Jiang H, Deng YX, Xu M. Qubit mapping based on tabu search. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY : 1–17

Qubit Mapping Based on Tabu Search

Hui Jiang, Yu-Xin Deng*, *Senior Member*, *CCF*, Ming Xu

Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China

E-mail: yhq_jh@126.com; yxdeng@sei.ecnu.edu.cn; mxu@cs.ecnu.edu.cn;

Abstract The goal of qubit mapping is to map a logical circuit to a physical device by introducing additional gates as few as possible in an acceptable amount of time. We present an effective approach called Tabu Search-based Adjustment (TSA) algorithm to construct the mappings. It consists of two key steps: one makes use of a combined subgraph isomorphism and completion to initialize some candidate mappings, and the other dynamically modifies the mappings by TSA. Our experiments show that, compared with state-of-the-art methods in the literature, TSA can generate mappings with a smaller number of additional gates and have better scalability for large-scale circuits.

Keywords qubit mapping, initial mapping, tabu search

1 Introduction

Quantum computing has attracted more and more interest in the last decades, since it provides the possibility to efficiently solve important problems such as integer factorization[1], unstructured search[2], and solving linear equations[3]. However, the (great) improvements in computer science driven by quantum technology are still in the early stage, since large-scale quantum computers have not yet been built. IBM has been developed the first 5-qubit backend called IBM QX2, followed by the 16-qubit backend IBM QX3. The revised versions of them are called IBM QX4 and IBM QX5, respectively. Google announced the realization of quantum supremacy, with the 53-qubit quantum processor Sycamore[4]. IBM Q Experience[1] provides the public with free quantum computing resources on the cloud and Qiskit[2], an open source quantum computing software framework.

Users of early quantum computers mainly rely on quantum circuits to implement quantum algorithms. There is a gap between the design and the implementation of a quantum algorithm[5]. In the design stage, we usually do not consider any hardware connectivity constraints. But in order to implement an algorithm on a quantum physical device, physical constraints have to be taken into account. For example, IBM physical devices only support 1-qubit gates and the 2-qubit $\boldsymbol{CX}$ gate between two adjacent qubits. Hence, it is necessary to transform the circuits for quantum algorithms to satisfy both logical and physical constraints. It is called qubit mapping, which maps a logical circuit to a physical device by inserting additional gates. A major challenge for quantum information processing is quantum decoherence. Quantum gates are applied in a coherent period but the qubits stay in the coher-

Regular Paper

This work was supported by the National Natural Science Foundation of China under Grant Nos. 61832015, 62072176, 12271172 and 11871221, the Research Funds of Happiness Flower ECNU under Grant No. 2020ECNU-XFZH005, the Fundamental Research Funds for the Central Universities under Grant No. 2021JQRH014, Shanghai Trusted Industry Internet Software Collaborative Innovation Center, and the "Digital Silk Road" Shanghai International Joint Lab of Trustworthy Intelligent Software under Grant No. 22510750100.

*Corresponding Author

[1] https://www.ibm.com/quantum-computing/, Mar. 2022

[2] https://www.qiskit.org/, Mar. 2022

ent state for a very short time. The longest coherence time of a superconducting quantum chip is still within 10–100us[6]. Thus, the main goal of qubit mapping is to reduce the number of additional gates and the depth of output circuits in an efficient way.

In the current work, we use In-Memory Subgraph Matching (IMSM)[7] to generate partial isomorphic subgraphs of logical circuits and physical ones as a set of partial initial mappings. By exploiting an appropriate subgraph isomorphism and the connectivity of the logical circuits and the physical ones, we get a dense (clustered nodes) initial mapping, which avoids some nodes from being mapped to remote positions. Note that both subgraph isomorphism and the adjustment of qubit mapping are NP-complete[8]. Thus, to be practically efficient, we propose to use tabu search[9] to generate logical circuits that will be executed on the physical device. The advantage of tabu search is to jump out of local optima and ensure the diversity of the transformed results. We insert $\boldsymbol{SWAP}$ gates, abbreviated as $\boldsymbol{SW}$, associated with the gates on the shortest path to the candidate set, which greatly reduces the search space and improves the search speed. We design three evaluation functions that consider not only the current gates but also the constraints of the gates already processed. Our experiments have been conducted by using the architectures of IBM Q Tokyo and Sycamore as the target physical devices. The experimental results show that the evaluation function based on calculating the number of additional gates inserts the smallest number of gates. We test several combinations of state-of-the-art initial mapping and adjustment algorithms aiming to insert fewer additional gates after qubit mapping. Generally speaking, Tabu Search-based Adjustment (TSA) outperforms the Zulehner-Paler-Wille (ZPW) algorithm[10], SWAP-based Bidi-REctional heuristic search algorithm (SABRE)[11] and Filtered Depth-Limited Search (FiDLS)[12] in different aspects. When compared with the Dynamic Look-ahead Heuristic technique (DLH)[13], which uses the maximum consecutive positive effect of a $\boldsymbol{SW}$ operation (MCPE) and the optimized version (MCPE_OP) as the heuristic cost function, the additional gates inserted by TSA in the DLH benchmarks have been reduced by 27.32% and 12.42%, respectively.

The main contributions of this article are summarized as follows.

1. We extend IMSM, which only generates a set of partial initial mappings, by completing the mapping based on the connectivity between qubits.
2. We propose a heuristic circuit adjustment algorithm based on tabu search, TSA, which can adjust large-scale circuits much more efficiently than existing precise search and heuristic algorithms.
3. We propose three look-ahead evaluation functions for the circuit adjustment; one employs configuration checking with aspiration (CCA)[14], and the other two use the number of additional gates and the depth of the generated circuit as evaluation criteria, taking into account both the current gates and some gates yet to be processed.
4. We compare several state-of-the-art initial mapping and adjustment algorithms, and the results show that the initial mapping generated by our method requires inserting fewer SWAP gates, and TSA has better scalability than them for adjusting the mapping for large-scale circuits.

The rest of this article is organized as follows. In Section 2 we discuss the related work. In Section 3 we recall some background in quantum computing and quantum information. In Section 4 we introduce the problem of qubit mapping and provide our detailed solution. Section 5 reported the experimental results.

We conclude in the last section and discuss the future work.

2 Related work

Paler[15] has shown that initial mappings have an important impact on qubit mapping. Just by placing qubits in different positions from the default trivial placement in the circuit instances on actual Noisy intermediate-scale quantum (NISQ) devices, the cost can be reduced by up to 10%. One important goal of circuit adjustment algorithms is to minimize the number of additional gates. There are currently five main methods to attack the qubit mapping problem.

- *Unitary matrix decomposition algorithm.* It is used to rearrange a quantum circuit from the beginning while retaining the input circuit[16, 17]. It can be applied to a broad class of circuits consisting of generic gate sets, but the results are not as efficient as a compiler designed specifically for this task.

- *Converting into some existing problems.* This approach converts the qubit mapping problem into some existing problems, such as AI planning[18, 19], integer linear programming[20] and satisfiability modulo theories (SMT)[21], and then uses existing tools to find the optimums in an acceptable amount of time for the problem. Furthermore, as the time cost is usually high, it can only process small-scale quantum circuits.

- *Exact methods.* Siraichi *et al.* have proposed an exact method[8]. It iterates over all possible mappings, thus it is only suitable for simple quantum circuits and cannot be extended to complex ones.

- *Graph theory.* Shafaei *et al.* have used the minimum linear permutation solution in graph theory to model the problem of reducing the interaction distance[22]. A two-step method is used to reduce the qubit mapping to a graph problem to minimize the number of additional gates[23, 24].

- *Heuristic search.* Existing solutions mainly aim at inserting as few $\boldsymbol{SW}$ gates as possible[8, 10, 11, 12, 13, 25, 26], using the fidelity of the generated circuit as the objective function[27] or minimizing the overall circuit latency[28]. At present, there are a number of methods[10, 11, 13, 22] that exploit the look-ahead idea. In particular, Zhu *et al.* proposed to dynamically adjust the number of look-ahead gates[13]. SABRE[11] depends on a random initial mapping. SAHS[26] is an annealing algorithm to find an initial mapping, but it is unstable. FiDLS[12] tends to search through all possible combinations of $\boldsymbol{SW}$ gates to minimize the number of executable 2-qubit gates. But the cost of a thorough search is very high, especially when dealing with medium-scale and large-scale circuits. DLH[13] can deal with some large-scale benchmarks. We will give a quantitative comparison with that method in Section 5. A variation-aware qubit movement strategy[27] is proposed, which takes advantage of the change in error rate and a change-aware qubit mapping strategy by trying to select the route with the lowest probability of failure. Lao *et al.* have shown that the fidelity of a circuit is related to the delay and the number of gates[28]. Now some heuristic methods are also applied to other platforms such as Surface-17[28, 29] and Sycamore[12].

3 Preliminary

In this section, we introduce some notions and notations of quantum computing. Let $\mathbb{C}$ denote the set of all complex numbers.

Classical information is stored in bits, while quan-

tum information is stored in qubits. Besides two basic states $|\mathbf{0}\rangle$ and $|\mathbf{1}\rangle$, a qubit can be in any linear superposition state like $|\phi\rangle = a\,|\mathbf{0}\rangle + b\,|\mathbf{1}\rangle$, where $a, b \in \mathbb{C}$ satisfy the condition $|a|^2 + |b|^2 = 1$. The intuition is that $|\phi\rangle$ is in the state $|\mathbf{0}\rangle$ with probability $|a|^2$ and in the state $|\mathbf{1}\rangle$ with probability $|b|^2$. We use the letter Q (resp. q) to denote a physical (resp. logical) qubit.

A quantum gate acts on a qubit to change the state of the qubit. For example, the Hadamard ($\boldsymbol{H}$) gate is applied on a qubit, and the $\boldsymbol{CX}$ gate is applied on two qubits. Their symbols and matrix forms are shown in Fig. 1. The $\boldsymbol{H}$ gate turns state $|\mathbf{0}\rangle$ (resp. $|\mathbf{1}\rangle$) into $\frac{1}{\sqrt{2}}(|\mathbf{0}\rangle + |\mathbf{1}\rangle)$ (resp. $\frac{1}{\sqrt{2}}(|\mathbf{0}\rangle - |\mathbf{1}\rangle)$). The $\boldsymbol{CX}$ gate is a generalization of the classical $\boldsymbol{XOR}$ gate, since the action of the gate may be summarized as $|\boldsymbol{A}, \boldsymbol{B}\rangle \rightarrow |\boldsymbol{A}, \boldsymbol{B} \oplus \boldsymbol{A}\rangle$, where $\oplus$ is addition modulo two, which is exactly what the $\boldsymbol{XOR}$ gate does. That is, the control qubit and the target qubit are $\boldsymbol{XOR}$ed and stored in the target qubit. Here $|\boldsymbol{A}, \boldsymbol{B}\rangle$ is a shorthand of the product state $|\boldsymbol{A}\rangle\,|\boldsymbol{B}\rangle = |\boldsymbol{A}\rangle \otimes |\boldsymbol{B}\rangle$. We use a $\boldsymbol{SW}$ gate to exchange the states between two adjacent qubits, and multiple operations simulate moving non-adjacent qubits to adjacent positions. A $\boldsymbol{SW}$ gate can be implemented by three $\boldsymbol{CX}$ gates, or by inserting four $\boldsymbol{H}$ gates to change the direction of the middle $\boldsymbol{CX}$ gate, as shown in Fig. 2.

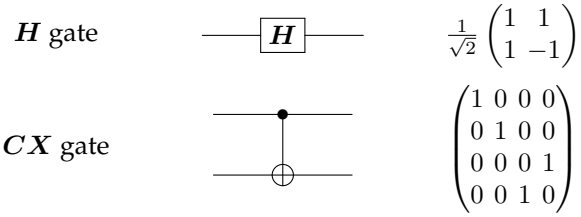


Fig.1. The symbols of two quantum gates and their matrices.

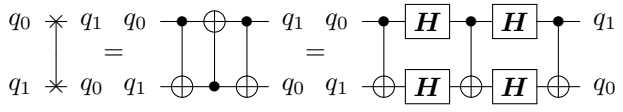


Fig.2. Implementing a $\boldsymbol{SW}$ gate by $\boldsymbol{CX}$ gates and $\boldsymbol{H}$ gates.

In a quantum circuit, each line represents a *wire*. The wire does not necessarily correspond to a physical wire but may correspond to the passage of time or a physical particle that moves from one location to another through space. The interested reader can find more details of these gates from the standard textbook[30]. The execution order of a quantum logical circuit is from left to right. The width of a circuit refers to the number of qubits in the circuit. The depth of a circuit refers to the number of layers executable in parallel. For example, the depth of the circuit in Fig. 3 (a) is 6, and the width is 5. We refer to a circuit with the number of 2-qubit gates no more than 100 as a small-scale circuit, a circuit with the number of 2-qubit gates more than 1000 as a large-scale circuit, and the rest are medium-scale circuits. It is unnecessary to consider quantum gates acting on single qubits since 1-qubit gates are *local*[22], which do not need to move the involved qubits for gate applications.

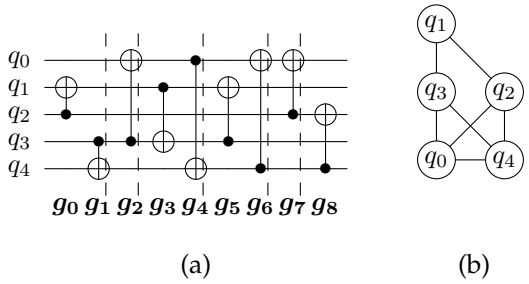


Fig.3. (a) The original quantum circuit. (b) The logical interaction graph of (a).

In the current work, we mainly consider the physical circuits of the IBM Q series, called coupling graphs. Let $\mathcal{CG} = (V_{\mathcal{C}}, E_{\mathcal{C}})$ denote the coupling graph of a physical device, where $V_{\mathcal{C}}$ is the set of physical qubits and $E_{\mathcal{C}}$ is the set of edges representing the connectivity between qubits related by $\boldsymbol{CX}$ gates. Fig. 4 (a)–(e) are the coupling graphs of the 5-qubit IBM QX2, IBM QX4, 16-qubit IBM QX3, and IBM QX5, the 20-qubit IBM Q Tokyo, respectively. The control of one qubit to a neighbor is unilateral, but for IBM Q Tokyo the control between two adjacent qubits is bilateral. The direction in each edge indicates the control direction

of each 2-qubit gate, and 2-qubit gates can only be performed between two adjacent qubits.

Given an interaction graph $\mathcal{IG}$, a coupling graph $\mathcal{CG}$, an initial mapping τ, and a $\boldsymbol{CX}$ gate $\boldsymbol{g} = \langle q_i, q_j \rangle$ where q_i is the control qubit and q_j is the target qubit, if the gate $\boldsymbol{g}$ is executable on coupling graph $\mathcal{CG}$, then $\langle \tau[q_i], \tau[q_j] \rangle$ should be a directed edge on $\mathcal{CG}$.

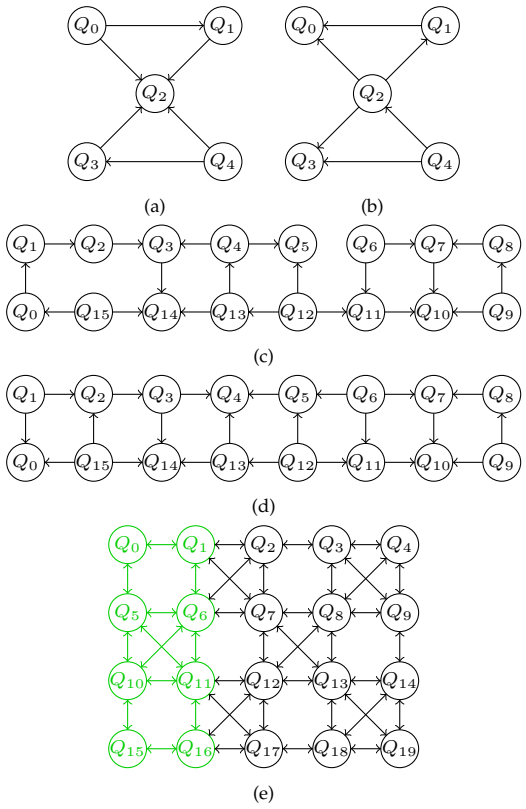


Fig.4. The coupling graphs of IBM Q series. (a) IBM QX2, (b) IBM QX4, (c) IBM QX3, (d) IBM QX5, (e) IBM Q Tokyo.

Example 3.1. *Consider the logical interaction graph $\mathcal{IG}$ and a coupling graph $\mathcal{CG}$ shown in Fig. 3 (b) and the green part of Fig. 4 (e). Let the initial mapping be as follows,*

$$\tau = \{q_0 \to Q_{10}, q_1 \to Q_0, q_2 \to Q_6, q_3 \to Q_5, q_4 \to Q_{11}\}.$$

Then the 2-qubit gate $\boldsymbol{g_0} = \langle q_2, q_1 \rangle$ is not executable, since the edge $\langle \tau[q_2], \tau[q_1] \rangle = \langle Q_6, Q_0 \rangle$ does not exist in $\mathcal{CG}$. However, the gate $\boldsymbol{g_1} = \langle q_3, q_4 \rangle$ is executable, since the edge $\langle \tau[q_3], \tau[q_4] \rangle = \langle Q_5, Q_{11} \rangle$ exists in $\mathcal{CG}$.

4 Qubit Mapping

Assume that the input circuit has only 1-qubit gates and $\boldsymbol{CX}$ gates[31, 32]. We expect to find a qubit mapping algorithm that, when given an input circuit, can produce an output circuit with a small number of additional gates in an acceptable amount of time. Roughly speaking, we propose a method of qubit mapping with the following three steps.

1. *Preprocessing.* This step includes extracting the interaction graph from the input circuit and calculating the shortest paths of the coupling graph.

2. *Isomorphism and completion.* This step first uses the subgraph isomorphism algorithm to find a set of partial initial mappings[7]. Then we perform a mapping completion to process the remaining nodes that do not satisfy all isomorphism requirements, according to the connectivity between the unmapped nodes and the mapped ones.

3. *Adjustment.* After the second step, some logically adjacent nodes may be mapped to physically non-adjacent nodes, therefore, the quantum circuit is not executable on the coupling graph. We use a tabu search-based adjustment algorithm to generate circuits that can be physically executed.

4.1 Preprocessing

In the preprocessing step, we adjust the input circuit described by an openQASM program[33] to extract the interaction graph from the input circuit and calculate the shortest paths of the coupling graph.

Quantum gates acting on different qubits can be executed in parallel. The notation $\mathcal{L}(\mathcal{C}) = \{\mathcal{L}_0, \mathcal{L}_1, ..., \mathcal{L}_n\}$ denotes the layered form of circuit $\mathcal{C}$, where $\mathcal{L}_i$ $(0 \leq i \leq n)$ stands for a set of quantum gates that can be executed in parallel. The quantum gate set separated by the dotted lines in Fig. 3 (a) are the follow-

ing: $\mathcal{L}_0 = \{\boldsymbol{g_0}, \boldsymbol{g_1}\}, \mathcal{L}_1 = \{\boldsymbol{g_2}\}, \mathcal{L}_2 = \{\boldsymbol{g_3}, \boldsymbol{g_4}\}, \mathcal{L}_3 = \{\boldsymbol{g_5}, \boldsymbol{g_6}\}, \mathcal{L}_4 = \{\boldsymbol{g_7}\}, \mathcal{L}_5 = \{\boldsymbol{g_8}\}$.

At the same time of layering, we generate an interaction graph $\mathcal{IG} = (V_\mathcal{I}, E_\mathcal{I})$, which is an undirected graph with $V_\mathcal{I}$ being the set of vertices, and $E_\mathcal{I}$ the set of undirected edges that denotes the connectivity between qubits related by $\boldsymbol{CX}$ gates. Given a coupling graph and assume the distance of each edge is 1, we use the Floyd-Warshall algorithm[34] to calculate the shortest distance matrix, with $\boldsymbol{D}[i][j]$ denoting the shortest distance from Q_i to Q_j.

Consider a $\boldsymbol{CX}$ gate $\boldsymbol{g} = \langle q_i, q_j \rangle$. If q_i and q_j are mapped to Q_m and Q_n, respectively, then the cost of executing $\boldsymbol{g}$ under the shortest path is denoted by $cost_{\boldsymbol{g}} = 7 \times (\boldsymbol{D}[m][n] - 1)$ on devices with unilateral control. For IBM Q Tokyo, the cost is $cost_{\boldsymbol{g}} = 3 \times (\boldsymbol{D}[m][n] - 1)$.

Example 4.1. *Consider the QX5 coupling graph (cf. Fig. 4 (d)). Given a $\boldsymbol{CX}$ gate $\boldsymbol{g} = \langle q_1, q_2 \rangle$, with q_1 mapped to Q_6 and q_2 mapped to Q_{13}, the shortest distance between them is $\boldsymbol{D}[6][13] = 3$. There are three shortest paths of moving from Q_6 to an adjacent position of Q_{13} : $\pi_0 = Q_6 \to Q_5 \to Q_4 \to Q_{13}$, $\pi_1 = Q_6 \to Q_5 \to Q_{12} \to Q_{13}$, $\pi_2 = Q_6 \to Q_{11} \to Q_{12} \to Q_{13}$. Their costs are given by $cost_{\pi_0} = 18$, $cost_{\pi_1} = 14$, and $cost_{\pi_2} = 14$, respectively. Here $cost_\pi$ stands for the cost of swapping the qubits q_i and q_j along the path π.*

4.2 Isomorphism and Completion

Generally speaking, in a coupling graph, it is almost impossible to find a subgraph that exactly matches the interaction graph. We regard the mapping with the largest number of mapped nodes as a good partial mapping. IMSM compares various compositions of several state-of-the-art subgraph isomorphism algorithms. Since IMSM cannot process disconnected graphs, we manually create connected graphs by linking isolated nodes to the ones with the largest degree in the interaction graph. Note that this does not change the architecture of the original circuit.

The input of Algorithm 1 is a coupling graph $\mathcal{CG}$, an interaction graph $\mathcal{IG}$, and a partial mappings set T. Line 2 selects the largest number n of mapped nodes, and the partial mappings with n mapped nodes are used by the candidate set. Lines 3–22 complete the partial mappings. The function $length(\tau)$ returns the size of τ. In Line 5, we initialize an empty queue I, which stores unmapped logical qubits, traverse the mapping τ and add the unmapped qubits to I. We then loop until I is empty, and all logical qubits are mapped to physical qubits. Line 7 takes out the first element in I to s. Lines 8 and 9 get the adjacency matrices of $\mathcal{CG}$ and $\mathcal{IG}$, respectively. Line 10 initializes a map C, sorted by a descending order of the degree of connectivity between s and u. Lines 11–21 traverse C and select the node u that has been mapped to the physical node t in the coupling graph and has the largest number of logical connections to s in C. Line 14 deletes the node from C. Lines 15–19 select the node k adjacent to t in the adjacency matrix, and map s to that node. Finally, we generate a dense mapping.

Example 4.2. *Consider the interaction graph shown in Fig 3 (a) and the coupling graph in Fig. 4 (e). Suppose we have a partial mapping set $T = \{\tau_0, \tau_1, ..., \tau_n\}$. We take one of the partial mappings as an example.*

$$\tau_0 = \{q_0 \to Q_{10}, q_1 \to -1, q_2 \to Q_6, q_3 \to Q_5, q_4 \to Q_{11}\},$$

where $q_1 \to -1$ means that q_1 is not mapped to any physical qubit, therefore we need the mapping completion algorithm. The maximum number of mapped nodes is 4. We demonstrate how τ_0 is completed. We add all unmapped nodes to the queue I; in this example we have $I = \{q_1\}$. Then we loop until I is empty. We pop the first element s of I, get the adjacency matrix of the query graph and the target graph, and traverse the adjacency matrix. We put the nodes u adjacent to s into the candidate nodes list C, which is sorted by the connectivity of s and u. We get $C = \{q_3, q_2, q_4, q_0\}$.

Next, we traverse C and take out the first element q_3 in C, and calculate the physical node $t = Q_5$ as $\tau_0[q_3] = Q_5$. Finally, we map s to the node connected to t but not yet mapped. In this example, it can be directly mapped to Q_0. In the end, we obtain the mapping

$\tau_0 = \{q_0 \rightarrow Q_{10}, q_1 \rightarrow Q_0, q_2 \rightarrow Q_6, q_3 \rightarrow Q_5, q_4 \rightarrow Q_{11}\}$.

Algorithm 1: Complete the initial mapping

Input: $\mathcal{CG}$: a coupling graph; $\mathcal{IG}$: an interaction graph; T: a partial mapping set obtained by IMSM;
Output: $results$: a set of mapping relations between $\mathcal{IG}$ and $\mathcal{CG}$;

Initialize $results = \emptyset$;
$n \leftarrow \max_{\tau \in T} |\{i : \tau[i] \neq -1,\ i \leq length(\tau),\ i \in \mathbb{N}\}|$
for $\tau \in T$ **do**
  if $n = length(\tau)$ **then**
    $I \leftarrow \{i : \tau[i] = -1,\ i \leq length(\tau),\ i \in \mathbb{N}\}$;
    while $I \neq \emptyset$ **do**
      $s \leftarrow I.poll()$;
      $\boldsymbol{P} \leftarrow \mathcal{CG}.adjacencyMatrix()$;
      $\boldsymbol{L} \leftarrow \mathcal{IG}.adjacencyMatrix()$;
      $C \leftarrow \{u : \boldsymbol{L}[s][u] \neq 0\}$;
      while $C \neq \emptyset$ **do**
        $t \leftarrow \tau[C[0]]$;
        $k \leftarrow 0$;
        $C \leftarrow C \backslash C[0]$;
        while $k < length(\boldsymbol{P}[t])$ **do**
          if ($\boldsymbol{P}[t][k]$ *or* $\boldsymbol{P}[k][t] \neq 0$ *and* $!\tau.contains(k)$) **then**
            $\tau[s] \leftarrow k$;
            break;
          $k \leftarrow k + 1$;
        if $k \neq length(\boldsymbol{P}[t])$ **then**
          break;
    $results.add(\tau)$;
return $results$;

4.3 Adjustment

4.3.1 Tabu search

Tabu search uses a tabu list to avoid searching repeated spaces and deadlock and amnesty rules to jump out of the local optima to ensure the diversity of transformed results. Our circuit adjustment mainly relies on the tabu search algorithm, aiming to adjust those large-scale circuits that the existing algorithms are difficult to process and generate a circuit closer to the optimal solution.

Algorithm 2: Calculate the candidate set

Input: s: the current node; P: the shortest paths set of coupling graph; $\boldsymbol{D}$: the distance matrix between nodes in the coupling graph; M_p: the mapping from physical qubits to logical qubits; M_l: the mapping from logical qubits to physical qubits; L: gates included in the current layer of circuits;
Output: $results$: the set of candidate mapping;

Initialize $results \leftarrow \emptyset$; $N \leftarrow \emptyset$;
foreach $\boldsymbol{g} \in L$ **do**
  if $\boldsymbol{g}$ *is executable* **then**
    $L \leftarrow L \backslash \{\boldsymbol{g}\}$;
  else
    $N \leftarrow N \cup \{\boldsymbol{g}.c, \boldsymbol{g}.t\}$;
foreach $\boldsymbol{g} \in L$ **do**
  foreach $p \in P[M_\mathrm{l}[\boldsymbol{g}.c]][M_\mathrm{l}[\boldsymbol{g}.t]]$ **do**
    foreach $e \in p$ **do**
      if $e.s$ *and* $e.t \notin N$ **then**
        continue;
      $M_\mathrm{p}^{'} \leftarrow M_\mathrm{p}$; $M_\mathrm{l}^{'} \leftarrow M_\mathrm{l}$;
      $q_1 \leftarrow M_\mathrm{p}^{'}[e.s]$; $q_2 \leftarrow M_\mathrm{p}^{'}[e.t]$;
      $M_\mathrm{p}^{'}[e.s] \leftarrow q_2$; $M_\mathrm{p}^{'}[e.t] \leftarrow q_1$;
      if $q_1 \neq -1$ **then**
        $M_\mathrm{l}^{'}[q_1] \leftarrow q_2$;
      if $q_2 \neq -1$ **then**
        $M_\mathrm{l}^{'}[q_2] \leftarrow q_1$;
      $s \leftarrow \emptyset$;
      $s.swaps \leftarrow s.swaps \cup \{e\}$;
      $s.value \leftarrow evaluate(\boldsymbol{D}, M_\mathrm{l}^{'}, L)$;
      $results \leftarrow results \cup \{s\}$;
return $results$;

The calculation of the candidate set is shown in Algorithm 2. The input M_p is a mapping from physical qubits to logical ones, where $j = M_\mathrm{p}[i]$ means that the i-th physical qubit is mapped to the j-th logical qubit. The set M_l denotes the mapping of logical qubits to physical ones, where $j = M_\mathrm{l}[i]$ means that the i-th logical qubit is mapped to the j-th physical qubit. The set

L includes all the gates in the current layer, and the output is a candidate mapping set of the current mapping. The set P (resp. $\boldsymbol{D}$) contains the edges (resp. distance) of all the shortest paths in the coupling graph. Lines 3–6 delete the gate $\boldsymbol{g}$ that can be executed in L under the current mapping and gather the control qubit $\boldsymbol{g}.c$ and target qubit $\boldsymbol{g}.t$ of gate $\boldsymbol{g}$ that cannot be executed into the set N. Lines 7–22 traverse gates $\boldsymbol{g}$ in L, and calculate the shortest paths between the nodes of $\boldsymbol{g}$. If the endpoints $e.s$ and $e.t$ of edge e intersect with N on the shortest path, then e is an element of the candidate set. Lines 12–18 update the mapping after the swap. Lines 18–20 generate a new candidate solution. Line 19 stores the swapped edges that will be used in the output circuit, and Line 21 calculates the swap scores using an evaluation function.

Example 4.3. *Let us consider the mapping*

$$\tau_0 = \{q_0 \to Q_{10}, q_1 \to Q_0, q_2 \to Q_6, q_3 \to Q_5, q_4 \to Q_{11}\},$$

with $\mathcal{L}_0 = \{\boldsymbol{g_0}, \boldsymbol{g_1}\}$, $cost_{\boldsymbol{g_1}} = 0$ *and* $cost_{\boldsymbol{g_0}} = 3$. *The gate* $\boldsymbol{g_1}$ *can be executed directly in the* τ_0 *mapping, therefore we delete it from* $\mathcal{L}_0$, *but* $\boldsymbol{g_0}$ *cannot be executed in the mapping* τ_0. *The nodes that cannot be executed join the set* $N = \{Q_0, Q_6\}$. *The set of shortest paths is*

$$P[6][0] = \{\{Q_6 \to Q_1 \to Q_0\}, \{Q_6 \to Q_5 \to Q_0\}\}.$$

We traverse the shortest paths and calculate the candidate set. The current candidate set is $\{(Q_6, Q_1), (Q_1, Q_0), (Q_6, Q_5), (Q_5, Q_0)\}$.

TSA takes a layered circuit and an initial mapping as input and outputs a circuit that can be executed in the specified coupling graph, as shown in Algorithm 3. The adjusted circuit mapping of each layer is used as the initial mapping of the next layer. Line 1 regards the initial mapping τ_{ini} as the best mapping τ_{best}. Lines 3–12 cyclically check whether all the gates in the current layer can be executed under the mapping τ_{best}. If all the gates are executable or the number of iterations reaches the given bound, the search is completed. Otherwise, the search continues. Line 4 gets the current mapping candidate, and Line 7 finds the best mapping in the candidate set. Note that if the edge swapped by a candidate appears in the tabu list, the candidate will be removed from the candidate set. Then from the remaining candidates, we choose a mapping with the lowest cost. Line 9 takes the amnesty rules. If the best candidate is not found, the amnesty rules will select the mapping with the lowest cost in the candidate set as the best mapping. Lines 10–12 update the best mapping τ_{best} and insert the swapped edge performed by the best mapping to the tabu list t_{l}. The motivation is to execute the generated circuit in parallel as much as possible and to avoid swapping the edges in the tabu list. Then it will check whether the termination condition of the algorithm is satisfied. The condition determines whether the number of iterations reaches the given bound, or the current mapping ensures all the gates in the current layer can be executed.

Algorithm 3: Tabu search

```
   Input: τ_ini: the initial mapping; t_l: tabu list;
   Output: τ_best: the best mapping;
1  Initialize τ_best ← τ_ini;
2  n ← 1; /*the number of iterations*/
3  while not mustStop(n, τ_best) do
4      C ← τ_best.candidates() /*candidate set*/
5      if C is empty then
6          break;
7      c_best ← best_candidates(C, t_l);
8      if c_best is empty then
9          c_best ← amnesty_candidates(C, t_l);
10     τ_best ← c_best;
11     t_l ← t_l ∪ {c_best.swap};
12     n ← n + 1;
13 return τ_best
```

4.3.2 Evaluation functions with look ahead

We propose three evaluation functions: one introduces CCA, one uses the number of additional gates in the generated circuit as an evaluation criterion as given in (1), and the last one uses the depth of the generated circuit as an evaluation criterion as given in (2).

They give rise to three variants of TSA called TSA_{cca}, TSA_{num}, and TSA_{dep}, respectively.

CCA has mainly been used for Boolean Satisfiability (SAT) problems. We apply the idea of CCA to adjust circuits. Let $submake$ represent the number of qubits for which two qubits are closer after a $\boldsymbol{SW}$ gate, and $subbreak$ represent the number of qubits for which two qubits are farther apart after a $\boldsymbol{SW}$ gate. We introduce $subscore = submake - subbreak$ into the evaluation function, and adjust the weight with the Smooth Weight based Threshold (SWT) scheme[14].

The output of the i-th layer, with i smaller than the depth of the circuit d, is used as the input of the $(i+1)$-th layer. Note that any $\boldsymbol{SW}$ gate in the i-th layer will affect the mapping of the $(i+1)$-th layer. If we only consider the gate of the current layer when selecting the $\boldsymbol{SW}$ gate, only the requirements of the i layer will be satisfied, not necessarily the next layer. Therefore, we take the gates from the i-th to the $(i+l_{\text{a}})$-th layer, with $i + l_{\text{a}} \leq d$, into consideration, where l_{a} is the number of look-ahead layers. It is necessary to give a higher priority to executing the gates in the i-th layer, therefore we introduce an attenuation factor δ to control the influence of the gates in the look-ahead layers.

$$cost_{\text{num}}(Q_m, Q_n) = \sum_{\boldsymbol{g} \in \mathcal{L}_i} 3 \times (\boldsymbol{D}[\tau[\boldsymbol{g}.c]][\tau[\boldsymbol{g}.t]] - 1) + \delta \times \left(\sum_{j=i}^{i+l_{\text{a}}} \sum_{\boldsymbol{g} \in \mathcal{L}_j} 3 \times (\boldsymbol{D}[\tau[\boldsymbol{g}.c]][\tau[\boldsymbol{g}.t]] - 1] \right), \quad (1)$$

$$cost_{\text{dep}}(Q_m, Q_n) = Depth \left(\bigcup_{j=i}^{i+l_{\text{a}}} \mathcal{L}_j \right). \quad (2)$$

Here $cost_{\text{num}}(Q_m, Q_n)$ (resp. $cost_{\text{dep}}(Q_m, Q_n)$) denotes the distance (resp. depth) of all the gates in layer $\mathcal{L}_j$ $(i \leq j \leq i + l_{\text{a}})$, after swapping the state of Q_m with that of Q_n. The notation $\boldsymbol{g}.c$ (resp. $\boldsymbol{g}.t$) stands for the control (resp. target) qubit of gate $\boldsymbol{g}$.

Example 4.4. *Let us continue the previous example. We select the one with the lowest evaluation score from the candidate set. Assuming* $\delta = 0.5$ *and* $l_{\text{a}} = 2$*, for* $\mathcal{L}_1 = \{\boldsymbol{g_2}, \boldsymbol{g_0}\}$*, the candidate set is* $\{(Q_6, Q_1), (Q_1, Q_0), (Q_6, Q_5), (Q_5, Q_0)\}$*, and the costs are given as follows:*

$$cost_{\text{num}}(Q_6, Q_1) = 0, \quad cost_{\text{num}}(Q_1, Q_0) = 1.5,$$
$$cost_{\text{num}}(Q_6, Q_5) = 1.5, \quad cost_{\text{num}}(Q_5, Q_0) = 1.5.$$

The algorithm chooses the first $\boldsymbol{SW}$ *with the smallest score, and the mapping becomes*

$$\tau_0 = \{q_0 \to Q_{10}, q_1 \to Q_0, q_2 \to Q_1, q_3 \to Q_5, q_4 \to Q_{11}\}.$$

It can be seen that the current mapping ensures that $\boldsymbol{g_0}$ *is executable. Thus we can continue to the next layer.*

4.3.3 Complexity

Given an interaction graph $\mathcal{IG} = (V_{\mathcal{I}}, E_{\mathcal{I}})$ and a coupling graph $\mathcal{CG} = (V_{\mathcal{C}}, E_{\mathcal{C}})$, we assume that the depth of the circuit is d and there are g gates in one layer. The candidate set consists of the edges connected to the control or target qubits, thus the size of the $\boldsymbol{SW}$ candidate set is $8 \times g$. The worst-case time complexity is $O(d \times g \times (8 \times g)^{(|E_{\mathcal{C}}|-1)})$, and the space complexity is $O(g)$.

5 Experiments

We compare TSA with several state-of-the-art algorithms for qubit mapping, namely ZPW[10], SABRE[11], FiDLS[12], and DLH[13]. Notice that other algorithms such as SAHS[26] and $t\,|ket\rangle$[25] are not listed because Li *et al.*[12] have been pointed out that FiDLS is superior to SAHS and the latter outperforms $t\,|ket\rangle$[26]. The implementation in Python is available online[3]. All the experiments are conducted on a Ubuntu machine with a 2.2GHz CPU and 64G memory. We take the logarithm $\log_{10}$ of both the *x*-axis and *y*-axis such that the experimental results are easy to observe. The time limit for each benchmark is one hour. Among the 159 benchmarks we have considered, 158

[3] https://github.com/Holly-Jiang/QCTSA

of them are taken from some functions of RevLib[35] and one is added by our own. This data set has also been adopted in several related work. We believe that it is representative and our comparative experiments are carried out on it. With the 159 benchmarks, we compare TSA with ZPW, SABRE, and FiDLS on IBM Q Tokyo, and with FiDLS on Sycamore. Since the code of DLH is not available online, we only compare with this algorithm on the number of inserted additional gates but not the running time. Note that SABRE uses a random initial mapping, thus for every benchmark we execute it five times, each with a different initial mapping, and report the best result out of the five trials. TSA uses the unsorted candidates, and thus we execute it five times and take the best result. Other algorithms are deterministic, therefore they only run once. Fig. 5 illustrates the entire process of our experiments. Below we go through it in more detail.

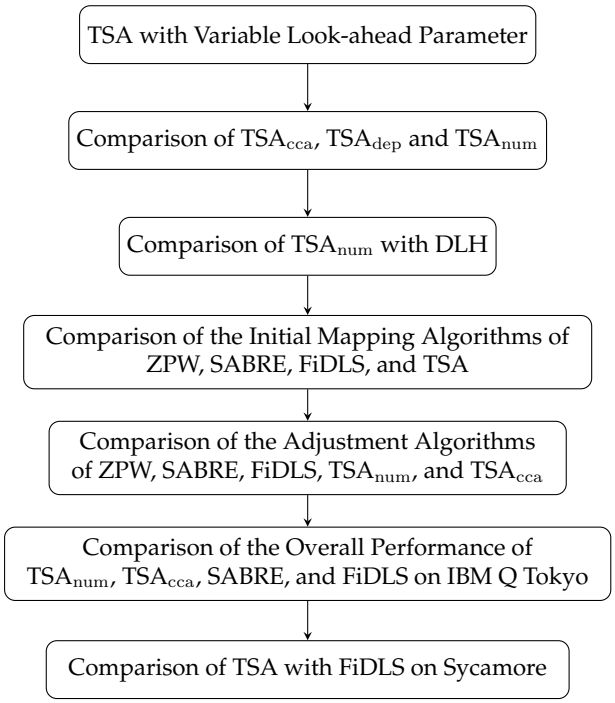


Fig.5. Sketch of the experiments.

Firstly, we test TSA with fixed and variable look-ahead parameter l_{a}. In Fig. 6, different colors represent the logarithms of the numbers of additional gates. The lower the points in the figure, the fewer the number of additional gates inserted. As to the look-ahead parameter l_{a}, the optimal parameter for each circuit may be different. We have done thousands of experiments and found that when $l_{\text{a}} = 2$, the number of additional gates is relatively small for all benchmarks. It means that a 2-layer look-ahead already gives a good performance for TSA.

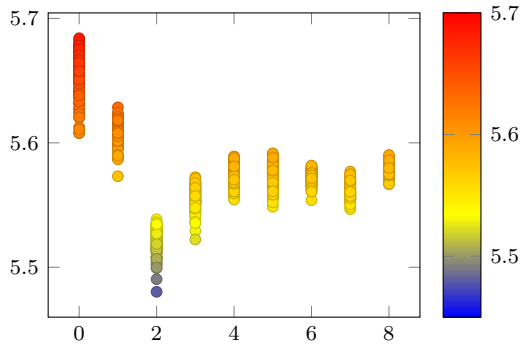


Fig.6. The impact of the look-ahead parameter l_{a} on search results. The x-axis represents l_{a}, the y-axis represents the number of additional gates.

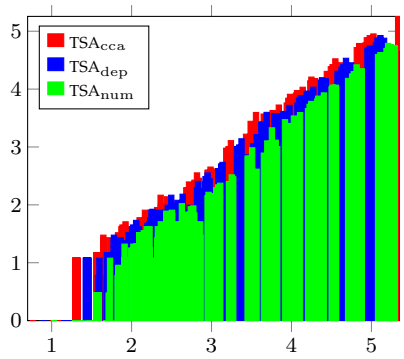


Fig.7. Comparison of the number of additional gates inserted by TSA$_{\text{dep}}$, TSA$_{\text{cca}}$, and TSA$_{\text{num}}$. The x-axis represents the number of 2-qubit gates in the benchmark, the y-axis represents the number of additional gates.

In Fig. 7, we compare TSA$_{\text{cca}}$, TSA$_{\text{dep}}$ and TSA$_{\text{num}}$ using the 159 benchmarks mentioned above. Compared with TSA$_{\text{cca}}$ (resp. TSA$_{\text{num}}$), the depth of the generated circuits by TSA$_{\text{dep}}$ is reduced by 2.37% (resp. 3.42%) on average. Compared with TSA$_{\text{cca}}$ (resp. TSA$_{\text{dep}}$), the number of additional gates by TSA$_{\text{num}}$ is reduced by 2.69% (resp. 9.56%) on average. Therefore, it is preferable to use either TSA$_{\text{dep}}$ or TSA$_{\text{num}}$, depending on the optimization objective to

Table 1. Comparison of MCPE, MCPE_OP and $\mathrm{TSA}_{\mathrm{num}}$

Benchmark	#circ.		MCPE	MCPE_OP	$\mathrm{TSA}_{\mathrm{num}}$	$\Delta_0(\%)$	$\Delta_1(\%)$
	n	g	g_0	g_1	g_2		
4mod5-v1_22	5	21	0	0	0	0	0
mod5mils_65	5	35	0	0	0	0	0
alu-v0_27	5	36	3	3	6	−100	−100
decod24-v2_43	4	52	0	0	0	0	0
4gt13_92	5	66	21	21	0	100	100
ising_model_10	16	786	0	0	0	0	0
ising_model_13	16	786	0	0	0	0	0
ising_model_16	16	786	0	0	0	0	0
qft_10	10	200	39	39	57	−46.15	−46.15
qft_16	16	512	225	192	189	16.00	1.56
rd84_142	15	343	153	108	99	35.29	8.33
adr4_197	13	3439	1566	1224	1029	34.29	15.93
radd_250	13	3213	1353	1047	852	37.03	18.62
z4_268	11	3073	1071	855	915	14.57	−7.02
sym6_145	14	3888	1017	1017	681	33.04	33.04
misex1_241	15	4813	2118	1098	1032	51.27	6.01
rd73_252	10	5321	2352	2193	1629	30.74	25.72
cycle10_2_110	12	6050	2226	1968	1890	15.09	3.96
square_root_7	15	7630	2061	1788	1509	26.78	15.60
sqn_258	10	4459	3708	3057	3093	16.59	−1.18
rd84_253	12	13658	6411	5697	4605	28.17	19.17
co14_215	15	17936	5634	5062	6813	−20.93	−34.59
sym9_193	10	34881	15420	13746	12315	20.14	10.41
urf5_158	9	164416	69852	58947	56253	19.47	4.57
hwb9_119	10	207775	93219	89355	78753	15.52	11.87
urf4_187	11	512064	220329	168366	141768	35.66	15.80
sum	–	1307223	428778	355782	311598	27.32	12.42

Note: n is the number of qubits, g is the number of gates in the input circuit, g_0–g_2 are the numbers of additional gates inserted by MCPE, MCPE_OP and $\mathrm{TSA}_{\mathrm{num}}$, respectively, and $\Delta_i = (g_i - g_2)/g_i$.

be either the depth or the number of additional gates of the resulting circuits.

Secondly, we use the benchmarks[13] to compare $\mathrm{TSA}_{\mathrm{num}}$ with DLH. Note that two heuristic cost functions MCPE and MCPE_OP are used in DLH. Since there is no code available online for DLH, we only compare the number of additional gates inserted with the circuits[13], as shown in Table 1. Compared with MCPE and MCPE_OP, $\mathrm{TSA}_{\mathrm{num}}$ reduces the total number of additional gates by 27.32% and 12.42%.

Thirdly, we compare the combinations of several algorithms for inserting fewer additional gates. We use the initial mapping and adjustment algorithms from ZPW[10], SABRE[11], FiDLS[12], and TSA.

In Table 2, we compare the performance of the four initial mapping algorithms from ZPW, SABRE, FiDLS, and TSA under the specific adjustment algorithms. For example, in the first row the adjustment algorithm is fixed to be that of ZPW; there are 115 circuits that all of the four initial mapping algorithms can successfully transform and we compare the number of additional gates. As we can see, the initial mapping algorithm of TSA performs best when used in conjunction with the five adjustment algorithms. It leads to a reduction of 41%, 30% and 37% of additional gates than the initial mapping algorithms of ZPW, SABRE, and FiDLS.

We then compare the five adjustment algorithms ZPW, SABRE, FiDLS, $\mathrm{TSA}_{\mathrm{num}}$, and $\mathrm{TSA}_{\mathrm{cca}}$ under spe-

Table 2. Comparison of the initial mapping algorithms ZPW, SABRE, FiDLS, and TSA

Algorithm	n	g	g_0	g_1	g_2	g_3	$\Delta_0(\%)$	$\Delta_1(\%)$	$\Delta_2(\%)$
ZPW	115	63666	29640	24951	27651	17412	41.26	30.22	37.03
SABRE	108	77790	28671	26079	26412	16068	43.96	38.39	39.16
FiDLS	120	209433	29484	28434	30195	25950	11.99	8.74	14.06
$\mathrm{TSA_{num}}$	120	163485	54969	52512	62817	45948	12.50	26.85	18.59
$\mathrm{TSA_{cca}}$	120	163485	57777	53922	61668	46305	19.86	14.13	24.91

Note: n is the number of circuits that all the four initial mapping algorithms can successfully transform, g is the number of gates in the input circuits, g_0–g_3 are the numbers of additional gates inserted by ZPW, SABRE, FiDLS, and TSA, respectively, and $\Delta_i = (g_i - g_3)/g_i$.

Table 3. Comparison of the adjustment algorithms ZPW, SABRE, FiDLS, $\mathrm{TSA_{num}}$, and $\mathrm{TSA_{cca}}$

Algorithm	n	g	g_0	g_1	g_2	g_3	g_4	$\Delta_0(\%)$	$\Delta_1(\%)$	$\Delta_2(\%)$	$\Delta_4(\%)$
ZPW	94	29443	14472	11244	4938	10173	10389	29.71	9.53	−106.01	2.08
SABRE	105	49987	19053	16632	6204	12072	11904	36.61	27.41	−94.58	−1.41
FiDLS	109	105428	45813	31011	16668	37800	37851	17.49	−21.89	−126.78	0.13
TSA	124	150464	49620	30447	19068	40461	40629	18.46	−32.89	−112.19	0.41

Note: n is the number of circuits that all the five adjustment algorithms can successfully transform, g is the number of gates in the input circuits, g_0–g_4 are the numbers of additional gates inserted by ZPW, SABRE, FiDLS, $\mathrm{TSA_{num}}$, and $\mathrm{TSA_{cca}}$ respectively, and $\Delta_i = (g_i - g_3)/g_i$.

Table 4. Comparison of the runtime and the number of circuits successfully transformed by SABRE, FiDLS, $\mathrm{TSA_{num}}$, $\mathrm{TSA_{cca}}$, respectively

Scale	n	g	SABRE			FiDLS			$\mathrm{TSA_{num}}$			$\mathrm{TSA_{cca}}$		
			n_0	g_0	t_0	n_1	g_1	t_1	n_2	g_2	t_2	n_3	g_3	t_3
small	66	5997	66	2301	2	66	1329	7	66	894	16	66	897	21
medium	49	21618	49	10218	22	49	5328	90	49	5199	57	49	5280	62
large	44	3289162	29	162522	12412	44	532485	63744	44	1013196	2392	44	1037427	2440
sum	159	3312734	144	175041	12436	159	539142	63841	159	1015521	2465	159	1043604	2523

Note: n is the number of test circuits, g is the number of gates in the input circuits, n_0–n_3 are the numbers of circuits successfully transformed by SABRE, FiDLS, $\mathrm{TSA_{num}}$, and $\mathrm{TSA_{cca}}$ respectively, t_0–t_3 are the runtime of SABRE, FiDLS, $\mathrm{TSA_{num}}$, and $\mathrm{TSA_{cca}}$, respectively in seconds, and g_0–g_3 are the numbers of additional gates inserted by SABRE, FiDLS, $\mathrm{TSA_{num}}$, and $\mathrm{TSA_{cca}}$, respectively.

Table 5. Comparing the initial mapping and adjustment algorithms of FiDLS and TSA on Sycamore

Algorithm	n	g	FiDLS		$\mathrm{TSA_{num}}$		$\mathrm{TSA_{cca}}$		$\Delta_1(\%)$	$\Delta_2(\%)$
			g_0	t_0	g_1	t_1	g_2	t_2		
FiDLS	159	3312734	2311560	31896	2245314	233	2257371	3392	2.86	2.56
TSA			2305125	31211	2234937	1795	2252271	3390	3.04	2.29

Note: n is the number of test circuits, g is the number of gates in the input circuits, g_0–g_2 are the numbers of additional gates inserted by FiDLS, $\mathrm{TSA_{num}}$ and $\mathrm{TSA_{cca}}$, respectively, t_0–t_2 are the runtime of FiDLS, $\mathrm{TSA_{num}}$, and $\mathrm{TSA_{cca}}$, respectively in seconds, and $\Delta_i = (g_0 - g_i)/g_0$.

cific initial mapping algorithms in Table 3. FiDLS gives rise to the fewest additional gates. For example, in the second row, SABRE is used as the adjustment algorithm, 16632 (resp. 12072) gates are inserted under the initial mapping of SABRE (resp. TSA). The SABRE adjustment algorithm combined with the initial mapping provided by TSA has fewer gates inserted than the SABRE initial mapping algorithm in those benchmarks. This shows that the initial mapping of TSA is better than that of SABRE. FiDLS uses a deep search on the circuits, calculates the full permutation of all edges, and then selects the best among all the permutations according to an evaluation function. FiDLS takes large-scale search space and long search time for large-

scale circuits. Overall, TSA performs well on large-scale circuits, trading off additional gates and runtime.

Fourthly, we compare the overall performance of TSA_{num} and TSA_{cca} with SABRE and FiDLS on IBM Q Tokyo. We test 159 circuits, including 66 small-scale circuits, 49 medium-scale circuits and 44 large-scale ones. Note that in Table 4 and Fig. 8 we do not display the data for ZPW. Instead, we compare with SABRE because it is already shown that SABRE is much more scalable than ZPW[11]. In Fig. 8, the number of additional gates introduced by the blue bars is the largest, followed by the red ones. We can see that the yellow bars are the shortest when the x-axis is greater than 3, indicating that FiDLS has inserted the fewest gates in the large-scale circuits. The green bars are for TSA_{num}. The number of additional gates it introduces is slightly larger than that of FiDLS. It can also be seen from Table 4 that TSA_{num} takes much less time than FiDLS in general. SABRE successfully transforms 144 circuits, including all the small-scale and medium-scale circuits, and 29 large-scale ones, which takes 12436 seconds. FiDLS successfully transforms 159 circuits, which takes 63841 seconds. TSA_{num} and TSA_{cca} are much faster, as they successfully transform all the 159 circuits, taking 2465 seconds and 2523 seconds, respectively. Compared with SABRE, the number of additional $\boldsymbol{SW}$ gates generated by TSA_{num} is reduced by 51% on average, among the 115 small-scale and medium-scale circuits that both of them can successfully transform.

In small scale (resp. middle scale) circuits, TSA_{num} generates an average of 33% (resp. 2%) fewer additional $\boldsymbol{SW}$ gates compared to FiDLS. Specifically, FiDLS inserts 1329 (resp. 5328) additional gates, while the number is 894 (resp. 5199) for TSA_{num}. When dealing with large-scale circuits, although TSA_{num} inserts more additional gates, it can convert large-scale circuits more than 25 times faster than FiDLS, as we can see in the fourth row and the t_2-column of Table 4.

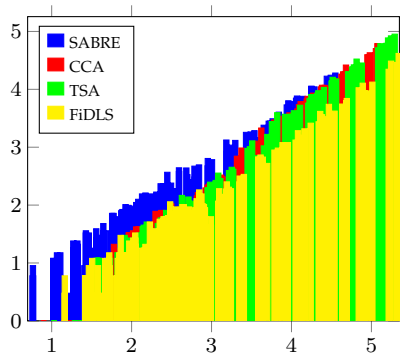


Fig.8. Comparison of SABRE, FiDLS, TSA_{num}, and TSA_{cca} on IBM Q Tokyo. The x-axis represents the number of 2-qubit gates in the benchmark, the y-axis represents the number of additional gates.

Finally, we set our target device to be the 53-qubit quantum processor Sycamore and compare TSA with FiDLS still on the 159 benchmarks. In each row of Table 5, the same initial mapping algorithm is used, and in each column, the same adjustment algorithm is used. Generally speaking, TSA leads to a reduction of 2%-3% for the number of inserted additional gates. In the experiment, we find that the degrees of the Sycamore nodes are small and the maximum is 4. If the degrees of nodes in the interaction graph are generally greater than the maximum degree of Sycamore, it is not very suitable to use subgraph isomorphism to generate the set of partial initial mappings. The algorithm tempts to first match the node with the largest degree. If the node with the maximum degree does not satisfy the isomorphism condition, the initial mapping generated by the subgraph isomorphism algorithm is not friendly. However, the adjustment of TSA is still very effective because the time cost is drastically lowered, going from 31896 seconds for FiDLS to 1795 seconds for TSA_{num}, that is, the latter is more than 17 times faster than the former.

6 Conclusions

We propose a scalable algorithm for qubit mapping. We first use a subgraph isomorphism algorithm and a mapping completion algorithm based on the connectivity between qubits to generate a high-quality

initial mapping. Then we employ a look-ahead heuristic search to adjust the mapping, which takes into account the influence of the gates yet to be processed to reduce the number of additional gates. We compared the performance of the initial mapping and adjustment algorithms with state-of-the-art algorithms ZPW, SABRE, and FiDLS, using the architectures of IBM Q Tokyo and Sycamore as target devices. Our experimental results show that the initial mapping of TSA gives rise to fewer $\boldsymbol{SW}$ gates inserted and the adjustment algorithm can be obtained in an acceptable amount of time. Most small-scale and medium-scale circuits can be transformed in a few seconds. For large-scale circuits, the results can be obtained within a few minutes. In the future, we will investigate how to reduce the number of additional gates inserted and increase the speed. We will also apply the proposed method to more NISQ devices.

References

[1] Shor P W. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[2] Grover L K. A fast quantum mechanical algorithm for database search. In *Proc. 28th Annual ACM Symposium on the Theory of Computing*, STOC '96, 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[3] Harrow A W, Hassidim A, Lloyd S. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 2009, 103:150502:1–150502:4. DOI: 10.1103/PhysRevLett.103.150502.

[4] Arute F, Arya K, al. Quantum supremacy using a programmable superconducting processor. *Nature*, 2019, 574:505–510. DOI: 10.1038/s41586-019-1666-5.

[5] Almudever C G, Lao L, Wille R, Guerreschi G G. Realizing quantum algorithms on real quantum computing devices. In *Proceedings of the 23rd Conference on Design, Automation and Test in Europe*, DATE '20, 2020, pp. 864–872. DOI: 10.23919/DATE48585.2020.9116240.

[6] Reagor M, Pfaff W, Axline C, Heeres R W, Ofek N, Sliwa K, Holland E, Wang C, Blumoff J, Chou K, Hatridge M J, Frunzio L, Devoret M H, Jiang L, Schoelkopf R J. Quantum memory with millisecond coherence in circuit qed. *Phys. Rev. B*, 2016, 94:014506:1–014506:8. DOI: 10.1103/PhysRevB.94.014506.

[7] Sun S, Luo Q. In-memory subgraph matching: An in-depth study. In *Proceedings of the 2020 International Conference on Management of Data*, SIGMOD '20, 2020, pp. 1083–1098. DOI: 10.1145/3318464.3380581.

[8] Siraichi M Y, Santos V F, Collange S, Pereira F M Q. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, CGO 2018, 2018, pp. 113–125. DOI: 10.1145/3168822.

[9] Glover F W. Tabu search - part II. *Informs Journal on Computing*, 1990, 2(1):4–32. DOI: 10.1287/ijoc.2.1.4.

[10] Zulehner A, Paler A, Wille R. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019, 38(7):1226–1236. DOI: 10.1109/TCAD.2018.2846658.

[11] Li G, Ding Y, Xie Y. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, 2019, pp. 1001–1014. DOI: 10.1145/3297858.3304023.

[12] Li S, Zhou X, Feng Y. Qubit mapping based on subgraph isomorphism and filtered depth-limited search. *IEEE Transactions on Computers*, 2021, 70(11):1777–1788. DOI: 10.1109/TC.2020.3023247.

[13] Zhu P, Guan Z, Cheng X. A dynamic look-ahead heuristic for the qubit mapping problem of NISQ computers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020, 39(12):4721–4735. DOI: 10.1109/TCAD.2020.2970594.

[14] Cai S, Su K. Local search for boolean satisfiability with configuration checking and subscore. *Artificial Intelligence*, 2013, 204(9):75–98. DOI: 10.1016/j.artint.2013.09.001.

[15] Paler A. On the influence of initial qubit placement during nisq circuit compilation. In Feld S, Linnhoff-Popien C, editors, *Quantum Technology and Optimization Problems*, 2019, pp. 207–217. DOI: 10.1007/978-3-030-14082-3_18.

[16] Kissinger A, Griend A M. CNOT circuit extraction for topologically-constrained quantum memories. *Quantum Inf. Comput.*, 2020, 20(7&8):581–596. DOI: 10.26421/QIC20.7-8-4.

[17] Nash B, Gheorghiu V, Mosca M. Quantum circuit optimizations for NISQ architectures. *Quantum Science and Technology*, 2020, 5(2):025010:1–025010:14. DOI: 10.1088/2058-9565/ab79b1.

[18] Venturelli D, Do M, Rieffel E G, Frank J. Temporal planning for compilation of quantum approximate optimization circuits. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 2017, pp. 4440–4446. DOI: 10.1007/978-3-030-14082-3_18.

[19] Bernal D E, Booth K E C, Dridi R, Alghassi H, Tayur S R, Venturelli D. Integer programming techniques for minor-embedding in quantum annealers. In *Proceedings of the 17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2020, pp. 112–129. DOI: 10.1007/978-3-030-58942-4_8.

[20] Almeida A A A, Dueck G W, Silva A C R. Finding optimal qubit permutations for IBM's quantum computer architectures. In *Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design*, SBCCI '19, 2019, pp. 13:1–13:6. DOI: 10.1145/3338852.3339829.

[21] Wille R, Burgholzer L, Zulehner A. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, 2019, pp. 142:1–142:6. DOI: 10.1145/3316781.3317859.

[22] Shafaei A, Saeedi M, Pedram M. Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 41:1–41:6. DOI: 10.1145/2463209.2488785.

[23] Guerreschi G G, Park J. Two-step approach to scheduling quantum circuits. *Quantum Science and Technology*, 2018, 3(4):045003:1–045003:15. DOI: 10.1088/2058-9565/aacf0b.

[24] Matsuo A, Yamashita S. An efficient method for quantum circuit placement problem on a 2-d grid. In *Proceedings of the 11th International Conference on Reversible Computation*, volume 11497, 2019, pp. 162–168. DOI: 10.1007/978-3-030-21500-2_10.

[25] Cowtan A, Dilkes S, Duncan R, Krajenbrink A, Simmons W, Sivarajah S. On the qubit routing problem. In *Proceedings of the 14th Conference on the Theory of Quantum Computation, Communication and Cryptography*, volume 135 of *LIPIcs*, 2019, pp. 5:1–5:32. DOI: 10.1007/978-3-030-14082-3_18.

[26] Zhou X, Li S, Feng Y. Quantum circuit transformation based on simulated annealing and heuristic search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020, 39(12):4683–4694. DOI: 10.1109/TCAD.2020.2969647.

[27] Tannu S S, Qureshi M K. Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, 2019, pp. 987–999. DOI: 10.1145/3297858.3304007.

[28] Lao L, Someren H, Ashraf I, Almudever C G. Timing and resource-aware mapping of quantum circuits to superconducting processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021, 41(2):359–371. DOI: 10.1109/TCAD.2021.3057583.

[29] Gian Giacomo Guerreschi. Scheduler of quantum circuits based on dynamical pattern improvement and its application to hardware design. arXiv:1912.00035, 2019. https://arxiv.org/abs/1912.00035, Mar. 2022.

[30] Nielsen M A, Chuang I L. *Quantum Computation and Quantum Information: 10th Anniversary edition*. Cambridge University Press, 10th edition, 2011.

[31] Barenco A, Bennett C, Cleve R, DiVincenzo D, Margolus N, Shor P, Sleator T, Smolin J, Weinfurter H. Elementary gates for quantum computation. *Physical Review A*, 1995, 52(5):3457–3467. DOI: 10.1103/PhysRevA.52.3457.

[32] Mottonen M, Vartiainen J J. Decompositions of general quantum gates. *Frontiers in Artificial Intelligence and Applications*, 2005, 57(8):1263–1270. DOI: 10.1103/PhysRevLett.93.130502.

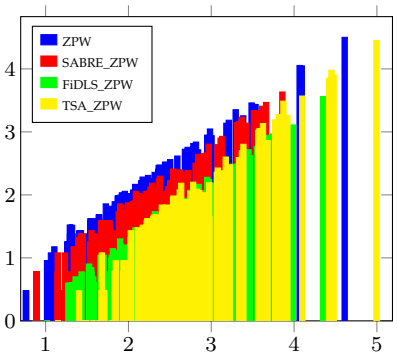
[33] Cross A, Javadi-Abhari A, Alexander T, Beaudrap N D, Bishop L S, Heidel S, Ryan C A, Sivarajah P, Smolin J, Gambetta J M, Johnson B R. OpenQASM 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, 2022, 3(3):12:1–12:50. DOI: 10.1145/3505636.

[34] Floyd R W. Algorithm 97: Shortest path. *Commun. ACM*, 1962, 5(6):344–348. DOI: 10.1145/367766.368168.

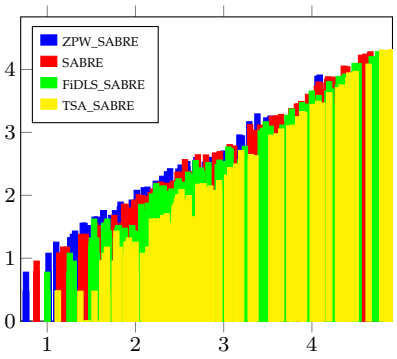
[35] Wille R, Groe D, Teuber L, Dueck G W, Drechsler R. Revlib: An online resource for reversible functions and reversible circuits. In *Proceedings of the 38th International Symposium on Multiple Valued Logic*, ISMVL '08, 2008, pp. 220–225. DOI: 10.1109/ISMVL.2008.43.

Appendix A Combinations of Initial Mapping and Adjustment Algorithms

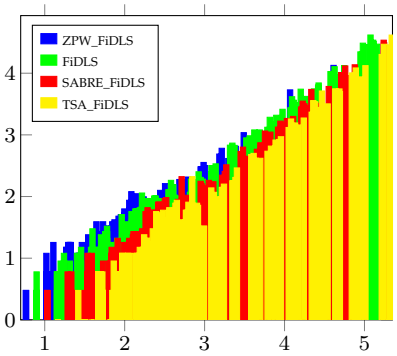
In order to visualize the difference between the four initial mapping algorithms from ZPW, FiDLS, SABRE, and TSA, we plot the data in Table 2. The five rows in the table correspond to Fig.A. 1 (a)–(e). The lower the y-axis of the bar graph means the fewer additional gates are inserted. The smaller the range on the x-axis means the smaller the circuit scale it can handle. In the figures, the yellow (resp. blue) color indicates the performance of TSA (resp. ZPW) as the initial mapping. In terms of different adjustment algorithms, the initial mapping algorithms of SABRE (red) and FiDLS (green) are not absolutely good or bad, but overall, the initial mapping of TSA has the fewest number of additional gates inserted in each adjustment algorithm, and the number of additional gates inserted in the initial mapping of ZPW is the most.



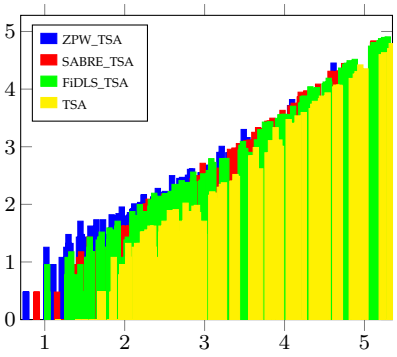
(a)



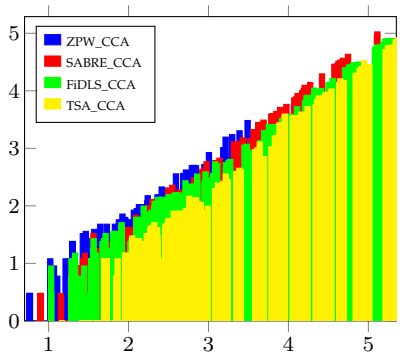
(b)



(c)



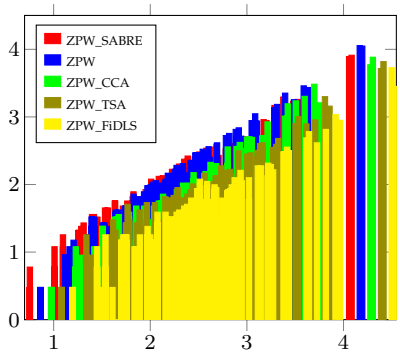
(d)



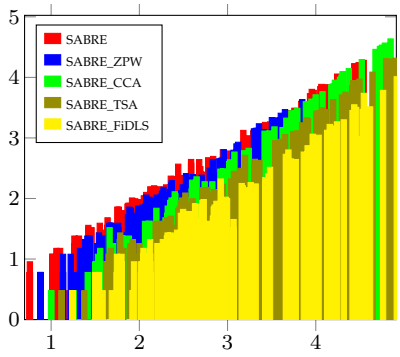
(e)

Fig.A.1. Comparison of the initial mapping algorithms of ZPW, SABRE, FiDLS, and TSA, using the adjustment algorithm of (a) ZPW, (b) SABRE, (c) FiDLS, (d) $\mathrm{TSA}_{\mathrm{num}}$, and (e) $\mathrm{TSA}_{\mathrm{cca}}$, respectively. The x-axis represents the number of 2-qubit gates in the benchmark, the y-axis represents the number of additional gates.

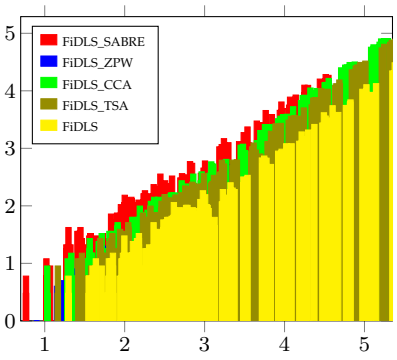
We then compare the five adjustment algorithms ZPW, SABRE, FiDLS, $\mathrm{TSA}_{\mathrm{num}}$, and $\mathrm{TSA}_{\mathrm{cca}}$ under specific initial mapping algorithms. The four rows in Table 3 correspond to Fig.A. 2 (a)–(d). As we can see, yellow bars (FiDLS) are the lowest. Red bars (SABRE) are the highest. The second-lowest are olive bars ($\mathrm{TSA}_{\mathrm{num}}$), which can handle all benchmarks. $\mathrm{TSA}_{\mathrm{num}}$ has fewer gates and shorter running time than ZPW in all initial mapping algorithms.



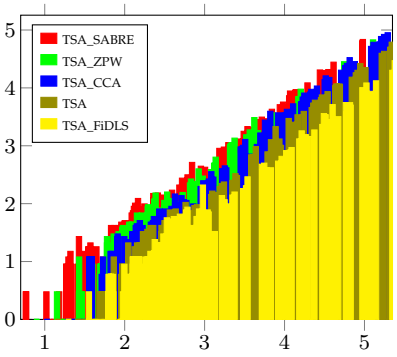
(a)



(b)



(c)



(d)

Fig.A.2. Comparison of the adjustment algorithms of ZPW, SABRE, FiDLS, TSA_{num}, and TSA_{cca}, using the initial mapping algorithm of (a) ZPW, (b) SABRE, (c) FiDLS, and (d) TSA, respectively. The x-axis represents the number of 2-qubit gates in the benchmark, the y-axis represents the number of additional gates.

Hui Jiang received her B.Eng. degree in computer science and technology from Sichuan Agriculture University, Ya'an, in 2019. She is currently a Ph.D. candidate at Shanghai Key Laboratory of Trustworthy Computing, School of Software Engineering, East China Normal University, Shanghai. Her research interests include quantum circuit compilation and optimization.

Yu-Xin Deng received his B.Eng. degree in thermal energy engineering and M.Sc. degree in computer science from Shanghai Jiao Tong University, Shanghai, in 1999 and 2002, respectively, and his Ph.D. degree in computer science from Ecole des Mines de Paris, Paris, in 2005. He is a professor at East China Normal University, Shanghai. His research interests include concurrency theory, especially about process calculi, and formal semantics of programming languages, as well as quantum computing. He authored the book titled Semantics of Probabilistic Processes: An Operational Approach (Springer, 2015).

Ming Xu received his Ph.D. degree in system sciences from East China Normal University (ECNU), Shanghai, in 2010. He is currently an associate research professor at Shanghai Key Laboratory of Trustworthy Computing, School of Software Engineering, ECNU. His research interests include computer algebra, program verification and quantum computing.