Jiang H, Deng YX, Xu M. Qubit mapping based on tabu search. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 39(2): 421-433 Mar. 2024. DOI: 10.1007/s11390-023-2121-5

Qubit Mapping Based on Tabu Search

Hui Jiang (蒋　慧), Yu-Xin Deng* (邓玉欣), *Senior Member, CCF*, and Ming Xu (徐　鸣)

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

E-mail: 52215902019@stu.ecnu.edu.cn; yxdeng@sei.ecnu.edu.cn; mxu@cs.ecnu.edu.cn

Received December 29, 2021; accepted August 31, 2023.

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 is making use of a combined subgraph isomorphism and completion to initialize some candidate mappings, and the other is dynamically modifying the mappings by TSA. Our experiments show that, compared with state-of-the-art methods, TSA can generate mappings with a smaller number of additional gates and have better scalability for large-scale circuits.

Keywords quantum computing, qubit mapping, initial mapping, tabu search, logical circuit

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 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 Experience① provides the public with free quantum computing resources on the cloud and Qiskit②, 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 **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 coherent state for a very short time. The longest coherence time of a superconducting quantum chip is still within 10 μs–100 μs[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.

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 of East China Normal University under Grant No. 2020ECNU-XFZH005, the Fundamental Research Funds for the Central Universities of China 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

①https://www.ibm.com/quantum-computing/, Mar. 2024.

②https://www.qiskit.org/, Mar. 2024.

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 optimum and ensure the diversity of the transformed results. We insert **SWAP** gates, 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 BidiREctional 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 an **SWAP** 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 reports the experimental results. We conclude in the last section and discuss 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 so efficient as a compiler designed specifically for this task.

• *Converting into 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 optimum 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.* 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.* used the minimum linear permutation solution in graph theory to model the problem of reducing the interaction distance[22]. A two-step method was 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 **SWAP** gates as possible[8, 10–13, 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 **SWAP** 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 this method in Section 5. A variation-aware qubit movement strategy[27] was 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.* showed 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 Preliminaries

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 quantum information is stored in qubits. Besides two basic states $|0\rangle$ and $|1\rangle$, a qubit can be in any linear superposition state like $|\phi\rangle = a\,|0\rangle + b\,|1\rangle$, where $a, b \in \mathbb{C}$ satisfy the condition $|a|^2 + |b|^2 = 1$. The intuition is that $|\phi\rangle$ is in the state $|0\rangle$ with probability $|a|^2$ and in the state $|1\rangle$ 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 (**H**) gate is applied on a qubit, and the **CX** gate is applied on two qubits. Their symbols and matrix forms are shown in Fig.1. The **H** gate turns state $|0\rangle$ (resp. $|1\rangle$) into $(|0\rangle + |1\rangle)/\sqrt{2}$ (resp. $(|0\rangle - |1\rangle)/\sqrt{2}$). The **CX** gate is a generalization of the classical XOR gate, since the action of the gate may be summarized as $|\boldsymbol{A}, \boldsymbol{B}\rangle \to |\boldsymbol{A}, \boldsymbol{B} \oplus \boldsymbol{A}\rangle$, where $\oplus$ is addition modulo two, which is exactly what the XOR gate does. That is, the control qubit and the target qubit are XORed 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 an **SWAP** gate to exchange the states between two adjacent qubits, and multiple operations simulate moving non-adjacent qubits to adjacent positions. An **SWAP** gate can be implemented by three **CX** gates, or by inserting four **H** gates to change the direction of the middle **CX** gate, as illustrated in Fig.2.

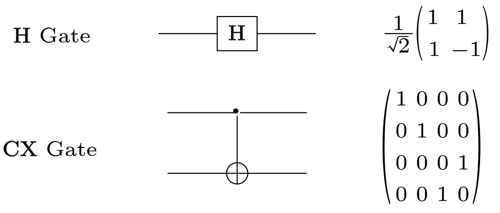

H Gate
H
$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
CX Gate
$\begin{pmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{pmatrix}$

Fig.1. Symbols of two quantum gates and their matrices.

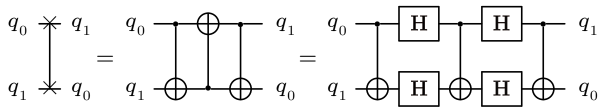
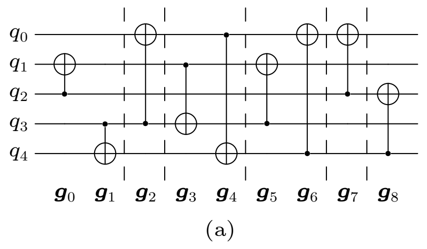


Fig.2. Implementing an **SWAP** gate by **CX** gates and **H** gates.

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 1 000 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.



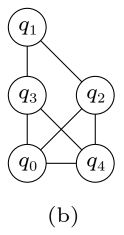


Fig.3. (a) Original quantum circuit. (b) Logical interaction graph of (a).

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 **CX** gates. Figs.4(a)–4(e) are the coupling graphs of the 5-qubit IBM QX2, IBM QX4, 16-qubit IBM QX3, IBM QX5, and 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.

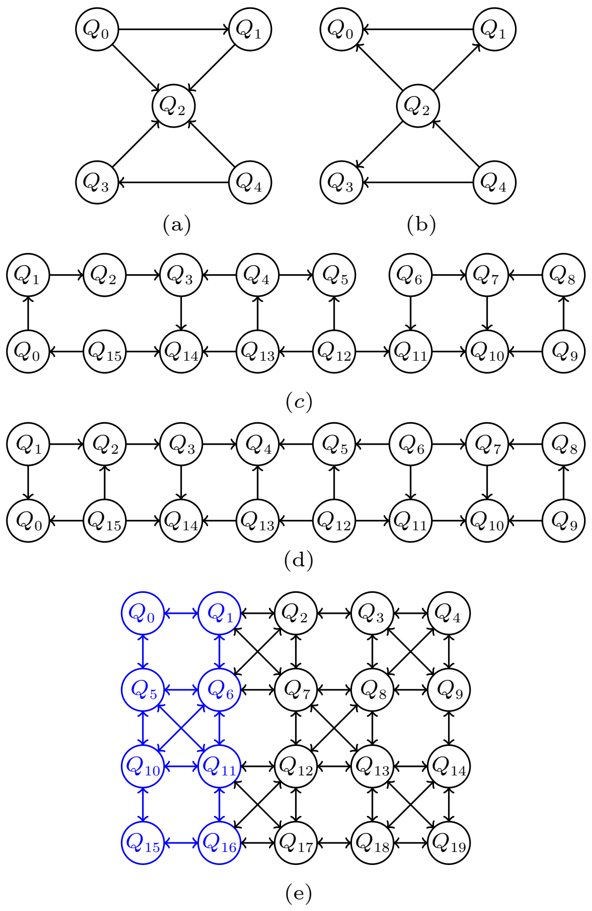


Fig.4. Coupling graphs of IBM Q series. (a) IBM QX2. (b) IBM QX4. (c) IBM QX3. (d) IBM QX5. (e) IBM Q Tokyo.

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

Example 1. Consider the logical interaction graph $\mathcal{IG}$ and a coupling graph $\mathcal{CG}$ shown in Fig.3(b) and the blue 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 **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.* First, the subgraph isomorphism algorithm is used 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 $\mathcal{L}(\mathcal{C}) = \{\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_n\}$ denotes the layered form of circuit $\mathcal{C}$, where $\mathcal{L}_i$ $(0 \leqslant i \leqslant n)$ stands for a set of quantum gates that can be executed in parallel. The quantum gate sets separated by the dotted lines in Fig.3(a) are the following: $\mathcal{L}_0 = \{\boldsymbol{g}_0, \boldsymbol{g}_1\}, \mathcal{L}_1 = \{\boldsymbol{g}_2\}, \mathcal{L}_2 = \{\boldsymbol{g}_3, \boldsymbol{g}_4\}, \mathcal{L}_3 = \{\boldsymbol{g}_5, \boldsymbol{g}_6\}, \mathcal{L}_4 = \{\boldsymbol{g}_7\}, \mathcal{L}_5 = \{\boldsymbol{g}_8\}$.

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 **CX** gates. Given a coupling graph and assuming that the distance of each edge is 1, we use the Floyd-Warshall algorithm[34] to calculate the shortest distance matrix, with $D[Q_{\rm c}][Q_{\rm t}]$ denoting the shortest distance from $Q_{\rm c}$ to $Q_{\rm t}$.

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

Example 2. Consider the QX5 coupling graph (cf. Fig.4(d)). Given a **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 $D[Q_6][Q_{13}] = 3$. 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 $cost_{\pi_i}$ for $0 \leqslant i \leqslant 2$ stands for the cost of swapping the qubits Q_6 to Q_{13} along the path π_i.

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 l of mapped nodes, and the partial mappings with l mapped nodes are used by the candidate set. Lines 3–22 complete the partial mappings. The function $len(\tau)$ returns the size of τ. In line 5, we initialize an empty queue V, which stores unmapped logical qubits, traverse the mapping τ, and add the unmapped qubits to V. We then loop until V is empty, and all logical qubits are mapped to physical qubits. Line 7 takes out the first element in V to q. Line 8 gets the adjacency matrix of $\mathcal{CG}$. Line 9 initializes a list U, sorted by descending degree of connectivity to q. Lines 10–20 traverse U and select the node $U[0]$ that has been mapped to the physical node Q in the coupling graph and has the largest number of logical connections to q in U. Line 13 deletes the node $U[0]$ from U. Lines 15–17 select the node k adjacent to Q in the adjacency matrix and map q to that node. Finally, we generate a dense mapping.

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 $\mathcal{CG}$;
1  Initialize $results = \emptyset$;
2  $l \leftarrow \max_{\tau \in T} |\{i : \tau[i] \neq -1,\ i \leqslant len(\tau),\ i \in \mathbb{N}\}|$
3  for $\tau \in T$ do
4      if $l = len(\tau)$ then
5          $V \leftarrow$ the unmapped logical qubits in $\tau$;
6          while $len(V) > 0$ do
7              $q \leftarrow V$.poll();
8              $\boldsymbol{P} \leftarrow \mathcal{CG}.adjacencyMatrix()$;
9              $U \leftarrow$ the neighbors of $q$ in $\mathcal{IG}$;
10             while $len(U) > 0$ do
11                 $Q \leftarrow \tau[U[0]]$;
12                 $k \leftarrow 0$;
13                 $U \leftarrow U \backslash U[0]$;
14                 while $k < len(\boldsymbol{P}[Q])$ do
15                     if $(\boldsymbol{P}[Q][k]$ or $\boldsymbol{P}[k][Q] \neq 0$ and not $\tau.contains(k))$
                       then
16                         $\tau[q] \leftarrow k$;
17                         break;
18                     $k \leftarrow k + 1$;
19                 if $k \neq len(\boldsymbol{P}[Q])$ then
20                     break;
21         $results$.add($\tau$);
22 return $results$;
```

Example 3. Consider the interaction graph shown in Fig.3(a) and the coupling graph in Fig.4(e). Suppose we have a partial mapping set $T = \{\tau_0, \tau_1, \ldots, \tau_n\}$. 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 V; in this example, we have $V = \{q_1\}$. Then we loop until V is empty. We pop the first element q_1 of V, get the adjacency matrix of the target graph, and the candidate nodes list $U = \{q_3, q_2, q_4, q_0\}$. Next, we traverse U and take out the first element q_3 in U, and calculate the physical node Q_5 as $\tau_0[q_3] = Q_5$. Finally, we map q_1 to the node connected to q_3 but not yet mapped. In this example, it can be directly mapped to Q_0. In the end, we obtain the mapping $\tau_0 = \{q_0 \to Q_{10}, q_1 \to Q_0, q_2 \to Q_6, q_3 \to Q_5, q_4 \to Q_{11}\}$.

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 optimum 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.

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 $\mathcal{L}$ includes all the gates in the current layer, and the output is a candidate mapping set of the current mapping. The set E and the matrix $\boldsymbol{D}$ contain the shortest paths and distances of all nodes in the coupled graph, respectively. Lines 3–7 delete the gate $\boldsymbol{g}$ that can be executed in $\mathcal{L}$ under the current mapping and gather the control qubit q_c and target qubit q_t of gate $\boldsymbol{g}$ that cannot be executed into the set B. Lines 8–24 traverse gates $\boldsymbol{g}$ in $\mathcal{L}$, and calculate the shortest paths between the nodes of $\boldsymbol{g}$. If the endpoints $e.s$ and $e.t$ of edge e intersect with B on the shortest path, then e is an element of the candidate set. Lines 14–20 update the mapping after the swap. Lines 21–24 generate a new candidate solution. Line 22 stores the swapped edges that will be used in the output circuit, and line 23 calculates the swap scores using an evaluation function.

Algorithm 2. Calculate the Candidate Set

```
Input: $M_\mathrm{p}$: the mapping from physical qubits to logical qubits; $M_\mathrm{l}$: the mapping from logical qubits to physical qubits; $\mathcal{L}$: gates included in the current layer of circuits; $E$: the shortest paths set of the coupling graph; $\boldsymbol{D}$: the distance matrix between nodes in the coupling graph;
Output: $results$: the set of candidate mapping;
1  Initialize $results \leftarrow \emptyset$; $B \leftarrow \emptyset$;
2  for $\boldsymbol{g} \in \mathcal{L}$ do
3      if $\boldsymbol{g}$ is executable then
4          $\mathcal{L} \leftarrow \mathcal{L} \backslash \{\boldsymbol{g}\}$;
5      else
6          $q_\mathrm{c}, q_\mathrm{t} \leftarrow$ the operating qubits of gate $\boldsymbol{g}$;
7          $B \leftarrow B \cup \{q_\mathrm{c}, q_\mathrm{t}\}$;
8  for $\boldsymbol{g} \in \mathcal{L}$ do
9      $q_\mathrm{c}, q_\mathrm{t} \leftarrow$ the operating qubits of gate $\boldsymbol{g}$;
10     for $p \in E[M_\mathrm{l}[q_\mathrm{c}]][M_\mathrm{l}[q_\mathrm{t}]]$ do
11         for $e \in p$ do
12             if $e.s$ and $e.t \notin B$ then
13                 continue;
14             $M'_\mathrm{p} \leftarrow M_\mathrm{p}$; $M'_\mathrm{l} \leftarrow M_\mathrm{l}$;
15             $Q_1 \leftarrow M'_\mathrm{p}[e.s]$; $Q_2 \leftarrow M'_\mathrm{p}[e.t]$;
16             $M'_\mathrm{p}[e.s] \leftarrow Q_2$; $M'_\mathrm{p}[e.t] \leftarrow Q_1$;
17             if $Q_1 \neq -1$ then
18                 $M'_\mathrm{l}[Q_1] \leftarrow Q_2$;
19             if $Q_2 \neq -1$ then
20                 $M'_\mathrm{l}[Q_2] \leftarrow Q_1$;
21             $s \leftarrow \emptyset$;
22             $s.swaps \leftarrow s.swaps \cup \{e\}$;
23             $s.value \leftarrow evaluate(\boldsymbol{D}, M'_\mathrm{l}, \mathcal{L})$;
24             $results \leftarrow results \cup \{s\}$;
25 return $results$;
```

Example 4. 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 $B = \{q_2, q_1\}$. The set of shortest paths is

$$\{\{Q_6 \to Q_1 \to Q_0\}, \{Q_6 \to Q_5 \to Q_0\}\}.$$

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 tb. 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: $\tau_{\text{ini}}$: the initial mapping; $tb$: the tabu list;
  Output: $\tau_{\text{best}}$: the best mapping;
1 Initialize $\tau_{\text{best}} \leftarrow \tau_{\text{ini}}$;
2 $it \leftarrow 1$; /*the number of iterations*/
3 while not $mustStop(it, \tau_{\text{best}})$ do
4     $C \leftarrow \tau_{\text{best}}.candidates()$ /*candidate set*/
5     if $C$ is empty then
6         break;
7     $c_{\text{best}} \leftarrow best\_candidates(C, tb)$;
8     if $c_{\text{best}}$ is empty then
9         $c_{\text{best}} \leftarrow amnesty\_candidates(C, tb)$;
10    $\tau_{\text{best}} \leftarrow c_{\text{best}}$;
11    $tb \leftarrow tb$.add $(c_{\text{best}}.swaps)$;
12    $n \leftarrow n + 1$;
13 return $\tau_{\text{best}}$;
```

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$_{\text{cca}}$, TSA$_{\text{num}}$, and TSA$_{\text{dep}}$, 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 an **SWAP** gate, and *subbreak* represent the number of qubits for which two qubits are farther apart after an **SWAP** 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 **SWAP** 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 **SWAP** gate, only the requirements of the i layer will be satisfied, not necessarily those of the next layer. Therefore, we take the gates from the i-th to the $(i+l_{\text{a}})$-th layer, with $i + l_{\text{a}} \leqslant d$, 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.

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

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

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

Example 5. 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 **SWAP** with the smallest score, and the mapping becomes $\tau_0 = \{q_0 \rightarrow Q_{10}, q_1 \rightarrow Q_0, q_2 \rightarrow Q_1, q_3 \rightarrow Q_5, q_4 \rightarrow Q_{11}\}$.

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 2-qubit gates in one layer. The candidate set consists of the edges connected to the control or target qubits, thus the size of the **SWAP** 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⟩[25] are not listed because Li *et al.*[12] have pointed out that FiDLS is superior to SAHS and the latter outperforms t|ket⟩[26]. The implementation in Python is available online③. All the experiments are conducted on a Ubuntu machine with a 2.2 GHz CPU and 64 GB 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 of them are taken from some functions of RevLib[35], and one is added by our own. This dataset 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 make comparison 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 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.

Firstly, we test TSA with fixed and variable look-

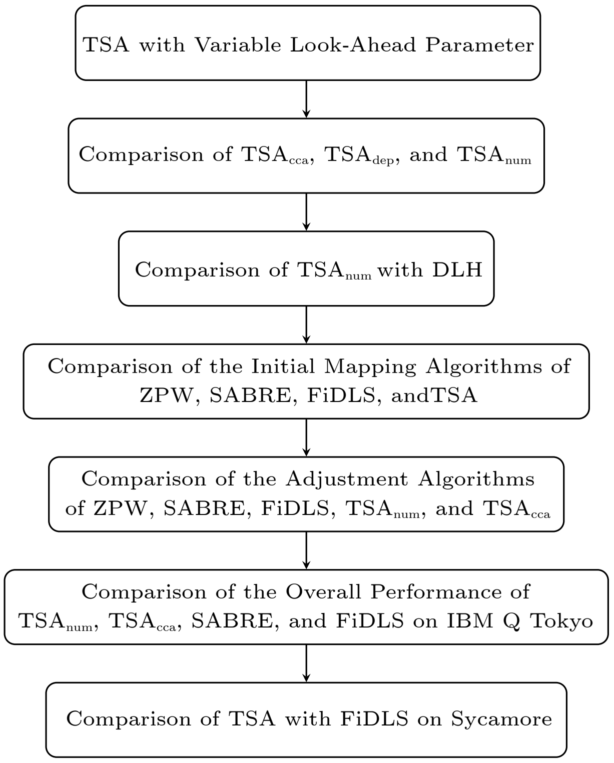

TSA with Variable Look-Ahead Parameter
Comparison of TSA$_{\rm cca}$, TSA$_{\rm dep}$, and TSA$_{\rm num}$
Comparison of TSA$_{\rm num}$ with DLH
Comparison of the Initial Mapping Algorithms of ZPW, SABRE, FiDLS, andTSA
Comparison of the Adjustment Algorithms of ZPW, SABRE, FiDLS, TSA$_{\rm num}$, and TSA$_{\rm cca}$
Comparison of the Overall Performance of TSA$_{\rm num}$, TSA$_{\rm cca}$, SABRE, and FiDLS on IBM Q Tokyo
Comparison of TSA with FiDLS on Sycamore

Fig.5. Sketch of the experiments.

ahead parameter $l_{\rm a}$. In Fig.6, different colors represent the logarithms of the number of additional gates. The lower the points in the figure, the fewer additional gates inserted. As for the look-ahead parameter $l_{\rm a}$, the optimal parameter for each circuit may be different. We have done thousands of experiments and found that when $l_{\rm a} = 2$, 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.

In Fig.7, we compare TSA$_{\rm cca}$, TSA$_{\rm dep}$ and TSA$_{\rm num}$ using the 159 benchmarks mentioned above. Com-

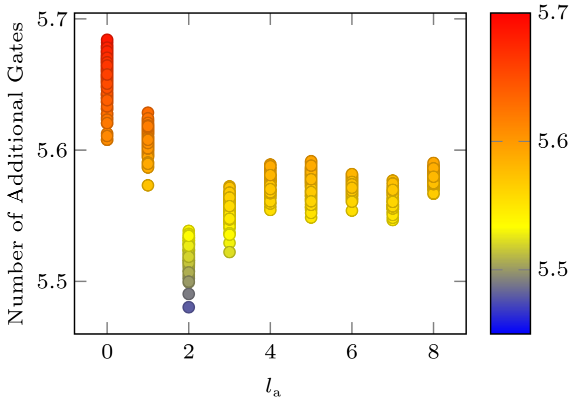


Fig.6. Impact of the look-ahead parameter $l_{\rm a}$ on search results.

③https://github.com/Holly-Jiang/QCTSA, Mar. 2022.

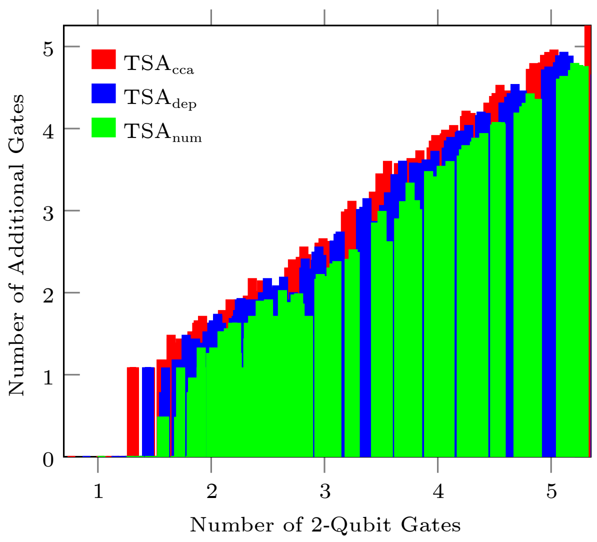


Fig.7. Comparison of the number of additional gates inserted by TSA$_{dep}$, TSA$_{cca}$, and TSA$_{num}$.

pared with TSA$_{cca}$ (resp. TSA$_{num}$), the depth of the generated circuits by TSA$_{dep}$ is reduced by 2.37% (resp. 3.42%) on average. Compared with TSA$_{cca}$ (resp. TSA$_{dep}$), the number of additional gates by TSA$_{num}$ is reduced by 2.69% (resp. 9.56%) on average. Therefore, it is preferable to use either TSA$_{dep}$ or TSA$_{num}$, depending on the optimization objective to be either the depth or the number of additional gates of the resulting circuits.

Secondly, we use the benchmarks[13] to compare TSA$_{num}$ 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, TSA$_{num}$ reduces the total number of additional gates by 27.32% and 12.42%, respectively.

Thirdly, we compare the combinations of several algorithms for inserting fewer additional gates. In order to visualize the differences between ZPW, FiDLS, SABRE, and TSA, we have plotted a series of figures

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

Benchmark	n	G	G_0 (MCPE)	G_1 (MCPE_OP)	G_2 (TSA$_{num}$)	Δ_0(%)	Δ_1(%)
4mod5-v1_22	5	21	0	0	0	0.00	0.00
mod5mils_65	5	35	0	0	0	0.00	0.00
alu-v0_27	5	36	3	3	6	–100.00	–100.00
decod24-v2_43	4	52	0	0	0	0.00	0.00
4gt13_92	5	66	21	21	0	100.00	100.00
ising_model_10	16	786	0	0	0	0.00	0.00
ising_model_13	16	786	0	0	0	0.00	0.00
ising_model_16	16	786	0	0	0	0.00	0.00
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	3 439	1 566	1 224	1 029	34.29	15.93
radd_250	13	3 213	1 353	1 047	852	37.03	18.62
z4_268	11	3 073	1 071	855	915	14.57	–7.02
sym6_145	14	3 888	1 017	1 017	681	33.04	33.04
misex1_241	15	4 813	2 118	1 098	1 032	51.27	6.01
rd73_252	10	5 321	2 352	2 193	1 629	30.74	25.72
cycle10_2_110	12	6 050	2 226	1 968	1 890	15.09	3.96
square_root_7	15	7 630	2 061	1 788	1 509	26.78	15.60
sqn_258	10	4 459	3 708	3 057	3 093	16.59	–1.18
rd84_253	12	13 658	6 411	5 697	4 605	28.17	19.17
co14_215	15	17 936	5 634	5 062	6 813	–20.93	–34.59
sym9_193	10	34 881	15 420	13 746	12 315	20.14	10.41
urf5_158	9	164 416	69 852	58 947	56 253	19.47	4.57
hwb9_119	10	207 775	93 219	89 355	78 753	15.52	11.87
urf4_187	11	512 064	220 329	168 366	141 768	35.66	15.80
Sum	–	1 307 223	428 778	355 782	311 598	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 TSA$_{num}$, respectively, and $\Delta_i = (G_i - G_2)/G_i$.

available online as supplementary materials[④]. 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 of 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 the best when used in conjunction with the five adjustment algorithms. It leads to a reduction of 41%, 30%, and 37% of additional gates compared with the initial mapping algorithms of ZPW, SABRE, and FiDLS.

We then compare the five adjustment algorithms from ZPW, SABRE, FiDLS, TSA_{num}, and TSA_{cca} under specific 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, 16 632 (resp. 12 072) 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 these 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 make comparison 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 12 436 seconds. FiDLS successfully transforms 159 circuits, which takes 63 841 seconds. TSA_{num} and TSA_{cca} are much faster, as they successfully transform all the 159 circuits, taking 2 465 seconds and 2 523 seconds, respectively. Compared with SABRE, the number of additional **SWAP** gates

Table 2. Comparison of the Initial Mapping Algorithms of ZPW, SABRE, FiDLS, and TSA

Algorithm	N	G	G_0	G_1	G_2	G_3	$\Delta_0(\%)$	$\Delta_1(\%)$	$\Delta_2(\%)$
ZPW	115	63 666	29 640	24 951	27 651	17 412	41.26	30.22	37.03
SABRE	108	77 790	28 671	26 079	26 412	16 068	43.96	38.39	39.16
FiDLS	120	209 433	29 484	28 434	30 195	25 950	11.99	8.74	14.06
TSA_{num}	120	163 485	54 969	52 512	62 817	45 948	12.50	26.85	18.59
TSA_{cca}	120	163 485	57 777	53 922	61 668	46 305	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 of ZPW, SABRE, FiDLS, TSA_{num}, and 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	29 443	14 472	11 244	4 938	10 173	10 389	29.71	9.53	−106.01	2.08
SABRE	105	49 987	19 053	16 632	6 204	12 072	11 904	36.61	27.41	−94.58	−1.41
FiDLS	109	105 428	45 813	31 011	16 668	37 800	37 851	17.49	−21.89	−126.78	0.13
TSA	124	150 464	49 620	30 447	19 068	40 461	40 629	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, TSA_{num}, and TSA_{cca}, respectively, and $\Delta_i = (G_i - G_3)/G_i$.

[④]https://github.com/Holly-Jiang/QCTSA/blob/master/QCTSA_Appendix.pdf, Jun. 2023.

Table 4. Comparison of Runtime and Number of Circuits Successfully Transformed by SABRE, FiDLS, TSA$_{\text{num}}$, and TSA$_{\text{cca}}$

Scale	N	G	SABRE			FiDLS			TSA$_{\text{num}}$			TSA$_{\text{cca}}$		
			N_0	G_0	t_0 (s)	N_1	G_1	t_1 (s)	N_2	G_2	t_2 (s)	N_3	G_3	t_3 (s)
Small	66	5 997	66	2 301	2	66	1 329	7	66	894	16	66	897	21
Medium	49	21 618	49	10 218	22	49	5 328	90	49	5 199	57	49	5 280	62
Large	44	3 289 162	29	162 522	12 412	44	532 485	63 744	44	1 013 196	2 392	44	1 037 427	2 440
Sum	159	3 312 734	144	175 041	12 436	159	539 142	63 841	159	1 015 521	2 465	159	1 043 604	2 523

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, TSA$_{\text{num}}$, and TSA$_{\text{cca}}$ respectively, t_0–t_3 are the runtime of SABRE, FiDLS, TSA$_{\text{num}}$, and TSA$_{\text{cca}}$, respectively, and G_0–G_3 are the numbers of additional gates inserted by SABRE, FiDLS, TSA$_{\text{num}}$, and TSA$_{\text{cca}}$, respectively.

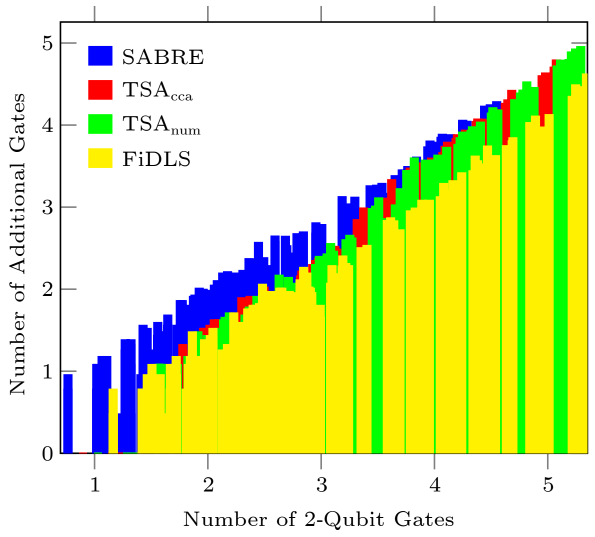


Fig.8. Comparison of SABRE, TSA$_{\text{cca}}$, TSA$_{\text{num}}$ and FiDLS on IBM Q Tokyo.

generated by TSA$_{\text{num}}$ 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$_{\text{num}}$ generates an average of 33% (resp. 2%) fewer additional **SWAP** gates compared with FiDLS. Specifically, FiDLS inserts 1 329 (resp. 5 328) additional gates, while the number is 894 (resp. 5 199) for TSA$_{\text{num}}$. When dealing with large-scale circuits, although TSA$_{\text{num}}$ 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.

Finally, we set the 53-qubit quantum processor Sycamore as our target device 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 31 896 seconds for FiDLS to 1 795 seconds for TSA$_{\text{num}}$, that is, the latter is more than 17 times faster than the former.

6 Conclusions

We proposed a scalable algorithm called Tabu Search-Based Adjustment (TSA) for qubit mapping. We first used a subgraph isomorphism algorithm and a mapping completion algorithm based on the connectivity between qubits to generate a high-quality initial mapping. Then we employed a look-ahead heuristic search to adjust the mapping, which takes into account the influence of the gates yet to be processed to

Table 5. Comparing the Initial Mapping and Adjustment Algorithms of FiDLS and TSA on Sycamore

Algorithm	FiDLS		TSA$_{\text{num}}$		TSA$_{\text{cca}}$		Δ_1(%)	Δ_2(%)
	G_0	t_0 (s)	G_1	t_1 (s)	G_2	t_2 (s)		
FiDLS	2 311 560	31 896	2 245 314	233	2 257 371	3 392	2.86	2.56
TSA	2 305 125	31 211	2 234 937	1 795	2 252 271	3 390	3.04	2.29

Note: The number of test circuits, N, is 159, the number of gates in the input circuits, G, is 3 312 734, G_0–G_2 are the numbers of additional gates inserted by FiDLS, TSA$_{\text{num}}$ and TSA$_{\text{cca}}$, respectively, t_0–t_2 are the runtime of FiDLS, TSA$_{\text{num}}$, and TSA$_{\text{cca}}$, respectively, and $\Delta_i = (G_0 - G_i)/G_0$.

reduce the number of additional gates. We compared the performance of the initial mapping and adjustment algorithms of TSA with state-of-the-art algorithms ZPW, SABRE, and FiDLS, using the architectures of IBM Q Tokyo and Sycamore as the target devices. Our experimental results show that the initial mapping of TSA gives rise to fewer **SWAP** 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 noisy intermedia-to-scale quantum (NISQ) devices.

Conflict of Interest The authors declare that they have no conflict of interest.

References

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

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

[3] Harrow A W, Hassidim A, Lloyd S. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 2009, 103(15): 150502. DOI: 10.1103/PhysRevLett.103.150502.

[4] Arute F, Arya K, Babbush R *et al.* Quantum supremacy using a programmable superconducting processor. *Nature*, 2019, 574(7779): 505–510. DOI: 10.1038/s41586-019-1666-5.

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

[6] Reagor M, Pfaff W, Axline C *et al.* Quantum memory with millisecond coherence in circuit QED. *Physical Review B*, 2016, 94(1): 014506. DOI: 10.1103/PhysRevB.94.014506.

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

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

[9] Glover F. Tabu search-part II. *ORSA 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 S, Ding Y F, Xie Y. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proc. the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr. 2019, pp.1001–1014. DOI: 10.1145/3297858.3304023.

[12] Li S J, Zhou X Z, 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 C, Guan Z J, Cheng X Y. 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 W, Su K L. Local search for Boolean Satisfiability with configuration checking and subscore. *Artificial Intelligence*, 2013, 204: 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 *Proc. the 1st International Workshop on Quantum Technology and Optimization Problems*, Mar. 2019, pp.207–217. DOI: 10.1007/978-3-030-14082-3_18.

[16] Kissinger A, van de Griend A M. CNOT circuit extraction for topologically-constrained quantum memories. *Quantum Information and Computation*, 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. DOI: 10.1088/2058-9565/ab79b1.

[18] Venturelli D, Do M, Rieffel E, Frank J. Temporal planning for compilation of quantum approximate optimization circuits. In *Proc. the 26th International Joint Conference on Artificial Intelligence*, Aug. 2017, pp.4440–4446.

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

[20] de Almeida A A A, Dueck G W, da Silva A C R. Finding optimal qubit permutations for IBM's quantum computer architectures. In *Proc. the 32nd Symposium on Integrated Circuits and Systems Design*, 2019, Article No. 13. 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 *Proc. the 56th Annual Design Automation Conference 2019*, Jun. 2019, Arti-

cle No. 142. 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 *Proc. the 50th Annual Design Automation Conference*, May 2013, Article No. 41. 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. 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 *Proc. the 11th International Conference on Reversible Computation*, Jun. 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 *Proc. the 14th Conference on the Theory of Quantum Computation, Communication and Cryptography*, May 2019, Article No. 5. DOI: 10.4230/LIPIcs.TQC.2019.5.

[26] Zhou X Z, Li S J, 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 *Proc. the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr. 2019, pp.987–999. DOI: 10.1145/3297858.3304007.

[28] Lao L L, van 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*, 2022, 41(2): 359–371. DOI: 10.1109/TCAD.2021.3057583.

[29] Guerreschi G G. 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. 2024.

[30] Nielsen M A, Chuang I L. Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press, 2010.

[31] Barenco A, Bennett C H, Cleve R, DiVincenzo D P, Margolus N, Shor P, Sleator T, Smolin J A, 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 *et al.* OpenQASM 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, 2022, 3(3): 12. DOI: 10.1145/3505636.

[34] Floyd R W. Algorithm 97: Shortest path. *Communications of the ACM*, 1962, 5(6): 345. DOI: 10.1145/367766.368168.

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

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, East China Normal University (ECNU), 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 of East China Normal University (ECNU), Shanghai. His research interests include concurrency theory, especially about process calculi, formal semantics of programming languages, as well as quantum computing.

Ming Xu received his B.Eng. degree in software engineering and Ph.D. degree in system sciences from East China Normal University (ECNU), Shanghai, in 2005 and 2010, respectively. He is currently an associate research professor at Shanghai Key Laboratory of Trustworthy Computing, ECNU. His research interests include computer algebra, program verification, and quantum computing.