

# Dynamic Circuit Mapping Algorithms for Networked Quantum Computers

Natasha Valentina Santoso

23223788

Supervisors:

Abhishek Agarwal (NPL), Lachlan Lindoy (NPL)

A project report presented in partial fulfillment of the degree

*MSc Quantum Technologies*

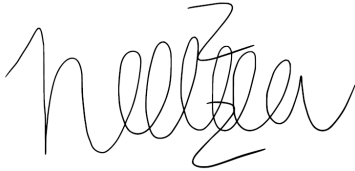
Department of Physics and Astronomy

University College London

August 2024

# Declaration

I, Natasha Valentina Santoso, declare that the thesis has been composed by myself and that the work has not be submitted for any other degree or professional qualification. I confirm that the work submitted is my own. The report may be freely copied and distributed provided the source is explicitly acknowledged.



Natasha Valentina Santoso

22 August 2024

---

*Signature*

---

*Date*

# Table of Contents

<b>List of Tables</b>	<b>v</b>
<b>List of Algorithms</b>	<b>vi</b>
<b>1 Introduction and Background</b>	<b>1</b>
1.1 Distributed Quantum Computing . . . . .	1
1.2 Quantum Computing Architecture . . . . .	2
1.3 Quantum Circuit Mapping . . . . .	3
1.4 Qiskit Framework . . . . .	4
1.4.1 Pass Manager . . . . .	4
1.4.2 Initialization stage . . . . .	5
1.4.3 Layout Stage . . . . .	6
1.4.4 Routing Stage . . . . .	6
1.4.5 Translation Stage . . . . .	7
<b>2 Methodology</b>	<b>8</b>
2.1 Layout . . . . .	9
2.1.1 Distributed Coupling Graph . . . . .	9
2.1.2 Interaction Mapping . . . . .	9
2.1.3 Analysis Pass . . . . .	14
2.2 Routing . . . . .	16
2.2.1 Lookahead Swap . . . . .	16
2.2.2 Transformation Pass . . . . .	20
2.3 Integration . . . . .	22
2.3.1 Staged Pass Manager . . . . .	22
2.3.2 Verification . . . . .	23
<b>3 Results and Analysis</b>	<b>25</b>
3.1 Configurations . . . . .	25
3.2 Evaluation . . . . .	26
3.2.1 Distribution of Additional Swap Gates and Circuit Depth . . . . .	26
3.2.2 The Total Number of Algorithms Successfully Run for Each Layout	29

3.2.3	Layouts for Algorithms . . . . .	30
3.2.4	GHZ . . . . .	31
3.2.5	Deutsch-Jozsa . . . . .	32
3.2.6	Graph State . . . . .	32
3.2.7	VQE . . . . .	32
3.2.8	Portfolio QAOA . . . . .	33
<b>4</b>	<b>Discussion</b>	<b>35</b>
4.1	Time Complexity . . . . .	35
4.1.1	Interaction Mapping Layout . . . . .	35
4.1.2	Lookahead Swap Routing . . . . .	35
4.2	Coupling Graph Options . . . . .	36
4.2.1	Choosing Layout . . . . .	36
4.2.2	Choosing Number of Groups . . . . .	37
4.3	Limitation and Errors . . . . .	38
4.4	Direction of Future Work . . . . .	39
<b>5</b>	<b>Conclusion</b>	<b>41</b>
	<b>Appendix A Source Code</b>	<b>48</b>
	<b>Appendix B Coupling Graph by Group</b>	<b>49</b>
	<b>Appendix C Qubit Mapping Illustration</b>	<b>50</b>
	<b>Appendix D Failed Algorithms by Layout and Group</b>	<b>51</b>
	<b>Appendix E Benchmark Result Table by Group</b>	<b>52</b>

# List of Tables

2.1	Definition of Notations . . . . .	8
2.2	QPI value of gates on logical qubits $(q_i, q_j)$ . . . . .	12
3.1	Number of qubits and group for layouts . . . . .	25
3.2	Circuit size and circuit depth of benchmark algorithms for 5, 10, and 15 qubits . . . . .	26
D.1	Failed algorithms grouped by circuit size . . . . .	51

# List of Algorithms

1	Interaction Layout Mapping . . . . .	13
2	Lookahead Swap Routing . . . . .	21

# Acronyms

**DAG** Directed Acyclic Graph. 12, 13, 16, 17, 20, 35, 36

**DQC** Distributed quantum computing. 1

**ECR** Echoed Cross-Resonance. 3

**ISA** Instruction Set Architecture. 6, 7

**QBN** Qubit Interaction Neighborhood. 12, 14

**QPI** Qubit Pair Interaction. v, 11, 12, 13, 14, 35

**QPU** Quantum Processing Unit. 1, 3, 9, 40

## 2 | Methodology

One of the main challenges in transpilation is mapping logical qubits to physical qubits, especially since physical qubits often have limited connectivity. The qubit-mapping problem is classified as NP-hard [1], meaning that while optimal solutions may be achievable for small quantum circuits, the problem becomes increasingly difficult for larger circuits. Because most quantum hardware only allows two-qubit gates to operate between physically adjacent qubits, two important strategies are required: determining the initial placement of qubits and inserting swap gates when a gate execution violates these connectivity constraints [2].

The initial step in addressing the qubit-mapping problem involves utilizing the physical qubit connectivity and gate priorities to generate an initial qubit mapping that is more efficient than a simple one-to-one layout. The subsequent step introduces a dynamic lookahead window to insert swap gates more efficiently taking into account gates that give positive effect to the swap candidate. This proposed algorithm successfully addresses the coupling limitations of quantum devices and reduces the overhead of inserting swap gates, leading to better performance on noisy quantum hardware.

Table 2.1 provides the definitions for the notations used in the analysis:

Table 2.1: Definition of Notations

Notation	Definition
$n$	number of logical qubits
$q_{1,2,\dots,n}$	logical qubits $q_L$ in quantum circuit
$N$	number of physical qubits
$Q_{1,2,\dots,N}$	physical qubits $Q_P$ on quantum device
$C_{1,2,\dots,N}$	classical register for quantum device
$g_i$	number of gates in the circuit
$d$	depth of the circuit
$G(V, E)$	the coupling graph of the backend
$D[ ][ ]$	the distance matrix of the physical qubits between $Q_i, Q_j$
$\pi()$	a mapping from $q_{1,2,\dots,n}$ to $Q_{1,2,\dots,N}$
$\pi^{-1}()$	a mapping from $Q_{1,2,\dots,N}$ to $q_{1,2,\dots,n}$
$F$	Front layer of quantum circuit



## 2.1 Layout

### 2.1.1 Distributed Coupling Graph

Simulating a distributed quantum computer involves creating a coupling map that represents the connectivity of qubits across different "groups", with each group acting as a separate node or quantum processor within the quantum network. These groups consist of multiple Quantum Processing Unit (QPU), with connections both within each group and between different groups. The coupling graph defines the topology of the QPU within each group, and this is managed using `CouplingMap` [3] class in Qiskit. In this coupling graph, the last qubit of one group is connected to the first qubit of the next group, forming a larger distributed coupling map. By modifying the internal topology of a node (quantum computer) and the distribution of qubits across different groups, various configurations of the quantum network can be explored, as illustrated in Appendix B.

The following functions are used to define the topology within each group:

- `from_line`: creates a coupling map of  $n$  qubits connected in a line.
- `from_grid`: creates a coupling map of qubits arranged in a grid with a specified number of rows and columns.
- `from_ring`: creates a coupling map of  $n$  qubits connected in a ring.
- `from_full`: creates a fully connected coupling map on  $n$  qubits.
- `from_t_horizontal`: creates a coupling map of five qubits arranged in a T-shape, connected at the longer end.
- `from_t_vertical`: creates a coupling map of five qubits arranged in a T-shape, connected at the shorter end.

### 2.1.2 Interaction Mapping

In this section, an algorithm called `InteractionLayoutMapping` will be discussed to determine the most appropriate mapping from logical qubits to physical qubits. During the initialization step, the connectivity of the physical qubits on the backend is determined, and the priority of two-qubit gates is assigned based on their order of appearance in the quantum circuit, with gates appearing earlier (on the left side) being given higher priority [4]. Additionally, a metric is calculated to measure qubit interactions that quantify how qubits interact with each other within a circuit, particularly on how qubits are connected

through multi-qubit gates, such as CNOT gates [5]. This calculation will help to define the order in which quantum gates are prioritized.

**Definition 1.** Physical Connectivity is the number of neighbours of physical qubits ( $Q_P$ ).

A physical quantum device is represented by a coupling graph, which is a directed graph  $(V, E)$ . In this graph,  $Q = \{Q_0, Q_1, \dots, Q_{N-1}\}$  represents the set of physical qubits as vertices ( $V$ ), and the edges ( $E$ ) represent the direct connections between two physical qubits,  $Q_i$  and  $Q_j$ , where a two-qubit gate can be applied. When a physical qubit has higher connectivity, the logical qubit mapped to it has a greater chance of connecting to other logical qubits without needing additional swap gates. Conversely, if a physical qubit has low connectivity, more qubit movements will be needed to satisfy the coupling constraints before a logical gate can be operated. These additional movements increase the need for swap gate insertions, leading to a larger circuit size and depth, which may negatively impact gate fidelity and runtime. For example, Qiskit FakeLondonV2 [6] (Figure 2.1) is a simulated 5-qubit backend with a coupling graph that illustrates the physical connectivity on each qubit.

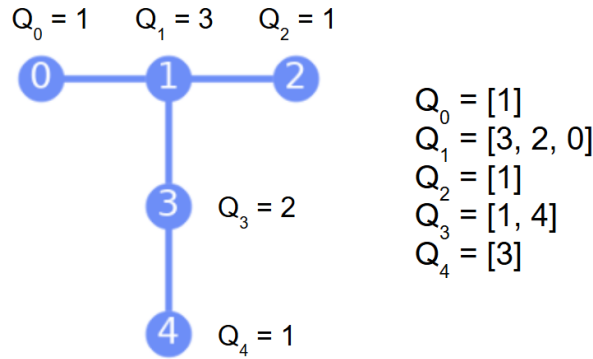


Figure 2.1: Fake London coupling map and its physical connectivity list containing the number of neighbours of each physical qubit  $Q_P$

**Definition 2.** Logical Priority calculates the number of two-qubit gates acting on each logical qubit in a quantum circuit.

If two logical qubits have the same priority value, the one with the earlier index is given precedence. The logical neighbors list identifies other logical qubits that interact with a specific logical qubit  $q_i$  in the circuit. Figure 2.2 lists the number two-qubit gates

operating on the wire of the quantum circuit.

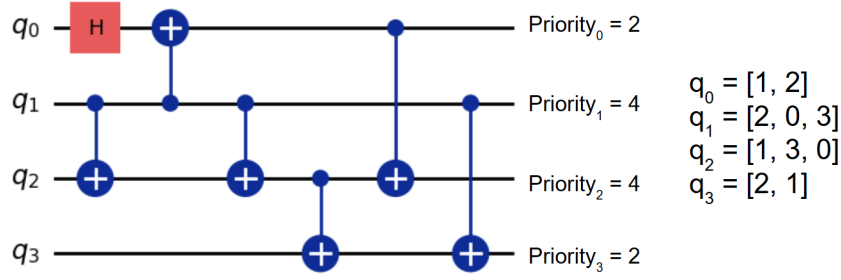


Figure 2.2: Logical priority of a quantum circuit and its logical neighbours list containing the indices of qubits that interact with the logical qubit  $q_i$

**Definition 3.** Gate Weight reflects the importance of each gate  $g$  in a quantum circuit, based on when it occurs in the sequence of operations.

In a quantum circuit, consider a set of logical qubits  $q_L = \{q_0, q_1, \dots, q_{n-1}\}$  and an ordered sequence of gates  $g = \{g_0, g_1, \dots, g_{m-1}\}$ . It is important to note that the ordering of gates introduces some arbitrariness, as certain gates can be implemented in parallel. For example, the last two gates in Figure 2.3,  $(q_0, q_2)$  and  $(q_1, q_3)$ , could be executed simultaneously, making the weights of  $g_6$  and  $g_7$  interchangeable.

Each logical gate  $g_i$  is assigned a weight  $w_i$ , which is calculated as:

$$w_i = m - i \quad (2.1)$$

where  $m$  is the total number of gates in the quantum circuit, and  $i$  is the gate's position in the sequence. A higher weight suggests that the gate occurs earlier in the sequence, potentially having a greater influence on the subsequent operations and the final state of the quantum system. Figure 2.3 shows the gate weights in descending order, with earlier gates assigned higher values.

**Definition 4.** Qubit Pair Interaction (QPI) refers to the measure of interaction logical qubit pair  $(q_i, q_j)$  when they are involved in a multi-qubit quantum gate, such as controlled-NOT (CNOT) gate or any other two-qubit gate.

QPI is determined by summing all gate weights that involve the qubit pair and is repre-

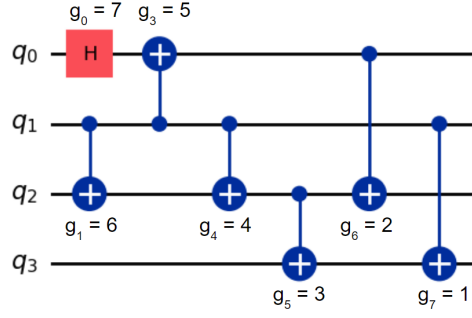


Figure 2.3: Gate weight of a quantum circuit, with earlier gates assigned higher weights

sented as a 2D matrix, as in Table 2.2:

$$\text{QPI}(q_i, q_j) = \sum_{k \in \mathcal{G}(q_i, q_j)} w_k \quad (2.2)$$

where  $\mathcal{G}(q_i, q_j)$  represents the set of indices  $k$  corresponding to gates that involve both qubits  $q_i$  and  $q_j$ , and  $w_k$  is the weight of gate  $g_k$ .

$(q_i, q_j)$	(0, 1)	(0, 2)	(1, 2)	(1, 3)	(2, 3)
QPI	5	2	10	1	3

Table 2.2: QPI value of gates on logical qubits  $(q_i, q_j)$

**Definition 5.** Qubit Interaction Neighborhood (QBN) refers to the sum of weights of the qubit pairs interactions (QPI values) that involve the logical qubit being considered and its neighbouring physical qubits.

A higher QBN value indicates that the qubit is more connected or plays a more significant role within the quantum circuit. The QBN value is defined as:

$$\text{QBN}(q_i) = \sum_j \text{QPI}(q_i, Q_{P,j}) \quad (2.3)$$

where  $q_i$  is the logical qubit under consideration,  $Q_{P,j}$  denotes each physical qubit that is adjacent to or interacts with  $q_i$ , and  $\text{QPI}(q_i, Q_{P,j})$  represents the interaction strength (weight) between  $q_i$  and the  $j$ -th adjacent physical qubit.

Algorithm 1 `InteractionLayoutMapping` outlines the steps for creating an initial qubit mapping, where the inputs are the device's coupling map and the circuit's Directed

Acyclic Graph (DAG), and the output is a layout that maps logical qubits ( $q_L$ ) to physical qubits ( $Q_P$ ). The algorithm starts by calculating physical connectivity from the coupling map and determining the logical qubits' priorities, which are sorted in descending order. The logical qubit with the highest priority is then mapped to the physical qubit with the most connections. For subsequent placements, the algorithm checks if the already assigned physical qubits have adjacent neighbours. If they do, it evaluates Qubit Pair Interaction (QPI) between the logical neighbours and their corresponding qubits, selecting the highest QPI candidate for mapping. If no adjacent neighbours meet this criterion, the algorithm assigns the next highest physically connected qubits to the logical qubits. This process generates all possible mappings for the quantum circuit, which are then ranked based on their total QPI index, and the final output is one initial mapping with the highest QPI rank. If multiple mappings have the same top QPI rank, the final output may be different with each iteration.

---

**Algorithm 1** Interaction Layout Mapping

---

**Input** Coupling Graph  $G(V, E)$ , Circuit DAG

---

```
1: physical connectivity  $\leftarrow E$  on  $Q_P$ 
2: logical priority  $\leftarrow g$  on  $q_L$ 
3: if coupling graph is mostly fully connected then
4:   return one-to-one index mapping
5: end if
6: maps = [ ]
7: while logical priority is not empty do
8:   for maps do
9:      $q_i \leftarrow \max(\text{logical priority})$ 
10:    temp physical connectivity  $\leftarrow$  neighbor of assigned  $Q_X$ 
11:    if all  $q_L$  neighbor already assigned then
12:       $Q_X \leftarrow \max(\text{temp physical connectivity})$ 
13:      maps  $\leftarrow (q_i, Q_X)$ 
14:    else
15:       $Q_X \text{ neighbors}[ ] \leftarrow$  adjacent assigned  $Q_P$ 
16:      Get all assigned  $q_L$  from maps
17:      for available  $Q_X$  neighbors do
18:         $q_j \leftarrow \pi^{-1}(\text{available } Q_X)$ 
19:        QBN candidate  $\leftarrow \text{QPI}[q_i][q_j]$ 
20:      end for
21:       $Q_X \leftarrow \max(\text{QBN value})$ 
22:      maps  $\leftarrow (q_i, Q_X)$ 
23:    end if
24:  end for
25: end while
```

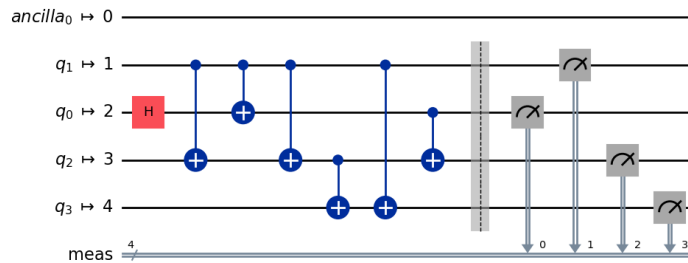
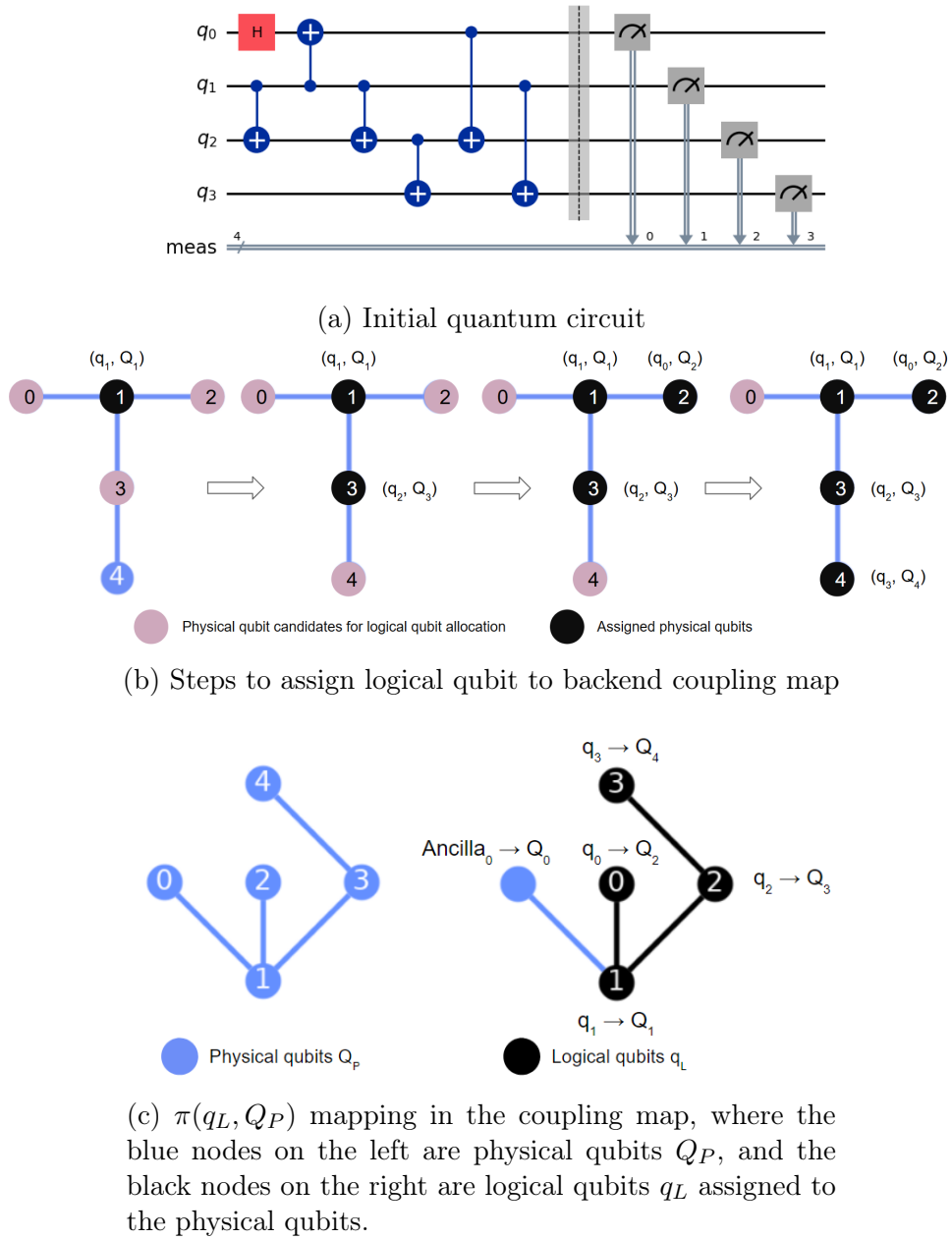
---

**Example 1.** The initial quantum circuit shown in Figure 2.4a is processed through Algorithm 1 `InteractionLayoutMapping` to obtain a more efficient solution for qubit placement. The backend device used is a simulated five-qubit system, `FakeLondonV2`.

First, logical qubit  $q_1$  is directly assigned to  $Q_1$ , which has the most connections with three neighbours. The current mapping is  $[(1, 1)]$ , where the first value in the tuple represents the logical qubit ( $q_L$ ) and the second value represents the physical qubit ( $Q_P$ ). Next, logical qubit  $q_2$  is to be assigned to the available physical candidates  $Q_3$ ,  $Q_2$ , and  $Q_0$ . The  $QPI(q_1, q_2)$  value is 10 for all these candidates, so they are all included in the mapping. The current mapping is  $[(1, 1), (2, 3)], [(1, 1), (2, 2)], [(1, 1), (2, 0)]$ , and the sum of QBN is recorded. Figure 2.4b illustrates the path when physical qubit  $Q_3$  is chosen for logical qubit  $q_2$ . The next logical qubit,  $q_0$ , has possible physical candidates  $Q_0$ ,  $Q_2$ , and  $Q_4$ . Using the mapped logical qubits and their assigned physical qubits, the QPI values are calculated, and the QBN values are summed to determine the maximum value. For the neighbour of  $Q_1$ , the QPI between  $q_0$  and  $q_1$  is 5, while for the neighbor of  $Q_3$ , the QPI between  $q_0$ ,  $q_2$  is 2. The maximum value occurs for both  $Q_0$  and  $Q_2$ . Therefore, the updated mapping for the path  $[(1, 1), (2, 3)]$  is  $[(1, 1), (2, 3), (0, 2)], [(1, 1), (2, 3), (0, 0)]$ . With the last mapping of  $(q_0, Q_2)$ , the last logical qubit is for  $q_3$  has available candidates  $Q_0$  and  $Q_4$ . If  $q_3$  is assigned to  $Q_0$ , the QPI value is 1, while for  $Q_4$ , the QPI value is 3. The final assignment is  $(q_3, Q_4)$  and QBN rank is updated accordingly. The algorithm not only generates all possible paths for mapping but also identifies the path with the highest QBN rank. The chosen layout is  $[(1, 1), (2, 3), (0, 0), (3, 4)]$ , as shown in Figure 2.4c, with a total rank of 18.0. The initial quantum circuit is redrawn as in Figure 2.4d

### 2.1.3 Analysis Pass

Qiskit framework provides specific `AnalysisPass` [7] to change the property set of a quantum circuit. The custom analysis pass requires layout mapping as input, builds the mapping to Qiskit `Layout` [8] class, and implements the abstract run function. The class `layout` constructs a bijective dictionary, mapping virtual qubits  $q_L$  to physical qubits  $Q_P$ . An `AnalysisPass` might be used to evaluate the effectiveness of a given qubit layout, guiding the optimization process and ensuring that the final circuit is well-suited for the physical quantum hardware it will run on.

Figure 2.4: Interaction layout mapping  $(q_L, Q_P)$

## 2.2 Routing

### 2.2.1 Lookahead Swap

When addressing the qubit mapping problem, all two-qubit gates must meet the hardware connectivity constraints. Single-qubit gates, which act on individual physical qubits, are not relevant and can be ignored for the mapping problem. In most cases, it is not feasible to find a mapping that satisfies all connectivity constraints for the entire quantum circuit. Therefore, whenever a gate does not meet these constraints, the mapping between logical and physical qubits must be updated using swap gates. However, adding swap gates can greatly affect the order of subsequent logical gates. The main metrics for evaluating these adjustments are the number of additional swap gates needed and the resulting circuit depth.

**Definition 6.** DAGCircuit [9] is a Directed Acyclic Graph (DAG) representation of a quantum circuit. This intermediate representation allows the manipulation and optimization of quantum circuits.

In the DAG, each node corresponds to a quantum gate or a measurement applied to qubits, while the edges indicate the flow of quantum information between operations, connecting the output of one operation to the input of another. The DAG structure can be analysed by counting the number of qubits and assessing the circuit depth, which may help simplify the circuit by merging gates, eliminating redundant operations, and applying other techniques to reduce the circuit's complexity. Figure 2.5 illustrates the flow of a quantum circuit, where green nodes represent logical qubit inputs, blue nodes represent logical gates  $g$ , and red nodes represent logical qubit outputs. The edges in the graph depict the quantum information flow from input to output. Additionally, each level in the graph corresponds to the quantum circuit depth, so when multiple nodes appear at the same level, it indicates that the gate operations are applied to disjoint physical qubits.

**Definition 7.** The Gate Front Layer refers to a series of quantum gates that have no pending predecessors on their associated qubits.

A layer in a quantum circuit consists of gates that operate on non-overlapping qubits, and



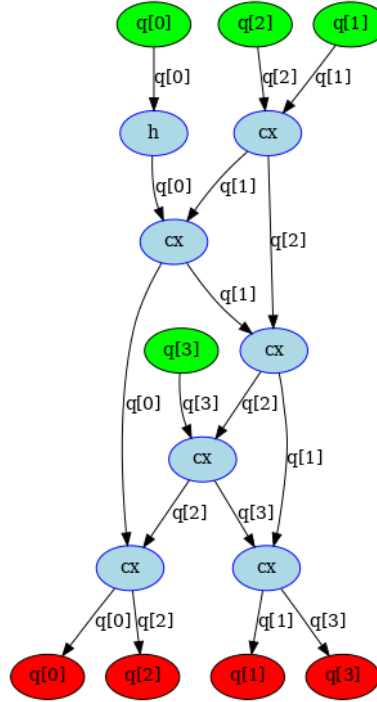


Figure 2.5: Directed Acyclic Graph (DAG) of the previous quantum circuit (Figure 2.4a). DAG consists of three nodes: DAG input nodes (green), DAG operation nodes (blue), and DAG output nodes (red). The arrows indicate the quantum information flow.

the total number of layers corresponds to the circuit depth, denoted as  $d$ . From Qiskit documentation, these layers are generated using a greedy algorithm. The resulting layer includes new DAG operation nodes, DAG input nodes, and DAG output nodes. In the DAG illustrated in Figure 2.5, the front layer consists of the  $CX$  and  $H$  gates.

**Definition 8.** Nearest neighbour distance refers to the shortest distance between a given qubit and its closest neighbouring qubit in a quantum device.

In most quantum hardware, two-qubit gates (like CNOT or CZ gates) can only be directly implemented between qubits that are physically adjacent. If the two qubits involved in the gate are not neighbours, they need to be moved closer together through a series of SWAP operations, which exchange the positions of qubits on the device.

In the setup Figure 2.6, the nearest neighbour distance between  $Q_0$  and  $Q_1$  is 1, the nearest neighbour distance between  $Q_1$  and  $Q_2$  is 1, and the nearest neighbour distance between  $Q_0$  and  $Q_2$  is 2, as they are not direct neighbours.

**Definition 9.** The dependency list for  $q_i$  contains the quantum gates  $g_i$  that are located at logical qubits  $q_L$ . This list only considers two-qubit gates.



Figure 2.6: Example of nearest neighbour distance setup, where distance  $Q_0 - Q_1$  is 1, and distance  $Q_0 - Q_2$  is 2.

The dependency list starts with the front layer of quantum gates that have no preceding gates, which is the leftmost gate in each dependency list<sub>i</sub>. A two-qubit gate is included in two related dependency lists and becomes an active gate when it is the front gate for both associated logical qubits at the same time. As shown in Figure 2.7, there are three front gates ( $g_1, g_2$ , and  $g_4$ ) with gate  $g_1$  being the front gate for both  $dlist_1$  and  $dlist_3$ , making it an active gate. Single-qubit gates are not included in the dependency list because they can be directly executed on a physical qubit without requiring special consideration.

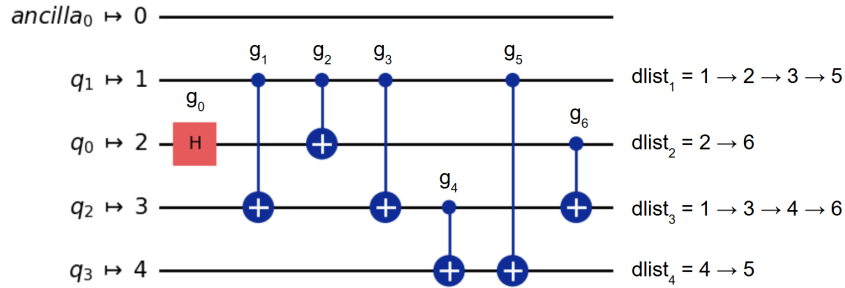


Figure 2.7: The dependency list of a quantum circuit, where  $dlist_1$  contains  $g_1, g_2, g_3$ , and  $g_5$ , which are two-qubit gates that lie on  $q_1$ .

**Definition 10.** The effect of a swap gate on a specific two-qubit gate is defined as the change in the nearest neighbour distance of that gate before and after the swap is applied.

When a swap gate is applied, it alters the nearest neighbour distance for subsequent two-qubit gates that share a logical qubit with the swap. Each swap impacts only one distance, leading to one of three outcomes: the distance can decrease by 1 (positive effect), increase by 1 (negative effect), or remain unchanged (no effect) [10]. In figure 2.8, each rectangle represents a two-qubit gate, with “+” indicates a non-negative effect (0 or +1), while “-” signifies a negative effect (-1).

The effect of a swap gate applied to  $(Q_i, Q_j)$  changes the nearest neighbour distance of  $g_i$ , as determined by the logical to physical mapping before the swap ( $\pi_1$ ) and after the swap ( $\pi_2$ ).

$$\text{effect}(\text{SWAP}, g_i) = \text{dist}(g_i, \pi_1) - \text{dist}(g_i, \pi_2) \quad (2.4)$$

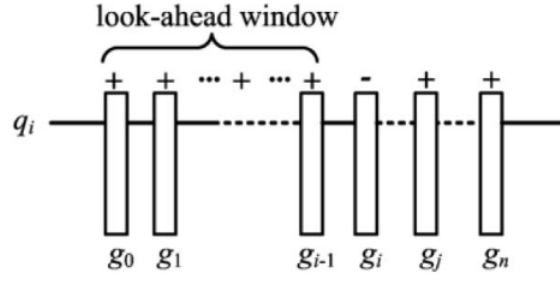


Figure 2.8: Diagram for the dynamic look-ahead technique. Adapted from [10]

**Definition 11.** The look-ahead technique is a heuristic cost function used to evaluate how effectively a swap gate ensures that the circuit meets connectivity constraints.

The method used in Qiskit relies on a fixed window size that is determined before the mapping process begins [11]. This fixed window approach has limitations because it considers a predetermined number of gates across all qubits, regardless of their relevance. Additionally, with a fixed-size window, the heuristic cost function might select a swap gate that helps some gates scheduled later but negatively impacts those that are executed earlier. In extreme cases, the chosen swap could be harmful to current active gates but may be beneficial for future ones, potentially slowing down or even blocking the heuristic algorithm to progress.

To overcome these issues, a dynamic look-ahead technique is introduced as the heuristic cost function. This approach evaluates the effect of a swap gate by focusing only on the two-qubit gates directly affected by the swap. It selectively looks ahead at the two logical qubits ( $q_i$  and  $q_j$ ) that experience a positive effect, stopping when a negative effect is encountered. As illustrated in Figure 2.8, lookahead window only considers from  $g_0$  to  $g_{i-1}$ , where the values are non-negative. The look-ahead window sums all the non-negative (1 or 0) effects on the logical qubit  $q_L$ :

$$\text{Lookahead}(\text{SWAP}, q_i) = \sum_{g \in \text{window}} \text{effect}(g) \quad (2.5)$$

The value of Lookahead includes both logical qubits  $q_i$  and  $q_j$  where the swap gate is inserted,

$$\text{Lookahead}(\text{SWAP}) = \text{Lookahead}(\text{SWAP}, q_i) + \text{Lookahead}(\text{SWAP}, q_j) \quad (2.6)$$

Figure 2.9b demonstrates the transpilation process using Qiskit `TrivialLayout` [12] and `BasicSwap` [13] routing methods. The `TrivialLayout` assigns virtual qubits to physical qubits by mapping  $n$  logical qubits to device qubits  $0, 1, \dots, N - 1$  in ascending order. The `BasicSwap` then performs minimal adjustments by inserting one or more swap gates before a two-qubit gate until the quantum information is located at adjacent physical qubits.

In contrast, Figure 2.9d illustrates the process using the interaction layout from Algorithm 1. Initially, the algorithm converts the circuit to DAG and creates a dependency list for all two-qubit gates, as shown in Figure 2.7. It then iterates over the DAG's level layers, checking the gate operation's coupling map distance  $D[q_i][q_j]$ . If the distance is 1, indicating that the logical qubits are adjacent, the gate is directly applied, and the gate nodes are added to the DAG circuit. However, if the distance is greater than 1, the node is inserted into the active list.

The first four layers have a distance of 1, allowing gates  $g_0$  through  $g_4$  to be executed directly. The next layer includes  $g_5(q_1, q_3)$  and  $g_6(q_0, q_2)$ , for which the algorithm generates a list of physical swap candidates for each gate. For gate  $g_5$ , the swap candidates are  $[(Q_2, Q_1), (Q_3, Q_1), (Q_3, Q_4)]$ , while gate  $g_6$  adds the swap candidate  $(Q_1, Q_0)$ . The algorithm then calculates the lookahead heuristic value for subsequent gates on the dependency list. If a negative effect is encountered during the iteration, the process terminates, and the lookahead window only considers non-negative values. After the calculation, the swap between  $Q_3$  and  $Q_1$  has the highest lookahead value. The next step is to recalculate the distance between the new swap positions for the gates. If the gates can now be executed directly, the gate operation is added to the DAG. The final step is to determine the order of the measurement operations to the classical registers, taking into account the updated logical-to-physical qubit mapping after the swaps.

## 2.2.2 Transformation Pass

The Pass Manager responsible for routing requires the use of a `TransformationPass` [14] to modify the DAG structure representing the quantum circuit. This pass manager takes the device's coupling map and the circuit's DAG as inputs and modifies the DAG by inserting the necessary swap gates to ensure that the circuit adheres to the hardware coupling constraints. The output of this process is a transpiled quantum circuit that is compatible with the physical device and ready for execution.

---

**Algorithm 2** Lookahead Swap Routing

---

**Input** Coupling Graph  $G(V, E)$ , Circuit DAG**Output** Circuit DAG

```
1: procedure CHECK_GATE_CONNECTIVITY( $g_i$ )
2:    $Q_i \leftarrow \pi(q_i)$ 
3:    $Q_j \leftarrow \pi(q_j)$ 
4:    $D[Q_i][Q_j]$ 
5:   if  $D[Q_i][Q_j] == 1$  then
6:      $dlist[q_i]$  pop front
7:      $dlist[q_j]$  pop front
8:     new_dag apply operation back( $g_i$ )
9:   else
10:    act_list.append( $g_i$ )
11:  end if
12: end procedure

13:  $dlist[ ] \leftarrow g_i$  on  $q_n$  iff two-qubit gates
14: new_dag = dag.copy_empty()
15: act_list = [ ]
16: for dag layers do ▷ traverse circuit depth per level
17:   for  $g_i$  in dag operation nodes do
18:     if  $g_i$  is a two-qubit gate of  $(q_i, q_j)$  then
19:       check gate connectivity( $g_i$ )
20:     else
21:       if  $g_i$  is 'measure' operation then
22:          $C_i \leftarrow q_i$  the corresponding updated  $Q_P$  for register
23:       end if
24:       new_dag apply operation back( $g_i$ )
25:     end if
26:   end for
27:   while act_list is not empty do
28:     check gate connectivity( $g_i$ )
29:     candidate list = generate possible physical swaps
30:     for swap( $Q_x, Q_y$ ) in candidate list do
31:       Lookahead (swap( $Q_x, Q_y$ )) =  $\sum_g \text{effect}(Q_x(g)) + \sum_g \text{effect}(Q_y(g))$ 
32:     end for
33:     highest swap ( $Q_x, Q_y$ )  $\leftarrow \max(\text{lookahead})$ 
34:     new_dag apply operation back(SwapGate( $Q_x, Q_y$ ))
35:   end while
36: end for
```

---

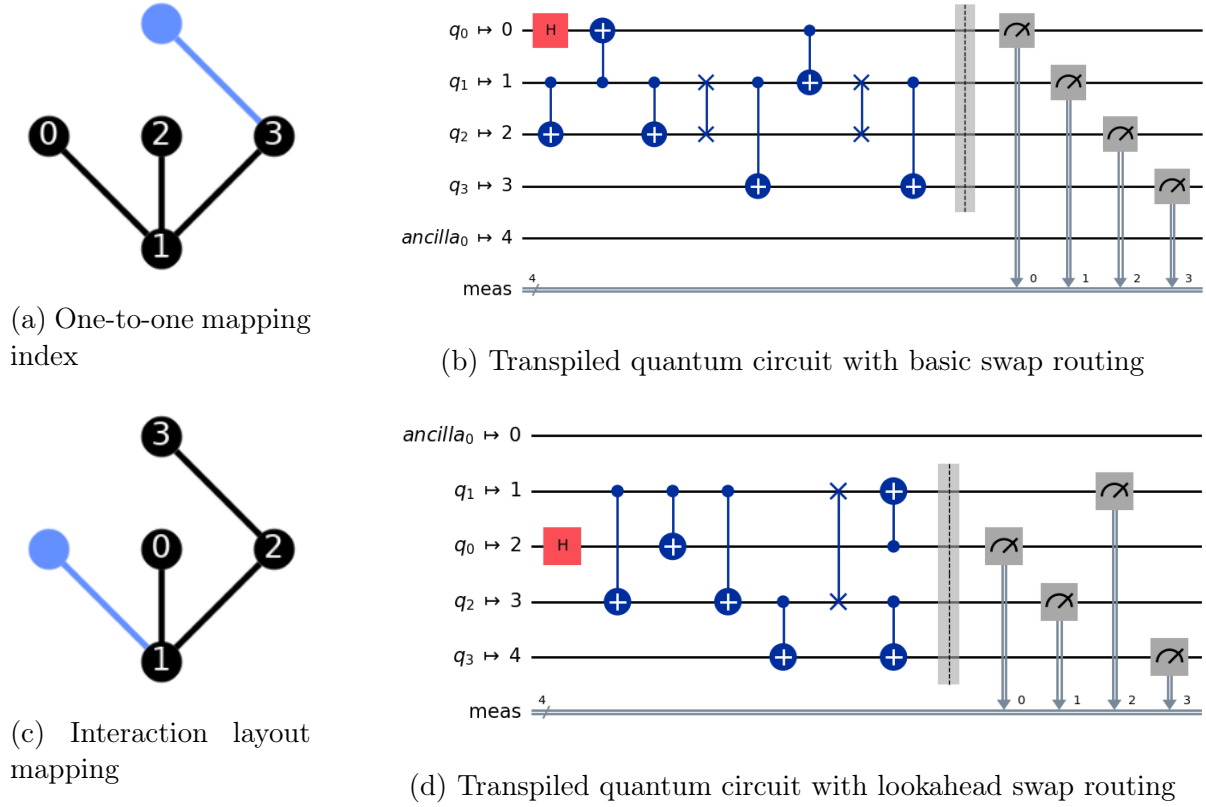


Figure 2.9: Comparison between layout and routing implementation

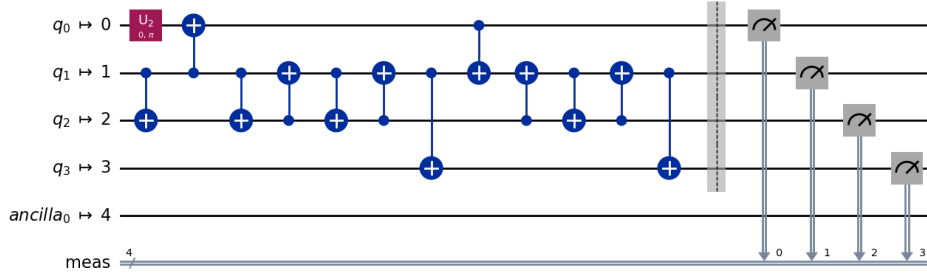
## 2.3 Integration

### 2.3.1 Staged Pass Manager

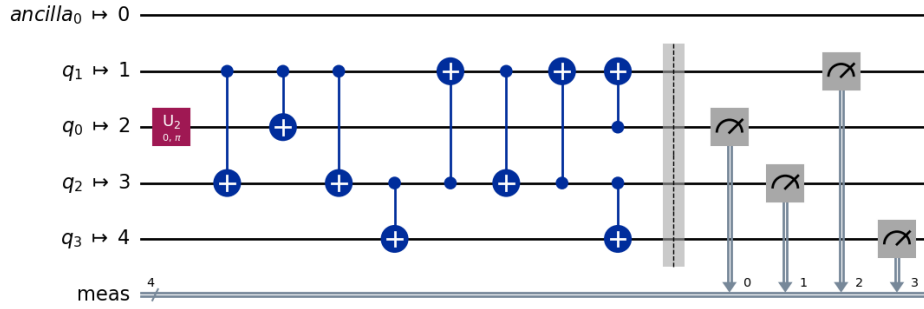
The final step involves integrating both Algorithm 1 for Interaction Layout Mapping and Algorithm 2 for Dynamic Lookahead Swap Routing into the Staged Pass Manager. In this pass manager, the `init` stage unrolls gates involving more than three qubits, and `layout` applies the most effective interaction layout mapping. The `routing` generates the minimal number of SWAP gates required to satisfy the device's coupling constraints. The final stage, `translation`, converts the logical gates into the available basis gate set for the selected device backend. By employing this staged pass manager, the initial quantum circuit is fully translated and optimized for execution on the target backend.

For instance, the initial circuit shown in Figure 2.4a has a circuit size of 11 and a circuit depth of 6. When using the `TrivialLayout` and `BasicSwap` methods, as depicted in Figure 2.9b, the circuit size increases to 17 and depth to 13 (Figure 2.10a). In contrast, the `InteractionLayout` and `DynamicLookaheadSwap` methods, as shown in Figure 2.9d, produce a

more compact circuit with a size of 14 and a depth of 9 (Figure 2.10b).



(a) Translated final layout with BasicSwap routing



(b) Translated final layout with Lookahead routing

Figure 2.10: Comparison for swap gate routing methods

### 2.3.2 Verification

The verification process involves retrieving the results of a quantum computation after a job has been executed on a quantum device or simulator. For this work, `GenericBackend2` [15] is used, and the backend is run without noise. The result is returned as a dictionary, where the keys represent the bitstrings of the measured qubits, and the values represent the number of times each bitstring was measured. This output allows for the analysis of the probability distribution of the qubit states after the quantum circuit has been executed.

The `Result` class [16] of the transpiled quantum circuit, run on Qiskit's preset pass manager, is then compared with the result obtained from the algorithm's swap implementation. It is observed that several of the highest occurrences are identical in both cases (Figure 2.11a and Figure 2.11b), indicating consistency between the two methods.

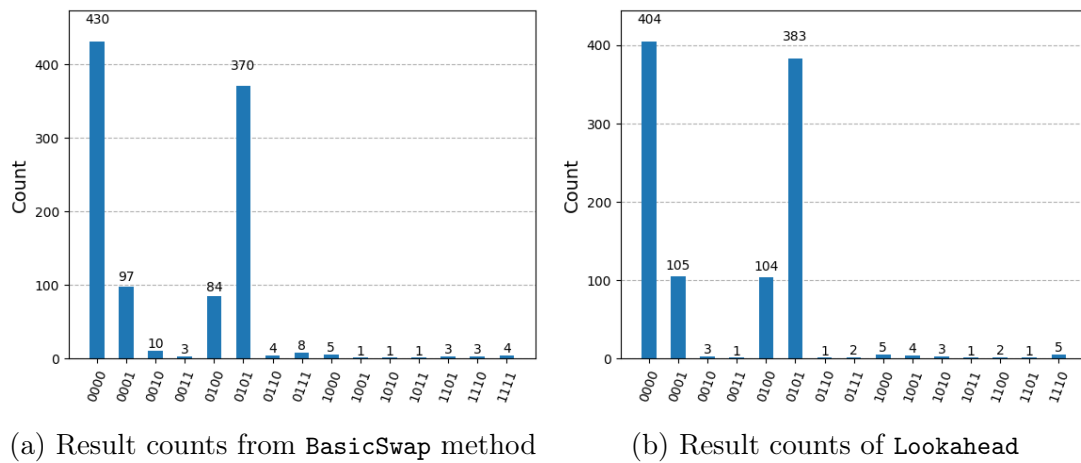


Figure 2.11: Result counts from running FakeLondonV2 backend. The counts vary due to being drawn from probability distributions, with the highest occurrences being compared.



# References

- [1] A. Botea, A. Kishimoto, and R. Marinescu, “On the Complexity of Quantum Circuit Compilation,” en, *Proceedings of the International Symposium on Combinatorial Search*, vol. 9, no. 1, pp. 138–142, Sep. 2021, ISSN: 2832-9163, 2832-9171. DOI: [10.1609/socs.v9i1.18463](https://doi.org/10.1609/socs.v9i1.18463).
- [2] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, “On the Qubit Routing Problem,” en, *arXiv preprint arXiv:1902.08091*, 32 pages, 1190396 bytes, 2019, ISSN: 1868-8969. DOI: [10.4230/LIPICS.TQC.2019.5](https://doi.org/10.4230/LIPICS.TQC.2019.5).
- [3] IBMQuantum, *CouplingMap*, en. [Online]. Available: <https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.CouplingMap> (visited on 07/18/2024).
- [4] H. Liu, B. Zhang, Y. Zhu, H. Yang, and B. Zhao, “QM-DLA: An efficient qubit mapping method based on dynamic look-ahead strategy,” en, *Scientific Reports*, vol. 14, no. 1, p. 13 118, Jun. 2024, ISSN: 2045-2322. DOI: [10.1038/s41598-024-64061-0](https://doi.org/10.1038/s41598-024-64061-0). (visited on 07/01/2024).
- [5] M. Bandic, C. G. Almudever, and S. Feld, “Interaction graph-based characterization of quantum benchmarks for improving quantum circuit mapping techniques,” *Quantum Machine Intelligence*, vol. 5, no. 2, p. 40, Oct. 2023, ISSN: 2524-4914. DOI: [10.1007/s42484-023-00124-1](https://doi.org/10.1007/s42484-023-00124-1).
- [6] IBMQuantum, *FakeLondonV2*, en. [Online]. Available: [https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit\\_ibm\\_runtime.fake\\_provider.FakeLondonV2](https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.fake_provider.FakeLondonV2) (visited on 07/28/2024).
- [7] IBMQuantum, *AnalysisPass*, en. [Online]. Available: <https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.AnalysisPass> (visited on 07/28/2024).
- [8] IBMQuantum, *Layout*, en. [Online]. Available: <https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.Layout> (visited on 07/29/2024).
- [9] IBMQuantum, *DAGCircuit*, en. [Online]. Available: <https://docs.quantum.ibm.com/api/qiskit/qiskit.dagcircuit.DAGCircuit> (visited on 08/11/2024).
- [10] P. Zhu, Z. Guan, and X. Cheng, “A Dynamic Look-Ahead Heuristic for the Qubit Mapping Problem of NISQ Computers,” en, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4721–4735, Dec. 2020, ISSN: 0278-0070, 1937-4151. DOI: [10.1109/TCAD.2020.2970594](https://doi.org/10.1109/TCAD.2020.2970594).

- [11] IBMQuantum, *LookaheadSwap*, en. [Online]. Available: <https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.LookaheadSwap> (visited on 07/31/2024).
- [12] IBMQuantum, *TrivialLayout*, en. [Online]. Available: <https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.TrivialLayout> (visited on 07/17/2024).
- [13] IBMQuantum, *BasicSwap*, en. [Online]. Available: <https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.BasicSwap> (visited on 07/31/2024).
- [14] IBMQuantum, *TransformationPass*, en. [Online]. Available: <https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.TransformationPass> (visited on 07/11/2024).
- [15] IBMQuantum, *GenericBackendV2*, en. [Online]. Available: [https://docs.quantum.ibm.com/api/qiskit/qiskit.providers.fake\\_provider.GenericBackendV2](https://docs.quantum.ibm.com/api/qiskit/qiskit.providers.fake_provider.GenericBackendV2) (visited on 07/29/2024).
- [16] IBMQuantum, *Result*, en. [Online]. Available: <https://docs.quantum.ibm.com/api/qiskit/qiskit.result.Result> (visited on 08/14/2024).

