Quantum Circuit Transformation
Based on Tabu Search

Hui Jiang, Yuxin Deng, and Ming Xu

Abstract—The goal of quantum circuit transformation is to map a logical circuit to a physical device by inserting additional gates as few as possible in an acceptable amount of time. We present an effective approach called TSA to construct the mapping. It consists of two key steps: one makes use of a combined subgraph isomorphism and completion to initialize some candidate mappings, the other dynamically modifies the mappings by using tabu search-based adjustment. Our experiments show that, compared with state-of-the-art methods GA, SABRE and FiDLS proposed in the literature, TSA can generate mappings with a smaller number of additional gates and it has a better scalability for large-scale circuits.

Index Terms—Quantum circuit transformation, Tabu search, subgraph isomorphism, initial mapping

I. INTRODUCTION

As we all know, quantum technology has been applied in practice. However, the (great) improvements of computer science driven by quantum technology are still in the early stage, since large quantum computers have not yet been built. In 2017, IBM 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. IBM Q Experience [1] provides the public with free quantum computing resources on the cloud and opens source the quantum computing software framework Qiskit [2].

Users of these 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 [3]. In the design stage, we usually do not consider any hardware connectivity constraints. But in order to implement an algorithm on a quantum device, physical constraints have to be taken into account. For example, IBM physical devices only support 1-qubit gates and CNOT gates between two adjacent qubits. Hence, it is necessary to transform the circuits for quantum algorithms to satisfy both logical and physical constraints. This process is called quantum circuit transformation, which maps a logical circuit to a physical device by inserting additional gates. A big challenge for quantum information is the problem of quantum decoherence. Due to the decoherence of qubits, quantum gates need to be applied in a coherent period as the time for a qubit to stay in a coherent state is very short. The longest coherence time of a superconducting quantum chip is still within 10–100us [4]. Therefore, the main goal of quantum circuit transformation is to reduce the number of additional gates and the depth of output circuits in an efficient way.

In the current work, we shorten the lifetime of qubits by parallelization, and use IMSM [5] to generate partial isomorphic subgraphs of logical circuits and physical ones as part of the initial mapping. The advantage of the initial mapping is that we use the appropriate subgraph isomorphism and the two-way connection of the logical circuits and the physical ones to obtain a dense (clustered nodes) initial mapping, which avoids certain nodes from being mapped to remote locations. We use tabu search [6] to generate logical circuits that can be executed on the physical device. Our approach of quantum circuit transformation is thus called TSA. Tabu search can avoid falling into local optima and swapping the recently swapped qubits, thereby improving the parallelism of quantum gates. 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 [7]. We design three evaluation functions that consider not only the current gates but also the constraints of the gates already processed. Our experiment has been conducted by using the IBM Q20 architecture as the target physical device. The experimental results show that the evaluation function based on calculating the number of additional gates inserts the fewest additional gates. We test several combinations of state-of-the-art initial mapping and adjustment algorithms aiming to insert fewer additional gates after quantum circuit transformation. Generally speaking, TSA outperforms GA [8], SABRE [7] and FiDLS [9] in different aspects. When compared with DLH [10] which consists of two evaluation functions MCPE and MCPE_OP, TSA performs better on large-scale circuits on the DLH benchmarks.

The main contributions of this paper are summarized as follows.

1) We extend IMSM, which only generates part of initial mappings, by completing the mapping based on the connectivity between qubits.
2) We propose a heuristic circuit adjustment algorithm based on tabu search, 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) [11], 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 to insert fewer SWAP gates, and TSA has a better scalability than them for adjusting the mapping for large-scale circuits.

The rest of this article is organized as follows. In Section II, we discuss some related work. In Section III we recall some background of quantum computing and quantum information. In Section IV we introduce the problem of quantum circuit transformation and provide our detailed solution. The experimental results are reported in Section V. We conclude in the last section and discuss some future work.

II. RELATED WORK

There exist several initial mapping methods. Paler [12] has shown that initial mapping have an important impact on quantum circuit transformation. He has proposed a heuristic method to find the initial mapping. Just by placing qubits in different positions from the default trivial placement in the circuit instances on actual NISQ devices, the cost can be reduced by up to 10%. Li et al. [7] have proposed a novel reverse traversal technique, which determines the initial mapping by considering the entire circuit. Zhou et al. [13] have put forward an annealing algorithm to find an initial mapping, but it is unstable. In [10], Li et al. have considered the subgraph isomorphism algorithm FiDLS to generate an initial mapping. Zhu et al. [10] have proposed an expansion-from-center scheme to determine the initial mapping. Starting from the center of the interaction graph, they arrange all logical qubits in the order defined by breadth-first search (BFS), and explore all neighboring nodes at the current depth in a strict chronological order. The first mapped node of this method has an important impact on the entire initial mapping. If the relationship between the logical interaction graph and the coupling graph is not considered, the mapping from the center will lead to more additional gates to be inserted.

One important goal of circuit adjustment algorithms is to minimize the number of additional gates. There are currently five main methods to solve the quantum circuit adjustment problem.

- **Unitary matrix decomposition algorithm.** It is used in [14], [15] to rearrange a quantum circuit from the beginning while retaining the input circuit. 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 quantum circuit transformation problem into some existing problems, such as AI planning [16], [17], integer linear programming [18] and satisfiability modulo theories (SMT) [19], and then uses existing tools to find the optimal results in an acceptable amount of time for the problem. Furthermore, as the time cost is usually long, it can only process small-scale quantum circuits.

- **Exact methods.** Siraichi et al. have proposed an exact method [20]. It will iterate over all possible mappings for all dependencies, so it is only suitable for simple quantum coupling graphs and cannot be extended to complex ones.

- **Graph theory.** In [21], Shafaei et al. have used the minimum linear permutation solution in graph theory to model the problem of reducing the interaction distance. The main idea is to divide a given circuit into several sub-circuits and apply the minimum linear permutation solution, respectively. Then, by inserting additional gates, all gates in the sub-circuits can be executed. Finally, a bubble sort is used to calculate the number of inserted SWAP gates. In [22], [23], a two-step method is used to reduce the quantum circuit transformation to a graph problem to minimize the number of additional gates, based on the graph coloring problem and the largest subgraph isomorphism problem.

- **Heuristic search.** Shafaei et al. have used reversal, bridge or swap to achieve quantum circuit transformation [20]. Zulehner et al. [8] have suggested to layer the circuits, then determine compatible mappings for each of these layers to insert as few additional gates as possible. Cowtan et al. [24] have given a qubit routing method \( t(bket) \), which defines a distance vector to approximate the number of SWAP gates in order to compute a sequence of sets of candidate SWAPs. Li et al. [7] have proposed a SWAP-based search algorithm called SABRE. It uses a heuristic evaluation function that trades off the number of 2-qubit gates and the depth of the circuit. Compared with previous search algorithms based on exhaustive mapping, SABRE can handle large-scale quantum circuits. But it depends on a random initial mapping, which does not seem to be the best choice in our opinion. Zhou et al. [13] have designed a heuristic search algorithm with a novel selection mechanism. Instead of choosing the operation with the lowest cost to apply, one can look ahead one step and then choose the best continuous operation. In this way, the algorithm can effectively avoid local optima. Moreover, a pruning mechanism has been introduced to reduce the size of search space and ensure that the program terminates in a reasonable amount of time. Li et al. [9] have suggested to use filtered depth-limit search and feedback to minimize the number of SWAP gates. Their method, FiDLS, tends to search through all possible combinations of SWAP gates to maximize the number of executable 2-qubit gates. But the cost of a thorough search is very high, thus it sets a fixed limit and “filters” out those SWAP gates that do not interact with the gates in the front layer of the circuit. Using exhaustive search, the number of auxiliary gates introduced by FiDLS is small, but the time cost is very high, especially when dealing with medium-scale and large-scale circuits. Zhu et al. [10] have put forward a dynamical look-ahead heuristic cost function to adjust the window size according to the details of the quantum circuit. With the support of the dynamic look-ahead technique, they can deal with some large-scale benchmarks. We will give a quantitative comparison with that method in Section V. In [25], a variation-aware qubit movement strategy is proposed. It takes advantage of the change in error rate and a change-aware quantum circuit transformation strategy by trying to select the route with the lowest probability of failure. This strategy uses the error rate of SWAPs to allocate logical qubits to physical ones, thus avoiding paths with
high error rates as much as possible. In [26] Lao et al. have shown that the fidelity of a circuit is related to the delay and the number of gates. Now some heuristic methods are also applied to other platforms such as Surface-17 [26], [27].

A heuristic search algorithm often uses an evaluation function to obtain an optimal solution. Existing solutions mainly aim at inserting as few SWAP gates as possible [7–10, 24] or using the fidelity of the generated circuit as the objective function [25] or minimizing the overall circuit latency [26]. We provide three options of evaluation functions, considering the number of additional SWAP gates, the depth of the output circuit, or configuration checking with aspiration. We also equip evaluation functions with a look-ahead parameter. At present, there are a number of methods that exploit the look-ahead idea, such as [7], [8], [10], [21]. In particular, the method of [10] can dynamically adjust the number of look-ahead gates. Inspired by it, we allow a dynamic look-ahead parameter and adjust the parameter according to the number of layers of the input circuit, but mainly focusing on the nearest gates.

III. PRELIMINARY

In this section, we introduce some notions and notations of quantum computing.

Classical information is stored in bits, while quantum information is stored in qubits. Besides two basic states |0⟩ and |1⟩, a qubit can be in any linear superposition state like |φ⟩ = a|0⟩ + b|1⟩, where a, b ∈ C satisfy the condition |a|^2 + |b|^2 = 1. The intuition is that |φ⟩ is in the state |0⟩ with probability |a|^2 and in the state |1⟩ with probability |b|^2. We use the letter q (resp. g) to denote a physical qubit (resp. logical qubit).

A quantum gate acts on a qubit to change the state of the qubit. For example, the Hadamard gate (H gate) is applied on a qubit, and the CNOT gate is applied on two qubits. Their symbols and matrix forms are shown in Fig. 1. We use a SWAP gate to exchange the states between two adjacent qubits, and multiple operations simulate moving non-adjacent qubits to adjacent positions. A SWAP gate can be implemented by three CNOT gates, or inserting four H gates to change the direction of the middle CNOT gate, as shown in Fig. 2.

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 many textbooks such as [28]. 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 call a circuit with depth no more than 100 as a small-scale circuit, a circuit with depth more than 1000 as a large-scale circuit, and the rest are medium-scale circuits. It is unnecessary to consider quantum gates acting on a single qubits in circuit adjustments, since 1-qubit gates are local [21].

In the current work, we mainly consider the physical circuits of the IBM Q series, called coupling graphs. Let CG = (V_C, E_C) denote the coupling graph of a physical device, where V_C is the set of physical qubits and E_C is the set of edges representing the connectivity between qubits related by CNOT gates. In Fig. 3 (a) and (b) are the coupling graphs of the 5-qubit IBM QX2 and IBM QX4, respectively; (c) and (d) are the coupling graphs of the 16-qubit IBM QX3 and IBM QX5, respectively; and (e) is the coupling graph of the IBM Q20. 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.

For IBM QX2, QX3, QX4, and QX5, the control of one qubit to a neighbour is unilateral, but for IBM Q20 the control between two adjacent qubits is bilateral.

Assume an interaction graph IG, a coupling graph CG, an
initial mapping $\tau$, and a CNOT gate $g = \langle q_i, q_j \rangle$, where $q_i$ is the control qubit and $q_j$ is the target qubit. If the gate $g$ is executable on coupling graph $CG$, then $\langle \tau(q_i), \tau(q_j) \rangle$ must be a directed edge on $CG$.

**Example 1:** Suppose that a logical interaction graph $IG$ and a coupling graph $CG$ are shown in Fig. 5. Suppose that the initial mapping is as follows

$$\tau = \{q_0 \rightarrow q_{10}, q_1 \rightarrow q_0, q_2 \rightarrow q_6, q_3 \rightarrow q_5, q_4 \rightarrow q_{11}\}.$$ 

Then the 2-qubit gate $g_0 = \langle q_2, q_1 \rangle$ is not executable, since the edge $\langle \tau(q_2), \tau(q_1) \rangle = \langle q_6, q_0 \rangle$ does not exist in $CG$. However, the gate $g_3 = \langle q_1, q_3 \rangle$ is executable, since the edge $\langle \tau(q_1), \tau(q_3) \rangle = \langle q_0, q_6 \rangle$ exists in $CG$.

### IV. Quantum Circuit Transformation

Assume that the input circuit has only 1-qubit gates and CNOT gates [29], [30]. We insert additional gates to move two non-adjacent qubits to adjacent positions or change the direction of CNOT gates. Inserting more gates increases the risk of introducing more noise. Therefore, we expect to find a quantum circuit transformation algorithm that, when given an input circuit, can produce an output circuit with a small number of additional gates and a small depth in an acceptable amount of time.

Roughly speaking, we propose a method of quantum circuit transformation with the following three steps.

1) **Preprocessing.** This step includes extracting the interaction graph from the input circuit, shortening the lifetime of qubits as in [31], and calculating the shortest paths of the coupling graph.

2) **Isomorphism and completion.** This step first uses the subgraph isomorphism algorithm to find part of the initial mapping [5]. 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. It is necessary to adjust the quantum circuits by inserting additional gates. We use a tabu search-based adjustment algorithm to generate circuits that can be physically executed.

Note that isomorphism and adjustment are both NP-complete [20]. Thus, we make use of some heuristics. Below we give a detailed account of each step.

#### A. Preprocessing

In the preprocessing step, we adjust the input circuit described by an openQASM program [32] to shorten the lifetime of qubits. Then we use a BFS search to calculate the shortest distance between each pair of nodes on the coupling graph.

We use a layered method to analyze the lifetime of qubits and pack the gates that can be executed in parallel into a bundle, forming a layered bundle format [31]. Quantum gates acting on different qubits can be executed in parallel. Therefore, we classify the gates that can be executed in parallel into one layer; otherwise, we insert a new layer. The notation $L(C_i) = \{L_0, L_1, \ldots, L_n\}$ denotes the layered circuit, where $L_i$ $(0 \leq i \leq n)$ stands for a quantum gate set that can be executed in parallel. The quantum gate set separated by the dotted lines in Fig. 3(b) are the following: $L_0 = \{g_1\}, L_1 = \{g_2, g_0\}, L_2 = \{g_3, g_4\}, L_3 = \{g_5, g_6\}, L_4 = \{g_7\}, L_5 = \{g_8\}$.

At the same time of layering, we generate an interaction graph $IG = (V_I, E_I)$, which is an undirected graph with $V_I$ being the set of vertices, and $E_I$ the set of undirected edges that denotes the connectivity between qubits related by CNOT gates. Given a coupling graph and assume the distance of each edge is 1, we use the Floyd-Warshall algorithm [33] to
calculate the shortest distance matrix, with \( dist[i][j] \) denoting the shortest distance from \( q_i \) to \( q_j \).

Consider a CNOT gate \( g = (q_1, q_2) \). If \( q_i \) and \( q_j \) are mapped to \( q_{m_i} \) and \( q_{m_j} \), respectively, then the cost of executing \( g \) under the shortest path is denoted by \( \text{cost}_g = 7 \times (dist[m][n] - 1) \) on devices with unilateral control. For IBM Q20, the cost is \( \text{cost}_g = 3 \times (dist[m][n] - 1) \).

**Example 2:** Consider the QX5 coupling graph (cf. Fig. 4(d)). Given a CNOT gate \( g = (q_1, q_2) \), with \( q_1 \) mapped to \( q_6 \) and \( q_2 \) mapped to \( q_{13} \), the shortest distance between them is \( dist[6][13] = 3 \). There are three shortest paths of moving from \( q_6 \) to an adjacent position of \( q_{13} \): \( \pi_0 = q_6 \rightarrow q_5 \rightarrow q_4 \rightarrow q_{13} \), \( \pi_1 = q_6 \rightarrow q_5 \rightarrow q_{12} \rightarrow q_{13} \), \( \pi_2 = q_6 \rightarrow q_{11} \rightarrow q_{12} \rightarrow q_{13} \). Their costs are given by \( \text{cost}_{\pi_0} = 18 \), \( \text{cost}_{\pi_1} = 14 \), and \( \text{cost}_{\pi_2} = 14 \), respectively. Here \( \text{cost}_{\pi} \) stands for the cost of swapping the qubits \( q_i \) and \( q_j \) by following the path \( \pi \).

**B. 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. It shows that the best performance can be achieved by using filters and the sorting ideas of the GraphQL algorithm to process candidate nodes, and the local candidates calculation method LFTI based on set-intersection to enumerate the results. Since IMSM cannot process disconnected graphs, we artificially create connected graphs by linking isolated nodes to the nodes with the largest degree in the interaction graph.

The input of Algorithm 1 is a coupling graph \( CG \), an interaction graph \( IG \), and a partial mappings set \( T \). We initialize an empty queue \( Q \), and mark \( q \) with integers such as \( q_1 = 1, q_2 = 2, q_3 = 1 \). Then we traverse a mapping \( \tau \) and add the unmapped nodes to the queue \( Q \). For the unmapped nodes, we try to map them to the nodes adjacent to the mapped node in \( CG \). Finally, we generate a dense mapping. In principle, we could try to match the remaining unmapped nodes randomly, but it may lead to a mapping with a node far away from other nodes. If an unmapped node has an edge adjacent to a matched node in the interaction graph, it will first be matched to one of the adjacent nodes. In this way, we can obtain all candidate mappings.

In Algorithm 1, 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 3-23 complete the partial mappings. In Line 5, we initialize an empty queue \( Q \), which stores unmapped logical qubits, traverse the mapping \( \tau \) and add the unmapped qubits to \( Q \). We then loop until \( Q \) is empty, and all logical qubits are mapped to physical qubits. We take out the first element in \( Q \) to \( q \). Lines 8-9 get the adjacency matrices of \( CG \) and \( IG \), respectively. Line 10 initializes an empty map \( C \), sorted by a descending order of the degree of connectivity between \( q_m \) and \( q \). Lines 11-21 traverse \( C \), and select the node \( q_m \) that has been mapped to the node \( q \) in the coupling graph and has the largest number of logical connections to \( q \) in \( C \). Line 14 deletes the node from \( C \). Lines 15-19 select the node \( k \) adjacent to \( q_m \) in the adjacency matrix, and map \( q \) to that node.

**Example 3:** Consider the interaction graph shown in Fig. 5(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.

\[
\tau_0 = \{q_0 \rightarrow q_{10}, q_1 \rightarrow -1, q_2 \rightarrow q_6, q_3 \rightarrow q_5, q_4 \rightarrow q_{11}\}
\]

where \( q_1 \rightarrow -1 \) means that \( q_1 \) is not mapped to any physical qubit, so we need the mapping completion algorithm. The maximum number of mapped nodes is 4. We demonstrate how \( \tau_0 \) is completed. We add all unmapped nodes to the queue \( Q \); in this example we have \( Q = \{q_1\} \). Then we loop until \( Q \) is empty. We pop the first element \( q \) of \( Q \), get the adjacency matrix of the query graph and the target graph, and traverse the adjacency matrix. We put the nodes \( q_m \) adjacent to \( q \) into the candidate nodes list \( C \), which is sorted by the connectivity of \( q_m \) and \( q \). We get \( C = \{q_1, q_2, q_4, q_9\} \). Next, we traverse \( C \) and take out the first element \( q_j \) in \( C \), and calculate the physical node \( q_m = q_5 \) so \( \tau_0(q_j) = q_5 \). Finally, we map \( q \) to

---

**Algorithm 1:** Complete the initial mapping

**Input:** \( CG \): A coupling graph;  
\( IG \): An interaction graph;  
\( T \): A partial mapping set obtained by IMSM;  
**Output:** \( \text{results} \): A set of mapping relations between \( IG \) and \( CG \);  

1. Initialize \( \text{results} = \emptyset \);  
2. \( n = \max_{\tau \in T} \{|i : \tau[i] \neq -1, i \leq \tau.\text{length}, i \in N\} \);  
3. for \( \tau \in T \) do  
   4. if \( n = \tau.\text{length} \) then  
      5. \( Q \leftarrow \{i : \tau[i] = -1, i \leq \tau.\text{length}, i \in N\} \);  
      6. \( \text{f*an empty unmapped node queue} \);  
      7. while \( Q \neq \emptyset \) do  
         8. \( q \leftarrow Q.\text{poll}(); \)  
         9. \( M_p \leftarrow CG.\text{adjacencyMatrix}(); \)  
        10. \( M_L \leftarrow IG.\text{adjacencyMatrix}(); \)  
        11. \( C \leftarrow C \cup \{q_m : M_L[q_m] \neq 0\}; \)  
        12. \( \text{f*sorted by degree} \);  
        13. while \( C \neq \emptyset \) do  
           14. \( q_m \leftarrow \tau[C[0]]; \)  
           15. \( k \leftarrow 0; \)  
           16. \( C \leftarrow C \setminus C[0]; \)  
           17. while \( k < M_p[q_m].\text{length} \) do  
              18. if \( (M_p[q_m][k] \neq 0 \text{ and } M_L[q_m][k] \neq 0 \text{ and } \tau[q] \neq k) \) then  
                 19. \( k \leftarrow k + 1; \)  
                 20. break;  
            21. if \( k \neq M_p[q_m].\text{length} \) then  
               22. break;  
        23. \( \text{results}.\text{add}()\);  
5. return \( \text{results} \);
the node connected to \( g_m \) but not yet mapped. If the nodes connected to \( g_m \) have been mapped, the loop continues. In this example, it can be directly mapped to \( q_0 \). In the end, we obtain the mapping

\[
\tau_0 = \{ q_0 \rightarrow q_{10}, q_1 \rightarrow q_0, q_2 \rightarrow q_6, q_3 \rightarrow q_5, q_4 \rightarrow q_{11} \}.
\]

C. Adjustment

1) Tabu search: Tabu search is a type of heuristic algorithm. It uses a tabu list to avoid searching repeated spaces and deadlock. The algorithm uses amnesty rules to jump out of 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.

The following objects are defined in tabu search: neighborhoods, neighborhood action, tabu list, candidate set, tabu object, evaluation function, and amnesty rule. All the edges that can be swapped are the neighborhoods. Obviously, it is not helpful to swap the edges that are not connected to the mapped nodes, so we only add those edges on the shortest path to the candidate set. This does not affect the number of inserted additional gates, but saves significant amount of time rather than collect all the edges in the coupling graph into the candidate set. The tabu list avoids local optima and and tries to parallelize the inserted auxiliary gates. Tabu object is the object in the tabu list. We try not to use the recently swapped qubits as much as possible, which are added to the tabu list. Evaluation function selects an element from the candidate set that can make the nodes of the gate closer. Amnesty rules are used when all the objects in the candidate set are banned, or after banning an object, the target value will be greatly increased.

The calculation of the candidate set is shown in Algorithm 2. The input \( M_p \) contains the mapping from physical qubits to logical ones, where \( j = M_p[i] \) 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 qubits, where \( j = M_l[i] \) 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 layer mapping. The set \( D \) contains the edges of all the shortest paths in the coupling graph. Lines 3-5 delete the gates that can be executed in \( L \) under the current mapping. Lines 6-19 traverse gates in \( L \), and calculate the shortest paths between the nodes of \( g \). The edges involved in the shortest path are all the elements of the candidate set. Lines 9-15 update the mapping after the swap. Lines 16-19 generate a new candidate solution. Line 17 stores the swapped edges that will be used in the output circuit, and Line 18 calculates the swap scores.

**Example 4:** Let us consider the mapping

\[
\tau_0 = \{ q_0 \rightarrow q_{10}, q_1 \rightarrow q_0, q_2 \rightarrow q_6, q_3 \rightarrow q_5, q_4 \rightarrow q_{11} \},
\]

with \( L_1 = \{ g_2, g_6 \} \), \( \text{cost}_{g_2} = 3 \) and \( \text{cost}_{g_6} = 6 \). The gate \( g_2 \) can be executed directly in the \( \tau_0 \) mapping, so we delete it from \( L_1 \), but \( g_0 \) cannot be executed in the mapping \( \tau_0 \). The nodes that cannot be executed join the set \( \text{swap_nodes} = \{ q_0, q_6 \} \). The set of shortest paths is

\[
\text{paths} = \{ \{ q_6 \rightarrow q_1 \rightarrow q_0 \}, \{ q_6 \rightarrow q_5 \rightarrow q_0 \} \}.
\]

We traverse the shortest paths to calculate the candidate set. The two endpoints of an edge passed by one of the shortest paths should intersect with the SWAP set and join 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 \( \tau_{\text{ini}} \) as the best mapping \( \tau_{\text{best}} \). Lines 3-12 cyclically check whether all the gates in the current layer can be executed under the mapping \( \tau_{\text{best}} \). If all the gates are executable or the number of iterations has reached 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 a SWAP appears in the tabu list, its corresponding mapping will be removed from the candidate set. Then from the remaining candidates, we choose

**Algorithm 2:** Calculate the candidate set

**Input:** \( P \): The shortest paths set of coupling graph; \( D \): The distance 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:** \( \text{results} \): The set of candidate mapping:

1. Initialize \( \text{results} \leftarrow \emptyset \);
2. \( N \leftarrow \) An empty set of candidate swap nodes;
3. foreach \( g \in L \)
   4. if \( g \) is executable then
      5. \( L \leftarrow L \setminus \{ g \} \);
4. foreach \( g \in L \)
   7. \( p \in P[N][g \cdot \text{control}][|M_l[g \cdot \text{target}]|] \)
      8. foreach \( e \in p \)
         9. \( M'_q \leftarrow M_q \); \( M'_l \leftarrow M_l \);
         10. \( q_1 \leftarrow M'_q[e \cdot \text{s}] ; q_2 \leftarrow M'_q[e \cdot \text{t}] ; \)
         11. \( M'_q[e \cdot \text{s}] \leftarrow q_2 ; M'_q[e \cdot \text{t}] \leftarrow q_1 ; \)
         12. if \( q_1 \neq -1 \)
            13. \( M'_l[q_1] \leftarrow q_2 ; \)
         14. if \( q_2 \neq -1 \)
            15. \( M'_l[q_2] \leftarrow q_1 ; \)
         16. \( s \leftarrow \emptyset ; \)
         17. \( \text{swaps} \leftarrow \text{swaps} \cup \{ D[e \cdot \text{s}] [e \cdot \text{t}] \} ; \)
         18. \( \text{value} \leftarrow \text{evaluate}(D, M'_l, L) ; \)
         19. \( \text{results} \leftarrow \text{results} \cup \{ s \} ; \)
5. return \( \text{results} ; \)
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 \( \tau_{\text{best}} \), and insert the SWAP performed by the best mapping to the tabu list \( t_t \), indicating that the SWAP in the tabu list should not be used as much as possible recently. The algorithm would try to avoid re-swapping the just swapped qubits. Then it will check whether the termination condition of the algorithm is satisfied. The condition determines whether the number of iterations has reached the given bound, or the current mapping ensures all the gates in the current layer can be executed.

**Example 5:** Let us continue the previous example. We select the one with the lowest evaluation scores from the candidate set. For \( \mathcal{L}_1 = \{ g_2, g_0 \} \), the candidate set is \( \{(q_6, q_1), (q_1, q_6), (q_5, q_0), (q_0, q_5)\} \), and the costs are given as follows:

\[
\begin{align*}
\text{cost}_{\text{num}}(q_6, q_1) &= 3.0, \\
\text{cost}_{\text{num}}(q_1, q_6) &= 3.0, \\
\text{cost}_{\text{num}}(q_5, q_0) &= 3.0, \\
\text{cost}_{\text{num}}(q_0, q_5) &= 3.0.
\end{align*}
\]

The algorithm chooses the first SWAP with the smallest score, and the mapping becomes

\[
\tau_0 = \{ q_0 \rightarrow q_{10}, q_1 \rightarrow q_0, q_2 \rightarrow q_1, q_3 \rightarrow q_5, q_4 \rightarrow q_{11} \}.
\]

It can be seen that the current mapping ensures that \( g_0 \) is executable. So we can continue to the next layer.

2) **Evaluation functions:** Evaluation functions can control the search direction. 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.

\[
\begin{align*}
\text{cost}_{\text{num}}(q_m, q_n) &= \sum_{g \in \mathcal{L}} \text{dist}([\tau(g.\text{control})][\tau(g.\text{target})]) \\
\text{cost}_{\text{depth}}(q_m, q_n) &= \text{Depth}(L)
\end{align*}
\]

Here \( \text{cost}_{\text{num}}(q_m, q_n) \) (resp. \( \text{cost}_{\text{depth}}(q_m, q_n) \)) denotes the distance (resp. depth) between two qubits of all the gates in the layer \( L \) after swapping \( q_m \) with \( q_n \).

CCA is a heuristic method, mainly used for SAT problems. We apply the idea of CCA to adjust circuits. Let \( \text{submake} \) represent the number of qubits for which two qubits are closer after a SWAP operation, and \( \text{subbreak} \) represent the number of qubits for which two qubits are farther apart after a SWAP operation. We introduce \( \text{subscore} = \text{submake} - \text{subbreak} \) into the evaluation function, and adjust the weight with Smooth Weight based Threshold (SWT) scheme (11). The application of SWT in our experiment is mainly to add 1 to the edge of swap. When the weight of an edge exceeds the threshold, the weight of all edges becomes \( \rho \ast w(c_i) + (1 - \rho) \ast \bar{w} \), where \( \rho \) is the influence factor of the weight of edge \( c_i \) in the adjustment, and \( \bar{w} \) represents the average weight. Adjusting the \( \rho \) and threshold parameters has limited effect on our experiments, especially on large-scale circuits, because these two parameters only ensure that the weight is close to the threshold.

3) **Look ahead:** 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 operation in the \( i \)-th layer will affect the mapping of the \( (i + 1) \)-th layer. If we only consider the gates in the current layer when choosing the SWAP gates, the SWAP only satisfies the requirement of the \( i \)-th layer, not necessarily the next layer. Therefore, we take the gates from the \( i \)-th to the \( (i + n) \)-th layer, with \( i + n \leq d \), into consideration, where \( n \) is the number of look-ahead layers. However, it is necessary to give a higher priority to the execution of the gates in the \( i \)-th layer, so we introduce an attenuation factor \( \delta \), which controls the influence of the gates in the look-ahead layers. Our evaluation functions in (1) and (2) can be modified as (3) and (4), respectively.

\[
\begin{align*}
\text{cost}_{\text{num}}(q_m, q_n) &= \left( \sum_{g \in \mathcal{L}_i} \text{dist}([\tau(g.\text{control})][\tau(g.\text{target})]) \right) \\
&\quad + \delta \times \left( \sum_{j=i}^{i+n} \sum_{g \in \mathcal{L}_j} \text{dist}([\tau(g.\text{control})][\tau(g.\text{target})]) \right) \\
\text{cost}_{\text{depth}}(q_m, q_n) &= \text{Depth}(L_i) + \delta \times \text{Depth}(\sum_{j=i}^{i+n} L_j)
\end{align*}
\]

4) **Complexity:** Given an interaction graph \( \mathcal{G} = (\mathcal{V}_g, \mathcal{E}_g) \) and a coupling graph \( \mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c) \), we assume that the depth of the circuit is \( d \). Tabu search processes each layer one by one, and searches at most \( d \) times. Starting from the initial mapping, we first delete the executable gates of the first layer under the
initial mapping. Then, the edges of all the shortest paths of all the gates that are not executable in the current layer are added to the candidate set where at least one node is in the gate mapping. In the worst case, the length of the shortest path is \(|E_c| - 1\) as well as the size of the candidate set. Each SWAP will make the gates closer. In the worst case, the number of SWAPs is \((|E_c| - 1)^{|E_c| - 2}\), but our selection strategy will make the number of SWAPs significantly reduced. The time complexity in the worst case is \(O(d \times (|E_c| - 1)^{|E_c| - 2})\), and the space complexity is the size of our candidate set \((E_c - 1)\), which is in PSPACE.

V. EXPERIMENTS

We compare TSA with several state-of-the-art algorithms for quantum circuit transformation, namely GA [8], SABRE [7], FiDLS [9] and DLH [10]. Notice that other algorithms such as SAHS [13] and \(t\ket{ket}\) [24] are not listed because it has been pointed out in [2] that FiDLS is superior to SAHS and the latter outperforms \(t\ket{ket}\) as shown in [13]. The implementation in Python is available at [https://github.com/Holly-Jiang/QCTSA](https://github.com/Holly-Jiang/QCTSA). All the experiments are conducted on a Ubuntu machine with 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. Since SABRE uses a random initial mapping, for every test case we execute it five times, each with a different initial mapping for each benchmark and reports the best result out of the five attempts. Other algorithms are deterministic, so it suffices to run them only once.

Firstly, we test TSA with fixed and variable look-ahead parameters \(l_a\). In Fig. 6 different colors represent the logarithms of the number of additional gates. The experiments show that the influence of look-ahead parameter \(l_a\) is more significant than attenuation factor \(\delta\). The optimal attenuation factor for each circuit may be different. We have done thousands of experiments and found that when \(\delta = 0.5\) and \(l_a = 2\), the number of additional gates are relatively small for all benchmarks [8]. It means that the current layer is roughly twice as important as the later layers, and a 2-layer look-ahead already gives a good performance for TSA.

In Fig. 7 we compare three evaluation functions \(TSA_{cca}\), \(TSA_{dep}\) and \(TSA_{num}\). Using these indicators as objective functions, we test 159 benchmarks [8]. Compared with \(TSA_{cca}\) (resp. \(TSA_{dep}\)), the depth of the generated circuits by \(TSA_{num}\) is reduced by 4.02\% (resp. 3.24\%) on average, and the number of additional gates are reduced by 26.22\% (resp. 24.52\%) on average. The evaluation function of \(TSA_{cca}\) in quantum circuit transformation does not seem to have obvious advantage over \(TSA_{num}\). Inserting a SWAP gate requires inserting 3 CNOT gates, and the depth will increase by 3. Therefore, inserting fewer SWAP gates may have a smaller circuit depth.

Secondly, we compare TSA with DLH, using the benchmarks in [10]. Note that two evaluation functions MCPE and MCPE_OP are used in DLH. Since there is no code for DLH available online, we only compare the number of additional gates inserted in the output circuits generated by DLH and TSA, as we can see in Table I. Compared with MCPE and MCPE_OP, TSA reduces the total number of additional gates by 21.07\% and 4.90\%, respectively.

Thirdly, we compare the combinations of several algorithms in the hope of inserting fewer additional gates. We use the initial mapping and adjustment algorithms from GA [8], SABRE [7], FiDLS [9], and TSA.

We compare the performance of the four initial mapping algorithms from GA, FiDLS, SABRE and TSA under specific adjustment algorithms. The five rows in Table II correspond to Figs. 8-12. For example, in the first row the adjustment algorithm is fixed to be that of GA; there are 101 circuits that all the four initial mapping algorithms can successfully transform and we compare the number of additional gates. In Fig. 8 the adjustment algorithm is fixed to be that of GA. It can be seen that the initial mapping algorithm of TSA leads
TABLE I: Comparison of MCPE, MCPE_OP and TSA

![Table Image]

"n": the number of qubits. g: the number of gates in the input circuit. g0-g2: the number of additional gates inserted by MCPE, MCPE_OP and TSA, respectively.

TABLE II: Comparison of the initial mapping algorithms of GA, FiDLS, SABRE and TSA.

![Table Image]

"n": the number of circuits that all the four initial mapping algorithms can successfully transform. g: the number of 2-qubit gates in the input circuits. g0-g4: the number of additional gates inserted by GA, SABRE, FiDLS and TSA, respectively.

TABLE III: Comparison of the adjustment algorithms of GA, SABRE, FiDLS, TSA_num and TSA_csa.

![Table Image]

"n": the number of test circuits. g: the number of 2-qubit gates in the input circuits. n0-n3: the number of circuits successfully transformed by SABRE, FiDLS, TSA_num and TSA_csa, respectively. l0-l3: runtime of SABRE, FiDLS, TSA_num and TSA_csa, respectively, in seconds. g0-g4: the number of additional gates inserted by SABRE, FiDLS, TSA_num and TSA_csa, respectively.

TABLE IV: Comparison of runtime and the number of circuits successfully transformed by SABRE, FiDLS, TSA_num, TSA_csa, respectively.

![Table Image]
Fig. 8: Comparison of the initial mapping algorithms of GA, SABRE, FiDLS and TSA, using the adjustment algorithm of GA.

Fig. 9: Comparison of the initial mapping algorithms of GA, SABRE, FiDLS and TSA, using the adjustment algorithm of FiDLS.

Fig. 10: Comparison of the initial mapping algorithms of GA, SABRE, FiDLS and TSA, using the adjustment algorithm of FiDLS.

Fig. 11: Comparison of the initial mapping algorithms of GA, SABRE, FiDLS and TSA, using the adjustment algorithm of TSA.

to a reduction of 54%, 43% and 33% of additional gates than the initial mapping algorithms of GA, SABRE and FiDLS, respectively. In Figs. [9][12] the adjustment algorithm is fixed to be that of SABRE, FiDLS, TSA\textsubscript{num}, and TSA\textsubscript{cca}, respectively. As can be seen from Table [II] on all the benchmarks, the initial mapping algorithm of TSA performs best when used in conjunction with the five adjustment algorithms.

In Figs. [13][16] we compare the five adjustment algorithms GA, SABRE, FiDLS, TSA\textsubscript{num} and TSA\textsubscript{cca} under specific initial mapping algorithms. For small-scale and medium-scale circuits FiDLS gives rise to the fewest additional gates. Since it is based on depth-first search, FiDLS takes large search space and long search time and the scale of circuits it can process is very limited. On the contrary, TSA performs well on large circuits in terms of additional gates and runtime. The four rows in Table [III] correspond to Figs. [13][16] For example, in the first row the initial mapping algorithm is fixed to be that of GA; there are 94 circuits that all the five adjustment algorithms can transform. In columns 4-8 we list the number of additional gates required by the five adjustment algorithms.

Finally, we compare the overall performance of TSA\textsubscript{num} and TSA\textsubscript{cca} with SABRE and FiDLS. We test 159 circuits, including 66 small-scale circuits, 49 medium-scale circuits and 44 large-scale ones. Note that in Table [IV] and Fig. [17] we do not display the data for GA. Instead, we compare with SABRE because it is already shown in [7] that SABRE is much more scalable than GA. SABRE successfully transforms 144 circuits, including all the small-scale and medium-scale circuits,
and 29 large-scale ones, which takes 12433 seconds. FiDLS successfully transforms 93 circuits, including all the small-scale circuits, 21 medium-scale circuits, and 6 large-scale circuits, which takes 9075 seconds. TSA\textsubscript{num} and TSA\textsubscript{cca} are much faster, as they successfully transform all the 159 circuits, which takes 1674 seconds and 2312 seconds, respectively. Compared with SABRE, the number of additional SWAP gates generated by TSA\textsubscript{num} is reduced by 61% on average, among the 115 small-scale and medium-scale circuits that both of them can successfully transform. More specifically, TSA\textsubscript{num} has 105 circuits with fewer additional gates than SABRE, and 9 circuits with equal numbers of gates. Compared with TSA\textsubscript{num}, the number of additional SWAP gates generated by FiDLS is reduced by 10% on average, among the 87 small-scale and medium-scale circuits successfully transformed by it. Specifically, FiDLS inserts 555 additional gates, and for TSA\textsubscript{num} the number is 618. Although TSA\textsubscript{num} inserts a little more additional gates, it can transform large-scale circuits much more quickly, as we can see in Table IV.
We proposed a scalable algorithm for quantum circuit transformation. 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 took into account the influence of the gates yet to be processed to reduce the number of additional gates inserted. Compared with DLH, TSA performed better on large-scale circuits and took a shorter time in the DLH benchmarks. We compared the performance of the initial mapping and adjustment algorithm with the state-of-the-art algorithms GA, SABRE, FiDLS and TSA. Our experimental results showed that the initial mapping of TSA gave rise to fewer SWAP gates inserted and the adjustment algorithm could be obtained in an acceptable amount of time. Most small-scale and medium-scale circuits could be transformed in a few seconds. For large-scale circuits, the results could be obtained within a few minutes. In the future, we would investigate how to reduce the number of additional gates inserted and increase the speed. We would also apply the proposed method to more NISQ devices.

VI. CONCLUSION

We proposed a scalable algorithm for quantum circuit transformation. 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 took into account the influence of the gates yet to be processed to reduce the number of additional gates. Compared with DLH, TSA performed better on large-scale circuits and took a shorter time in the DLH benchmarks. We compared the performance of the initial mapping and adjustment algorithm with the state-of-the-art algorithms GA, SABRE, FiDLS and TSA. Our experimental results showed that the initial mapping of TSA gave rise to fewer SWAP gates inserted and the adjustment algorithm could be obtained in an acceptable amount of time. Most small-scale and medium-scale circuits could be transformed in a few seconds. For large-scale circuits, the results could be obtained within a few minutes. In the future, we would investigate how to reduce the number of additional gates inserted and increase the speed. We would also apply the proposed method to more NISQ devices.

REFERENCES

[1] IBM Q Experience, https://www.ibm.com/quantum-computing/
[2] Qiskit, https://www.qiskit.org/
[3] C. G. Almudever, L. Lao, R. W. Heeres, N. Ofek, K. Sliwa, E. Holland, C. Wang, J. Blumoff, K. Chou, M. J. Hatridge, L. Frunzio, M. H. Devoret, L. Jiang, and R. J. Schoeklopf, “Quantum memory with millisecond coherence in circuit qed,” Phys. Rev. B, vol. 94, p. 014506, Jul. 2016.
[4] S. Sun and Q. Luo, “In-memory subgraph matching: An in-depth study,” in Proceedings of the 2020 International Conference on Management of Data. ACM, 2020, pp. 1083–1098.
[5] F. W. Glover, “Tabu search - part II,” Informs Journal on Computing, vol. 2, no. 1, pp. 4–32, 1990.
[6] A. Paler, “On the influence of initial qubit placement during NISQ architectures,” in Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2019, pp. 1001–1014.
[7] G. Li, Y. Ding, and Y. Xie, “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. ACM, 2019, pp. 1001–1014.
[8] A. Zalehner, A. Paler, and R. Wille, “An efficient methodology for mapping quantum circuits to the IBM QX architectures,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 7, pp. 1226–1236, 2019.
[9] S. Li, X. Zhou, and Y. Feng, “Qubit mapping based on subgraph isomorphism and filtered depth-limited search,” IEEE Transactions on Computers, pp. 1–1, 2020.
[10] P. Zha, Z. Guan, and X. Cheng, “A dynamic look-ahead heuristic for the qubit mapping problem of NISQ systems,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 12, pp. 4721–4735, 2020.
[11] S. Cai and K. Su, “Local search for boolean satisfiability with configuration checking and subscore,” Artif. Intell., vol. 204, pp. 75–98, 2013.
[12] A. Paler, “On the influence of initial qubit placement during NISQ circuit compilation,” in Quantum Technology and Optimization Problems. Springer International Publishing, 2019, pp. 207–217.
[13] X. Zhou, S. Li, and Y. Feng, “Quantum circuit transformation based on simulated annealing and heuristic search,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 12, pp. 4683–4694, 2020.
[14] A. Kissinger and A. M. de Griend, “CNOT circuit extraction for topologically-constrained quantum memories,” Quantum Inf. Comput., vol. 20, no. 7 & 8, pp. 581–596, 2020.
[15] B. Nash, V. Gheorghiu, and M. Mosca, “Quantum circuit optimizations for NISQ architectures,” Quantum Science and Technology, vol. 5, no. 2, p. 025010, 02 2020.
[16] D. Venturelli, M. Do, E. G. Rieffel, and J. Frank, “Temporal planning for compilation of quantum approximate optimization circuits,” in Proceedings of the 26th International Joint Conference on Artificial Intelligence. ijcai.org, 08 2017, pp. 4440–4446.
[17] D. E. Bernal, K. E. C. Booth, R. Dridi, H. Alghassi, S. R. Tayur, and D. Venturelli, “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, ser. Lecture Notes in Computer Science, vol. 12296. Springer, 2020, pp. 112–129.
[18] A. A. A. de Almeida, G. W. Dueck, and A. C. R. da Silva, “Finding optimal qubit permutations for IBM’s quantum computer architectures,” in Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design. ACM, 2019, p. 13.
[19] P. Murali, N. M. Linke, M. Martonosi, A. Javadi-Abhari, N. H. Nguyen, and C. H. Alderate, “Full-stack, real-system quantum computer studies: architectural comparisons and design insights,” in Proceedings of the 46th International Symposium on Computer Architecture. ACM, 2019, pp. 527–540.
[20] M. Y. Siraichi, V. F. dos Santos, S. Collange, and F. M. Q. Pereira, “Qubit allocation,” in Proceedings of the 2018 International Symposium on Code Generation and Optimization. ACM, 2018, pp. 113–125.

[21] A. Shafaei, M. Saeedi, and M. Pedram, “Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures,” in Proceedings of the 50th Annual Design Automation Conference. ACM, 2013, pp. 41:1–41:6.

[22] G. G. Guerreschi and J. Park, “Two-step approach to scheduling quantum circuits,” Quantum Science and Technology, vol. 3, no. 4, p. 045003, 06 2018.

[23] A. Matsuo and S. Yamashita, “An efficient method for quantum circuit placement problem on a 2-d grid,” in Proceedings of the 11th International Conference on Reversible Computation, ser. LNCS, vol. 11497. Springer, 2019, pp. 162–168.

[24] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, “On the qubit routing problem,” in Proceedings of the 14th Conference on the Theory of Quantum Computation, Communication and Cryptography, ser. LIPIcs, vol. 135, 2019, pp. 5:1–5:32.

[25] S. S. Tannu and M. K. Qureshi, “Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers,” in Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2019, pp. 987–999.

[26] L. Lao, H. van Someren, I. Ashraf, and C. G. Almudever, “Timing and resource-aware mapping of quantum circuits to superconducting processors,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 1–1, 2021.

[27] G. G. Guerreschi, “Scheduler of quantum circuits based on dynamical pattern improvement and its application to hardware design,” 2019.

[28] M. A. Nielsen and I. L. Chuang, Quantum Computation and Quantum Information (10th Anniversary edition). Cambridge University Press, 2016.

[29] A. Barenco, C. Bennett, R. Cleve, D. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, “Elementary gates for quantum computation,” Physical Review A, vol. 52, no. 5, pp. 3457–3467, Nov 1995.

[30] M. Möttönen and J. Vartiainen, Decompositions of general quantum gates. Nova Science Publishers Inc, 2006.

[31] Y. Zhang, H. Deng, Q. Li, H. Song, and L. Nie, “Optimizing quantum programs against decoherence: Delaying qubits into quantum superposition,” in 2019 International Symposium on Theoretical Aspects of Software Engineering. IEEE, 2019, pp. 184–191.

[32] A. Cross, L. Bishop, J. Smolin, and J. Gambetta, “Open quantum assembly language,” 07 2017.

[33] R. W. Floyd, “Algorithm 97: Shortest path,” Commun. ACM, vol. 5, no. 6, p. 345, 1962.