Single-Qubit Gates Matter for Optimising Quantum Circuit Depth in Qubit Mapping

Sanjiang Li
Centre for Quantum Software & Information, *University of Technology Sydney*, Sydney, Australia
sanjiang.li@uts.edu.au
Ky Dan Nguyen
School of Computer Science, *University of Sydney*, Sydney, Australia
kngu7458@uni.sydney.edu.au
Zachary Clare
School of Computer Science, *University of Technology Sydney*, Sydney, Australia
zachary.clare@student.uts.edu.au
Yuan Feng
Centre for Quantum Software & Information, *University of Technology Sydney*, Sydney, Australia
yuan.feng@uts.edu.au

***Abstract*—Quantum circuit transformation (QCT, a.k.a. qubit mapping) is a critical step in quantum circuit compilation. Typically, QCT is achieved by finding an appropriate initial mapping and using SWAP gates to route the qubits such that all connectivity constraints are satisfied. The objective of QCT can be to minimise circuit size or depth. Most existing QCT algorithms prioritise minimising circuit size, potentially overlooking the impact of single-qubit gates on circuit depth. In this paper, we first point out that a single SWAP gate insertion can double the circuit depth, and then propose a simple and effective method that takes into account the impact of single-qubit gates on circuit depth. Our method can be combined with many existing QCT algorithms to optimise circuit depth. The Qiskit SABRE algorithm has been widely accepted as the state-of-the-art algorithm for optimising both circuit size and depth. We demonstrate the effectiveness of our method by embedding it in SABRE, showing that it can reduce circuit depth by up to 50% and 27% on average on, for instance, Google Sycamore and 117 real quantum circuits from MQTBench.**

I. Introduction

Current Noisy Intermediate-Scale Quantum (NISQ) devices have strict connectivity constraints that limit the execution of 2-qubit gates (such as CX or CZ) to neighbouring qubits only. This requires a transformation of the ideal circuits before running them on real quantum devices, which is commonly known as Quantum Circuit Transformation (QCT), qubit mapping, or layout synthesis. In this paper, we use the terms *qubit mapping* and *(quantum) circuit transformation* interchangeably to refer to this procedure. QCT is an essential component of quantum circuit compilation and has gained widespread interest in areas such as quantum computing [1], [2], [3], [4], [5], electronic design automation [6], [7], [8], [9], [10], [11], [12], [13], and computer architecture [14], [15], [16], [17], [18].

Over the past few years, numerous QCT algorithms have been developed to transform ideal quantum circuits into circuits that can be executed on a specific quantum device with connectivity constraints. The input to these algorithms includes an architecture graph that specifies the connectivity constraints of the targeted quantum device, and an ideal quantum circuit that contains only single- and 2-qubit gates defined in the basic gate library of the device. QCT algorithms typically achieve this transformation by first applying an initial mapping, followed by repeatedly adding SWAP gates to schedule 2-qubit gates for execution on the target device. The objective of QCT can be to minimise circuit size or depth, both in turn can increase the overall fidelity of the transformed circuit. The transformation cost of such a transformation depends on the optimisation objective, which can be measured either by the number of SWAP gates inserted or the difference in circuit depth before and after transformation. Due to the NP-complete nature of the qubit mapping problem [19], [20], exact algorithms can only handle–within reasonable time–circuits with up to 10 qubits, and therefore, most QCT algorithms are heuristic or approximate in nature.

⋆Accepted to *The 2023 International Conference on Computer-Aided Design* (IEEE/ACM ICCAD'23)

The error rates of 2-qubit gates in present NISQ devices are often 10 times higher than those of single-qubit gates. Consequently, numerous QCT algorithms [13], [14], [21], [12], [11] prioritise minimising the number of 2-qubit gates as their primary objective, which is essentially determined by the total number of SWAP gates inserted. In many cases, QCT algorithms execute a gate as early as possible and single-qubit gates are often not considered, see, e.g., [14], [1], [2], [21]. In fact, FiDLS [21] and an initial implementation of SABRE [14] even completely remove single-qubit gates from the circuit before transformation.

In the past several years, we have seen the size of quantum computer increases from 5 to 433 qubits[1], but qubit coherence time in NISQ devices remains very short. This implies that we can only run a very limited number of quantum operations on each qubit; in other words, we cannot extract meaningful information from very deep quantum circuits on NISQ devices. Thus, minimising the depth of transformed circuits is perhaps a more important objective. There are several QCT algorithms targeting circuit depth, see, e.g., Qiskit's `StochasticSWAP` and [1], [22], which is often achieved by encouraging parallel SWAPs or minimising the depth of inserted SWAP circuits. Tan and Cong [23] and Zhang et al. [18] propose exact QCT algorithms for minimising circuit depth. Again, these algorithms can transform only small circuits. Both exact algorithms are also relaxed to obtain approximate algorithms, which could tackle circuits with more qubits than their exact version while still can obtain much better depth results than heuristic-based algorithms like SABRE. However, based on SMT or A^* search, the two approximate algorithms are still not scalable to circuits with 50 or more qubits (see experiments reported in Sec. IV-D).

First described in [14], SABRE was later implemented in Qiskit and is now its default transpiler. As a randomised algorithm SABRE can be used for both circuit size and depth optimisations: we need only run it multiple times and select the best size or depth result. In addition, SABRE introduces a 'decay' factor to discourage applying SWAP gates on a qubit which was recently swapped. Extensive evaluation on several quantum devices shows that SABRE significantly outperforms many state-of-the-art QCT algorithms in both circuit size and depth [24].

Despite this outstanding performance in depth optimisation, the impact of single-qubit gates is overlooked. This is because SABRE executes a gate whenever it is allowed to do so. In particular, it greedily executes every single-qubit gate if its predecessor has been executed. This sometimes results in unnecessarily much deeper transformed circuits (cf. the example in Fig.s 2 and 3).

In this paper, we first show by an example that single-qubit gates are also important and **a single SWAP gate insertion may double the circuit depth**, and then propose a method that takes into account the impact of single-qubit gates on circuit depth. The idea is to record the *transformation progress* of each physical qubit, hold single-qubit gates until we meet a new 2-qubit gate after them, and introduce a 'delay' component based on the qubit progress to discourage those SWAPs that have progressed too much on their qubits. Our method can be combined with many existing QCT algorithms for optimising circuit depth. We demonstrate the effectiveness of our method by embedding it in `SabreSWAP`—the Qiskit implementation of SABRE's routing process, showing that it can reduce circuit depth by up to 50% and 27% on average on, for instance, Google Sycamore (54Q) and 117 real quantum circuits from MQTBench [25]. Similar results are also observed on IBM Q Tokyo (20Q) and Rochester (53Q) and synthesised `QUEKNO` benchmarks for evaluating depth optimality [24], where the average improvements over SABRE are 15% and 17%, respectively.

It was found that the relaxed TOQM algorithm [18] performs very well on 20-qubit IBM Q Tokyo and could beat SABRE in terms of circuit depth by on average 20%. This paper will also compare our algorithm and TOQM on larger quantum devices. Results (reported in Sec. IV-D) show that the relaxed version of TOQM is not yet scalable to larger devices such as the 54Q Google Sycamore, while SABRE and our algorithm, called SQGM (for **S**ingle-**Q**ubit **G**ates **M**atter), can process within a few seconds.

Another recent work [15] combines qubit mapping with 2-qubit block re-synthesis and commutation-based gate cancellation. Their algorithm, called NASSC, enriches SABRE with the above optimisation techniques. In this paper, we also empirically compare NASSC with our SQGM. Results show that, on MQTBench circuits and 54Q Sycamore, SQGM outperforms NASSC by 12% in terms of circuit depth; and, if we apply a post-routing commutative gate cancellation (which NASSC has already included) to our algorithm, then SQGM outperforms NASSC by 17%.

The remainder of this paper is organised as follows.

[1]https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two

We recall relevant backgrounds about quantum circuits and quantum circuit transformation as well as SABRE in Sec. II, and present our method in Sec. III. We then evaluate and compare our method with SABRE, TOQM [18], NASSC [15] in Sec. IV. The last section concludes the paper.

II. Preliminaries

This section first recalls some relevant background in quantum computing and then introduces the preliminaries of quantum circuit transformation. In the end of this section, we recall the QCT algorithm SABRE.

A. Quantum Circuits

Quantum algorithms are commonly described using quantum circuits, which are analogous to classical combinational circuits. A quantum circuit comprises a sequence of quantum gates that act on qubits (quantum bits). Quantum gates are unitary transformations. An n-qubit gate is represented as a $2^n \times 2^n$ unitary matrix. Some well-known single-qubit gates include:

$$X=\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix},\quad H=\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix},\quad S=\begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix},\quad T=\begin{pmatrix}1 & 0\\ 0 & e^{i\frac{\pi}{4}}\end{pmatrix}.$$

CX (also called CNOT) and CZ are two most common 2-qubit gates. For any computational basis state $|i\rangle\,|j\rangle$, CX and CZ map $|i\rangle\,|j\rangle$ to, respectively, $|i\rangle\,|i\oplus j\rangle$ and $(-1)^{i\cdot j}\,|i\rangle\,|j\rangle$, where $\oplus$ denotes exclusive-or and $\cdot$ denotes logical conjunction.

Any quantum gate can be implemented using, i.e., *decomposed* into, single-qubit and CX gates. Furthermore, we can approximate any quantum gate to arbitrary accuracy using the H, S, T, and CX gates. The SWAP gate, which swaps the states of two qubits, can be implemented using three CX gates. That is, $\textsc{swap}(p,q)=\mathrm{CX}(p,q)\mathrm{CX}(q,p)\mathrm{CX}(p,q)$.

Although different quantum devices may have different universal sets of quantum gates, the 2-qubit gate in these sets is usually either CX or CZ. Since the functionality of a single-qubit gate is not (directly) relevant in quantum circuit transformation, a gate acting on qubit q_i is simply denoted as $\langle q_i\rangle$, while a CX or CZ gate with control qubit q_i and target qubit q_j is denoted as $\langle q_i, q_j\rangle$.

A circuit C is usually given as a sequence of gates $(g_0, g_1, \ldots, g_{m-1})$, but this does not mean that the $(i+s)$-th gate should be executed after the i-th gate for all $i \geq 0, s \geq 1$ with $0 < i+s < m$. In fact, two gates can be executed in parallel if they do not act on a common qubit. Naturally, we partition C into layers while putting each gate as front as possible (so that it can be executed at the earliest time). The number of layers is called the *depth* of the circuit. For example, the circuit in Fig. 1 can be represented as

$$C=\big(\langle q_0,q_1\rangle, \langle q_0\rangle, \langle q_1,q_3\rangle, \langle q_0,q_3\rangle, \langle q_1,q_2\rangle,\\ \langle q_0\rangle, \langle q_0\rangle, \langle q_0\rangle, \langle q_0\rangle, \langle q_1\rangle, \langle q_1\rangle, \langle q_1\rangle, \langle q_1\rangle\big).$$

The circuit has 7 layers and thus a depth of 7. Specifically, its third layer contains two gates, viz. $\langle q_0,q_3\rangle, \langle q_1,q_2\rangle$.

This paper represents a quantum device as an undirected graph $G = (V, E)$, called the *architecture graph* of the quantum device, where V denotes the set of physical qubits and E the set of permitted 2-qubit interactions. In other words, (v, v') is an edge in E iff a 2-qubit gate acting on qubits v, v' is executable on the device. As G is an undirected graph, $(v, v') \in E$ iff $(v', v) \in E$. Fig. 1 (right) shows the architecture graph of an artificial device.

B. Quantum Circuit Transformation

In the quantum circuit model, it is common for the algorithm designer to not have a targeted quantum device in mind when developing the algorithm. Let C be an ideal circuit representing a quantum algorithm and $G = (V, E)$ the architecture graph of the target quantum device. We assume gates in C have been decomposed into elementary gates supported by the physical device and our task is to transform C into a functionally equivalent circuit where every 2-qubit gate acts on two neighbouring nodes in G.

In the following, we refer to C as a *logical* circuit and call the transformed circuit a *physical* circuit. The qubits in the logical (physical) circuit are called logical (physical) qubits. It is worth noting that, in the context of QCT, the use of the term "logical" should not be confused with its usage in error correction. Device-supported single-qubit gates can be executed directly if its predecessor in the circuit has been executed. A 2-qubit gate is *directly executable* on G if its two qubits are neighbours in G and its predecessors in the circuit have been executed.

The circuit transformation task involves two steps: *initial mapping* and *qubit routing*. A typical QCT algorithm runs as follows: Select an initial mapping τ_0. After removing all or a subset of executable gates under τ_0 from C, repeat the following two procedures alternatively until there are no gates left: (i) transform the mapping into a new mapping by inserting one or several SWAP gates and (ii) remove all or a subset of executable gates.

Notation: Throughout this paper, we will use p, q, p', q', possibly with subscripts, to denote logical qubits, and use

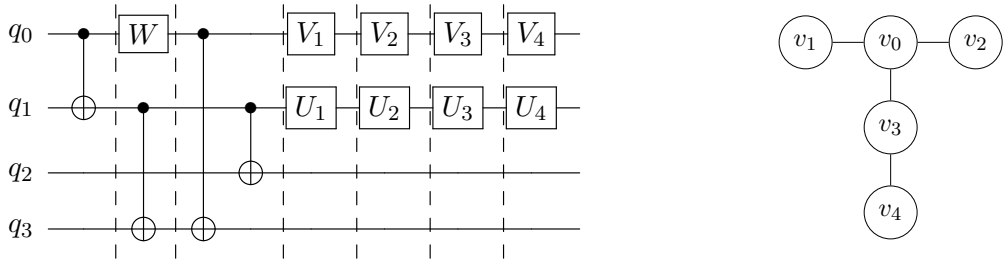

Fig. 1: A logical circuit with depth 7 (left) and the architecture graph of IBM Q Ourense (right)

u, v, u', v', possibly with subscripts, to denote physical qubits, i.e., vertices in the architecture graph.

C. SABRE

We first recall the heuristics of SABRE [14]. Let S be a set of 2-qubit gates and $\pi : P \to V$ the current mapping, where P and V denote the set of logical qubits in a given circuit C and physical qubits in a given architecture graph G. Define

$$V(\pi, S) \triangleq \sum_{g \in S} \text{dist}\big(\pi(g.p_0), \pi(g.p_1)\big), \tag{1}$$

where $g.p_0$ and $g.p_1$ are the first and second qubits that gate g acts on, and $\text{dist}(v, v')$ is the length of a shortest path from v to v' in the architecture graph. That is, $V(\pi, S)$ measures the total distances between the qubit pair of all 2-qubit gates in S, under the mapping π. The smaller $V(\pi, S)$ is the better π is.

Let F be the 2-qubit gates in the current front layer and X the set of 2-qubit gates in the extended set (which are 2-qubit gates that follow those in the front layer but could occupy on several subsequent layers), which are used to 'look ahead' and increase the applicability of a candidate mapping to 'future' gates. For each edge (v_i, v_j) in the architecture graph, we define three heuristics:

$$H_{\text{basic}} = V(\pi', F) \tag{2}$$

$$H_{\text{lookahead}} = \frac{1}{|F|} V(\pi', F) + w \cdot \frac{1}{|X|} V(\pi', X) \tag{3}$$

$$H_{\text{decay}} = \max\big(\text{decay}(v_i), \text{decay}(v_j)\big) \cdot H_{\text{lookahead}}, \tag{4}$$

where $\pi' \triangleq \pi_{i,j} \circ \pi$, $\pi_{i,j}$ is the permutation obtained by swapping v_i, v_j, $\text{decay}(v_i) \geq 1$ is a *decay factor* and $w \in [0, 1]$ is a weight. The initial decay factor of a qubit is 1, and is incremented by some pre-defined *decay rate* δ every time that qubit is involved in a SWAP, thus encouraging the algorithm to insert SWAP gates acting on different qubits, i.e., they can be executed in parallel. In the search procedure, SABRE inserts $\text{SWAP}(v_i, v_j)$ if it has the minimum H value.

The key factor driving SABRE's success is its unique approach to integrating the initial mapping and routing processes. Initially, a random mapping is applied, and then the routing process is run using either the "lookahead" or "decay" search strategies until all gates in the circuit $C = (g_1, \ldots, g_m)$ have been executed. The final mapping is then used as the initial mapping for transforming the reverse circuit $C^{-1} = (g_m, \ldots, g_1)$, which is routed using the same approach. This forward-backward transformation may iterate multiple times, and the final mapping obtained is used as the actual initial mapping to transform C. As a result, the algorithm takes into account the global information of the circuit when generating the initial mapping.

III. Our Method

In this section we describe our method. We start with a motivating example and then describe in detail our approach to enhance a QCT algorithm like SABRE.

A. A Motivating Example

Consider the logical circuit as well as the architecture graph of IBM Q Ourense shown on Fig. 1, which will be the running example in this paper. Fig. 2 and Fig. 3 show two examples of a SABRE transformation, where the initial mapping τ is defined as $q_0 \mapsto v_1, q_1 \mapsto v_0, q_2 \mapsto v_2, q_3 \mapsto v_3$. Since $\tau(q_1) = v_0$, the CX gates $CX(q_0, q_1), CX(q_1, q_3), CX(q_1, q_2)$ are all executable. We transform these gates immediately. Since single-qubit gates are always executable, we transform W and U_1 to U_4 immediately. As $\tau(q_0) = v_1$ and $\tau(q_3) = v_3$ are not neighbours in G, $CX(q_0, q_3)$ is not executable. Note that $CX(q_0, q_3)$ is the only CX gate in the current front layer and there are no CX gates in the extended set. We have two SWAP gate candidates, viz. $\text{SWAP}(v_0, v_1)$ and $\text{SWAP}(v_0, v_3)$, each corresponding to an edge in G. In general, a candidate SWAP gate $\text{SWAP}(v, v')$ corresponds to an edge (v, v') in G such that either v or v' is in the image of τ and either $\tau^{-1}(v)$ or $\tau^{-1}(v')$ is in F, where τ^{-1} is the inverse of τ. Suppose we use the "lookahead" heuristics. By Eq. 3, it is easy to see

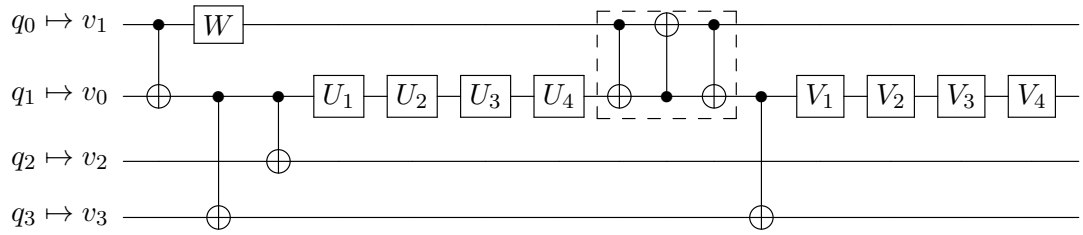

Fig. 2: A SABRE transformation with depth 15.

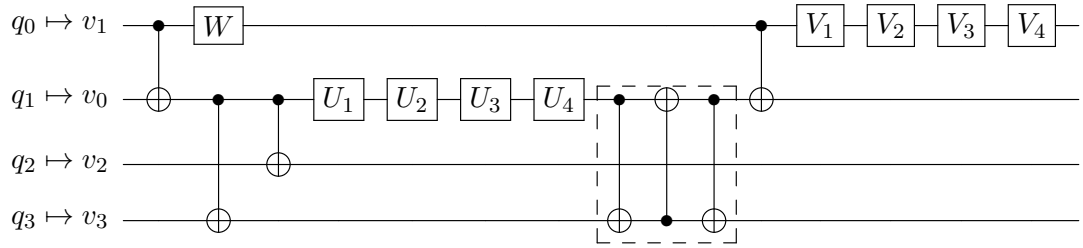

Fig. 3: Another SABRE transformation with depth 15.

that both $\text{SWAP}(v_0, v_1)$ and $\text{SWAP}(v_0, v_3)$ have score $H = 1$ and we can insert either to transform the circuit, resulting in the physical circuits depicted in Fig.s 2 and 3 respectively. After the insertion, all the remaining gates are executable and the transformation is completed.

These two transformations show that sometimes **a single SWAP insertion can double the circuit depth**! In fact, if we replace the single-qubit gate lists $V_1, \ldots, V_4$ and $U_1, \ldots, U_4$ in Fig. 1 with $V_1, \ldots, V_k$ and $U_1, \ldots, U_k$ for $k \in \mathbb{N}$, then a single SWAP insertion as shown in Fig. 2 or Fig. 3 increases the circuit depth from $k+3$ to $2k+7$! This, however, can be avoided by, for example, inserting the same SWAP gate $\text{SWAP}(v_0, v_1)$ before the single-qubit gate U_1, resulting a better transformation with depth $k+7$ (see Fig. 4).

The example shown above suggests that single-qubit gates can significantly affect the depth of the transformed circuit. In the following, we show how the intuition illustrated in Fig. 4 can be developed into a general method for improving the depth optimality of QCT algorithms.

B. Qubit Progress

To track the transformation progress of a quantum (logical) circuit C on a target architecture G, each physical qubit v on G is associated with a progress indicator denoted as $PG(v)$.

Initially, all $PG(v)$ values are set as 0. Whenever a single-qubit gate $U(p)$ acting on logical qubit p is transformed, we increment the progress on the corresponding physical qubit $v = \tau(p)$ by 1, where τ is the current mapping. If a 2-qubit gate (CX or CZ) $V(p, p')$ acting on logical qubits p and p' is transformed, the progress on the physical qubits $v = \tau(p)$ and $v' = \tau(p')$ should be **realigned**, by setting both $PG(v)$ and $PG(v')$ to $\max(PG(v), PG(v'))+1$. When a SWAP gate is inserted on $v = \tau(p)$ and $v' = \tau(p')$, the progress on the physical qubits v and v' should also be realigned, by setting both $PG(v)$ and $PG(v')$ to $\max(PG(v), PG(v')) + 3$, considering that each SWAP gate is decomposed into three consecutive CX gates.

With the transformation progress indicator PG, it is possible to identify which qubit is progressing too quickly and which is lagging behind. For example, suppose we apply the transformation depicted in Fig. 2 and have transformed $CX(q_0, q_1)$, $CX(q_1, q_3)$, $CX(q_1, q_2)$, and single-qubit gates W and all U_i for $1 \leq i \leq 4$. Then, the progress on the qubits are as follows: $PG(v_0) = 7$, $PG(v_1) = 2$, $PG(v_2) = 3$, and $PG(v_3) = 2$, see Table I. Apparently, v_0 is progressing too rapidly.

TABLE I: The qubit progress immediately after a particular gate has been transformed as in Fig. 2.

	$CX(q_0, q_1)$	W	$CX(q_1, q_3)$	$CX(q_1, q_2)$	U_4
$PG(v_0)$	1	1	2	3	7
$PG(v_1)$	1	2	2	2	2
$PG(v_2)$	0	0	0	3	3
$PG(v_3)$	0	0	2	2	2

C. Single-qubit Gate Buffer

In contrast to SABRE and many other QCT algorithms, when a single-qubit gate is next in line to be transformed on physical qubit v, it is not transformed

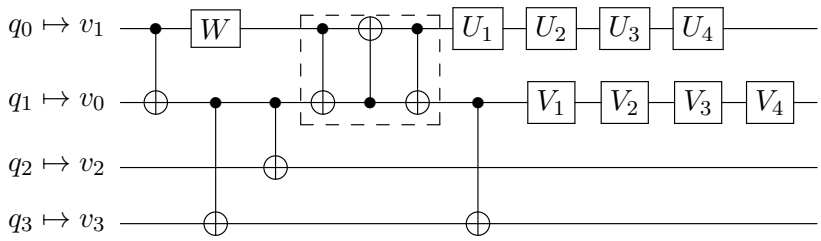

Fig. 4: A transformation with depth 11.

immediately. Instead, it is placed in a *buffer* (a temporary storage), which, denoted as $S(v)$, is an ordered set. The search for the next executable 2-qubit gate continues. When a new executable 2-qubit gate $CX(p, p')$ is encountered, all single-qubit gates in $S(v)$ and $S(v')$, where $v = \tau(p)$ and $v' = \tau(p')$, are transformed one by one. If no such executable 2-qubit gate is found, these single-qubit gates in $S(v)$ and $S(v')$ remain reserved until the next SWAP gate is selected for insertion. For example, in Fig. 2, before inserting the SWAP gate, we store the executable single-qubit gates W and $U_i, 1 \leq i \leq 4$ in, respectively, $S(v_1)$ and $S(v_0)$. Meanwhile, $S(v_2) = S(v_3) = \varnothing$. Since these single-qubit gates are reserved, the qubit progress at this time is $PG(v_0) = PG(v_2) = 3, PG(v_1) = 1, PG(v_3) = 2$.

D. SWAP Selection and Insertion

We can enhance the heuristic H adopted by a QCT algorithm with the qubit progress indicator PG. Take SABRE as an example. For each SWAP on some edge (v, v') in $G = (V, E)$, let $\nu(v, v') = \max(PG(v), PG(v'))$. Then, the enhanced heuristic score associated with that SWAP is

$$H_{\mathsf{sqgm}} = H + \frac{\nu(v, v')}{|V|}, \tag{5}$$

where H is the score computed by Eq. 3 or Eq. 4. Continuing with our running example, suppose that all gates executable by the initial mapping have been processed (i.e., executed and removed or stored in their buffers). Recall that we have two candidate SWAPs, i.e., those on edges (v_0, v_1) and (v_0, v_3). Recall also that the front layer contains one gate $CX(q_0, q_3)$ and the extended set is empty. By Eq. 3, the lookahead scores of these edges are $H(v_0, v_1) = H(v_0, v_3) = 1$. Thus, SABRE will select one SWAP from (v_0, v_1) or (v_0, v_3). Observe that $\nu(v_0, v_1) = \nu(v_0, v_3) = 3$ (cf. Sec. III-C). Thus, the enhanced heuristic scores for both (v_0, v_1) or (v_0, v_3) are $1+3/5$ (since IBM Q Ourense has 5 qubits). Hence, the enhanced SABRE also picks randomly from (v_0, v_1) and (v_0, v_3).

Suppose the selected best SWAP operation acts on physical qubits $v = \tau(p), v' = \tau(p')$. We need to decide where to insert this SWAP. This depends on the qubit progress and the buffer sizes of v, v'. If $PG(v) = PG(v')$, then we insert the SWAP gate in the front of $S(v), S(v')$, i.e., before the first single-qubit gates in the buffers of v, v'. If $PG(v) < PG(v')$, then we insert the SWAP gate in front of the k-th single-qubit gate in $S(v)$ (after transforming them one by one), where $k = \min(PG(v') - PG(v), |S(v)|)$. The case when $PG(v) > PG(v')$ is analogous. After swapping, we need to exchange the buffers of v, v'.

In our example, suppose we picked (v_0, v_1) as the SWAP gate. Since $PG(v_0) = 3 > 1 = PG(v_1)$ and $S(v_1) = (W), S(v_0) = (U_1, \ldots, U_4)$, we first transform W (and remove it from $S(v_1)$) and then insert $\textsc{swap}(v_0, v_1)$ immediately after $CX(q_1, q_2)$. Finally, we swap the buffers for v_1 and v_0. That is, after the swap insertion, we have $PG(v_0) = PG(v_1) = 6$ and $S(v_1) = (U_1, \ldots, U_4)$ and $S(v_0) = S(v_2) = S(v_3) = \varnothing$. This is precisely the transformation described in Fig. 4. Note that after swap insertion, the remaining gates are executable and we also transform all single-qubit gates in the buffer if there are any.

Algorithm 1 presents the pseudocode of our algorithm SQGM. The procedure $\textsc{getSwapCands}(F, G)$ on line 7 obtains SWAP candidates from the physical neighbours of the qubits involved in the front layer F. For every non-executable gate $g(p_0, p_1) \in F$,[2] we retrieve the corresponding physical qubits $v_0 = \tau(p_0), v_1 = \tau(p_1)$ and construct N_0, N_1 via the target architecture graph $G = (V, E)$, where we define $N_i = \{(v_i, v) : v \in V, (v_i, v) \in E\}$, i.e., the set of edges incident to v_i in G. Then, the SWAP candidates are precisely $N_0 \cup N_1$. The procedure $\textsc{addResolvedSuccessors}(F, g)$ on line 37 checks every successor gate h of g, and adds h to F if *all* of h's predecessor gates have been executed.

[2]Note that g is guaranteed to be a 2-qubit gate as all single-qubit gates before g are either executed or stored in a buffer.

Algorithm 1 Single-Qubit Gates Matter

Require: An ideal circuit C, a target architecture graph $G = (V, E)$, an initial mapping τ_0
Ensure: A transformation of C satisfying G's connectivity constraints

Initialise a buffer $S(v_i)$ for each $v_i \in V$
$F \leftarrow$ front layer of C
$\tau \leftarrow \tau_0$
while $F \neq \varnothing$ **do**
    executableGateList $\leftarrow$
        all $g \in F$ executable on G
    if executableGateList $= \varnothing$ **then**
        swapCands $\leftarrow$ getSwapCands(F, G)
        Compute H_{sqgm} for each swap $\in$ swapCands
        $(p_0, p_1) \leftarrow$ a swap with minimum H_{sqgm}
        if $PG(\tau(p_0)) < PG(\tau(p_1))$ **then**
            $v_0, v_1 \leftarrow \tau(p_0), \tau(p_1)$
        else
            $v_0, v_1 \leftarrow \tau(p_1), \tau(p_0)$
        end if
        $k \leftarrow \min(PG(v_1) - PG(v_0), |S(v_0)|)$
        for g_0 in the front k gates of $S(v_0)$ **do**
            Execute and remove g_0 from $S(v_0)$
            $PG(v_0) \leftarrow PG(v_0) + 1$
        end for
        $S(v_0), S(v_1) \leftarrow S(v_1), S(v_0)$
        $PG(v_0) \leftarrow PG(v_1) \leftarrow$
            $\max(PG(v_0), PG(v_1)) + 3$
        Update τ with (p, q)
    else
        for $g \in$ executableGateList **do**
            if $g(p)$ is a single-qubit gate **then**
                Add g to $S(\tau(p))$ *# Do not execute g yet*
            else if $g(p_0, p_1)$ is a 2-qubit gate **then**
                for $i \in \{0, 1\}$ and $g_i \in S(\tau(p_i))$ **do**
                    Execute and remove g_i from $S(\tau(p_i))$
                    $PG(\tau(p_i)) \leftarrow PG(\tau(p_i)) + 1$
                end for
                Execute g
                $PG(\tau(p_0)) \leftarrow PG(\tau(p_1)) \leftarrow$
                    $\max(PG(\tau(p_0)), PG(\tau(p_1))) + 1$
            end if
            Remove g from F
            addResolvedSuccessors(F, g)
        end for
        Clear executableGateList
    end if
end while

IV. Experiments and Evaluation

Our algorithm (implemented in Python 3) and benchmarks as well as experimental results are available at https://github.com/ebony72/sqgm. All our experiments were run on a computer with AMD Ryzen 5 5600 CPU, 32 GB RAM and AMD Radeon RX 6600 GPU.

SABRE has recently been assembled in Qiskit. In this paper, we choose this Qiskit (version 0.39.4) implementation of SABRE and adopt the advanced 'decay' heuristic, where the weight w and the decay rate δ are 0.5 and 0.001 as usual (cf. Sec. II-C). When comparing our algorithm with SABRE, or comparing different versions of SABRE, we take the same initial mapping generated from the current `SabreLayout` module (the initial mapping pass of SABRE), but use different routing modules.

The current Qiskit `SabreSWAP` (the routing pass of SABRE) uses internal accelerators written in Rust which are not open-source. Therefore, when implementing our SQGM routing module, we go back to Qiskit version 0.33.0 and modify the `SabreSWAP` module there. The 0.39.4 version of `SabreSWAP` (henceforth SABRE39) performs slightly (around 2-10%) better than the 0.33.0 version (henceforth SABRE33), thanks to several optimisations. Unless otherwise specified, the SABRE algorithm always refers to SABRE39.

A. Benchmarks and Architecture Graphs

To compare the depth optimality performance of our algorithm with SABRE and two other state-of-the-art algorithms, we considered extensive synthetic and real benchmarks on three architecture graphs: IBM Q Tokyo (20Q), IBM Q Rochester (53Q), and Google Sycamore (53Q and 54Q), see Fig. 5. Note that Google Sycamore has a bad qubit node, i.e., the black node in Fig. 5 (bottom left). Sometimes we will remove this node as well as its incident edges. We call this modified architecture the 53Q Sycamore.

The `QUEKO` benchmark [20] is designed to evaluate the depth optimality of QCT algorithms for various architectures and has been adopted by several researchers in evaluating QCT algorithms [23], [18], [24]. By design, each `QUEKO` circuit has a zero-cost optimal transformation, i.e., if we can find the right initial mapping, then no SWAP is required to transform the circuit. Our evaluation reveals that, on IBM Q Tokyo and `QUEKO` circuits, SABRE can often find an optimal transformation within 100 repeats. So, in the following, we only consider 54Q `QUEKO` benchmarks, for which an optimal transformation is difficult to find due to the large search space.

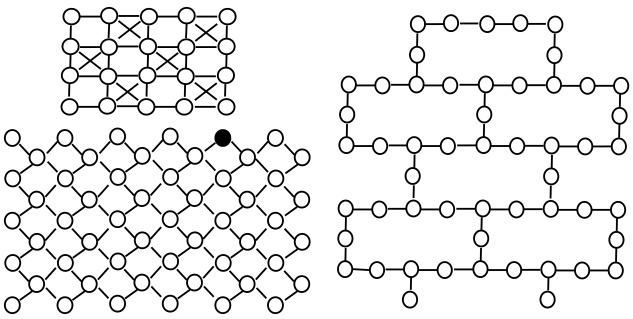

Fig. 5: The architecture graphs of IBM Q Tokyo (20Q, upper left), Google Sycamore (54Q, bottom left) and IBM Q Rochester (53Q, right), where the black node in Google Sycamore denotes a bad qubit node.

This shortcoming of QUEKO has been addressed in [24], where QUEKNO benchmarks are proposed for evaluating the optimality of QCT algorithms. Each QUEKNO circuit has a nonzero optimal transformation cost which can be closely estimated. For each architecture, we evaluate SQGM and compare algorithms on the corresponding QUEKNO benchmark set, which contains 120 circuits that all have a known *near-optimal* transformation cost.

Besides these synthesised benchmarks, we also extracted real quantum circuits from MQTBench [25], to be run on 54Q Google Sycamore. The circuit set we selected include 117 circuits, which are obtained by selecting

- All scalable benchmarks;
- Qubit Range: between 50 and 54;
- Target-independent Level: Qiskit as the used compiler; and
- Target-dependent Native Gates Level: targeted native gate-set from IBM, Qiskit as the used compiler, and optimisation level Opt. 3.

These circuits include well-known quantum circuits, such as Amplitude Estimation, Graph State, GHZ State, Grover's, QAOA, QFT, Quantum Phase Estimation, Quantum Walk, VQE, W-State. Before sending these circuits to a QCT algorithm, we decompose each non-standard gate into standard IBM basic gates. Table III shows name and depth of all those circuits with 53 qubits.

B. Compare SABRE and SQGM

We first compare the performances of SQGM and SABRE on several large benchmark sets. Recall that SABRE can often find the optimal solution for QUEKO benchmarks on IBM Q Tokyo. To make a meaningful comparison, we evaluate SQGM on QUEKNO benchmarks [24] and the three architecture graphs in Fig. 5. For each architecture, the corresponding QUEKNO benchmark set contains 120 circuits. We summarise the results in Table II, where we can see the improvement on 20Q Tokyo is about 15% and the improvements on 53Q Rochester and 53Q Sycamore are 17% and 23%. Note that here we used 53Q Sycamore instead of the 54Q version because the corresponding QUEKNO benchmarks were designed for 53Q Sycamore, for which we have known near-optimal depth costs.

TABLE II: Compare SABRE with SQGM on QUEKNO benchmarks [24] for evaluating depth optimality.

benchmark sets device	20Q_depth_Tokyo IBM Q Tokyo	53Q_depth_Rochester IBM Q Rochester	53Q_depth_Sycamore 53Q Google Sycamore
SABRE	2.177	2.748	2.577
SQGM	1.854	2.289	1.992
ratio	**0.852**	**0.833**	**0.773**

Note: each entry in row 'SABRE' is the geomean of 120 results, each of which is obtained as the best depth ratio (i.e., the ratio of the smallest depth of SABRE transformed circuit among 5 repeats and that of the input circuit); similar interpretation applies to entries in row 'SQGM'; entries in the 'ratio' row denote the ratios of the corresponding geomeans for SQGM and SABRE.

Since the above benchmarks are all synthesised, we also compared SABRE and SQGM on the 117 MQT benchmarks [25] and 54Q Google Sycamore. Table III shows the results for all 53Q circuits, from which we can see that the average improvement of SQGM over SABRE is 27% and the best improvement could be as high as 50%.

Role of the Heuristic The above evaluation shows that SQGM can significantly improve the depth optimality of SABRE. An experiment on 54Q Sycamore and ten QUEKO benchmarks '54QBT_25CYC_QSE' shows that, if we use H_{decay} (Eq. 4) instead of H_{sqgm} (Eq. 5), the improvement is around 9%, instead of 33% obtained from H_{sqgm}. This suggests that the SQGM heuristic (for selecting the right SWAP candidate) plays a more important role than simply adjusting the position of the selected SWAP by comparing the qubit progress.

Time Complexity SQGM has the same time complexity as SABRE, i.e., $O(m \cdot n^{2.5})$, where m is the number of gates and n the number of qubits in the quantum device. Using internal accelerators, SABRE39 is faster than SABRE33 and SQGM in practice. On the 117 MQT benchmarks, the runtimes of SQGM ranges from 0.09 to 28.01 seconds and, on average, SQGM takes 1.5x and 0.5x more time than SABRE39 and SABRE33.

TABLE III: Comparison of SQGM and NASSC against SABRE on 53-qubit MQT circuits [25] and Google Sycamore (54Q), where '+CC' indicates that the post-routing `Commutative Gate Cancellation` pass has been applied on the respective algorithm (cf. Sec. IV-E). The depth columns show the input depths or the depths of the circuits transformed by an algorithm; the ratio columns show the ratios of the transformed circuit depths of an algorithm and those of SABRE.

Circuit		SABRE	SABRE+CC		SQGM		SQGM+CC		NASSC	
name	depth	depth	depth	ratio	depth	ratio	depth	ratio	depth	ratio
ae_indep_qiskit_53.qasm	614	1961	1736	0.89	1489	0.76	1412	0.72	1756	0.90
ae_nativegates_ibm_qiskit_opt3_53.qasm	872	1845	1876	1.02	1718	0.93	1722	0.93	1666	0.90
dj_indep_qiskit_53.qasm	55	168	166	0.99	128	0.76	132	0.79	172	1.02
dj_nativegates_ibm_qiskit_opt3_53.qasm	58	180	176	0.98	135	0.75	132	0.73	171	0.95
ghz_indep_qiskit_53.qasm	54	157	149	0.95	79	0.50	75	0.48	149	0.95
ghz_nativegates_ibm_qiskit_opt3_53.qasm	56	126	137	1.09	81	0.64	75	0.60	134	1.06
graphstate_indep_qiskit_53.qasm	21	43	45	1.05	39	0.91	36	0.84	38	0.88
graphstate_nativegates_ibm_qiskit_opt3_53.qasm	21	35	30	0.86	29	0.83	29	0.83	23	0.66
qft_indep_qiskit_53.qasm	419	1859	1489	0.80	1282	0.69	1174	0.63	1459	0.78
qft_nativegates_ibm_qiskit_opt3_53.qasm	370	1491	1315	0.88	1312	0.88	1217	0.82	1412	0.95
qftentangled_indep_qiskit_53.qasm	421	1967	1811	0.92	1367	0.69	1245	0.63	1720	0.87
qftentangled_nativegates_ibm_qiskit_opt3_53.qasm	373	1569	1493	0.95	1305	0.83	1145	0.73	1373	0.88
qpeexact_indep_qiskit_53.qasm	619	2236	1877	0.84	1521	0.68	1447	0.65	1931	0.86
qpeexact_nativegates_ibm_qiskit_opt3_53.qasm	523	1743	1611	0.92	1404	0.81	1427	0.82	1527	0.88
qpeinexact_indep_qiskit_53.qasm	619	2384	1934	0.81	1608	0.67	1485	0.62	1952	0.82
qwalk-v-chain_indep_qiskit_53.qasm	29447	42033	41931	1.00	35973	0.86	34872	0.83	40850	0.97
realamprandom_indep_qiskit_53.qasm	214	2724	2475	0.91	1693	0.62	1529	0.56	2016	0.74
realamprandom_nativegates_ibm_qiskit_opt3_53.qasm	226	2605	2466	0.95	1671	0.64	1482	0.57	1965	0.75
su2random_indep_qiskit_53.qasm	214	2587	2243	0.87	1619	0.63	1423	0.55	1942	0.75
su2random_nativegates_ibm_qiskit_opt3_53.qasm	226	2667	2329	0.87	1674	0.63	1555	0.58	1774	0.67
twolocalrandom_indep_qiskit_53.qasm	214	2635	2290	0.87	1628	0.62	1541	0.58	1851	0.70
twolocalrandom_nativegates_ibm_qiskit_opt3_53.qasm	226	2643	2571	0.97	1635	0.62	1457	0.55	2019	0.76
wstate_indep_qiskit_53.qasm	160	200	208	1.04	178	0.89	170	0.85	180	0.90
wstate_nativegates_ibm_qiskit_opt3_53.qasm	213	259	258	1.00	235	0.91	226	0.87	153	0.59
geomean				**0.93**		**0.73**		**0.69**		**0.83**

C. Performance Under Different Repeats

SABRE and SQGM are randomised algorithms. By design, better results can be obtained if we run them more times. In above, we have seen that SQGM performs significantly better than SABRE, but the conclusion was obtained for a fixed repeat number, viz. 5. What may happen if we run both SABRE and SQGM, say, 50 times: will SQGM still perform significantly better? This subsection is devoted to examining their performance under different repeat numbers.

We compare the performance of SABRE33, SABRE39, and SQGM on the ten `QUEKO` '54QBT_25CYC_QSE' benchmarks under different repeats, ranging from 5 to 1000. For each algorithm and each repeat number k, let δ_i^k be the minimum depth after k repeats for circuit i. Let also

$$\Delta^k \equiv \text{the geomean of } \delta_i^k/\delta_i^5 \text{ for } 0 \le i \le 9. \quad (6)$$

Then, $1-\Delta^k$ quantifies the average improvement when the repeat number is increased from 5 to k. Fig. 6 (top) shows the change of Δ^k for SABRE33, SABRE39 and SQGM with the repeat number k ranging from 5 to 1000. We note that the scale in the x-axis of this figure is not uniform. It is clear that SABRE33 and SABRE39 have the same trends and the depth of the transformed circuit can be reduced on average by 15%, 20%, 25%, 30%, 40%, 55% if we repeat, respectively, 20, 40, 75, 250, 400, 750 times. A similar pattern is also observed for SQGM, though its circuit depth reduction decreases less rapidly than SABRE33 and SABRE39 with increasing repeat number.

In addition, we compare the performance of SABRE33 (SQGM) against that of SABRE39 on circuit i by setting λ_i^k as the ratio between the minimum depths after k repeats for SABRE33 (SQGM) and SABRE39. Let

$$\Lambda^k \equiv \text{the geomean of } \lambda_i^k \text{ for } 0 \le i \le 9. \quad (7)$$

Then $1-\Lambda^k$ quantifies the average improvement of SABRE33 (SQGM) against SABRE39. The result in Fig. 6 (bottom) shows that SABRE39 often outperforms SABRE33 and the improvement of SQGM against SABRE decreases with k from 35% to 15%. It suggests in particular that SQGM still outperforms SABRE significantly even after 1000 repetitions.

A similar pattern was also observed on the ten `QUEKNO` '53QBT_depth_Sycamore_large_opt_10_2.55' benchmarks.

D. Compare with TOQM

TOQM (**T**ime-**O**ptimal **Q**ubit **M**apping) [18] is an

TABLE IV: Comparison of SABRE and SQGM with TOQM on 20Q IBM Q Tokyo and 10 QUEKNO benchmarks "20QBT_depth_Tokyo_large_opt_10_2.55".

Circuit Information				TOQM		SABRE repeat=5		SQGM repeat=5		SABRE repeat=100		SQGM repeat=100	
No.	#gate	#CX	depth	depth	ratio	depth	ratio	depth	ratio	depth	ratio	depth	ratio
0	760	229	93	188	2.02	266	2.86	195	2.10	232	2.49	180	1.94
1	769	229	97	178	1.84	220	2.27	164	1.69	193	1.99	157	1.62
2	725	236	93	176	1.89	245	2.63	229	2.46	209	2.25	173	1.86
3	756	227	90	179	1.99	253	2.81	202	2.24	222	2.47	171	1.90
4	850	254	99	188	1.90	284	2.87	229	2.31	248	2.51	200	2.02
5	795	258	95	191	2.01	262	2.76	208	2.19	244	2.57	187	1.97
6	864	274	110	192	1.75	291	2.65	215	1.95	256	2.33	215	1.95
7	801	245	95	178	1.87	284	2.99	216	2.27	250	2.63	204	2.15
8	893	280	109	188	1.72	311	2.85	206	1.89	235	2.16	191	1.75
9	837	238	101	188	1.86	318	3.15	247	2.45	259	2.56	201	1.99
	geomean				**1.88**		**2.77**		**2.14**		**2.39**		**1.91**

TABLE V: Comparison of SABRE and SQGM with TOQM on 54Q Sycamore and 10 QUEKNO benchmarks "20QBT_depth_Tokyo_large_opt_10_2.55".

Circuit Information				TOQM		SABRE repeat=5		SQGM repeat=5		SABRE repeat=100		SQGM repeat=100	
No.	#gate	#CX	depth	depth	ratio	depth	ratio	depth	ratio	depth	ratio	depth	ratio
0	760	229	93	274	2.95	311	3.34	202	2.17	265	2.85	205	2.20
1	769	229	97	264	2.72	241	2.48	175	1.80	216	2.23	170	1.75
2	725	236	93	253	2.72	313	3.37	215	2.31	228	2.45	197	2.12
3	756	227	90	281	3.12	232	2.58	206	2.29	258	2.87	197	2.19
4	850	254	99	245	2.47	307	3.10	248	2.51	295	2.98	226	2.28
5	795	258	95	285	3.00	309	3.25	227	2.39	268	2.82	202	2.13
6	864	274	110	280	2.55	379	3.45	254	2.31	303	2.75	230	2.09
7	801	245	95	298	3.14	331	3.48	240	2.53	260	2.74	208	2.19
8	893	280	109	291	2.67	326	2.99	261	2.39	276	2.53	232	2.13
9	837	238	101	282	2.79	324	3.21	251	2.49	277	2.74	210	2.08
	geomean				**2.80**		**3.11**		**2.31**		**2.69**		**2.11**

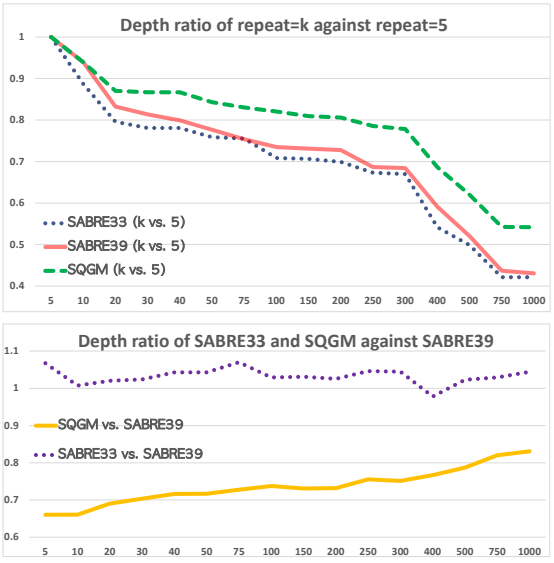

Fig. 6: Changes of Δ^k (Eq. 6) for SABRE33, SABRE39, and SQGM (top) and changes of Λ^k (Eq. 7) of SABRE33 and SQGM against SABRE39 (bottom) on the 10 QUEKO '54QBT_25CYC_QSE' benchmarks and 54Q Sycamore.

exact QCT algorithm for optimising the depth of the output circuit. Relaxation techniques are introduced to make TOQM scalable to 16 qubits and tens of thousands of gates. To evaluate the scalability of TOQM on large devices, we compared the performance of TOQM, SABRE, SQGM on ten QUEKNO '20Q_Tokyo_depth' benchmarks [24] on IBM Q Tokyo and 54Q Google Sycamore. The results are presented in Tables IV and V. While Tables IV confirms the efficacy of TOQM on the 20Q IBM Q Tokyo (1.88 vs. 2.39 and 1.91 for SABRE and SQGM with repeat=100), Table V shows that, running the same batch of benchmarks on 54Q Sycamore, TOQM does not perform better than SABRE and SQGM (2.80 vs. 2.69 and 2.11 for SABRE and SQGM with repeat=100). We also tried to run the relaxed version of TOQM with QUEKO '54QBT_5CYC_QSE' (depth=5) circuits on 54Q Sycamore; however, it did not halt and return a result within 1,800 seconds. This indicates that, despite the relaxation techniques, TOQM is still not scalable to quantum devices with medium to large number of qubits.

E. Compare with NASSC

While most current QCT algorithms perform qubit routing independent of circuit optimisation, NASSC (**N**ot **A**ll **S**WAPs have the **S**ame **C**ost) [15] combines

qubit mapping with two optimisation techniques: 2-qubit block re-synthesis and commutation-based gate cancellation. It observes that (i) a SWAP gate could be merged into the previous 2-qubit block by re-synthesis to possibly reduce CX count by 3; (ii) a SWAP gate could be merged with a previous CX or SWAP gate to cancel two CXs. NASSC enriches `SabreSWAP` with the above optimisation techniques.

Continuing with our running example, Fig. 7 shows another better transformation, which inserts $\text{SWAP}(v_0, v_3)$ instead of $\text{SWAP}(v_0, v_1)$ as in Fig. 4. Note that the first CX of $\text{SWAP}(v_0, v_3)$ is commutable with the CX before it. If we commute and cancel it with the first $CX(v_0, v_3)$, we shall have a circuit with depth 9. One merit of NASSC is its ability to identify this possibility and, then, use the right SWAP decomposition in the post-routing optimisation.

NASSC has the same time complexity as SABRE and experiments show that it can reduce by 21.30% and 7.61%, on average, the number of CX gates and circuit depth, respectively, when compared with SABRE. As NASSC applies the Qiskit `Commutative Gate Cancellation` pass, it is not clear how much of this reduction is contributed solely by NASSC's routing pass.

While there are eight combinations of the three optimisations (2-qubit gate block re-synthesis and two commutation-based optimisations), enabling all three optimisations produces a similar performance to the best combination [15]. Hence, we assume all three optimisations as well as the following are enforced in NASSC:[3]

- *pre-routing optimisation*: involves combining chains of single-qubit gates into a single gate (single-qubit decomposition optimisation),
- *commutative gate cancellation (CC)*: involves iteratively cancelling gates based on commutation rules until no changes in depth are observed, and
- *post-routing optimisation*: involves both single-qubit decomposition optimisation and commutative gate cancellation.

For a fair comparison with NASSC, we also enhance the base SABRE and SQGM algorithms with a post-routing *commutative gate cancellation (CC)*.

We then compare the performance of SABRE, SQGM, and NASSC on selected `QUEKO` and MQTBench library circuits [25] on the 54Q Google Sycamore. To ensure fairness, the same initial mapping generated by Qiskit's SabreLayout was used across the algorithms for each repeat.

[3]Unitary synthesis was initially included for both pre- and post-optimisations, but was removed in our experiments as it was found that it led to significant increase in transformed circuit depth and running time.

TABLE VI: Comparison of SABRE and SQGM with NASSC on 10 `QUEKO` '54QBT_25CYC_QSE' benchmarks.

algorithm	SABRE	SABRE+CC	SQGM	SQGM+CC	NASSC
avg. depth ratio	5.18	4.00	3.34	2.88	3.96
ratio with SABRE	-	**0.77**	**0.64**	**0.56**	**0.76**

Note: each entry in row 'avg. depth ratio' is the geomean of best depth ratios obtained from 5 repeats for ten `QUEKO` circuits.

The first set of tests comprised the aforementioned `QUEKO` '54QBT_25CYC_QSE' circuits. Considering the base algorithms alone, NASSC and SQGM yielded much lower average depth ratios than SABRE (3.96 and 3.34 vs. 5.18). When post-routing CC optimisation is adopted, the ratios for SABRE+CC and SQGM+CC decrease sharply to 4.00 and 2.88. While NASSC remains slightly better than SABRE+CC, SQGM+CC beats NASSC by $1-2.88/3.96 = 27\%$. The unexpected sharp decrease obtained with CC is due to the presence of Pauli X gates as the sole kind of single-qubit gates in `QUEKO` circuits, thus enabling CC to cancel many gates.

The second set of tests comprised 117 real circuits from the MQT Bench library, with qubit number ranging from 50 to 54 qubits (cf. Sec. IV-A). For these circuits, the number of repeats was 5. Table III shows a part of the results for those circuits with 53 qubits. We observe from the table that, firstly, NASSC and SQGM reduce the depth by 17% and 27% against SABRE. Secondly, enhanced with post-routing CC, the depth of SABRE is reduced by 7%, which is quite modest compared with the result for `QUEKO` benchmarks shown in Table VI; this, however, should be more representative of the general case. Lastly, SQGM+CC obtains the best average result, which shows that enhancing with CC will further boost the depth optimality of SQGM.

V. Conclusion

While we have recently witnessed significant advances in building large quantum computers, qubit coherence time remains a strict restriction to NISQ devices. In this sense, it seems more important for qubit mapping to minimise the depth overhead. In this paper, we demonstrated that a simple SWAP gate insertion may double the depth of a circuit, from which we proposed a simple yet general method that takes into consideration the impact of single-qubit gates on circuit depth. The effectiveness of our method was demonstrated by embedding it in SABRE, and our experiments on three architectures and extensive benchmarks confirmed that

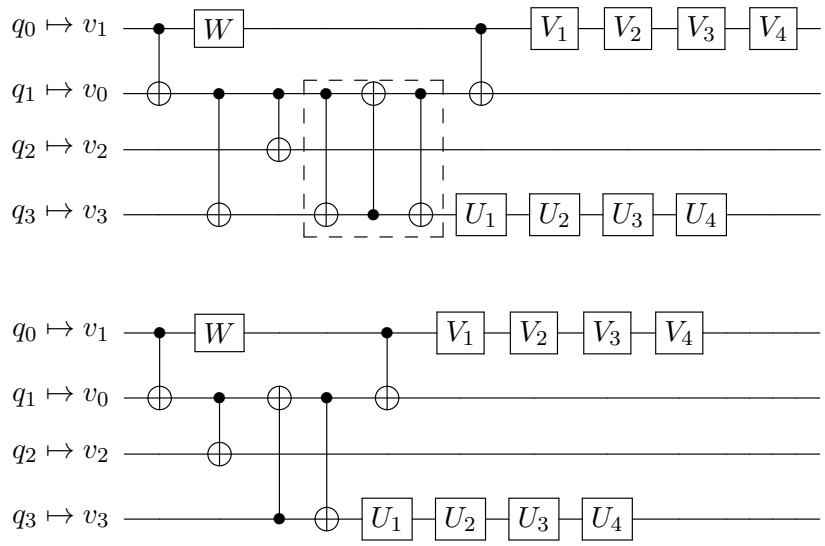

Fig. 7: Another better transformation with depth 11 (top). After applying commutative gate cancellation (cancelling the first two $CX(v_0, v_3)$ in the top circuit) the circuit depth is further reduced to 9 (bottom)

the new algorithm, called SQGM, is able to significantly reduce the depth overhead of circuit transformation when compared with SABRE. Compared with the relaxed version of TOQM [18], a time-optimal algorithm, our method is scalable to large devices. In addition, its performance can be further enhanced with post-routing commutative gate cancellation.

In real quantum devices, the reliability of qubits and qubits connectivity strengths vary spatially and temporally. Further work is required to extend our method to QCT algorithms which respect these hardware characteristics [16], [17] or those that mitigate crosstalk noises [10].

ACKNOWLEDGEMENTS

This work was partially supported by the Australian Research Council (Grant No.: DP220102059) and the National Science Foundation of China (Grant No.: 12071271). Ky Dan and Zachary were also partially supported by Sydney Quantum Academy (SQA) under two SQA Undergraduate Grants.

REFERENCES

[1] 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

[2] 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

[3] G. Nannicini, L. S. Bishop, O. Günlük, and P. Jurcevic, "Optimal qubit assignment and routing via integer programming," *ACM Transactions on Quantum Computing*, vol. 4, no. 1, oct 2022. [Online]. Available: https://doi.org/10.1145/3544563

[4] 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

[5] 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, 2018.

[6] A. Ash-Saki, M. Alam, and S. Ghosh, "Qure: Qubit re-allocation in noisy intermediate-scale quantum computers," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3316781.3317888

[7] H. Deng, Y. Zhang, and Q. Li, "Codar: A contextual duration-aware qubit mapping for various NISQ devices," in *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*. IEEE, 2020, pp. 1–6. [Online]. Available: https://doi.org/10.1109/DAC18072.2020.9218561

[8] T. Itoko, R. Raymond, T. Imamichi, A. Matsuo, and A. W. Cross, "Quantum circuit compilers using gate commutation rules," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ACM, 2019, pp. 191–196.

[9] B. Tan and J. Cong, "Optimal qubit mapping with simultaneous gate absorption," in *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021, Munich, Germany,*

November 1-4, 2021. IEEE, 2021, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ICCAD51958.2021.9643554

[10] L. Xie, J. Zhai, and W. Zheng, "Mitigating crosstalk in quantum computers through commutativity-based instruction reordering," in *58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021*. IEEE, 2021, pp. 445–450. [Online]. Available: https://doi.org/10.1109/DAC18074.2021.9586145

[11] X. Zhou, Y. Feng, and S. Li, "A Monte Carlo tree search framework for quantum circuit transformation," in *2020 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2020, pp. 1–7.

[12] 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*, vol. 39, no. 12, pp. 4683–4694, 2020.

[13] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2018.

[14] 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, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, I. Bahar, M. Herlihy, E. Witchel, and A. R. Lebeck, Eds. ACM, 2019, pp. 1001–1014. [Online]. Available: https://doi.org/10.1145/3297858.3304023

[15] J. Liu, P. Li, and H. Zhou, "Not all swaps have the same cost: A case for optimization-aware qubit routing," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 709–725.

[16] 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

[17] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for NISQ-era quantum computers," in *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. 987–999. [Online]. Available: https://doi.org/10.1145/3297858.3304007

[18] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen, and E. Z. Zhang, "Time-optimal qubit mapping," in *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*, T. Sherwood, E. D. Berger, and C. Kozyrakis, Eds. ACM, 2021, pp. 360–374. [Online]. Available: https://doi.org/10.1145/3445814.3446706

[19] 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.

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

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

[22] L. Lao, H. van Someren, I. Ashraf, and C. G. Almudéver, "Timing and resource-aware mapping of quantum circuits to superconducting processors," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 41, no. 2, pp. 359–371, 2022. [Online]. Available: https://doi.org/10.1109/TCAD.2021.3057583

[23] B. Tan and J. Cong, "Optimal layout synthesis for quantum computing," in *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2020, San Diego, CA, USA, November 2-5, 2020*. IEEE, 2020, pp. 137:1–137:9. [Online]. Available: https://doi.org/10.1145/3400302.3415620

[24] S. Li, X. Zhou, and Y. Feng, "On constructing benchmark quantum circuits with known near-optimal transformation cost," 2023. [Online]. Available: https://arxiv.org/abs/2301.08932

[25] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking software and design automation tools for quantum computing," 2022, MQT Bench is available at https://www.cda.cit.tum.de/mqtbench/.

[26] 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