	36	45	0	0
mini-alu_167	5	126	30	33	69	87	33	75
one-two-three-v0_97	5	128	42	78	72	90	66	66
rd53_135	7	134	60	84	99	111	54	48
decod24-enable_126	6	149	66	63	72	84	87	81
ham7_104	7	149	48	42	75	51	81	102
ising_model_16	16	150	0	0	48	48	0	0
mod8-10_178	6	152	69	33	69	87	21	162
rd84_142	15	154	84	126	87	108	102	198
ex3_229	6	175	24	81	87	102	18	174
4gt4-v0_73	6	179	99	42	117	120	42	177
mod8-10_177	6	196	123	78	72	87	39	135
alu-v2_31	5	198	78	90	60	99	54	63
rd53_131	7	200	63	78	93	81	90	87
C17_204	7	205	111	84	144	147	96	114
alu-v2_30	6	223	60	54	87	93	45	105
mod5adder_127	6	239	84	81	93	114	51	87
qft_16	16	240	189	135	204	231	135	195
rd53_133	7	256	60	174	138	150	105	159
majority_239	7	267	66	105	87	213	84	123
ex2_227	7	275	78	108	90	126	96	270
cm82a_208	8	283	117	105	102	225	84	222
sf_274	6	336	30	36	63	180	24	381
sf_276	6	336	36	36	102	159	24	384
con1_216	9	415	273	153	210	195	192	375
rd53_130	7	448	267	207	168	222	171	390
f2_232	8	525	336	126	192	312	213	225
rd53_251	8	564	201	195	225	240	204	309
hwb5_53	6	598	207	204	195	237	174	210
sum		8513	3165	2853	3582	4851	2640	5835
I-index			1.372	1.335	1.421	1.570	1.310	1.685

TABLE 8: Comparison on IBM Q Tokyo with medium circuits

Circuit name	qubit no.	input CNOT	topgr. added	wgtgr. added	empty added	naive added	SAHS added	Cambridge added
z4_268	11	1343	525	468	609	600	546	1671
radd_250	13	1405	633	567	555	549	669	1647
adr4_197	13	1498	681	741	642	807	711	1146
sym6_145	7	1701	540	585	540	765	744	2139
misex1_241	15	2100	621	726	858	924	921	1263
rd73_252	10	2319	1062	852	1227	1074	1065	2115
cycle10_2_110	12	2648	1125	1290	1320	1236	1038	2424
hwb6_56	7	2952	1077	1026	1098	1011	1104	1719
square_root_7	15	3089	1263	1374	1242	1470	1353	1326
sqn_258	10	4459	1467	1716	1638	1986	1953	3192
cm85a_209	14	4986	2073	2091	2289	2397	2337	4173
rd84_253	12	5960	2952	2841	3174	3009	3198	5286
root_255	13	7493	2928	3099	3468	3399	3525	5601
co14_215	15	7840	3975	4563	4629	4437	4356	7752
mlp4_245	16	8232	4275	4146	4173	4104	4116	6462
sym9_148	10	9408	1947	2166	1992	2388	2172	6438
urf2_277	8	10066	6285	6267	6135	6045	5934	8205
hwb7_59	8	10681	3684	3846	3696	3588	4602	6378
max46_240	10	11844	4410	4308	4560	4530	5289	9681
clip_206	14	14772	6762	6834	6963	6405	6843	12624
9symml_195	11	15232	5481	6462	5373	5682	6036	11454
sym9_193	11	15232	5481	6462	5373	5682	6123	11454
dist_223	13	16624	7470	6582	7107	7254	6936	12834
sao2_257	14	16864	7596	6756	8361	6792	7827	11742
urf5_280	9	23764	11988	11679	12060	11802	13065	20436
urf1_278	9	26692	13872	13809	14190	14022	15678	24600
sym10_262	12	28084	11490	10623	11520	10635	11697	20115
hwb8_113	9	30372	11295	11382	11769	11394	59977	35376
urf2_152	8	35210	18342	18342	18018	18489	18780	25857
sum		322870	141300	141603	144579	142476	198595	265110
I-index			1.438	1.439	1.448	1.441	1.615	1.821

TABLE 9: Comparison on IBM Q Tokyo with large circuits