

Quantum Circuit Transformation Based on Subgraph Isomorphism and Tabu Search^{*}

First Aaaaaauthor¹[0000-*ereer*1111-2222-3333], Second Author^{2,3}[1111-2222-3333-4444], and Third Author³[2222--3333-4444-5555]

¹ Princeton University, Princeton NJ 08544, USA

² Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany

lncs@springer.com

<http://www.springer.com/gp/computer-science/lncs>

³ ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
{abc,lncs}@uni-heidelberg.de

Abstract. The goal of quantum circuit adjustment 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 keys steps: one makes use of a combined subgraph isomorphism (CSI) to initialize a mapping, the other dynamically adjusts the mapping by using a Tabu search based adjustment (TSA). Our experiments show that, compared with the *wghtgraph* recently considered in the literature, CSI can save 22.26% of auxiliary gates and reduce the depths of output circuits by 11.76% on average in the initialization of the mapping, and TSA has a better scalability than many state-of-the-art algorithms for adjusting mappings.

Keywords: Quantum circuit transformation · Subgraph isomorphism
· Initial mapping · Tabu search

1 Introduction

The entanglement of a quantum system with its surrounding environment will lead to quantum decoherence. It is unrealistic to use quantum error correction in the circuit mapping process, since there are only dozens of available qubits for quantum devices in the NISQ era [16]. It is necessary to transform circuits by adding auxiliary gates to satisfy both logical and physical constraints, since quantum algorithms do not consider any hardware connectivity constraints. Hence, quantum circuit transformation is an important part of quantum circuit compilation. We need a set of highly efficient and automatic mapping procedures and adjustment routines to perform circuit transformation. In this process noise may be introduced, which brings a huge challenge to circuit compilation because noise has a significant impact on the final circuit and may make the result meaningless.

^{*} Supported by organization x.

In the current work, we adjust the lifetime of qubits through parallelization, and use SubgraphCompare [20] to generate partial isomorphic subgraphs of logical circuits LC and physical circuits PC as part of the initial mapping. The advantage of the initial mapping result is that we use the appropriate subgraph isomorphism and the two-way connection of the logical circuit and the physical circuit to obtain a dense initial mapping, which avoids certain nodes from being mapped to remote locations. We use Tabu search [6] to generate circuits that can be executed on physical devices. Tabu search can avoid falling into local optimum and swapping the recently swapped qubits, thereby improve the parallelism of quantum gates. We add the SWAPs associated with the gates on the shortest path to the candidate set, which greatly reduces the search space and improves the search speed. Our heuristic function not only considers the current gates but also the constraints of the gates already considered.

We compared CSI with the start-of-art initial mapping method wgtgraph [10] and optm [25]. On average, the auxiliary gates of the CSI algorithm are reduced by 22.44%, 27.02%, and the depths are reduced by 11.25%, 14.12%, respectively. We Compared with wgtgraph [10] and SABRE [9], the adjustment algorithm is compared. In contrast, TSA can deal 159 circuits in a few minutes, but other algorithms are difficult to handle middle or large circuits. The number of SWAP gates added by wgtgraph is 26.87% (resp. 24.89%) smaller than TSA_{num} (resp. TSA_{dep}) on average. SABRE tested 159 circuits and only 29 circuits were successfully mapped within the five-minute limit.

The main contributions of this paper are as follows.

1. We propose to use the combined subgraph isomorphism algorithm (CSI) to generate part of the initial mapping and then complete the mapping based on the connectivity between qubits.
2. We propose a heuristic circuit adjustment algorithm based on Tabu search (TSA) [6], which can handle large circuits in a shorter time at a lower cost, compared with existing precise search and heuristic algorithms.
3. We propose a look-ahead heuristic function that considers both the current gates and the gates yet to be processed. It filters out a swap that is beneficial to the current gates and also brings closer the gates to be processed.
4. We tested 159 circuits, and the results show that the initial mapping generated by our method can make the subsequent insertion of fewer gates, and the adjustment algorithm can be extended to medium and large circuits.

The rest of this paper is organized as follows. In Section 2 we introduce some related works. In Section 3 we recall some background of quantum computing and quantum information. In Section 4 we introduce the problem of the transformation of quantum circuits. In Section 5 we describe and analyze our algorithm in detail. The experimental results are reported in Section 6. The last section concludes the paper and discusses some future research.

2 Related work

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 theoretical 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 provides the public with free quantum computer resources on the cloud and opens source the quantum computing software framework *Qiskit* ⁴.

Users of these early quantum computers mainly rely on quantum circuits to implement quantum algorithms. They design logical circuits which then go through the step of circuit transformation in order to map 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.

There are several initial mapping methods. Paler [15] 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 in the actual circuit instance on the actual NISQ device, the time cost can be reduced by up to 10%. Li et al. [9] proposed a novel reverse traversal technique, which determines the initial mapping by considering the entire circuit. Zhou et al. [23] put forward an annealing algorithm to find an initial mapping, but it is unstable. In [10], Li et al. considered the subgraph isomorphism algorithm *wghtgraph* to generate an initial mapping, which is currently the most start-of-art, so we will compare it.

The goal of circuit adjustment algorithm is to find a minimum number of SWAPs. There are currently five main methods for solving the quantum circuit adjustment problem.

Unitary matrix decomposition algorithm. It is used in [8, 14] to rearrange the quantum circuit from the beginning while retaining the input circuit. It can be applied to a broader 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 [22, 3], Integer Linear Programming (ILP) [1], or Satisfiability Modulo Theories (SMT) [12]. 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 deal with small quantum circuits.

Exact methods. Siraichi et al. [19] have proposed an exact method. It will iterate all possible mappings for all dependencies, so it is only suitable for simple

⁴ <https://www.qiskit.org/>

quantum architecture and cannot be extended to complex quantum architectures.

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 idea is to first divide a given circuit into several subcircuits and apply the minimum linear permutation solution, respectively. Then we turn non-adjacent gates in the subcircuits into adjacent gates by adding auxiliary gates. Finally, we can use the minimum linear permutation solution to find an appropriate permutation and use bubble sort to calculate the number of necessary SWAP gates. In [7, 11], 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 [25] suggests to layer the circuits, then determines compatible mappings for each of these layers to add as few auxiliary gates as possible. Zhou et al. [23] 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 forward one step and then choose the best continuous operation. In this way, the algorithm can effectively avoid local minimum. Moreover, a pruning mechanism is introduced to reduce the search space’s size and ensure that the program terminates in a reasonable amount of time.

Li et al. [9] have proposed a SWAP-based search algorithm SABRE. Compared with previous search algorithms based on exhaustive mapping, SABRE ensures the scalability of SABRE to adapt to large quantum circuits in the NISQ era. In [4], 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 finally 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 handle 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.

3 Background

This section introduces some notions and notations of quantum computing and quantum information.

Qubits 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 letters \mathbf{q} and q in different fonts to represent physical qubits and logical qubits.

Quantum Gates By inserting quantum gates, the state of the qubits is changed. Some commonly used quantum gate symbols and their semantics in terms of matrices are shown in Fig. 1.

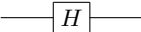
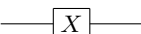
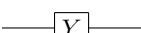
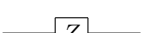
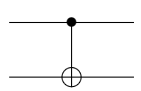
| | | |
|---------------|--|--|
| Hadamard gate |  | $\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ |
| Pauli-X gate |  | $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ |
| Pauli-Y gate |  | $\begin{bmatrix} 1 & -i \\ i & 0 \end{bmatrix}$ |
| Pauli-Z gate |  | $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ |
| CNOT gate |  | $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ |

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

Quantum Circuit A quantum logical circuit consists of quantum gates interconnected by quantum wires [5]; see Fig. 2 for an example. A quantum wire is a mechanism for moving quantum data from one location to another. Each line represents 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 w of a circuit refers to the number of qubits in the circuit. The depth d 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 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 adjust, since the single qubits are *local* [18].

Architectures We mainly discuss the physical circuits of IBM Q series. Let $\mathcal{AG}_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ denote the architecture graph of the physical circuit, where $V_{\mathcal{P}}$ denotes the set of physical qubits and $E_{\mathcal{P}}$ denotes the set of edges that connect CNOT gates. In Fig. 6, (a) and (b) are the physical architecture graphs of

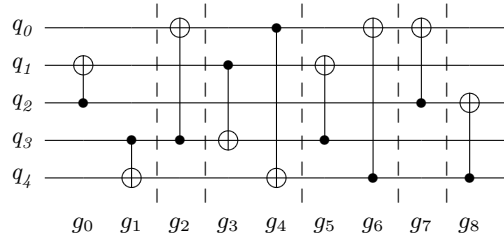


Fig. 2. Original circuit

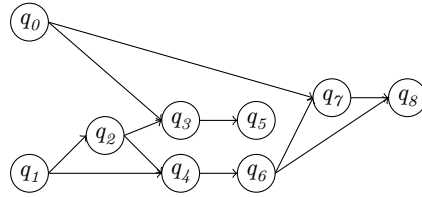


Fig. 3. The directed acyclic graph (DAG) of original circuit in Fig. 2.

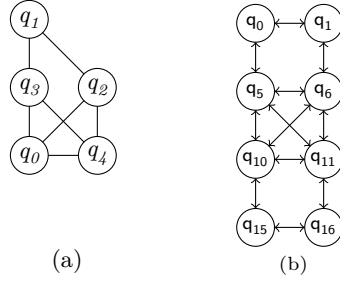


Fig. 4. (a) The architecture graph of original circuit in Fig. 2. (b) The partial architecture graph of IBM Q20.

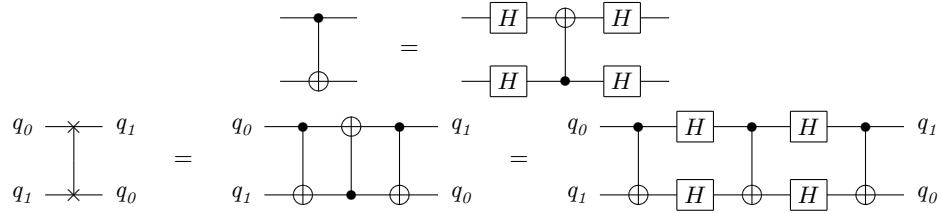


Fig. 5. The above circuits changes the direction of the CNOT gate by adding four H gates, and three equivalent circuits of SWAP gate are given below.

the 5-qubit IBM QX2 and IBM QX4, respectively. (d) and (c) are the physical architecture graphs of the 16-qubit IBM QX3 and IBM QX5, respectively, and (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 qubits with edges connected. IBM physical circuits only support single quantum gates and CNOT gates between two adjacent qubits.

Given a logical circuit LC , a physical structure \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, 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 structure of Fig. 2. Fig. 4 (b) is the partial architecture graph of IBM Q20. An initial mapping is

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

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 \mathcal{AG}_P . But $g_3 = \langle q_1, q_3 \rangle$ is executable, since the edge $\langle \tau(q_1), \tau(q_3) \rangle = \langle q_0, q_5 \rangle$ exists in \mathcal{AG}_P .

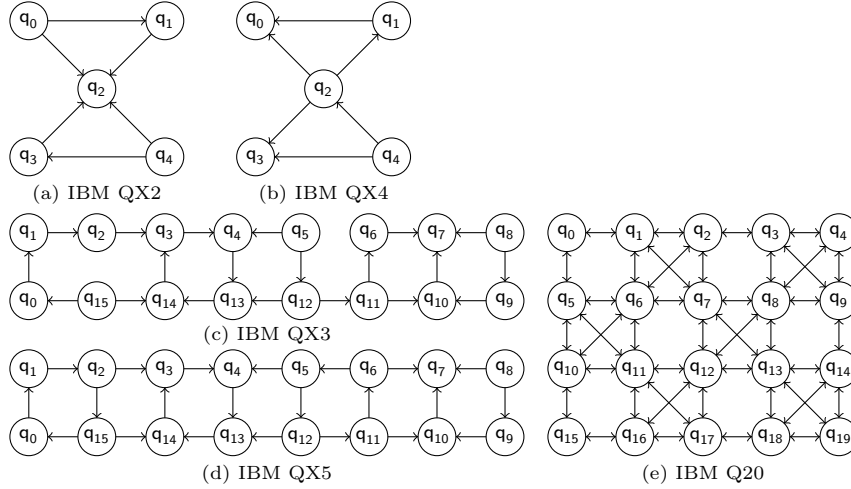


Fig. 6. IBM QX architectures

4 Problem Description and Solution Outline

Before the circuit transformation we are supposed to given a circuit that has only single quantum gates and CNOT gates [2, 13]. We add auxiliary gates (see Fig. 5)

to move two non-adjacent quantum positions to adjacent positions or change the direction of a CNOT gate, but this process may introduce errors. We hope to find a circuit transformation algorithm that can produce the output circuit with a minimal number of auxiliary gates and a small circuit depth in an acceptable amount of time. A quantum circuit transformation mainly includes the following four steps. Isomorphism and adjustment are both NP-complete [19].

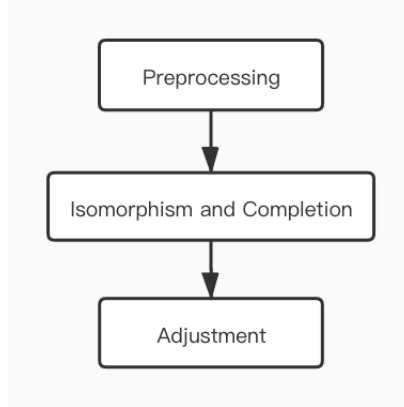


Fig. 7. Circuit transformation process

1. *Preprocess the logical quantum circuit.* This step includes extracting the logical architecture graph of the circuit, adjusting the life cycle of qubits as done in [24]), and calculating the shortest paths of the physical circuit.
2. *Compute isomorphic substructures and generate a high-quality initial mapping.* This step uses the subgraph isomorphism algorithm to find part of the initial mappings as done in [20]. 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 node and the mapped nodes.
3. *Transform logical circuits to meet physical constraints.* Circuit transformation need to be performed before compiling quantum circuits, since the design of quantum algorithms usually does not refer to the connectivity constraints of any hardware. Therefore, it is indispensable for any quantum compiler.

5 Solution Details

Our solution includes preprocessing, initial mapping, completion, and circuit adjustment, which will be presented in turn.

5.1 Preprocessing

Before a circuit adjustment, we can preprocess it by collect my data so as to shorten our search time and space. In the preprocessing step, we adjust the input circuit described by an openQASM program to shorten the life cycle of qubits. Then we use Breadth-First Search (BFS) to calculate the shortest distance between each node 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 [24]. A conversion method is designed to use the layered bundle format to determine which gates can be moved, which reduces the life cycle of these qubits. The algorithm reduces the error rate of quantum programs by 11% on average. In most quantum workloads, the longest qubit lifetime and the average qubit lifetime can be reduced by more than 20%, and the execution time of some quantum programs can also be reduced.

Shortest Distance 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 represents the shortest distance from q_i to q_j .

For IBM QX2, QX3, QX4, QX5, a SWAP needs 7 gates (3 CNOT gates and 4 H gates). Only 4 H gates are needed to change the direction of an adjacent CNOT gate. For a CNOT gate $g = \langle q_i, q_j \rangle$, the two qubits are mapped to q_m and q_n , respectively, with $\tau(q_i) = q_m, \tau(q_j) = q_n$. Then the cost of executing g under the shortest distance path is $cost_{cnot}(q_i, q_j) = 7 \times (dist[m][n] - 1)$. For IBM Q20, in which all edges are bidirected, a SWAP requires 3 CNOT gates. Thus the cost between them is $cost_{cnot}(q_i, q_j) = 3 \times (dist[m][n] - 1)$. The time complexity is $O(N^3)$.

Example 2. Take the QX5 (see in Fig. 6 (d)) structure as an example. Suppose there is a CNOT gate $g = \langle q_1, q_2 \rangle$, q_1 is mapped to q_1 , q_2 is mapped to q_{14} , and 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} : $\Pi = \{\pi_0, \pi_1, \pi_2\}$, $\pi_0 = q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_{14}$, $\pi_1 = q_1 \rightarrow q_2 \rightarrow q_{15} \rightarrow q_{14}$, $\pi_2 = q_1 \rightarrow q_0 \rightarrow q_{15} \rightarrow q_{14}$. Their costs are given by $cost_{\pi_0} = 18$, $cost_{\pi_1} = 14$, and $cost_{\pi_2} = 14$, respectively.

Circuit Layering 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 $L(LC) = \{\mathcal{L}_0, \mathcal{L}_1, \dots, \mathcal{L}_n\}$ represents 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, we generate 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 representing the set of undirected edges that the CNOT gates can execute.

5.2 Initial Mapping and Completion

We use the subgraph isomorphism algorithm to find a partial initial mapping that promissingly minimize auxiliary gates added by the output circuit.

As it is known, not all logical architecture graph have a subgraph that exactly matches all the nodes in physical architecture graph. In a physical architecture graph, it is almost impossible to find a subgraph that exactly matches all the nodes in a logical architecture graph. Thus, we regard the mapping with the largest number of matching nodes as the partial mapping. SubgraphCompare compares several state-of-the-art subgraph isomorphism algorithm compositions. 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. We artificially connect the isolating qubits to the qubits with the largest degree in the logical architecture graph, since SubgraphCompare cannot handle disconnected graphs. We would like to minimize the impact of logical dependency graphs, thus map isolated nodes to nodes with the largest degree.

The input of Algorithm 1 is a target graph (\mathcal{AG}_P), a query graph (\mathcal{AG}_L), and the partial mapping set T . First, 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 . $q \rightarrow -1$ means that q is not mapped to any physical node, so we need to perform a mapping completion. Finally, we generate a dense mapping, which can reduce the added auxiliary gates. 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.

In Algorithm 1, Line 2 selects the partial mapping with the most mapped nodes l as the candidate set. Lines 3-28 complete the partial mapping. In Line 6, we initialize an empty queue Q , which stores unmapped logical qubits. In Lines 8-10, we traverse the mapping τ and add the unmapped qubits to Q . We iterate the 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 13-14 are used to get the adjacency matrices of \mathcal{AG}_P and \mathcal{AG}_L , respectively. Line 15 initializes an empty map *cans*, sort by a descending order of whether q_m is mapped and the number of gate appearances. Line 16 store the nodes q_m connected to q in the adjacency matrix in *cans*. Lines 17-28 traverse *cans*, select the node q_m that has been mapped to the node (*cans.first*) on physical architecture graph, and has the largest number of connections to q in *cans*, with q being the node that q_m is mapped to in τ . Line 20 deletes the node from *cans*. Lines 21-26 select the node k adjacent to q in the adjacency matrix, and maps q to that node.

Example 3. Following the previous example, we first use the CSI algorithm for the logical architecture graph given in Fig. 4 (a) and the physical architecture graph given in Fig. 6 (e) to obtain the partial mapping set $T = \{\tau_0, \tau_1, \dots, \tau_n\}$. Then we use one of the partial mappings as an example. $\tau_0 = \{q_0 \rightarrow q_{10}, q_1 \rightarrow$

Algorithm 1: Initial mapping algorithm CSI

Input: $\mathcal{AG}_{\mathcal{L}}$: The architecture of logical circuit
 $\mathcal{AG}_{\mathcal{P}}$: The architecture of physical circuit
 T : A partial mapping set obtained by SubgraphCompare
Output: *result*: A collection of mapping relations between $\mathcal{AG}_{\mathcal{L}}$ and $\mathcal{AG}_{\mathcal{P}}$

```

1 Initialize result =  $\emptyset$ ;
2  $l \leftarrow \max_{\tau \in T} \tau.length$ ;
3 for  $\tau \in T$  do
4   if  $l = \tau.length$  then
5     result.add( $\tau$ );
6      $Q \leftarrow$  an empty unmapped node queue
7      $i \leftarrow 1$ ;
8     while  $i \leq \tau.length$  do
9        $Q.push(\{i, i \leq \tau.length\})$ 
10       $i \leftarrow i + 1$ ;
11     while  $Q$  is not empty do
12        $q \leftarrow Q.poll()$ ;
13        $targetAdj \leftarrow \mathcal{AG}_{\mathcal{P}}.adjacencyMatrix()$ ;
14        $queryAdj \leftarrow \mathcal{AG}_{\mathcal{L}}.adjacencyMatrix()$ ;
15        $cans \leftarrow$  an empty candidate node list sorted by degree
16        $cans \cup \{q_m, q_m \leq queryAdj[q].length\}$ ;
17       while  $cans$  is not empty do
18          $q \leftarrow \tau[cans.first]$ ;
19          $k \leftarrow 0$ ;
20          $cans \leftarrow cans \setminus cans.first$ ;
21         while  $k < targetAdj[q].length$  do
22           if ( $targetAdj[q][k] \neq -1$  or  $targetAdj[k][q] \neq -1$ )
23             and not  $\tau.contains(k)$  then
24                $\tau[q] \leftarrow k$ ;
25               break;
26            $k \leftarrow k + 1$ ;
27         if  $k \neq targetAdj[q].length$  then
28           break;
29 return result;

```

$-1, q_2 \rightarrow \mathbf{q}_6, q_3 \rightarrow \mathbf{q}_5, q_4 \rightarrow \mathbf{q}_{11}\}$, $0 \leq i < n$. Here $q_1 \rightarrow -1$ means that q_1 is not mapped to any physical node, so we need to perform a mapping completion. In this example, the maximum number of mapped nodes is 4. Next, we demonstrate how τ_0 is completed. We add all unmapped nodes to the queue Q , $Q = \{q_1\}$, and end the loop ends when Q is empty. We put the first element of Q into q , and delete it from Q . Then we 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 $cans = \{q_3, q_2, q_4, q_0\}$. Thereafter, we traverse $cans$ and take out of the first element $value = q_3$ in $cans$, and calculate the physical node $\mathbf{q} = \mathbf{q}_5$, $\tau_0(q_3) = \mathbf{q}_5$. Finally, we map q to the node connected to \mathbf{q} but not yet mapped. If the nodes connected to \mathbf{q} have been mapped, the loop continues. In this example, it can be directly mapped to \mathbf{q}_0 . In the end, we obtain the mapping

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

5.3 Adjustment

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 deal with those large circuits that the current algorithm is difficult to handle and produce an output circuit closer to the optimum solution in a short time.

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. The tabu list avoids local optimum and guarantees the parallelism of auxiliary gates. The 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, because 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 the candidate set. The evaluation function selects a SWAP evaluation formula from the candidate set, taking the objective function as the evaluation function in general. The evaluation function satisfies some gates, and the number of SWAP gates added or the depth of the entire circuit should be small. The 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 the neighborhoods is shown in Algorithm 2. The input is the current circuit mapping τ_p . The variable *qubits* contains the mapping of physical qubits to logical qubits, where $j = \text{qubits}[i]$ means that the i -th physical qubit has been mapped to the j -th logical qubit. The variable *locations* represents the mapping of logical qubits to physical qubits, where $j = \text{locations}[i]$ 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. Lines 17-37 swap all the edges of candidate set, and calculate the cost of them.

Example 4. Let us consider the mapping

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

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 $swap_nodes = \{q_0, q_6\}$. The set of shortest paths is $paths = \{\{q_6 \rightarrow q_1 \rightarrow q_0\}, \{q_6 \rightarrow q_5 \rightarrow q_0\}\}$, 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 the swap set and join the candidate set. The current candidate set is $\{SWAP(q_6, q_1), SWAP(q_1, q_0), SWAP(q_6, q_5), SWAP(q_5, q_0)\}$.

The circuit mapping algorithm based on Tabu search takes a layered circuit and an initial mapping as input and outputs a circuit that can be executed in the specified architecture graph (see Algorithm 3). The transformed circuit mapping of each layer is used as the initial mapping of the next layer of the circuit. Line 1 regard the initial mapping τ_{ini} as the best mapping τ_{best} . Lines 3-12 cyclically check whether all 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 are the amnesty rules. When the best candidate is not found, the candidate set elements are all the same as the tabu list elements. 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 judge 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 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

$$\{SWAP(q_6, q_1), SWAP(q_1, q_0), SWAP(q_6, q_5), SWAP(q_5, q_0)\},$$

Algorithm 2: Calculate the candidate sets

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

```

1 Initialize results  $\leftarrow \emptyset$ ;
2  $E_w \leftarrow$  Calculate the weight of each edge
3 swap_nodes  $\leftarrow$  An empty set of candidate swap nodes
4 foreach  $g \in cl$  do
5   if  $g$  is executable then
6      $cl \leftarrow cl \setminus \{g\}$ ;
7   else
8      $q_1 \leftarrow locations[g.control]$ ;
9      $q_2 \leftarrow locations[g.target]$ ;
10    swap_nodes.add( $q_1$ );
11    swap_nodes.add( $q_2$ );
12 foreach  $g \in cl$  do
13    $q_1 \leftarrow locations[g.control]$ ;
14    $q_2 \leftarrow locations[g.target]$ ;
15   foreach  $path \in paths[q_1][q_2]$  do
16     foreach  $e \in path$  do
17       if  $\{sour\_node, tar\_node\} \cap swap\_nodes \neq \emptyset$  then
18          $new\_qubits \leftarrow qubits$ ;
19          $new\_locations \leftarrow locations$ ;
20          $q_1 \leftarrow new\_qubits[e.source]$ ;
21          $q_2 \leftarrow new\_qubits[e.target]$ ;
22          $new\_qubits[e.source] \leftarrow q_2$ ;
23          $new\_qubits[e.target] \leftarrow q_1$ ;
24         if  $q_1 \neq -1$  then
25            $new\_locations[q_1] \leftarrow q_2$ ;
26         if  $q_2 \neq -1$  then
27            $new\_locations[q_2] \leftarrow q_1$ ;
28          $s \leftarrow \emptyset$ ;
29          $s.swaps \leftarrow p.swaps \cup \{distance.paths[path\_index][j]\}$ ;
30          $s.value \leftarrow compute\_evaluate\_value(dist, new\_locations, cl)$ ;
31          $results \leftarrow results \cup \{s\}$ ;
32 return results;

```

Algorithm 3: Tabu search

Input: τ_{ini} : The initial mapping
 tl : Tabu list
Output: τ_{best} : The final state and SWAPs

```

1 Initialize  $\tau_{best} \leftarrow \tau_{ini}$ ;
2  $iter \leftarrow 1$ ; // Number of iterations
3 while not musTSAop( $iter, \tau_{best}$ ) do
4    $C \leftarrow \tau_{ini}.candidates()$ ; // candidate set
5   if  $C$  is empty then
6      $\text{break}$ ;
7    $C_{best} \leftarrow \text{find\_best\_candidates}(C, tl)$ ;
8   if  $C_{best}$  is empty then
9      $C_{best} \leftarrow \text{find\_amnesty\_candidates}(C, tl)$ ;
10   $\tau_{best} \leftarrow C_{best}$ ;
11   $tl \leftarrow tl \cup \{C_{best}.swap\}$ ;
12   $iter \leftarrow iter + 1$ ;
13 return  $\tau_{best}$ 

```

and the costs are given as follows.

$$\begin{aligned} cost(SWAP(q_6, q_1)) &= 3.0, & cost(SWAP(q_1, q_0)) &= 3.0, \\ cost(SWAP(q_6, q_5)) &= 3.0, & cost(SWAP(q_5, q_0)) &= 3.0. \end{aligned}$$

The algorithm will choose the first SWAP, 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 the executability of g_0 . The algorithm continues to search for the next layer.

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).

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

$$cost(SWAP(q_m, q_n)) = Depth(L) \quad (2)$$

Here $cost(SWAP(q_m, q_n))$ represents the cost of executing all the gates of the current layer L after swapping q_m with q_n . We only calculate the depth between the unmapped gates as in (1) or the distance of the unmapped gates as in (2).

Look ahead We observe that the number of gates in each layer after layering is small. The output of the i -th ($i < n$) layer 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 ($i+x < n$) layer into consideration, where x is the number of forward-looking 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 is approximately the best. Our evaluation functions in (1) and (2) can be adjusted as (3) and (4), respectively.

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

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

Complexity Given a logical circuit architecture graph $\mathcal{AG}_{\mathcal{L}} = (V_L, E_L)$ and a physical circuit architecture graph $\mathcal{AG}_{\mathcal{P}} = (V_P, E_P)$, 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 a time, 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. Our time complexity is $d \times (|E_P| - 1)^{(|E_P| - 2)}$, and the space complexity is the size of our candidate set $(E_P - 1)$.

6 Experiments

We compare the CSI algorithm and the circuit adjustment algorithm based on Tabu search TSA with the wghtgraph in [10] and the heuristic algorithm A^* in [25]. All the experiments are conducted on a Windows machine with 3.3GHz CPU and 10G memory.

First, we compare the efficiency of initial mapping on optm [25], CSI and wghtgraph [10]. In order to observe the results of these two initial mapping

algorithms, we used the same circuit adjustment algorithm A^* [25]. We have tested 159 circuits. Within five minutes, *optm*, *wgtgraph*, and *CSI* can deal with 121, 106, and 131 circuits, respectively. There are 103 circuits that they can handle. We then compare the *wgtgraph* algorithm and the *CSI* algorithm more closely. The *wgtgraph* algorithm has 21 circuits with fewer auxiliary gates and 19 circuits with smaller depths, and the *CSI* algorithm has 54 circuits with fewer auxiliary gates and 60 circuits with smaller depths. They output 25 circuits with equal depth and 29 circuits with equal auxiliary gates. On average, the auxiliary gates of the *CSI* algorithm are reduced by 22.44%, and the depths are reduced by 11.25%.

Next, we compare the *optm* algorithm and the *CSI* algorithm. The *optm* algorithm has one circuit with fewer auxiliary gates and two circuits with a small depth, while the *CSI* algorithm has 99 circuits with fewer auxiliary gates and 98 circuits with a small depth. They have 4 circuits with equal depth and 4 circuits with equal auxiliary gates. The auxiliary gates of the *CSI* algorithm are relatively reduced by 27.02%, and the depths are reduced by 14.12%. Table 1 shows the experimental data on 104 circuits. The three initial mapping algorithms are compared according to the depths of the generated circuits using the same A^* algorithm, and the numbers of added auxiliary gates. The column headed by *CSI/optm* shows the efficiency improvement of the former upon the latter $(n_{optm} - n_{CSI})/n_{optm}$.

| | <i>optm</i> | <i>wgtgraph</i> | <i>CSI</i> | <i>CSI/optm</i> | <i>CSI/wgtgraph</i> |
|-----------------|-------------|-----------------|------------|-----------------|---------------------|
| Depths | 168895 | 163422 | 145040 | 14.12% | 11.25% |
| Auxiliary gates | 20439 | 19232 | 14916 | 27.02% | 22.44% |

Table 1. Compare *optm*, *wgtgraph*, and *CSI*.

We then compare 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 tested 159 circuits. The depths of the final circuits obtained by TSA_{num} are 1.93% smaller than TSA_{dep} on average, and the numbers auxiliary gates added are 4.53% smaller on average. When inserting a SWAP gate, the circuit needs to add 3 CNOT gates, and the depth will be increased by 3. Therefore, if fewer SWAP gates are added, the depths of the circuits will reduce accordingly.

Finally, we compare *TSA* and *wgtgraph*. Since the *wgtgraph* algorithm only uses 2-qubit gates, it is impossible to compare the depths 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 tested 159 circuits. It turns out that TSA_{num} only takes 461 seconds and TSA_{dep} takes 485 seconds. The *wgtgraph* algorithm takes 1908 seconds for the 159 circuits, but only produces valid results for 99 circuits, including 64 small circuits, 35 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 99 small and medium circuits, the number of SWAP gates added by wgtgraph is 26.87% (resp. 24.89%) smaller than TSA_{num} (resp. TSA_{dep}) on average. The Tabu search can quickly output converted circuits on large circuits, but wgtgraph cannot get results in a short time. Since our candidate set is too small to be able to process the circuit quickly, but this also leads to an increase 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 23 small circuits, 6 medium circuits, and 1 large circuit. The detailed results of the circuit comparisons are in the Appendix.

| benchmarks | #circ. | TSA_{num} | | TSA_{dep} | | wgtgraph | | SABRE | |
|------------|--------|-------------|------|-------------|------|----------|------|--------|---------|
| | | #succ. | time | #succ. | time | #succ. | time | #succ. | time |
| small | 66 | 66 | 32 | 66 | 29 | 64 | 587 | 23 | 12996 |
| medium | 49 | 49 | 45 | 49 | 40 | 35 | 1183 | 6 | 13019 |
| large | 44 | 44 | 407 | 44 | 432 | 0 | - | 0 | 1340719 |
| total | 159 | 159 | 484 | 159 | 501 | 99 | - | 29 | 1366734 |

Table 2. Compare optm, wgtgraph, SABRE TSA_{num} and TSA_{dep} .

7 Conclusion

We propose to use the combined subgraph isomorphism algorithm to generate high-quality initial mappings and a heuristic Tabu search to perform circuit adjustment. Experimental results show that the initial mappings generated by CSI have fewer SWAP gates inserted and the results can be obtained in a short time. Most small and medium circuits can be handled in a few seconds. For large circuits, the results can be obtained within a few minutes, but the cost of insertion may be larger than that of wgtgraph. We introduce a look-ahead method to make each selected SWAP more in line with the constraints of the gates to be processed. In the future, we will investigate how to reduce the number of auxiliary gates inserted and increase the speed. We will also apply the proposed method to more NISQ devices.

References

1. Almeida, A., Dueck, G., Silva, A.: Finding optimal qubit permutations for ibm’s quantum computer architectures pp. 1–6 (08 2019). <https://doi.org/10.1145/3338852.3339829>
2. Barenco, A., Bennett, C., Cleve, R., DiVincenzo, D., Margolus, N., Shor, P., Sleator, T., Smolin, J., Weinfurter, H.: Elementary gates for quantum computation. *Physical Review A* **52** (03 1995). <https://doi.org/10.1103/PhysRevA.52.3457>

3. Bernal, D., Booth, K., Dridi, R., Alghassi, H., Tayur, S., Venturelli, D.: Integer programming techniques for minor-embedding in quantum annealers (12 2019)
4. Cowtan, A., Dilkes, S., Duncan, R., Krajenbrink, A., Simmons, W., Sivarajah, S.: On the qubit routing problem (02 2019)
5. Daei, O., Navi, K., Zomorodi, M.: Optimized quantum circuit partitioning (05 2020)
6. Glover, F.: Tabu search—part ii. *ORSA Journal on Computing* **2**, 4–32 (02 1990). <https://doi.org/10.1287/ijoc.2.1.4>
7. Guerreschi, G.G., Park, J.: Two-step approach to scheduling quantum circuits. *Quantum Science and Technology* **3** (06 2018). <https://doi.org/10.1088/2058-9565/aac0b>
8. Kissinger, A., Meijer, A.: Cnot circuit extraction for topologically-constrained quantum memories (04 2019)
9. Li, G., Ding, Y., Xie, Y.: Tackling the qubit mapping problem for nisq-era quantum devices (09 2018)
10. Li, S., Zhou, X., Feng, Y.: Qubit mapping based on subgraph isomorphism and filtered depth-limited search (2020)
11. Matsuo, A., Yamashita, S.: An efficient method for quantum circuit placement problem on a 2-d grid pp. 162–168 (05 2019). https://doi.org/10.1007/978-3-030-21500-2_10
12. Murali, P., Linke, N., Martonosi, M., Abhari, A., Nguyen, N., Huerta Alderete, C.: Full-stack, real-system quantum computer studies: architectural comparisons and design insights pp. 527–540 (06 2019). <https://doi.org/10.1145/3307650.3322273>
13. Möttönen, M., Vartiainen, J.: Decompositions of general quantum gates. *Frontiers in Artificial Intelligence and Applications* (05 2005)
14. Nash, B., Gheorghiu, V., Mosca, M.: Quantum circuit optimizations for nisq architectures. *Quantum Science and Technology* **5** (02 2020). <https://doi.org/10.1088/2058-9565/ab79b1>
15. Paler, A.: On the influence of initial qubit placement during nisq circuit compilation (11 2018)
16. Preskill, J.: Quantum computing in the nisq era and beyond. *Quantum* **2** (2018)
17. Shafaei, A., Saeedi, M., Pedram, M.: Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. *Proceedings - Design Automation Conference* pp. 1–6 (05 2013). <https://doi.org/10.1145/2463209.2488785>
18. Shafaei, A., Saeedi, M., Pedram, M.: Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures (2013)
19. Siraichi, M.Y., dos Santos, V.F., Collange, S., Pereira, F.M.Q.: Qubit allocation (2018)
20. Sun, S., Luo, Q.: In-memory subgraph matching: An in-depth study pp. 1083–1098 (06 2020). <https://doi.org/10.1145/3318464.3380581>
21. Tannu, S., Qureshi, M.: Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers pp. 987–999 (04 2019). <https://doi.org/10.1145/3297858.3304007>
22. Venturelli, D., do, M., Rieffel, E., Frank, J.: Temporal planning for compilation of quantum approximate optimization circuits pp. 4440–4446 (08 2017). <https://doi.org/10.24963/ijcai.2017/620>
23. Xiangzhen, Z., Li, S., Feng, Y.: Quantum circuit transformation based on simulated annealing and heuristic search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **PP**, 1–1 (01 2020). <https://doi.org/10.1109/TCAD.2020.2969647>

24. Zhang, Y., Deng, H., Li, Q., Haoze, S., Nie, L.: Optimizing quantum programs against decoherence: Delaying qubits into quantum superposition pp. 184–191 (07 2019). <https://doi.org/10.1109/TASE.2019.000-2>
25. Zulehner, A., Paller, A., Wille, R.: Efficient mapping of quantum circuits to the ibm qx architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (12 2017). <https://doi.org/10.1109/TCAD.2018.2846658>

A Experimental details of the SWAP gates added by the output circuits

| Circuit name | qubit no. | CNOT no. | TSA _{num} added | TSA _{dep} added | optm added | wghtgr added | SABRE added |
|----------------------|-----------|----------|--------------------------|--------------------------|------------|--------------|-------------|
| decod24-enable_126 | 6 | 149 | 28 | 42 | 60 | 16 | 129 |
| 4mod5-v0_19 | 5 | 16 | 0 | 0 | 0 | 0 | - |
| 4mod5-v0_18 | 5 | 31 | 2 | 5 | 4 | 4 | - |
| mod5d2_64 | 5 | 25 | 5 | 6 | 8 | 3 | - |
| 4gt4-v0_72 | 6 | 113 | 14 | 10 | 33 | 14 | - |
| alu-v3_35 | 5 | 18 | 2 | 4 | 8 | 2 | - |
| 4gt4-v0_73 | 6 | 179 | 27 | 34 | 76 | 12 | - |
| alu-v3_34 | 5 | 24 | 2 | 3 | 7 | 2 | - |
| 3_17_13 | 3 | 17 | 0 | 0 | 6 | 0 | 0 |
| 4gt4-v0_78 | 6 | 109 | 12 | 8 | 48 | 4 | 124 |
| 4gt4-v0_79 | 6 | 105 | 17 | 17 | 48 | 3 | - |
| 4mod7-v1_96 | 5 | 72 | 16 | 19 | 27 | 7 | - |
| mod10_171 | 5 | 108 | 17 | 20 | 39 | 9 | - |
| ex2_227 | 7 | 275 | 48 | 59 | 121 | 33 | - |
| mod10_176 | 5 | 78 | 14 | 14 | 38 | 8 | - |
| 0410184_169 | 5 | 9 | 2 | 2 | 49 | 3 | - |
| 4mod5-v0_20 | 5 | 10 | 0 | 0 | 4 | 0 | - |
| aj-e11_165 | 5 | 69 | 8 | 8 | 33 | 7 | 63 |
| alu-v1_28 | 5 | 18 | 2 | 4 | 11 | 2 | - |
| 4gt12-v0_86 | 6 | 116 | 28 | 33 | 48 | 3 | - |
| 4gt12-v0_87 | 6 | 112 | 27 | 32 | 45 | 2 | - |
| 4gt12-v0_88 | 6 | 86 | 5 | 5 | 25 | 4 | - |
| alu-v1_29 | 5 | 17 | 4 | 4 | 11 | 2 | 13 |
| ham7_104 | 7 | 149 | 28 | 34 | 68 | 12 | - |
| C17_204 | 7 | 205 | 26 | 53 | 99 | 22 | - |
| xor5_254 | 6 | 5 | 0 | 0 | 1 | 0 | 3 |
| hwb4_49 | 5 | 107 | 14 | 15 | 38 | 11 | 157 |
| rd73_140 | 10 | 104 | 23 | 26 | 35 | 20 | - |
| decod24-v0_38 | 4 | 23 | 0 | 0 | 6 | 0 | - |
| rd53_131 | 7 | 200 | 39 | 39 | 98 | 24 | - |
| rd53_133 | 7 | 256 | 37 | 47 | 102 | 27 | 160 |
| rd53_135 | 7 | 134 | 28 | 29 | 38 | 23 | - |
| decod24-v2_43 | 4 | 22 | 0 | 0 | 9 | 0 | 39 |
| rd53_138 | 8 | 60 | 14 | 16 | 23 | 9 | - |
| rd32-v0_66 | 4 | 16 | 0 | 0 | 6 | 0 | - |
| 4gt13-v1_93 | 5 | 30 | 0 | 0 | 13 | 0 | - |
| graycode6_47 | 6 | 5 | 0 | 0 | 0 | 0 | - |
| 4mod5-bdd_287 | 7 | 31 | 3 | 6 | 8 | 6 | 34 |
| ham3_102 | 3 | 11 | 0 | 0 | 3 | 0 | 12 |
| 4gt4-v0_80 | 6 | 79 | 5 | 5 | 22 | 5 | - |
| ex-1_166 | 3 | 9 | 0 | 0 | 3 | 0 | - |
| mod5mils_65 | 5 | 16 | 0 | 0 | 6 | 0 | - |
| 0example | 5 | 9 | 1 | 2 | 3 | 3 | - |
| alu-v4_36 | 5 | 51 | 12 | 8 | 22 | 4 | 41 |
| alu-v4_37 | 5 | 18 | 2 | 4 | 8 | 2 | - |
| ex1_226 | 6 | 5 | 0 | 0 | 1 | 0 | 15 |
| one-two-three-v0_98 | 5 | 65 | 11 | 13 | 32 | 10 | - |
| one-two-three-v0_97 | 5 | 128 | 23 | 23 | 64 | 16 | - |
| one-two-three-v3_101 | 5 | 32 | 3 | 4 | 14 | 3 | - |
| rd32_270 | 5 | 36 | 3 | 3 | 6 | 6 | 24 |

Table 3. Comparison of the number of SWAP gates added by the output circuit on the IBM Q20

| Circuit name | qubit no. | CNOT no. | TSA _{num} added | TSA _{dep} added | optm added | wghtgr added | SABRE added |
|----------------------|-----------|----------|--------------------------|--------------------------|------------|--------------|-------------|
| rd53_130 | 7 | 448 | 89 | 100 | 190 | 49 | - |
| rd53_251 | 8 | 564 | 104 | 131 | 230 | 45 | - |
| 4mod5-v1_24 | 5 | 16 | 0 | 0 | 3 | 0 | - |
| mod5adder_127 | 6 | 239 | 21 | 56 | 111 | 20 | - |
| 4_49_16 | 5 | 99 | 20 | 17 | 40 | 10 | - |
| hwb5_53 | 6 | 598 | 141 | 168 | 173 | 59 | - |
| ex3_229 | 6 | 175 | 10 | 9 | 50 | 11 | - |
| 4gt10-v1_81 | 5 | 66 | 14 | 15 | 28 | 6 | - |
| alu-v2_32 | 5 | 72 | 15 | 17 | 27 | 7 | - |
| alu-v2_31 | 5 | 198 | 42 | 54 | 85 | 13 | - |
| alu-v2_30 | 6 | 223 | 41 | 45 | 96 | 20 | - |
| sf_276 | 6 | 336 | 12 | 52 | 138 | 12 | - |
| decod24-v1_41 | 5 | 38 | 4 | 4 | 14 | 3 | 28 |
| sf_274 | 6 | 336 | 34 | 21 | 82 | 12 | - |
| 4gt4-v1_74 | 6 | 119 | 17 | 24 | 37 | 9 | - |
| alu-v2_33 | 5 | 17 | 4 | 4 | 8 | 2 | - |
| cnt3-5_179 | 16 | 85 | 6 | 6 | 35 | 4 | - |
| 4mod5-v1_22 | 5 | 11 | 0 | 0 | 5 | 0 | 0 |
| 4mod5-v1_23 | 5 | 32 | 5 | 5 | 4 | 3 | 55 |
| mini_alu_305 | 10 | 77 | 10 | 20 | 28 | 8 | 123 |
| alu-v0_26 | 5 | 38 | 7 | 10 | 13 | 3 | - |
| alu-bdd_288 | 7 | 38 | 4 | 12 | 16 | 6 | - |
| alu-v0_27 | 5 | 17 | 2 | 4 | 11 | 2 | 9 |
| 4gt13_91 | 5 | 49 | 7 | 7 | 10 | 2 | - |
| 4gt5_77 | 5 | 58 | 12 | 12 | 20 | 6 | - |
| 4gt13_92 | 5 | 30 | 0 | 0 | 14 | 0 | - |
| 4gt5_76 | 5 | 46 | 7 | 10 | 24 | 5 | 39 |
| 4gt5_75 | 5 | 38 | 5 | 12 | 16 | 4 | 53 |
| 4gt12-v1_89 | 6 | 100 | 11 | 21 | 38 | 4 | - |
| one-two-three-v1_99 | 5 | 59 | 12 | 10 | 26 | 7 | - |
| 4gt13_90 | 5 | 53 | 7 | 7 | 13 | 3 | - |
| ising_model_10 | 10 | 90 | 0 | 0 | 5 | 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 | - |
| 4gt11_82 | 5 | 18 | 1 | 1 | 1 | 1 | - |
| decod24-v3_45 | 5 | 64 | 15 | 15 | 32 | 8 | 64 |
| rd32-v1_68 | 4 | 16 | 0 | 0 | 6 | 0 | 24 |
| mini-alu_167 | 5 | 126 | 27 | 27 | 49 | 11 | - |
| one-two-three-v2_100 | 5 | 32 | 3 | 4 | 8 | 3 | - |
| 4mod7-v0_94 | 5 | 72 | 8 | 13 | 36 | 9 | - |
| cm82a_208 | 8 | 283 | 41 | 69 | 84 | 33 | - |
| mod8-10_178 | 6 | 152 | 5 | 20 | 13 | 7 | - |
| mod8-10_177 | 6 | 196 | 14 | 33 | 58 | 13 | 230 |
| majority_239 | 7 | 267 | 39 | 43 | 105 | 33 | - |
| miller_11 | 3 | 23 | 0 | 0 | 9 | 0 | 36 |
| decod24-bdd_294 | 6 | 32 | 4 | 4 | 9 | 4 | 36 |
| total | 551 | 9244 | 1372 | 1738 | 3481 | 800 | - |

Table 4. Comparison of the number of SWAP gates added by the output circuit on the IBM Q20

| Circuit name | qubit no. | CNOT no. | TSA _{num} added | TSA _{dep} added | optm added | wghtgr added | SABRE added |
|----------------------------|--------------|-------------|-----------------------------|-----------------------------|---------------|-----------------|----------------|
| max46_240 | 10 | 11844 | 3473 | 4545 | - | - | - |
| rd73_252 | 10 | 2319 | 586 | 761 | - | - | - |
| cycle10_2_110 | 12 | 2648 | 919 | 1216 | 961 | - | - |
| sqrt8_260 | 12 | 1314 | 379 | 492 | 457 | - | - |
| urf4_187 | 11 | 224028 | 54785 | 60140 | - | - | - |
| sqn_258 | 10 | 4459 | 1199 | 1420 | - | - | - |
| f2_232 | 8 | 525 | 87 | 124 | 218 | - | - |
| radd_250 | 13 | 1405 | 386 | 489 | 511 | - | - |
| ham15_107 | 15 | 3858 | 1326 | 1689 | - | - | - |
| sao2_257 | 14 | 16864 | 5346 | 7178 | - | - | - |
| sym9_148 | 10 | 9408 | 1865 | 2432 | - | - | - |
| urf5_280 | 9 | 23764 | 6989 | 8730 | - | - | - |
| square_root_7 | 15 | 3089 | 812 | 2150 | - | - | - |
| sys6-v0_111 | 10 | 98 | 23 | 26 | 38 | - | - |
| hwb7_59 | 8 | 10681 | 2687 | 3551 | 3722 | - | - |
| sym9_146 | 12 | 148 | 38 | 55 | 54 | - | 248 |
| wim_266 | 11 | 427 | 93 | 120 | 147 | - | - |
| urf2_152 | 8 | 35210 | 9181 | 11921 | 10577 | - | - |
| urf5_159 | 9 | 71932 | 20258 | 25505 | - | - | - |
| urf2_277 | 8 | 10066 | 2807 | 3798 | 3782 | - | - |
| life_238 | 11 | 9800 | 2762 | 3576 | - | - | - |
| root_255 | 13 | 7493 | 2128 | 3035 | - | - | - |
| 9symml_195 | 11 | 15232 | 4553 | 5986 | - | - | - |
| sym10_262 | 12 | 28084 | 8534 | 11033 | - | - | - |
| dc1_220 | 11 | 833 | 226 | 207 | 371 | - | - |
| cm42a_207 | 14 | 771 | 182 | 229 | 294 | - | - |
| rd53_311 | 13 | 124 | 26 | 48 | 47 | - | - |
| dc2_222 | 15 | 4131 | 1383 | 1773 | - | - | - |
| rd84_142 | 15 | 154 | 49 | 58 | 50 | - | - |
| sym6_145 | 7 | 1701 | 317 | 449 | 750 | - | - |
| co14_215 | 15 | 7840 | 3078 | 3819 | - | - | - |
| cnt3-5_180 | 16 | 215 | 59 | 74 | 79 | - | - |
| cm152a_212 | 12 | 532 | 103 | 129 | 168 | - | - |
| sym6_316 | 14 | 123 | 30 | 39 | 56 | - | - |
| mlp4_245 | 16 | 8232 | 2780 | 3490 | - | - | - |
| hwb8_113 | 9 | 30372 | 10749 | 16489 | - | - | - |
| qft_16 | 16 | 240 | 90 | 147 | - | - | - |
| plus63mod4096_163 | 13 | 56329 | 19759 | 24273 | - | - | - |
| urf1_149 | 9 | 80878 | 22551 | 28516 | - | - | - |
| urf3_155 | 10 | 185276 | 50842 | 62903 | - | - | - |
| urf3_279 | 10 | 60380 | 17999 | 23318 | - | - | - |
| hwb9_119 | 10 | 90955 | 22946 | 30031 | - | - | - |
| plus63mod8192_164 | 14 | 81865 | 28022 | 36207 | - | - | - |
| pm1_249 | 14 | 771 | 182 | 229 | 294 | - | - |
| sym9_193 | 11 | 15232 | 4382 | 5518 | - | - | - |
| misex1_241 | 15 | 2100 | 480 | 754 | 600 | - | - |
| urf1_278 | 9 | 26692 | 8010 | 10217 | - | - | - |
| squar5_261 | 13 | 869 | 219 | 313 | 290 | - | - |
| ground_state_estimation_10 | 13 | 154209 | 11671 | 22886 | - | - | - |
| adr4_197 | 13 | 1498 | 516 | 670 | - | - | - |

Table 5. Comparison of the number of SWAP gates added by the output circuit on the IBM Q20

| Circuit name | qubit no. | CNOT no. | TSA _{num} added | TSA _{dep} added | optm added | wghtgr added | SABRE added |
|-----------------|--------------|-------------|-----------------------------|-----------------------------|---------------|-----------------|----------------|
| hwb6_56 | 7 | 2952 | 698 | 933 | 909 | - | - |
| clip_206 | 14 | 14772 | 5430 | 6865 | - | - | - |
| cm85a_209 | 14 | 4986 | 2088 | 2225 | - | - | - |
| rd84_253 | 12 | 5960 | 1849 | 2333 | - | - | - |
| dist_223 | 13 | 16624 | 5623 | 7431 | - | - | - |
| inc_237 | 16 | 4636 | 1193 | 1667 | - | - | - |
| qft_10 | 10 | 90 | 23 | 34 | 30 | - | - |
| urf6_160 | 15 | 75180 | 27524 | 32452 | - | - | - |
| con1_216 | 9 | 415 | 86 | 118 | 177 | - | - |

Table 6. Comparison of the number of SWAP gates added by the output circuit on the IBM Q20

B Experimental details of the depths of the output circuits

| Circuit name | qubit no. | CNOT no. | depths no. | TSA _{num} depths | TSA _{dep} depths | optm depths |
|--------------------|-----------|----------|------------|---------------------------|---------------------------|-------------|
| decod24-enable_126 | 6 | 149 | 190 | 233 | 275 | 470 |
| 4mod5-v0_19 | 5 | 16 | 21 | 16 | 16 | 21 |
| 4mod5-v0_18 | 5 | 31 | 40 | 37 | 46 | 54 |
| mod5d2_64 | 5 | 25 | 32 | 40 | 43 | 67 |
| 4gt4-v0_72 | 6 | 113 | 137 | 155 | 143 | 297 |
| alu-v3_35 | 5 | 18 | 22 | 24 | 30 | 60 |
| 4gt4-v0_73 | 6 | 179 | 227 | 260 | 281 | 586 |
| alu-v3_34 | 5 | 24 | 30 | 30 | 33 | 63 |
| 3_17_13 | 3 | 17 | 22 | 17 | 17 | 52 |
| 4gt4-v0_78 | 6 | 109 | 137 | 145 | 133 | 352 |
| 4gt4-v0_79 | 6 | 105 | 132 | 156 | 156 | 345 |
| 4mod7-v1_96 | 5 | 72 | 94 | 120 | 129 | 218 |
| mod10_171 | 5 | 108 | 139 | 159 | 168 | 335 |
| ex2_227 | 7 | 275 | 355 | 419 | 452 | 899 |
| mod10_176 | 5 | 78 | 101 | 120 | 120 | 104 |
| cycle10_2_110 | 12 | 2648 | 3386 | 5405 | 6296 | 7467 |
| 0410184_169 | 5 | 9 | 6 | 15 | 15 | 253 |
| 4mod5-v0_20 | 5 | 10 | 12 | 10 | 10 | 32 |
| sqrt8_260 | 12 | 1314 | 1661 | 2451 | 2790 | 3561 |
| aj-e11_165 | 5 | 69 | 86 | 93 | 93 | 250 |
| alu-v1_28 | 5 | 18 | 22 | 24 | 30 | 70 |
| f2_232 | 8 | 525 | 668 | 786 | 897 | 1672 |
| radd_250 | 13 | 1405 | 1781 | 2563 | 2872 | 3985 |
| 4gt12-v0_86 | 6 | 116 | 135 | 200 | 215 | 334 |
| 4gt12-v0_87 | 6 | 112 | 131 | 193 | 208 | 324 |
| 4gt12-v0_88 | 6 | 86 | 108 | 101 | 101 | 222 |
| alu-v1_29 | 5 | 17 | 22 | 29 | 29 | 64 |
| ham7_104 | 7 | 149 | 185 | 233 | 251 | 491 |
| C17_204 | 7 | 205 | 253 | 283 | 364 | 688 |
| xor5_254 | 6 | 5 | 5 | 5 | 5 | 10 |
| hwb4_49 | 5 | 107 | 134 | 149 | 152 | 308 |
| rd73_140 | 10 | 104 | 92 | 173 | 182 | 185 |
| decod24-v0_38 | 4 | 23 | 30 | 23 | 23 | 61 |
| rd53_131 | 7 | 200 | 261 | 317 | 317 | 677 |
| rd53_133 | 7 | 256 | 327 | 367 | 397 | 777 |
| rd53_135 | 7 | 134 | 159 | 218 | 221 | 331 |
| sys6-v0_111 | 10 | 98 | 75 | 167 | 176 | 188 |
| decod24-v2_43 | 4 | 22 | 30 | 22 | 22 | 75 |
| hwb7_59 | 8 | 10681 | 13437 | 18742 | 21334 | 29601 |
| rd53_138 | 8 | 60 | 56 | 102 | 108 | 114 |
| rd32-v0_66 | 4 | 16 | 20 | 16 | 16 | 51 |
| sym9_146 | 12 | 148 | 127 | 262 | 313 | 309 |
| 4gt13-v1_93 | 5 | 30 | 39 | 30 | 30 | 102 |
| graycode6_47 | 6 | 5 | 5 | 5 | 5 | 5 |
| wim_266 | 11 | 427 | 514 | 706 | 787 | 1180 |
| urf2_152 | 8 | 35210 | 44100 | 62753 | 70973 | 90299 |
| urf2_277 | 8 | 10066 | 11390 | 18487 | 21460 | 26548 |
| 4mod5-bdd_287 | 7 | 31 | 41 | 40 | 49 | 71 |
| ham3_102 | 3 | 11 | 13 | 11 | 11 | 28 |
| 4gt4-v0_80 | 6 | 79 | 101 | 94 | 94 | 206 |

Table 7. Comparison of the depth of the output circuit on the IBM Q20

| Circuit name | qubit no. | CNOT no. | depths no. | TSA _{num} depths | TSA _{dep} depths | optm depths |
|----------------------|--------------|-------------|---------------|------------------------------|------------------------------|----------------|
| ex-1_166 | 3 | 9 | 12 | 9 | 9 | 28 |
| mod5mils_65 | 5 | 16 | 21 | 16 | 16 | 52 |
| 0example | 5 | 9 | 6 | 12 | 15 | 15 |
| alu-v4_36 | 5 | 51 | 66 | 87 | 75 | 170 |
| alu-v4_37 | 5 | 18 | 22 | 24 | 30 | 60 |
| ex1_226 | 6 | 5 | 5 | 5 | 5 | 10 |
| one-two-three-v0_98 | 5 | 65 | 82 | 98 | 104 | 234 |
| one-two-three-v0_97 | 5 | 128 | 163 | 197 | 197 | 443 |
| one-two-three-v3_101 | 5 | 32 | 40 | 41 | 44 | 95 |
| rd32_270 | 5 | 36 | 47 | 45 | 45 | 76 |
| dc1_220 | 11 | 833 | 1041 | 1511 | 1454 | 2711 |
| rd53_130 | 7 | 448 | 569 | 715 | 748 | 1417 |
| rd53_251 | 8 | 564 | 712 | 876 | 957 | 1767 |
| cm42a_207 | 14 | 771 | 940 | 1317 | 1458 | 2279 |
| rd53_311 | 13 | 124 | 130 | 202 | 268 | 300 |
| 4mod5-v1_24 | 5 | 16 | 21 | 16 | 16 | 36 |
| mod5adder_127 | 6 | 239 | 302 | 302 | 407 | 817 |
| 4_49_16 | 5 | 99 | 125 | 159 | 150 | 320 |
| hwb5_53 | 6 | 598 | 758 | 1021 | 1102 | 1560 |
| ex3_229 | 6 | 175 | 226 | 205 | 202 | 462 |
| rd84_142 | 15 | 154 | 110 | 301 | 328 | 253 |
| 4gt10-v1_81 | 5 | 66 | 84 | 108 | 111 | 210 |
| alu-v2_32 | 5 | 72 | 92 | 117 | 123 | 215 |
| alu-v2_31 | 5 | 198 | 255 | 324 | 360 | 650 |
| alu-v2_30 | 6 | 223 | 285 | 346 | 358 | 734 |
| sym6_145 | 7 | 1701 | 2187 | 2652 | 3048 | 5716 |
| sf_276 | 6 | 336 | 435 | 372 | 492 | 1096 |
| decod24-v1_41 | 5 | 38 | 50 | 50 | 50 | 120 |
| sf_274 | 6 | 336 | 436 | 438 | 399 | 822 |
| 4gt4-v1_74 | 6 | 119 | 154 | 170 | 191 | 329 |
| alu-v2_33 | 5 | 17 | 22 | 29 | 29 | 59 |
| cnt3-5_180 | 16 | 215 | 209 | 392 | 437 | 482 |
| cm152a_212 | 12 | 532 | 684 | 841 | 919 | 1423 |
| cnt3-5_179 | 16 | 85 | 61 | 103 | 103 | 166 |
| sym6_316 | 14 | 123 | 135 | 213 | 240 | 378 |
| 4mod5-v1_22 | 5 | 11 | 12 | 11 | 11 | 37 |
| 4mod5-v1_23 | 5 | 32 | 41 | 47 | 47 | 55 |
| mini_alu_305 | 10 | 77 | 71 | 107 | 137 | 187 |
| alu-v0_26 | 5 | 38 | 49 | 59 | 68 | 108 |
| alu-bdd_288 | 7 | 38 | 48 | 50 | 74 | 112 |
| alu-v0_27 | 5 | 17 | 21 | 23 | 29 | 63 |
| 4gt13_91 | 5 | 49 | 61 | 70 | 70 | 108 |
| 4gt5_77 | 5 | 58 | 74 | 94 | 94 | 170 |
| 4gt13_92 | 5 | 30 | 38 | 30 | 30 | 103 |
| 4gt5_76 | 5 | 46 | 56 | 67 | 76 | 171 |
| 4gt5_75 | 5 | 38 | 47 | 53 | 74 | 127 |
| 4gt12-v1_89 | 6 | 100 | 130 | 133 | 163 | 313 |
| one-two-three-v1_99 | 5 | 59 | 76 | 95 | 89 | 194 |
| 4gt13_90 | 5 | 53 | 65 | 74 | 74 | 124 |
| pm1_249 | 14 | 771 | 940 | 1317 | 1458 | 2279 |

Table 8. Comparison of the depth of the output circuit on the IBM Q20

| Circuit name | qubit no. | CNOT no. | depths no. | TSA _{num} depths | TSA _{dep} depths | optm depths |
|----------------------|--------------|-------------|---------------|------------------------------|------------------------------|----------------|
| ising_model_10 | 10 | 90 | 52 | 90 | 90 | 107 |
| misex1_241 | 15 | 2100 | 2676 | 3540 | 4362 | 5326 |
| 4gt11_84 | 5 | 9 | 11 | 9 | 9 | 25 |
| 4gt11_83 | 5 | 14 | 16 | 14 | 14 | 16 |
| mod5d1_63 | 5 | 13 | 13 | 13 | 13 | 17 |
| 4gt11_82 | 5 | 18 | 20 | 21 | 21 | 25 |
| squar5_261 | 13 | 869 | 1051 | 1526 | 1808 | 2309 |
| decod24-v3_45 | 5 | 64 | 84 | 109 | 109 | 244 |
| rd32-v1_68 | 4 | 16 | 21 | 16 | 16 | 52 |
| hwb6_56 | 7 | 2952 | 3736 | 5046 | 5751 | 7773 |
| mini-alu_167 | 5 | 126 | 162 | 207 | 207 | 400 |
| one-two-three-v2_100 | 5 | 32 | 40 | 41 | 44 | 80 |
| 4mod7-v0_94 | 5 | 72 | 92 | 96 | 111 | 270 |
| cm82a_208 | 8 | 283 | 340 | 406 | 490 | 699 |
| mod8-10_178 | 6 | 152 | 193 | 167 | 212 | 243 |
| mod8-10_177 | 6 | 196 | 251 | 238 | 295 | 525 |
| majority_239 | 7 | 267 | 344 | 384 | 396 | 839 |
| qft_10 | 10 | 90 | 37 | 159 | 192 | 135 |
| miller_11 | 3 | 23 | 29 | 23 | 23 | 75 |
| decod24-bdd_294 | 6 | 32 | 40 | 44 | 44 | 86 |
| con1_216 | 9 | 415 | 508 | 673 | 769 | 1197 |
| total | 823 | 83416 | 103023 | 145372 | 164848 | 224731 |

Table 9. Comparison of the depth of the output circuit on the IBM Q20

| Circuit name | qubit no. | CNOT no. | depths no. | TSA _{num} depths | TSA _{dep} depths | optm depths |
|----------------------------|--------------|-------------|---------------|------------------------------|------------------------------|----------------|
| max46_240 | 10 | 11844 | 14257 | 22263 | 25479 | - |
| rd73_252 | 10 | 2319 | 2867 | 4077 | 4602 | - |
| urf4_187 | 11 | 224028 | 264330 | 388383 | 404448 | - |
| sqn_258 | 10 | 4459 | 5458 | 8056 | 8719 | - |
| ham15_107 | 15 | 3858 | 4819 | 7836 | 8925 | - |
| sao2_257 | 14 | 16864 | 19563 | 32902 | 38398 | - |
| sym9_148 | 10 | 9408 | 12087 | 15003 | 16704 | - |
| urf5_280 | 9 | 23764 | 27822 | 44731 | 49954 | - |
| square_root_7 | 15 | 3089 | 3847 | 5525 | 9539 | - |
| urf5_159 | 9 | 71932 | 89148 | 132706 | 148447 | - |
| life_238 | 11 | 9800 | 12511 | 18086 | 20528 | - |
| root_255 | 13 | 7493 | 8839 | 13877 | 16598 | - |
| 9symml_195 | 11 | 15232 | 19235 | 28891 | 33190 | - |
| sym10_262 | 12 | 28084 | 35572 | 53686 | 61183 | - |
| dc2_222 | 15 | 4131 | 5242 | 8280 | 9450 | - |
| co14_215 | 15 | 7840 | 8570 | 17074 | 19297 | - |
| mlp4_245 | 16 | 8232 | 10328 | 16572 | 18702 | - |
| hwb8_113 | 9 | 30372 | 38717 | 62619 | 79839 | - |
| qft_16 | 16 | 240 | 61 | 510 | 681 | - |
| plus63mod4096_163 | 13 | 56329 | 72246 | 115606 | 129148 | - |
| urf1_149 | 9 | 80878 | 99586 | 148531 | 166426 | - |
| urf3_155 | 10 | 185276 | 229365 | 337802 | 373985 | - |
| urf3_279 | 10 | 60380 | 70702 | 114377 | 130334 | - |
| hwb9_119 | 10 | 90955 | 116199 | 159793 | 181048 | - |
| plus63mod8192_164 | 14 | 81865 | 105142 | 165931 | 190486 | - |
| sym9_193 | 11 | 15232 | 19235 | 28378 | 31786 | - |
| ising_model_13 | 13 | 120 | 46 | 120 | 120 | - |
| urf1_278 | 9 | 26692 | 30955 | 50722 | 57343 | - |
| ising_model_16 | 16 | 150 | 57 | 150 | 150 | - |
| ground_state_estimation_10 | 13 | 154209 | 217236 | 189222 | 222867 | - |
| adr4_197 | 13 | 1498 | 1839 | 3046 | 3508 | - |
| clip_206 | 14 | 14772 | 17879 | 31062 | 35367 | - |
| cm85a_209 | 14 | 4986 | 6374 | 11250 | 11661 | - |
| rd84_253 | 12 | 5960 | 7261 | 11507 | 12959 | - |
| dist_223 | 13 | 16624 | 19694 | 33493 | 38917 | - |
| inc_237 | 16 | 4636 | 5864 | 8215 | 9637 | - |
| urf6_160 | 15 | 75180 | 93645 | 157752 | 172536 | - |

Table 10. Comparison of the depth of the output circuit on the IBM Q20