Quantum Circuit Transformation Based on Subgraph Isomorphism and Tabu Search

Hui Jiang, Yuxin Deng, and Ming Xu

***Abstract*—The goal of quantum circuit transformation is to construct mappings from logical quantum circuits to physical ones in an acceptable amount of time, and in the meantime to introduce as few auxiliary gates as possible. We present an effective approach to constructing the mappings. It consists of two key steps: one makes use of a combined subgraph isomorphism and complement (CSIC) to initialize a mapping, the other dynamically adjusts the mapping by using Tabu search-based adjustment (TSA). Our experiments show that, compared with the very recent method *wghtgraph* considered in the literature, CSIC can save 22.43% of auxiliary gates and reduce the depth of output circuits by 8.46% on average in the initialization of the mapping, and TSA has a better scalability than many state-of-the-art algorithms for adjusting mappings.**

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

I. Introduction

Quantum technology has been applied in practice, but large quantum computers have not yet been built. Most of the contributions of quantum information to computer science are still in the early stage. 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 computer 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 implementation of a quantum algorithm. In the design stage, we usually do not consider any hardware connectivity constraints. But in order to implement an algorithm on a physical device, some physical constraints must be taken into account. 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 logical qubits to physical ones before the logical circuits are executed in physical devices. 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 10us-100us. Therefore, two main goals of a circuit transformation is to reduce the depths of output circuits and the numbers of auxiliary gates introduced by the transformation.

H. Jiang, Y. Deng and M. Xu are with Shanghai Key Laboratory of Trustworthy Computing, East China Normal University.

In the current work, we adjust the lifetime of qubits through parallelization, and use *SubgraphMatching* [3] to generate partial isomorphic subgraphs of logical circuits and physical circuits 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 circuits to obtain a dense initial mapping, which avoids certain nodes from being mapped to remote locations. We use Tabu search [4] to generate circuits that can be executed on physical devices. Tabu search can avoid falling into local optima and swapping the recently swapped qubits, thereby improve the parallelism of quantum gates. We add 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. Our evaluation function not only considers the current gates but also the constraints of the gates already considered.

We compare CSIC with the state-of-the-art initial mapping methods *wghtgraph* [5] and *optm* [6]. On average, the auxiliary gates of the CSIC algorithm are reduced by 22.43% (resp. 27.14%), and the depths are reduced by 8.46% (resp. 10.74%). We compare TSA with wgtgraph [5] and SABRE [7]. TSA can transform 159 circuits in a few minutes, while the other two adjustment algorithms are difficult to transform medium or large circuits. Among the 159 circuits that we also test with SABRE, only 29 can be successfully mapped within the five-minute limit.

The main contributions of this paper are summarized as follows.

1) We use the combined subgraph isomorphism algorithm to generate part of the initial mapping and then complete the mapping based on the connectivity between qubits.
2) We present a heuristic circuit adjustment algorithm based on Tabu search, which can adjust large circuits much efficiently, compared with existing precise search and heuristic algorithms.
3) We propose a look-ahead evaluation function that considers both the current gates and the gates yet to be executed. It filters out SWAP gates that are beneficial to the current gates and also bring closer the gates to be executed.
4) We test 159 circuits, and the results show that the initial mapping generated by our method requires to insert fewer SWAP gates, and the adjustment algorithm can be extended to transform medium and large circuits.

The rest of this paper 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 povide our detailed solution. The experimental results are reported in Section V. The last section concludes the paper and discusses some future work.

II. Related work

There are several initial mapping methods. Paler [8] has showed that the initial mapping has an important influence on quantum circuit transformation. He proposed a heuristic method to find the initial mapping. Just by placing qubits in different positions from the default trivial placement in the actual circuit instances on the actual NISQ device, the time 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. [9] have put forward an annealing algorithm to find an initial mapping, but it is unstable. In [5], Li et al. have considered the subgraph isomorphism algorithm wghtgraph to generate an initial mapping, which is the most recent result, so we will compare with it.

One important goal of circuit adjustment algorithm is to minimize the number of auxiliary SWAP gates. There are currently five main methods for solving the quantum circuit adjustment problem.

- *Unitary matrix decomposition algorithm.* It is used in [10], [11] to rearrange the 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 [12], [13], Integer Linear Programming (ILP) [14], or Satisfiability Modulo Theories (SMT) [15]. Existing tools for those problems are then used to find acceptable results. The approach cannot take advantage of certain properties of quantum mapping, which is a drawback. Furthermore, as the time cost is usually long, it can only transform small quantum circuits.
- *Exact methods.* Siraichi et al. [16] have proposed an exact method. It will iterate all possible mappings for all dependencies, so it is only suitable for simple quantum architectures and cannot be extended to complex ones.
- *Graph theory.* In [17], 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 adding auxiliary gates, all gates in the sub-circuits become adjacent gates. Finally, bubble sort is used to calculate the number of necessary SWAP gates. In [18], [19], a two-step method is used to reduce the quantum circuit transformation to the graph problem to minimize the number of auxiliary gates, based on the graph coloring problem and the largest subgraph isomorphism problem.
- *Heuristic search.* Heuristic search uses an evaluation function to obtain an acceptable solution in exponential time. Zulehner et al. [6] have suggested to layer the circuits, then determine compatible mappings for each of these layers to add as few auxiliary gates as possible. Zhou et al. [9] 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 minima. Moreover, a pruning mechanism has been introduced to reduce the search space's size and ensure that the program terminates in a reasonable amount of time.
  Li et al. [7] have proposed a SWAP-based search algorithm SABRE. Compared with previous search algorithms based on exhaustive mapping, SABRE can adapt to large quantum circuits in the NISQ era. In [20], a routing algorithm called $t\,|ket\rangle$ ensures that any quantum circuit can be compiled into any architecture. The algorithm is divided into four stages: decomposing the input circuit into time steps, determining the initial mapping, routing across time steps, and cleaning up. The heuristics in $t\,|ket\rangle$ give the same or better results than other circuit transformation systems in terms of the depth and the total number of gates in the compiled circuit, with much shorter running time, and can transform larger circuits. In [21], 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 qubits, thus avoiding paths with high error rates as much as possible.

Among the existing methods above, wghtgraph [5] and optm [6] are probably the most effective initial mapping ones. As to circuit adjustment algorithms, wgtgraph [5] and SABRE [7] represent the state-of-the-art. Therefore, we choose to compare our solution with them in Section V.

III. Preliminary

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

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 the probability $|a|^2$ or in the state $|1\rangle$ with the probability $|b|^2$. We use the letter q (resp. q) to denote a physical qubit (resp. logical qubit).

By applying quantum gates to qubits, we can change their states. For example, the Hadamard gate (H gate) can be applied on a single qubit, while the CNOT gate can be applied on two qubits. Their representations in terms of gate symbols and their semantics in terms of matrices are shown in Fig. 1.

A quantum logical circuit consists of quantum gates interconnected by quantum wires [22]; see Fig. 2 for an example.

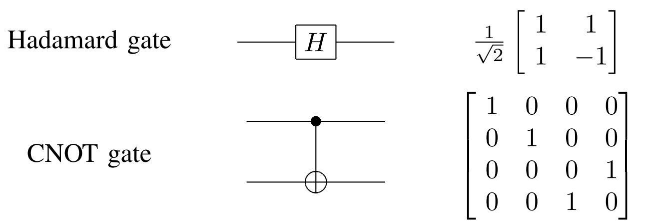


Fig. 1: The symbols of two quantum gates and their matrices

A quantum wire is a mechanism for moving quantum data from one location to another. Each line denotes a qubit, and the gates on the lines act on the corresponding qubits. 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 executing in parallel. For example, the depth of the circuit in Fig. 2 is 6, and the width is 5. In this paper, a circuit with a depth less than 100 is a called a small circuit, a circuit with a depth greater than 1000 is called a large circuit, and the rest are medium circuits. It is unnecessary to consider quantum gates acting on single qubits in circuit adjustments, since the single qubits are *local* [17].

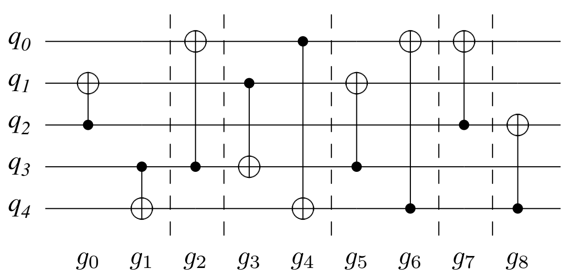


Fig. 2: A quantum circuit

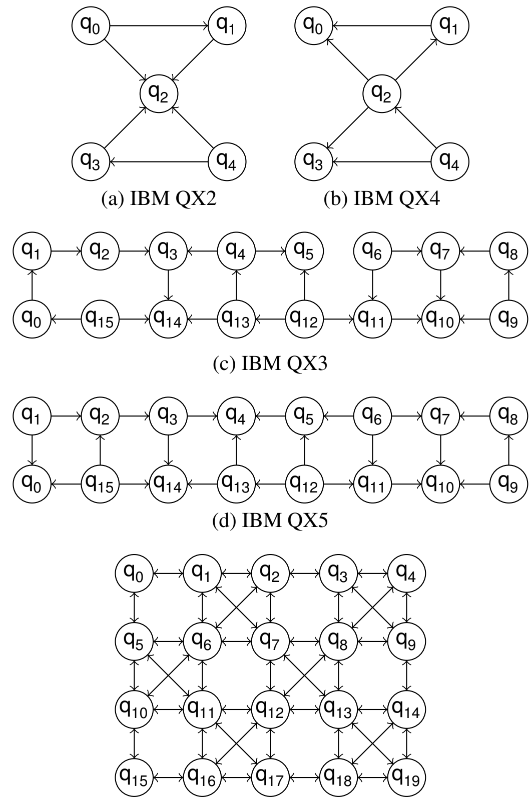
In the current work, we mainly consider the physical circuits of the IBM Q series. Let $\mathcal{AG}_{\mathcal{P}} = (V_P, E_P)$ denote the architecture graph of a physical circuit, where V_P denotes the set of physical qubits and E_P denotes the set of edges that connect CNOT gates. In Fig. 3, Diagrams (a) and (b) are the physical architecture graphs of the 5-qubit IBM QX2 and IBM QX4, respectively; Diagram (c) and (d) are the physical architecture graphs of the 16-qubit IBM QX3 and IBM QX5, respectively; Diagram (e) is the physical architecture graph of IBM Q20. The direction in each edge indicates the control direction of a 2-qubit gate, and 2-qubit gates can only be performed between on two adjacent qubits. IBM physical circuits only support single quantum gates and CNOT gates between two adjacent qubits.

Given a logical circuit LC, a physical architecture $\mathcal{AG}_P$, an initial mapping τ, and a CNOT gate $g = \langle q_i, q_j \rangle$, where q_i is the control qubit, q_j is the target qubit, if gate g is executable on a physical circuit with the architecture $\mathcal{AG}_P$, then $\langle \tau(q_i), \tau(q_j) \rangle$ must be a directed edge on $\mathcal{AG}_P$.

Example 1: Fig. 4 (a) is the logical architecture of Fig. 2. Fig. 4 (b) is the partial architecture graph of IBM Q20. An initial mapping is

$$\tau = \{q_0 \to \mathsf{q}_{10},\ q_1 \to \mathsf{q}_0,\ q_2 \to \mathsf{q}_6,\ q_3 \to \mathsf{q}_5,\ q_4 \to \mathsf{q}_{11}\}.$$

The 2-qubit gate $g_0 = \langle q_2, q_1 \rangle$ is not executable, since the edge $\langle \tau(q_2), \tau(q_1) \rangle = \langle \mathsf{q}_6, \mathsf{q}_0 \rangle$ does not exist in $\mathcal{AG}_P$. But $g_3 = \langle q_1, q_3 \rangle$ is executable, since the edge $\langle \tau(q_1), \tau(q_3) \rangle = \langle \mathsf{q}_0, \mathsf{q}_5 \rangle$ exists in $\mathcal{AG}_P$.



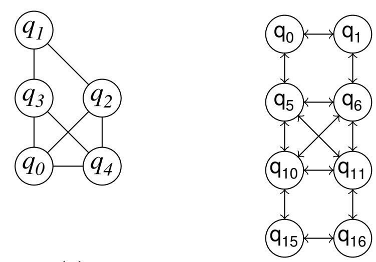
(a) IBM QX2 (b) IBM QX4

(c) IBM QX3

(d) IBM QX5

(e) IBM Q20

Fig. 3: IBM QX architectures



(a) (b)

Fig. 4: (a) The architecture graph of the original circuit in Fig. 2. (b) The partial architecture graph of IBM Q20.

IV. Quantum Circuit Transformation

It is a popular assumption that the input circuit has only single quantum gates and CNOT gates [23], [24]. We add auxiliary gates to move two non-adjacent qubits to adjacent position or change the direction of a CNOT gate. Adding more gates increases the risk of introducing more noise. Therefore, we expect to find a circuit transformation algorithm that, when given an input circuit, can produce an output circuit with a minimal number of auxiliary gates and a small circuit depth in an acceptable amount of time.

Roughly speaking, quantum circuit transformation includes the following three steps.

1) *Preprocessing.* This step includes extracting the logical architecture graph of the circuit, adjusting the life cycle of qubits as in [25], and calculating the shortest paths of the physical circuit.
2) *Isomorphism and completion.* This step uses the subgraph isomorphism algorithm to find part of the initial mapping [3]. Then we perform a mapping completion to include the remaining nodes that do not satisfy all isomorphism requirements, according to the connectivity between the unmapped nodes and the mapped nodes.
3) *Adjustment.* After the second step, some logically adjacent nodes may be mapped to physically non-adjacent nodes, thus quantum gates applied on them cannot be executable on physical devices. It is necessary to adjust the quantum circuits by adding auxiliary gates to swap qubits. We use Tabu search-based adjustment algorithm to generate circuits that can be physically executed.

Note that isomorphism and adjustment are both NP-complete [16]. Thus, we make use of some heuristics. Below we give a detailed consideration for each step.

A. Preprocessing

In the preprocessing step, we adjust the input circuit described by an openQASM program to shorten the life cycle of qubits. Then we use a Breadth-First Search (BFS) to calculate the shortest distance between each pair of nodes on the architecture graph.

We use a layered method to analyze the life cycle of qubits and pack the gates that can be executed in parallel into a $bundle$, forming a layered bundle format [25].

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 add a new layer. The notation $L(LC) = \{\mathcal{L}_0, \mathcal{L}_1, ..., \mathcal{L}_n\}$ denotes the layered circuit, where $\mathcal{L}_i$ $(0 \leq i \leq n)$ stands for a quantum gate set that can be executed in parallel. The quantum gate set separated by the dotted line in Fig. 2 are the following $\mathcal{L}_0 = \{g_0, g_1\}, \mathcal{L}_1 = \{g_2\}, \mathcal{L}_2 = \{g_3, g_4\}, \mathcal{L}_3 = \{g_5, g_6\}, \mathcal{L}_4 = \{g_7\}, \mathcal{L}_5 = \{g_8\}$.

At the same time of circuit layering, we generate a logical circuit architecture graph $\mathcal{AG}_{\mathcal{L}} = (V_L, E_L)$, which is an undirected graph with V_L being the set of vertices, and E_L the set of undirected edges that denote the connectivity between qubits related by CNOT gates. Given a physical architecture graph and assume the distance of each edge is 1, we can use Floyd-Warshall algorithm to calculate the shortest distance matrix $dist[i][j]$, which denotes the shortest distance from $\mathsf{q_i}$ to $\mathsf{q_j}$.

For IBM QX2, QX3, QX4, and QX5, the control of one qubit to a neighbour is unilateral. In this case, a SWAP gate can be implemented by using three CNOT gates and four H gates, as shown in Fig. 5. The four H gates are needed to change the direction of the middle CNOT gate. Consider a CNOT gate $g = \langle q_i, q_j \rangle$. If q_i and q_j are mapped to q_m and q_n, respectively, then the cost of executing g under the shortest path is $cost_{cnot}(q_i, q_j) = 7 \times (dist[m][n] - 1)$. For IBM Q20, where the control between two adjacent qubits is bilateral, a SWAP gate can be implemented by using three CNOT gates. Thus the cost is $cost_{cnot}(q_i, q_j) = 3 \times (dist[m][n] - 1)$.

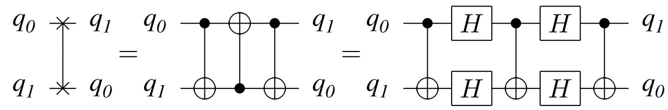


Fig. 5: Implementing a SWAP gate by using CNOT gates and H gates

Example 2: Take the QX5 (cf. Fig. 3 (d)) architecture as an example. Given a CNOT gate $g = \langle q_1, q_2 \rangle$, with q_1 mapped to q_1 and q_2 mapped to q_{14}, the shortest distance between them is $dist[1][14] = 3$. There are 3 shortest paths to move q_1 to the adjacent position of q_{14}: $\pi_0 = \mathsf{q}_1 \rightarrow \mathsf{q}_2 \rightarrow \mathsf{q}_3 \rightarrow \mathsf{q}_{14}$, $\pi_1 = \mathsf{q}_1 \rightarrow \mathsf{q}_2 \rightarrow \mathsf{q}_{15} \rightarrow \mathsf{q}_{14}$, $\pi_2 = \mathsf{q}_1 \rightarrow \mathsf{q}_0 \rightarrow \mathsf{q}_{15} \rightarrow \mathsf{q}_{14}$. Their costs are given by $cost_{\pi_0} = 18$, $cost_{\pi_1} = 14$, and $cost_{\pi_2} = 14$, respectively.

B. Isomorphism and Completion

Generally speaking, in a physical architecture graph, it is almost impossible to find a subgraph that exactly matches all the nodes in a logical architecture graph. We regard the mapping with the largest number of matching nodes as the partial mapping. SubgraphMatching 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 LFTJ based on set-intersection to enumerate the results. Since SubgraphMatching cannot handle disconnected graphs, we artificially create connected graphs by linking isolated nodes to the nodes with the largest degree in the logical architecture graph.

The input of Algorithm 1 is a target graph ($\mathcal{AG}_P$), a query graph ($\mathcal{AG}_L$), and the partial mapping set T. We initialize an empty queue Q. Then we traverse τ and add the unmapped nodes to the queue Q. For the unmapped nodes, we try to map them to the vicinity of the mapped nodes in $\mathcal{AG}_P$. If a node q is not mapped to any physical node, we need to perform such kind of mapping completion. Finally, we generate a dense mapping, which can reduce the added auxiliary gates. 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 query graph, it will be matched to one of the adjacent nodes first. In this way, we can obtain all initial candidate mappings.

Example 3: Following the previous example, we use the CSIC algorithm for the logical architecture graph given in Fig. 4 (a) and the physical architecture graph given in Fig. 3 (e) to obtain the partial mapping set $T = \{\tau_0, \tau_1, ..., \tau_n\}$. We take one of the partial mappings as an example.

$$\tau_0 = \{q_0 \rightarrow \mathsf{q}_{10}, q_1 \rightarrow -1, q_2 \rightarrow \mathsf{q}_6, q_3 \rightarrow \mathsf{q}_5, q_4 \rightarrow \mathsf{q}_{11}\}.$$

Algorithm 1: Complete initial mapping

Input: $\mathcal{AG_L}$: The architecture of logical circuit;
$\mathcal{AG_P}$: The architecture of physical circuit;
T: A partial mapping set obtained by SubgraphMatching;
Output: $result$: A collection of mapping relations between $\mathcal{AG_L}$ and $\mathcal{AG_P}$;

```
Initialize result = ∅;
l ← max_{τ∈T} τ.length;
for τ ∈ T do
    if l = τ.length then
        result.add(τ);
        Q ← an empty unmapped node queue;
        i ← 1;
        while i ≤ τ.length do
            Q.push({i, i ≤ τ.length });
            i ← i + 1;
        while Q is not empty do
            q ← Q.poll();
            targetAdj ← AG_P.adjacencyMatrix();
            queryAdj ← AG_L.adjacencyMatrix();
            cans ← an empty candidate node list
              sorted by degree;
            cans ∪ {q_m, q_m ≤ queryAdj[q].length};
            while cans is not empty do
                q ← τ[cans.first];
                k ← 0;
                cans ← cans\cans.first;
                while k < targetAdj[q].length do
                    if (targetAdj[q][k] ≠
                      −1 or targetAdj[k][q] ≠ −1)
                      and not τ.contains(k) then
                        τ[q] ← k;
                        break;
                    k ← k + 1;
                if k ≠ targetAdj[q].length then
                    break;
return result;
```

where $q_1 \rightarrow -1$ means that q_1 is not mapped to any physical node, so we need to perform a mapping completion. The maximum number of mapped nodes is 4. Next, we will demonstrate how τ_0 is completed. We add all unmapped nodes to the queue Q, $Q = \{q_1\}$. 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 $cans$, which is sorted by the connectivity of q_m and q. We get $cans = \{q_3, q_2, q_4, q_0\}$. Then, we traverse $cans$ and take out the first element q_3 in $cans$, and calculate the phycical node $\mathsf{q} = \mathsf{q}_5$, $\tau_0(q_3) = \mathsf{q}_5$. Finally, we map q to the node connected to q but not yet mapped. If the nodes connected to q have been mapped, the loop continues. In this example, it can be directly mapped to q_0. In the end, we obtain the mapping

$$\tau_0 = \{q_0 \rightarrow \mathsf{q}_{10}, q_1 \rightarrow \mathsf{q}_0, q_2 \rightarrow \mathsf{q}_6, q_3 \rightarrow \mathsf{q}_5, q_4 \rightarrow \mathsf{q}_{11}\}.$$

C. Adjustment

1) Tabu search: The Tabu search algorithm is a type of heuristic algorithm. It uses a tabu list to avoid searching repeated spaces, thereby avoiding deadlock. The algorithm uses amnesty rules to jump out of the local optimum to ensure the diversity of transformed results. The circuit adjustment mainly relies on the Tabu search algorithm, aiming to adjust those large circuits that the current algorithm is difficult to adjust and produce an output 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 in the current map are the neighborhoods. Tabu list avoids local optima and guarantees the parallelism of 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. We perform pruning to reduce search space, since only swaps adjacent to at least one gate node are meaningful. We select the edge in the shortest path that has an intersection with the qubits contained in the gate as a candidate set. Evaluation function selects an element from the candidate set that can make the distance of gates smaller. Amnesty rules are used when all objects in the candidate set are banned, or after banning an object, the target value will be greatly reduced.

The calculation of neighborhoods is shown in Algorithm 2. The input is the current circuit mapping τ_p. The set $qubits$ contains the mapping from physical qubits to logical qubits, where $j = qubits[i]$ means that the i-th physical qubit has been mapped to the j-th logical qubit. The set $locs$ denotes the mapping of logical qubits to physical qubits, where $j = locs[i]$ means that the i-th logical qubit has been mapped to the j-th physical qubit. The current layer list of all gates is cl, and the output is a candidate set of the current mapping. The set E contains the edges of all the shortest paths in the physical architecture graph of all the gates in the current layer.

Example 4: Let us consider the mapping

$$\tau_0 = \{q_0 \rightarrow \mathsf{q}_{10}, q_1 \rightarrow \mathsf{q}_0, q_2 \rightarrow \mathsf{q}_6, q_3 \rightarrow \mathsf{q}_5, q_4 \rightarrow \mathsf{q}_{11}\},$$

for $L_0 = \{g_0, g_1\}$, $dist_{cnot}(g_0) = 3$ and $dist_{cnot}(g_1) = 3$. Gate g_1 can be executed directly in the τ_0 mapping, so we delete it from L_0, but g_0 cannot be executed in the mapping τ_0. Thus, a circuit adjustment is required. Nodes that cannot be executed join the set $swap\_nodes = \{\mathsf{q}_0, \mathsf{q}_6\}$. The set of the shortest paths is

$$paths = \{\{\mathsf{q}_6 \rightarrow \mathsf{q}_1 \rightarrow \mathsf{q}_0\}, \{\mathsf{q}_6 \rightarrow \mathsf{q}_5 \rightarrow \mathsf{q}_0\}\},$$

and then 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 $\{(\mathsf{q}_6, \mathsf{q}_1), (\mathsf{q}_1, \mathsf{q}_0), (\mathsf{q}_6, \mathsf{q}_5), (\mathsf{q}_5, \mathsf{q}_0)\}$.

Algorithm 2: Calculate the candidate sets

Input: $dist$: The shortest paths of physical architecture;
$qubits$: The mapping from physical qubits to logical qubits;
$locs$: The mapping from logical qubits to physical qubits;
cl: Gates included in the current layer of circuits;
Output: $results$: The set of candidate solution;

Initialize $results \leftarrow \emptyset$;
$E_w \leftarrow$ Calculate the weight of each edge;
$swap\_nodes \leftarrow$ An empty set of candidate swap nodes;
foreach $g \in cl$ **do**
  if g is $executable$ **then**
    $cl \leftarrow cl \backslash \{g\}$;
  else
    $swap\_nodes.add(locs[g.control])$;
    $swap\_nodes.add(locs[g.target])$;
foreach $g \in cl$ **do**
  foreach $path \in paths[locs[g.control]][locs[g.target]]$ **do**
    foreach $e \in path$ **do**
      if $\{e.source,\ e.target\} \cap swap\_nodes \neq \emptyset$ **then**
        $new\_qubits \leftarrow qubits$;
        $new\_locs \leftarrow locs$;
        $q_1 \leftarrow new\_qubits[e.source]$;
        $q_2 \leftarrow new\_qubits[e.target]$;
        $new\_qubits[e.source] \leftarrow q_2$;
        $new\_qubits[e.target] \leftarrow q_1$;
        if $q_1 \neq -1$ **then**
          $new\_locs[q_1] \leftarrow q_2$;
        if $q_2 \neq -1$ **then**
          $new\_locs[q_2] \leftarrow q_1$;
        $s \leftarrow \emptyset$;
        $s.swaps \leftarrow p.swaps \cup \{distance.paths[e.source][e.target]\}$;
        $s.value \leftarrow evaluate(dist, new\_locs, cl)$;
        $results \leftarrow results \cup \{s\}$;
return $results$;

Algorithm 3: Tabu search

Input: τ_{ini}: The initial mapping
tl: Tabu list
Output: τ_{best}: The best mapping
1 **Initialize** $\tau_{best} \leftarrow \tau_{ini}$;
2 $iter \leftarrow 1$; `// Number of iterations`
3 **while** not $mustStop(iter, \tau_{best})$ **do**
4   $C \leftarrow \tau_{ini}.candidates()$; `// candidate set`
5   **if** C *is empty* **then**
6     $break$;
7   $C_{best} \leftarrow find\_best\_candidates(C, tl)$;
8   **if** C_{best} is $empty$ **then**
9     $C_{best} \leftarrow find\_amnesty\_candidates(C, tl)$;
10   $\tau_{best} \leftarrow C_{best}$;
11   $tl \leftarrow tl \cup \{C_{best}.swap\}$;
12   $iter \leftarrow iter + 1$;
13 return τ_{best}

The Tabu search-based adjustment algorithm takes a layered circuit and an initial mapping as input and outputs a circuit that can be executed in the specified architecture 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_{ini}. If not all the gates are executable or the number of iterations has not reached the given maximum number, the search will continue. Otherwise, the search will terminate. Line 4 gets the current mapping candidate, and Line 7 finds the best mapping in the candidate set. The mapping will first remove the overlapping elements of the candidate set and the tabu list. Then from the remaining candidates, we choose a mapping with the lowest cost. Line 9 takes the amnesty rules. When the best candidate is not found, all the elements in the candidate set will be put in the tabu list. The amnesty rules select the mapping with the lowest cost in the candidate set as the best candidate mapping. Lines 10-12 update the best mapping τ_{best} and the current mapping τ_{curr}, and add the SWAP performed by the best mapping to the tabu list tl, indicating that the SWAP has just been performed. 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 maximum number, or the current mapping ensures all the gates in the current layer can be executed.

Example 5: Let us continue the previous example. We start searching from the initial mapping. We need to get the candidate SWAP set and select the one with the lower evaluation scores. For $L_0 = \{g_0, g_1\}$, the candidate set is $\{(\mathsf{q}_6, \mathsf{q}_1), (\mathsf{q}_1, \mathsf{q}_0), (\mathsf{q}_6, \mathsf{q}_5), (\mathsf{q}_5, \mathsf{q}_0)\}$, and the costs are given as follows.

$$cost((\mathsf{q}_6, \mathsf{q}_1)) = 3.0,\ cost((\mathsf{q}_1, \mathsf{q}_0)) = 3.0,$$
$$cost((\mathsf{q}_6, \mathsf{q}_5)) = 3.0,\ cost((\mathsf{q}_5, \mathsf{q}_0)) = 3.0.$$

The algorithm will choose the first SWAP, the mapping becomes

$$\tau_0 = \{q_0 \rightarrow \mathsf{q}_{10}, q_1 \rightarrow \mathsf{q}_0, q_2 \rightarrow \mathsf{q}_1, q_3 \rightarrow \mathsf{q}_5, q_4 \rightarrow \mathsf{q}_{11}\}.$$

It can be seen that the current mapping ensures the executability of g_0. The algorithm continues to search for the next layer.

2) Evaluation functions: We can control the search direction by changing the evaluation functions. We test two evaluation functions: one uses the number of auxiliary gates in the generated circuit as the evaluation criterion (1), and the other uses the depth of the generated circuit as the evaluation criterion (2).

$$cost((\mathsf{q_m}, \mathsf{q_n})) = \sum_{g \in L} (dist[\tau(g.control)][\tau(g.target)]) \quad (1)$$

$$cost((\mathsf{q_m}, \mathsf{q_n})) = Depth(L) \quad (2)$$

where $cost((\mathsf{q_m}, \mathsf{q_n}))$ denotes the cost of executing all the gates of the current layer L after swapping $\mathsf{q_m}$ with $\mathsf{q_n}$. We only calculate the depth between the unmapped gates as in (1) or the distance of the unmapped gates as in (2).

3) Look ahead: We observe that the number of gates in each layer after layering is small. The output of the i-th layer, with $i < n$, 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 swapping gates, the swap only satisfies the requirement of the i-th layer, not necessarily the next layer. Therefore, we take the gates in the $(i+x)$-th layer, with $i+x < n$, into consideration, where x 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 δ, which controls the influence of the gates in the $(i+x)$-th layer. Heuristics show that for $x = 2,\ \delta = 0.9$, the final effect approaches the best. Our evaluation functions in (1) and (2) can be adjusted as (3) and (4), respectively.

$$\begin{aligned} cost((\mathsf{q_m}, \mathsf{q_n})) = &\sum_{g \in L_i} (dist[\tau(g.control)][\tau(g.target)]) + \\ &\delta \times \sum_{j=i}^{i+x} \sum_{g \in L_j} (dist[\tau(g.control)][\tau(g.target)]) \end{aligned} \quad (3)$$

$$cost((\mathsf{q_m}, \mathsf{q_n})) = Depth(L_i) + \delta \times Depth(\sum_{j=i}^{i+x} L_j). \quad (4)$$

4) Complexity: Given a logical circuit architecture graph $\mathcal{AG_L} = (V_L, E_L)$ and a physical circuit architecture graph $\mathcal{AG_P} = (V_P, E_P)$, we assume that the initial mapping is τ, the depth of the circuit is d, and the number of qubits is V_L. Tabu search deals with one layer at once, 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 $(|E_P| - 1)$ and the size of the candidate set is $(|E_P| - 1)$. Each swap will make the total distance between the gates smaller. In the worst case, the number of swaps is $(|E_P| - 1)^{|E_P| - 2}$, but our selection strategy will make the number of swaps significantly reduced. The time complexity in the worst case is O$(d \times (|E_P| - 1)^{(|E_P| - 2)})$, and the space complexity is the size of our candidate set $(E_P - 1)$, which is in PSPACE.

V. Experiments

We compare the CSIC algorithm and Tabu search-based adjustment TSA with the wghtgraph in [5] and the heuristic algorithm A^* in [6]. All the experiments are conducted on a Ubuntu machine with 2.2GHz CPU and 64G memory.

Firstly, we compare the efficiency of initial mapping on optm [6], CSIC and wghtgraph [5]. In order to observe the results of these two initial mapping algorithms, we used the same circuit adjustment algorithm A^* [6]. We test 159 circuits. Within five minutes, optm, wghtgraph, and CSIC can transform 125, 107, and 135 circuits, respectively. We then compare the wghtgraph algorithm and the CSIC algorithm more closely. The wghtgraph algorithm has 21 circuits with fewer auxiliary gates and 19 circuits with smaller depths, and the CSIC algorithm has 54 circuits with fewer auxiliary gates and 60 circuits with smaller depths. They output 27 circuits with equal depth and 31 circuits with equal auxiliary gates. On average, the auxiliary gates of the CSIC algorithm are reduced by 22.43%, and the depths are reduced by 8.46%.

Secondly, we compare the optm algorithm with the CSIC algorithm. The optm algorithm has 1 circuit with fewer auxiliary gates and 2 circuits with a small depth, while the CSIC algorithm has 101 circuits with fewer auxiliary gates and 100 circuits with a small depth. They have 4 circuits with equal depth and 4 circuits with equal auxiliary gates. The auxiliary gates of the CSIC algorithm are relatively reduced by 27.14%, and the depths are reduced by 10.74%. Table I shows the experimental data.The three initial mapping algorithms are compared according to the depth of the generated circuits using the same A^* algorithm, and the numbers of added auxiliary gates. The column headed by CSIC /optm shows the efficiency improvement of the former upon the latter $(n_{optm} - n_{CSIC})/n_{optm}$.

We then compare in Table II the use of two indicators TSA$_{dep}$ and TSA$_{num}$ that prioritize smaller depths and fewer auxiliary gates, respectively. Using the two indicators as objective functions, we test 159 circuits. The depth of the final circuits obtained by TSA$_{num}$ are 7.70% smaller than TSA$_{dep}$ on average, and the numbers of auxiliary gates added are 26.95% smaller on average. Inserting a SWAP gate requires adding 3 CNOT gates, and the depth will increase by 3. Therefore, the circuit adds less depth, since the number of SWAP gates added is less.

Finally, we compare TSA with wgtgraph. Since the wgtgraph algorithm only uses 2-qubit gates, it is impossible to compare the depth of the generated circuits. Instead, we compare the number of SWAP gates added and the time costs. We set a five-minute timeout period and test 159 circuits. It turns out that TSA$_{num}$ only takes 135 seconds and TSA$_{dep}$ takes 159 seconds. The wgtgraph algorithm takes overtime for the 159 circuts, but only produces valid results for 107 circuits, including 66 small circuits, 41 medium circuits, and no circuit output is produced for any of the 44 large circuits. Although Tabu search can quickly produce results on large circuits, in contrast, more auxiliary gates are added. In the generated circuits obtained by wgtgraph from 107 small and medium circuits, the number of SWAP gates added by wgtgraph is 38.97% smaller than TSA$_{num}$ on average. The Tabu search can quickly output transformed circuits on large circuits, but wgtgraph cannot get results. Since our candidate set is too small to be able to process the circuit quickly, this also leads

	optm	wgtgraph	CSIC	CSIC /optm	CSIC /wgtgraph
Depths	132709	129399	118450	10.74%	8.46%
Auxiliary gates	20477	19232	14918	27.14%	22.43%

TABLE I: Comparison of optm, wgtgraph and CSIC

benchmarks	#circ.	TSA$_{num}$		TSA$_{dep}$		wgtgraph		SABRE	
		#succ.	time	#succ.	time	#succ.	time	#succ.	time
small	66	66	5	66	5	66	254	19	9651
medium	49	49	8	49	8	41	1595	5	13270
large	44	44	122	44	146	0	-	-	-
total	159	159	135	159	159	107	-	-	-

TABLE II: Comparison of TSA$_{num}$, TSA$_{dep}$, wghtgraph and SABRE

to an increment in the insertion of additional gates. As to SABRE, when dealing with 159 circuits with a five-minute limit for each circuit, it successfully produces results for only 19 small circuits, 5 medium circuits. The detailed results of the circuit comparisons are in the Appendix.

VI. Conclusion

We have proposed a scalable algorithm for quantum circuit transformation. We first use a subgraph isomorphism algorithm and a mapping completion method based on the connectivity between qubits to generate a high-quality initial mapping. Then we exploit a look-ahead heuristic search taking into account the influence of the pre-layer circuit mapping to reduce the number of auxiliary gates, and complete the transformation. We have compared the influence of the initial mapping with the state-of-the-art algorithms wghtgraph and optm and also compared the overall efficiency with optm, wghtgraph, and SABRE. Experimental results have showed that the initial mapping generated by CSIC has fewer SWAP gates added and the results can be obtained in an acceptable amount of time. Most small and medium circuits can be transformed in a few seconds. For large circuits, the results can be obtained within a few minutes, but the cost of insertion may be lareger than wgtgraph. We have introduced a look-ahead method to make each selected SWAP more in line with the constraints of the gates to be executed. In the future, we will investigate how to reduce the number of inserted auxiliary gates and increase the speed. We will also apply the proposed method to more NISQ devices.

References

[1] [Online]. Available: https://www.ibm.com/quantum-computing/

[2] [Online]. Available: https://www.qiskit.org/

[3] 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.

[4] F. W. Glover, "Tabu search - part II," *Informs Journal on Computing*, vol. 2, no. 1, pp. 4–32, 1990.

[5] S. Li, Z. Xiangzhen, and Y. Feng, "Qubit mapping based on subgraph isomorphism and filtered depth-limited search," *IEEE Transactions on Computers*, pp. 1–1, 2020.

[6] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, no. 7, pp. 1226–1236, 2019.

[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. Paler, "On the influence of initial qubit placement during NISQ circuit compilation," 2018, abs/1811.08985.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] P. Murali, N. M. Linke, M. Martonosi, A. Javadi-Abhari, N. H. Nguyen, and C. H. Alderete, "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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] O. Daei, K. Navi, and M. Z. Moghadam, "Optimized quantum circuit partitioning," *CoRR*, vol. abs/2005.11614, 2020.

[23] 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.

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

[25] 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.

APPENDIX

EXPERIMENTAL DETAILS

In the following tables, qubit no. denotes the number of qubits in the circuit, CNOT no. denotes the number of initial CNOT gates, TSA$_{num}$, TSA$_{dep}$, optm, wghtgr, SABRE denote the numbers of CNOT gates added by the TSA using their respective circuit transformation methods. The symbol '-' means that the circuit cannot be successfully transformed within five minutes.

Tables III and IV show the circuits that TSA$_{num}$, TSA$_{dep}$, optm, wghtgr can successfully transform, and calculate the total numbers of gates added by them. Table V shows the circuits that optm, wghtgr, and SABRE may not be able to transform. Tables VI and VII show the depths of the circuits after TSA$_{num}$, TSA$_{dep}$, optm can successfully transform them. Table VIII shows the circuits that optm cannot transform.

Circuit name	qubit no.	CNOT no.	TSA$_{num}$ added	TSA$_{dep}$ added	optm added	wghtgr added	SABRE added
decod24-enable_126	6	149	27	50	60	16	-
4mod5-v0_19	5	16	0	0	0	0	11
4mod5-v0_18	5	31	2	5	4	2	-
mod5d2_64	5	25	4	5	8	3	-
4gt4-v0_72	6	113	14	10	33	13	60
alu-v3_35	5	18	2	4	8	3	12
4gt4-v0_73	6	179	25	40	76	19	-
alu-v3_34	5	24	2	3	7	3	28
3_17_13	3	17	0	0	6	0	-
4gt4-v0_78	6	109	12	8	48	4	281
4gt4-v0_79	6	105	16	17	48	3	-
4mod7-v1_96	5	72	16	19	27	6	-
mod10_171	5	108	17	20	39	8	-
ex2_227	7	275	46	55	121	25	-
mod10_176	5	78	14	14	38	5	-
0410184_169	14	104	11	13	49	1	-
4mod5-v0_20	5	10	0	0	4	0	12
aj-e11_165	5	69	8	8	33	6	-
alu-v1_28	5	18	2	4	11	3	-
f2_232	8	525	92	104	218	72	-
4gt12-v0_86	6	116	8	31	48	3	-
4gt12-v0_87	6	112	7	30	45	2	148
4gt12-v0_88	6	86	8	13	25	6	-
alu-v1_29	5	17	4	4	11	3	-
ham7_104	7	149	30	31	68	16	-
C17_204	7	205	26	53	99	23	-
xor5_254	6	5	0	0	1	0	-
hwb4_49	5	107	14	15	38	10	73
rd73_140	10	104	25	29	35	19	-
decod24-v0_38	4	23	0	0	6	0	12
rd53_131	7	200	21	39	98	12	-
rd53_133	7	256	39	55	102	17	-
rd53_135	7	134	22	33	38	22	-
sys6-v0_111	10	98	18	30	38	15	-
decod24-v2_43	4	22	0	0	9	0	36
rd53_138	8	60	13	19	23	8	-
rd32-v0_66	4	16	0	0	6	0	21
sym9_146	12	148	31	52	54	28	-
4gt13-v1_93	5	30	0	0	13	0	-
graycode6_47	6	5	0	0	0	0	-
4mod5-bdd_287	7	31	2	6	8	2	-
ham3_102	3	11	0	0	3	0	12
4gt4-v0_80	6	79	8	14	22	4	-
ex-1_166	3	9	0	0	3	0	-
mod5mils_65	5	16	0	0	6	0	16
0example	5	9	1	2	5	1	-
alu-v4_36	5	51	12	8	22	2	108
alu-v4_37	5	18	2	4	8	3	21
ex1_226	6	5	0	0	1	0	6
one-two-three-v0_98	5	65	11	13	32	5	-
one-two-three-v0_97	5	128	21	23	64	12	-
one-two-three-v3_101	5	32	3	4	14	3	-
rd32_270	5	36	3	3	6	4	-
rd53_130	7	448	89	100	190	52	-
rd53_251	8	564	74	131	230	44	-
4mod5-v1_24	5	16	0	0	3	0	-
mod5adder_127	6	239	21	47	111	21	-
4_49_16	5	99	20	16	40	6	-
hwb5_53	6	598	92	168	173	57	-
ex3_229	6	175	8	7	50	8	-
4gt10-v1_81	5	66	13	15	28	4	-
alu-v2_32	5	72	14	17	27	4	-
alu-v2_31	5	198	39	51	85	16	-
alu-v2_30	6	223	38	45	96	17	-
sf_276	6	336	51	38	138	12	-
decod24-v1_41	5	38	4	4	14	1	-
sf_274	6	336	10	21	82	11	370
4gt4-v1_74	6	119	9	24	37	6	-
alu-v2_33	5	17	4	4	8	2	14
cm152a_212	12	532	101	150	168	42	-
cnt3-5_179	16	85	6	6	35	1	-

TABLE III: Comparison of the numbers of SWAP gates added by the output circuits on IBM Q20

Circuit name	qubit no.	CNOT no.	TSA_{num} added	TSA_{dep} added	optm added	wghtgr added	SABRE added
sym6_316	14	123	32	33	56	18	-
4mod5-v1_22	5	11	0	0	5	0	21
4mod5-v1_23	5	32	5	5	4	2	-
mini_alu_305	10	77	11	19	28	9	-
alu-v0_26	5	38	7	10	13	4	-
alu-bdd_288	7	38	4	11	16	5	-
alu-v0_27	5	17	2	4	11	3	-
4gt13_91	5	49	7	7	10	2	-
4gt5_77	5	58	12	12	20	3	-
4gt13_92	5	30	0	0	14	0	51
4gt5_76	5	46	7	10	24	7	-
4gt5_75	5	38	5	11	16	3	-
4gt12-v1_89	6	100	19	23	38	9	-
one-two-three-v1_99	5	59	11	10	26	5	-
4gt13_90	5	53	7	7	13	3	53
ising_model_10	10	90	0	0	27	0	-
ising_model_13	13	120	0	0	9	0	-
4gt11_84	5	9	0	0	3	0	12
4gt11_83	5	14	0	0	0	0	-
mod5d1_63	5	13	0	0	1	0	28
4gt11_82	5	18	1	1	1	1	-
ising_model_16	16	150	0	0	5	0	-
decod24-v3_45	5	64	14	14	32	5	-
rd32-v1_68	4	16	0	0	6	0	-
mini-alu_167	5	126	27	27	49	10	-
one-two-three-v2_100	5	32	3	4	8	3	-
4mod7-v0_94	5	72	8	14	36	4	-
cm82a_208	8	283	45	75	84	16	-
mod8-10_178	6	152	5	13	13	25	-
mod8-10_177	6	196	14	25	58	34	-
majority_239	7	267	39	43	105	28	-
qft_10	10	90	23	34	30	15	-
miller_11	3	23	0	0	9	0	36
decod24-bdd_294	6	32	3	3	9	4	-
con1_216	9	415	75	101	177	56	-
total	664	11540	1618	2250	4260	990	-

TABLE IV: Comparison of the numbers of SWAP gates added by the output circuits on IBM Q20

Circuit name	qubit no.	CNOT no.	TSA$_{num}$ added	TSA$_{dep}$ added	optm added	wghtgr added	SABRE added
max46_240	10	11844	2648	3762	-	-	-
rd73_252	10	2319	521	751	708	-	-
cycle10_2_110	12	2648	712	1026	961	-	-
sqrt8_260	12	1314	378	481	457	-	-
urf4_187	11	224028	45463	60958	-	-	-
sqn_258	10	4459	1072	1455	-	-	-
radd_250	13	1405	342	458	511	-	-
ham15_107	15	3858	1228	1894	-	-	-
sao2_257	14	16864	5076	7050	-	-	-
sym9_148	10	9408	1812	2746	-	-	-
urf5_280	9	23764	-	-	-	-	-
square_root_7	15	3089	858	2212	-	-	-
hwb7_59	8	10681	2360	3539	3722	-	-
wim_266	11	427	74	118	147	-	-
urf2_152	8	35210	8458	11855	10577	-	-
urf5_158	9	71932	19415	25309	-	-	-
urf2_277	8	10066	2706	3760	3782	-	-
life_238	11	9800	2593	3579	-	-	-
root_255	13	7493	1857	2929	-	-	-
9symml_195	11	15232	4053	5660	-	-	-
sym10_262	12	28084	7806	11259	-	-	-
dc1_220	11	833	164	193	371	-	-
cm42a_207	14	771	129	211	294	-	-
rd53_311	13	124	37	49	51	-	-
dc2_222	15	4131	1438	1718	-	-	-
rd84_142	15	154	42	64	50	-	-
z4_268	11	1343	294	435	-	-	-
sym6_145	7	1701	250	449	750	-	-
co14_215	15	7840	2587	3535	-	-	-
cnt3-5_180	16	215	47	73	79	-	-
mlp4_245	16	8232	2771	3872	-	-	-
hwb8_113	9	30372	7057	10211	-	-	-
qft_16	16	240	77	147	-	-	-
plus63mod4096_163	13	56329	17748	23875	-	-	-
urf1_149	9	80878	21602	29125	-	-	-
urf3_155	10	185276	48390	63552	-	-	-
urf3_279	10	60380	16975	23211	-	-	-
hwb9_119	10	90955	21751	29612	-	-	-
plus63mod8192_164	14	81865	26663	34999	-	-	-
pm1_249	14	771	129	211	294	-	-
sym9_193	11	15232	4053	5660	-	-	-
misex1_241	15	2100	358	612	600	-	-
urf1_278	9	26692	7238	10150	-	-	-
squar5_261	13	869	181	283	290	-	-
ground_state_estimation_10	13	154209	13467	22255	15221	-	-
adr4_197	13	1498	417	532	-	-	-
hwb6_56	7	2952	619	933	909	-	-
clip_206	14	14772	4362	6089	-	-	-
cm85a_209	14	4986	1358	1874	-	-	-
rd84_253	12	5960	1704	2381	-	-	-
dist_223	13	16624	4665	6235	-	-	-
inc_237	16	4636	1048	1714	-	-	-
urf6_160	15	75180	24711	33358	-	-	-

TABLE V: Comparison of the numbers of SWAP gates added by the output circuits on IBM Q20

Circuit name	qubit no.	CNOT no.	depths no.	TSA$_{num}$ depths	TSA$_{dep}$ depths	optm depths
decod24-enable_126	6	149	190	230	299	358
4mod5-v0_19	5	16	21	16	16	21
4mod5-v0_18	5	31	40	37	46	48
mod5d2_64	5	25	32	37	40	53
4gt4-v0_72	6	113	137	155	143	233
alu-v3_35	5	18	22	24	30	44
4gt4-v0_73	6	179	227	254	299	442
alu-v3_34	5	24	30	30	33	49
3_17_13	3	17	22	17	17	40
4gt4-v0_78	6	109	137	145	133	266
4gt4-v0_79	6	105	132	153	156	259
4mod7-v1_96	5	72	94	120	129	168
mod10_171	5	108	139	159	168	257
ex2_227	7	275	355	413	440	679
mod10_176	5	78	101	120	120	204
rd73_252	10	2319	2867	3882	4572	4601
cycle10_2_110	12	2648	3386	4784	5726	5826
0410184_169	14	104	104	137	143	191
4mod5-v0_20	5	10	12	10	10	24
sqrt8_260	12	1314	1661	2448	2757	2787
aj-e11_165	5	69	86	93	93	184
alu-v1_28	5	18	22	24	30	50
f2_232	8	525	668	801	837	1268
radd_250	13	1405	1781	2431	2779	3093
4gt12-v0_86	6	116	135	140	209	254
4gt12-v0_87	6	112	131	133	202	246
4gt12-v0_88	6	86	108	110	125	176
alu-v1_29	5	17	22	29	29	46
ham7_104	7	149	185	239	242	367
C17_204	7	205	253	283	364	512
xor5_254	6	5	5	5	5	8
hwb4_49	5	107	134	149	152	236
rd73_140	10	104	92	179	191	147
decod24-v0_38	4	23	30	23	23	49
rd53_131	7	200	261	263	317	509
rd53_133	7	256	327	373	421	596
rd53_135	7	134	159	200	233	261
sys6-v0_111	10	98	75	152	188	142
decod24-v2_43	4	22	30	22	22	57
hwb7_59	8	10681	13437	17761	21298	23025
rd53_138	8	60	56	99	117	90
rd32-v0_66	4	16	20	16	16	39
sym9_146	12	148	127	241	304	235
4gt13-v1_93	5	30	39	30	30	76
graycode6_47	6	5	5	5	5	5
wim_266	11	427	514	649	781	912
urf2_152	8	35210	44100	60584	70775	71645
urf2_277	8	10066	11390	18184	21346	20336
4mod5-bdd_287	7	31	41	37	49	59
ham3_102	3	11	13	11	11	22
4gt4-v0_80	6	79	101	103	121	164
ex-1_166	3	9	12	9	9	22
mod5mils_65	5	16	21	16	16	40
0example	5	9	6	12	15	15
alu-v4_36	5	51	66	87	75	128
alu-v4_37	5	18	22	24	30	44
ex1_226	6	5	5	5	5	8
one-two-three-v0_98	5	65	82	98	104	174
one-two-three-v0_97	5	128	163	191	197	331
one-two-three-v3_101	5	32	40	41	44	73
rd32_270	5	36	47	45	45	64
dc1_220	11	833	1041	1325	1412	2035
rd53_130	7	448	569	715	748	1073
rd53_251	8	564	712	786	957	1341
cm42a_207	14	771	940	1158	1404	1739
rd53_311	13	124	130	235	271	234
4mod5-v1_24	5	16	21	16	16	30
mod5adder_127	6	239	302	302	380	609
4_49_16	5	99	125	159	147	240
hwb5_53	6	598	758	874	1102	1234
ex3_229	6	175	226	199	196	366

TABLE VI: Comparison of the depth of the output circuits on IBM Q20

Circuit name	qubit no.	CNOT no.	depths no.	TSA$_{num}$ depths	TSA$_{dep}$ depths	optm depths
rd84_142	15	154	110	280	346	195
4gt10-v1_81	5	66	84	105	111	158
alu-v2_32	5	72	92	114	123	163
alu-v2_31	5	198	255	315	351	490
alu-v2_30	6	223	285	337	358	552
sym6_145	7	1701	2187	2451	3048	4294
sf_276	6	336	435	489	450	830
decod24-v1_41	5	38	50	50	50	92
sf_274	6	336	436	366	399	663
4gt4-v1_74	6	119	154	146	191	259
alu-v2_33	5	17	22	29	29	43
cnt3-5_180	16	215	209	356	434	372
cm152a_212	12	532	684	835	982	1125
cnt3-5_179	16	85	61	103	103	124
sym6_316	14	123	135	219	222	280
4mod5-v1_22	5	11	12	11	11	27
4mod5-v1_23	5	32	41	47	47	49
mini_alu_305	10	77	71	110	134	139
alu-v0_26	5	38	49	59	68	83
alu-bdd_288	7	38	48	50	71	86
alu-v0_27	5	17	21	23	29	45
4gt13_91	5	49	61	70	70	88
4gt5_77	5	58	74	94	94	132
4gt13_92	5	30	38	30	30	77
4gt5_76	5	46	56	67	76	125
4gt5_75	5	38	47	53	71	95
4gt12-v1_89	6	100	130	157	169	239
one-two-three-v1_99	5	59	76	92	89	146
4gt13_90	5	53	65	74	74	100
pm1_249	14	771	940	1158	1404	1739
ising_model_10	10	90	52	90	90	116
ising_model_13	13	120	46	120	120	113
misex1_241	15	2100	2676	3174	3936	4258
4gt11_84	5	9	11	9	9	19
4gt11_83	5	14	16	14	14	16
mod5d1_63	5	13	13	13	13	15
4gt11_82	5	18	20	21	21	23
ising_model_16	16	150	57	150	150	85
squar5_261	13	869	1051	1412	1718	1801
decod24-v3_45	5	64	84	106	106	180
ground_state_estimation_10	13	154209	217236	194610	220974	260172
rd32-v1_68	4	16	21	16	16	40
hwb6_56	7	2952	3736	4809	5751	6129
mini-alu_167	5	126	162	207	207	304
one-two-three-v2_100	5	32	40	41	44	64
4mod7-v0_94	5	72	92	96	114	198
cm82a_208	8	283	340	418	508	553
mod8-10_178	6	152	193	167	191	221
mod8-10_177	6	196	251	238	271	415
majority_239	7	267	344	384	396	639
qft_10	10	90	37	159	192	103
miller_11	3	23	29	23	23	57
decod24-bdd_294	6	32	40	41	41	68
con1_216	9	415	508	640	718	919
total	884	240309	323327	338085	389022	440477

TABLE VII: Comparison of the depth of the output circuits on IBM Q20

Circuit name	qubit no.	CNOT no.	depths no.	TSA_{num} depths	TSA_{dep} depths	optm depths
max46_240	10	11844	14257	19788	23130	-
urf4_187	11	224028	264330	360417	406902	-
sqn_258	10	4459	5458	7675	8824	-
ham15_107	15	3858	4819	7542	9540	-
sao2_257	14	16864	19563	32092	38014	-
sym9_148	10	9408	12087	14844	17646	-
square_root_7	15	3089	3847	5663	9725	-
urf5_158	9	71932	89148	130177	147859	-
life_238	11	9800	12511	17579	20537	-
root_255	13	7493	8839	13064	16280	-
9symml_195	11	15232	19235	27391	32212	-
sym10_262	12	28084	35572	51502	61861	-
dc2_222	15	4131	5242	8445	9285	-
z4_268	11	1343	1644	2225	2648	-
co14_215	15	7840	8570	15601	18445	-
mlp4_245	16	8232	10328	16545	19848	-
hwb8_113	9	30372	38717	51543	61005	-
qft_16	16	240	61	471	681	-
plus63mod4096_163	13	56329	72246	109573	127954	-
urf1_149	9	80878	99586	145684	168253	-
urf3_155	10	185276	229365	330446	375932	-
urf3_279	10	60380	70702	111305	130013	-
hwb9_119	10	90955	116199	156208	179791	-
plus63mod8192_164	14	81865	105142	161854	186862	-
sym9_193	11	15232	19235	27391	32212	-
urf1_278	9	26692	30955	48406	57142	-
adr4_197	13	1498	1839	2749	3094	-
clip_206	14	14772	17879	27858	33039	-
cm85a_209	14	4986	6374	9060	10608	-
rd84_253	12	5960	7261	11072	13103	-
dist_223	13	16624	19694	30619	35329	-
inc_237	16	4636	5864	7780	9778	-
urf6_160	15	75180	93645	149313	175254	-

TABLE VIII: Comparison of the depth of the output circuits on IBM Q20