LONDON'S GLOBAL UNIVERSITY
Department of Computer Science

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

Abstract

This project focuses on simulating qubit placement on physical devices and incorporating swap gates to address connectivity constraints in networked quantum computers. A key challenge in transpilation is the NP-hard problem of mapping logical qubits to physical qubits, particularly given the limited connectivity of physical qubits. This problem requires strategies for initial qubit placement and the insertion of swap gates when connectivity constraints are violated. The proposed approach utilizes physical qubit connectivity and gate priorities to generate an efficient initial mapping, followed by a dynamic lookahead window to insert swap gates more effectively. Although the Lookahead Swap method introduces slightly more swap gates than the default optimized Swap method, it significantly reduces circuit depth, making it very effective for noisy quantum hardware. In distributed quantum computing settings, layouts with higher connectivity perform better because of shorter distance between nodes, and organizing qubits into fewer groups further improves efficiency by simplifying information exchange and reducing circuit depth. This approach is particularly beneficial for networked quantum computers.

Keywords— quantum computing - quantum circuit mapping - networked quantum computers

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.

[illegible]

Natasha Valentina Santoso	22 August 2024
Signature	*Date*

Table of Contents

List of Tables

List of Algorithms

Acronyms

DAG Directed Acyclic Graph. xii, xvi, xvii, xxi, xxxv, xxxvi

DQC Distributed quantum computing. 1

ECR Echoed Cross-Resonance. 3

ISA Instruction Set Architecture. 6, 7

QBN Qubit Interaction Neighborhood. xii, xiv

QPI Qubit Pair Interaction. v, xii, xiii, xiv, xxxv

QPU Quantum Processing Unit. 1, 3, ix, xl

4 | Discussion

4.1 Time Complexity

4.1.1 Interaction Mapping Layout

The Algorithm 1 `InteractionLayoutMapping`, particularly the `calculate_final_maps` method, can be computationally intensive, especially in the worst-case scenario. During its execution, several key operations contribute to its complexity. Initially, the algorithm performs pre-processing steps, such as generating the Qubit Pair Interaction (QPI) matrix and calculating logical priorities and physical connectivity. These steps run in polynomial time relative to the number of physical qubits and two-qubit operations in the DAG. However, the main challenge arises in the core loop of the `calculate_final_maps` function, where the algorithm iteratively assigns logical qubits to physical qubits.
During this process, the algorithm checks and updates possible mappings in a `while logical_priority` loop, which continue until all logical qubits are assigned. Each iteration may create several new mappings, causing the number of possibilities to increase exponentially, as the algorithm must consider at a rate of $O(2^k \times |P|)$, where k is the recursion depth and $|P|$ is the number of physical qubits. As a result, the overall time complexity in the worst case is **exponential**, potentially reaching $O(2^k \times |P| \times n)$, where n is the number of logical qubits. This exponential growth is due to the combinatorial nature of the problem, where multiple candidate mappings are explored and expanded. In practical implementations, this could result in significant computational overhead as the number of qubits increases.

4.1.2 Lookahead Swap Routing

The Algorithm 2 `DynamicLookaheadSwap` is a heuristic method designed to map logical qubits to physical qubits in quantum circuits, especially on hardware with limited connectivity. The goal of the algorithm is to reduce the number of additional swap gates required when implementing the circuit on the target hardware, consequently, reducing the circuit depth as well.
The algorithm begins with an initialization phase, where it preprocesses the quantum circuit and sets up necessary data structures. This phase runs in linear time proportional to the number of gates. The algorithm then categorizes and analyzes dependencies be-

tween gates, focusing on two-qubit operations. This step is also a linear time complexity of $O(G)$, where G is the total number of two-qubit gates in the DAG.

In the main loop, the algorithm iterates over each layer of the circuit and checks whether the current qubit layout allows direct execution of two-qubit gates or if a swap operation is necessary. This `check_gate_connectivity` has a time complexity of $O(G \times n^2)$, where n is the number of qubits. The algorithm then `generate_possible_swaps` operations by considering the neighbours of the involved qubits in the coupling map, with a complexity of $O(G \times n^2 \times D)$, where D is the degree of connectivity in the coupling map.

For each potential swap, the algorithm evaluates its effect on the circuit using the `sum_effect` method. This method assesses how the swap improves qubit placement for future gates by exchanging physical qubits and reducing the distance between qubits. This swap evaluation process can be computationally intensive, with a worst-case time complexity of $O(S \times G)$, where S is the number of candidate swaps. If a beneficial swap is identified, it is applied to the layout at $O(1)$, which further modifies the circuit structure.

Overall, the worst-case time complexity of the `DynamicLookaheadSwap` algorithm is approximately $O(G \times n^2 \times D + S \times G)$, reflecting the number of two-qubit gates, qubits, and the connectivity of the coupling map. Despite the potential for high computational costs, the algorithm's use of heuristics and the pruning of less beneficial swaps helps manage its complexity, making it a practical approach for optimizing quantum circuits on devices with limited qubit connectivity.

4.2 Coupling Graph Options

4.2.1 Choosing Layout

The algorithm tested involves around 20 qubits, which aligns with previous studies [1], [2] that used 5 to 20 qubits in quantum programs sourced from IBM Qiskit [3], [4] and RevLib [5]. The chosen layout configurations - *full*, *grid*, *ring*, *t_horizontal*, and *t_vertical* - were selected to determine which layout offers the best performance based on node connectivity. The *t_horizontal* and *t_vertical* layouts, derived from a 5-qubit T-shaped IBM backend, were specifically tested to assess how the chain's length at either end impacts qubit mapping performance. An illustration of how logical qubits are placed on physical qubits is provided in Appendix C.

When analyzing the distribution of additional swap gates, the *full* layout consistently outperforms the *grid* and *ring* layouts. Full connectivity allows any qubit to interact directly with any other qubit, eliminating the need for intermediate swaps and ensuring the shortest path between qubits is always direct [6]. The *grid* layout ranks second, as it has several nodes with three neighbours, compared to the *ring* layout, where each node has only two neighbours. The *grid* layout requires additional operations to manage qubit connectivity, and the *ring* linear layout structure further limits efficient algorithm implementation, leading to higher resource use and less efficient computation [7]. The *t_horizontal_5_4* and *t_vertical_5_4* layouts show no significant difference in additional swap gates and are similar to the *line_20_1* layout, indicating they share a linear arrangement. In summary, the number of neighbouring qubits significantly impacts the reduction of additional swap gates needed.

Surprisingly, the *grid* layout proved to be the most stable in terms of circuit depth, successfully running all algorithms compared to the *full* and *ring* layouts. The *t_horizontal* and *t_vertical* layouts also performed well, running all 15 algorithms. This stability likely stems from the presence of qubits with three neighbours in these layouts, which enhances their ability to handle circuit depth effectively. Although *t_horizontal_5_4* and *t_vertical_5_4* performed similarly to *line_20_1* in terms of additional swap gates, they exhibited slightly higher variance in circuit depth. The `DynamicLookaheadSwap` algorithm works well with these three layouts because it prioritizes executing as many active gates as possible from the dependency list while performing SWAP operations simultaneously [8]. In contrast, the *ring* layout is limited by its linear qubit configuration, and the *full* layout struggles with large traversal nodes when evaluating swap candidates. This suggests that an optimal number of neighbouring qubits - neither too many nor too few - is necessary to maintain a balanced circuit depth when using `InteractionLayoutMapping` algorithm.

4.2.2 Choosing Number of Groups

According to the results shown in Figure 3.3, Group 2 outperformed the other groups with low variability. This success is attributed to dividing tasks into fewer groups, which reduces the number of communication channels required between groups. With only two groups, communication overhead is minimized, as there is only one direct communication link, simplifying data exchange and reducing potential bottlenecks from waiting on other groups [9]. In contrast, increasing the number of groups complicates coordination, requiring more complex synchronization and qubit allocation across groups [10]. This added complexity can introduce overhead in the initial qubit mapping, as seen in the failed interaction mapping for the *full_5_4* layout. Fewer groups result in simpler coordination, fewer dependencies to manage, and less overall system complexity.

In the context of networked quantum computers, it is still crucial to maintain high connectivity within the individual quantum computers inside each group. Even as tasks are divided to reduce inter-group communication overhead, the internal connectivity of each quantum computer must remain robust to ensure efficient qubit interactions. High connectivity within a quantum computer allows for more flexible and efficient execution of quantum operations, minimizing the need for additional swap gates and reducing circuit depth. This balance between minimizing communication overhead across groups and maintaining strong internal connectivity is key to optimizing the overall performance of quantum networks.

4.3 Limitation and Errors

Appendix D details the layouts where timeouts occurred during quantum circuit transpilation. The `InteractionLayoutMapping` algorithm generally succeeds in mapping logical to physical qubits, except for the *full_5_4* layout, where it failed due to a timeout during qubit placement. This issue arises because the algorithm, when traversing possible neighbours, encounters an excessively large tree. To address this, the algorithm includes an exit function that checks if the coupling map is fully connected by calculating the number of neighbours for each physical qubit. If more than 80% of physical qubits have the maximum number of neighbours, the algorithm exits early and returns a one-to-one mapping. For example, in the best case, the *full_10_2* layout has 9 neighbours for 18 qubits and 10 neighbours for 2 qubits that connecting groups, allowing the algorithm to detect the full connectivity and return the one-to-one mapping quickly. However, the *full_5_4* layout, with 4 neighbours for 14 qubits and 5 neighbours for 6 qubits, fails to meet the 80% threshold, causing the algorithm to continue exploring all possibilities until it runs out of memory. A suggested solution to overcome this problem is to analyze the local graph characteristics and check if the graph is k-connected [11].

Similarly, the `DynamicLookaheadSwap` algorithm generally works well but begins to fail

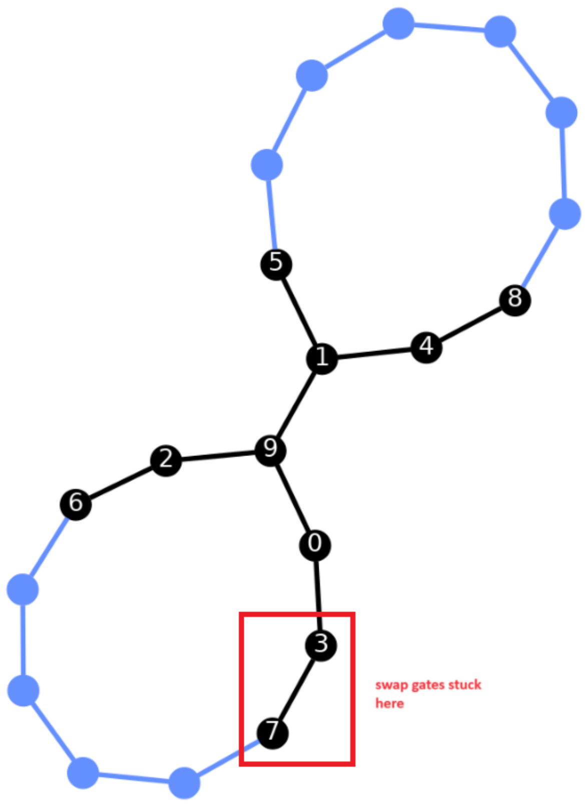


Figure 4.1: The swap operation is stuck because the highest Lookahead value is at Q_7 and Q_3.

for certain algorithms at a circuit size of 10 in the *ring_7_3* and *ring_5_4* layouts, and at a circuit size of 15 in the *full_7_3* layout, as well as the previous two layouts. The swap timeout error likely occurs due to the greedy nature of the qubit placement, which assigns logical qubits to the physical qubits with the highest connectivity. In ring layouts, the algorithm starts mapping from the center and branches out. If there are multiple active gates, the algorithm may direct one gate to a branch end but still fail to find adjacent physical qubits to operate the gate, leading to an infinite loop between two nodes at the end until timeout, as illustrated in Figure 4.1. One strategy to address this issue is to keep an empty list to track assigned physical qubits, but this can cause problems when multiple gates are active. If one gate completes, the next might get stuck because the necessary qubits to go back are already assigned. A potential solution could involve allowing the algorithm to backtrack to certain points in the layout and explore alternative paths [12].

4.4 Direction of Future Work

This section outlines potential future enhancements to optimize the current algorithm:

1. **Incorporating Noise**: The current work assumes quantum devices without noise. However, in practice, the noise levels can vary between physical qubits on the same device, which should be factored into the algorithm [13]. Future work could integrate noise considerations by configuring constraints, instruction sets, qubit properties, operation timing, and other parameters using the IBM `Target` [14] class.
2. **Addressing Communication Costs in Distributed QPU**: The current approach does not account for the communication cost between groups in a distributed QPU. Communication between distant quantum computers can introduce higher noise and errors, affecting gate fidelity and disrupting entanglement [15]. Future work could incorporate communication costs into the initial qubit mapping process [16], either by evenly distributing logical qubits when communication costs are low or by grouping them locally when communication costs are high.
3. **Unidirectional Connectivity Constraints**: The coupling graph used in this work assumes bidirectional connections, where two-qubit gates can operate in both directions. A future enhancement could involve considering unidirectional connectivity constraints, which are more restrictive, and refining the search strategy to improve performance under these conditions [17].
4. **Utilizing Optimization Techniques**: Currently, the algorithm only calculates the difference in additional swap gates before and after transpilation. Future work could introduce additional optimization stages, such as applying passes that check gate commutation rules [18] and optimizing gate decomposition [19], to further enhance performance.

5 | Conclusion

To address the connectivity constraints between logical and physical qubits, two algorithms were introduced: "Interaction Mapping Layout" and "Dynamic Lookahead Swap Routing". Although the `LookaheadSwap` method generates slightly more additional swap gates compared to the `SabreSwap` method, it excels in minimizing circuit depth, achieving the lowest circuit depth among the three strategies. In testing various layouts and groups, *full* and *grid* layouts demonstrated superior performance in distributed settings, offering better connectivity due to shorter paths between nodes and more even load distribution. Additionally, organizing qubits in fewer groups is generally preferable, as it simplifies information exchange leading to fewer additional swap gates and more direct operation of quantum gates within the groups, leading to a lower circuit depth. Overall, choosing layouts with higher connectivity and fewer groups can enhance circuit execution efficiency and performance on quantum hardware, making these approaches particularly well-suited for networked quantum architectures.

References

[1] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for NISQ-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, Providence RI USA: ACM, Apr. 4, 2019, pp. 1001–1014, ISBN: 978-1-4503-6240-5. DOI: 10.1145/3297858.3304023.

[2] 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.

[3] M. Y. Siraichi, V. F. D. Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation," en, in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, Vienna Austria: ACM, Feb. 2018, pp. 113–125, ISBN: 978-1-4503-5617-6. DOI: 10.1145/3168822. (visited on 04/23/2024).

[4] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the ibm qx architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2019. DOI: 10.1109/TCAD.2018.2846658.

[5] R. Wille, D. Gro, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: An Online Resource for Reversible Functions and Reversible Circuits," in *38th International Symposium on Multiple Valued Logic (ismvl 2008)*, ISSN: 0195-623X, Dallas, TX, USA: IEEE, May 2008, pp. 220–225, ISBN: 978-0-7695-3155-7. DOI: 10.1109/ISMVL.2008.43.

[6] B. Hayes, "Computing science: Graph theory in practice: Part i," *American Scientist*, vol. 88, no. 1, pp. 9–13, 2000, ISSN: 00030996. (visited on 08/22/2024).

[7] T. Peham, L. Burgholzer, and R. Wille, "On Optimal Subarchitectures for Quantum Circuit Mapping," en, *ACM Transactions on Quantum Computing*, vol. 4, no. 4, pp. 1–20, Dec. 2023, ISSN: 2643-6809, 2643-6817. DOI: 10.1145/3593594.

[8] C. Zhang, Y. Chen, Y. Jin, W. Ahn, Y. Zhang, and E. Z. Zhang, *A Depth-Aware Swap Insertion Scheme for the Qubit Mapping Problem*, 2020. DOI: 10.48550/ARXIV.2002.07289. (visited on 08/21/2024).

[9] D. Cuomo, M. Caleffi, K. Krsulich, *et al.*, "Optimized Compiler for Distributed Quantum Computing," en, *ACM Transactions on Quantum Computing*, vol. 4, no. 2, pp. 1–29, Jun. 2023, ISSN: 2643-6809, 2643-6817. DOI: 10.1145/3579367.

[10] C. Cicconetti, M. Conti, and A. Passarella, "Resource allocation in quantum networks for distributed quantum computing," in *2022 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2022, pp. 124–132. DOI: 10.1109/SMARTCOMP55677.2022.00032.

[11] A. Cornejo and N. Lynch, "Reliably detecting connectivity using local graph traits," in *Principles of Distributed Systems*, C. Lu, T. Masuzawa, and M. Mosbah, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 87–102, ISBN: 978-3-642-17653-1.

[12] P. Parízek and O. Lhoták, "Fast detection of concurrency errors by state space traversal with randomization and early backtracking," *International Journal on Software Tools for Technology Transfer*, vol. 21, no. 4, pp. 365–400, Aug. 2019, ISSN: 1433-2787. DOI: 10.1007/s10009-018-0484-7.

[13] S. Niu, A. Suau, G. Staffelbach, and A. Todri-Sanial, "A hardware-aware heuristic for the qubit mapping problem in the nisq era," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–14, 2020. DOI: 10.1109/TQE.2020.3026544.

[14] IBMQuantum, *Target*, en. [Online]. Available: https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.Target (visited on 07/30/2024).

[15] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers," en, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, Providence RI USA: ACM, Apr. 2019, pp. 1015–1029, ISBN: 978-1-4503-6240-5. DOI: 10.1145/3297858.3304075.

[16] M. Houshmand, Z. Mohammadi, M. Zomorodi-Moghadam, and M. Houshmand, "An evolutionary approach to optimizing teleportation cost in distributed quantum computation," *International Journal of Theoretical Physics*, vol. 59, no. 4, pp. 1315–1329, Apr. 2020, ISSN: 1572-9575. DOI: 10.1007/s10773-020-04409-0.

[17] S. Sanaei and N. Mohammadzadeh, "Qubit mapping of one-way quantum computation patterns onto 2d nearest-neighbor architectures," *Quantum Information*

Processing, vol. 18, no. 2, p. 56, Jan. 2019, ISSN: 1573-1332. DOI: 10.1007/s11128-019-2177-x.

[18] T. Itoko, R. Raymond, T. Imamichi, A. Matsuo, and A. W. Cross, "Quantum circuit compilers using gate commutation rules," en, in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, Tokyo Japan: ACM, Jan. 2019, pp. 191–196, ISBN: 978-1-4503-6007-4. DOI: 10.1145/3287624.3287701.

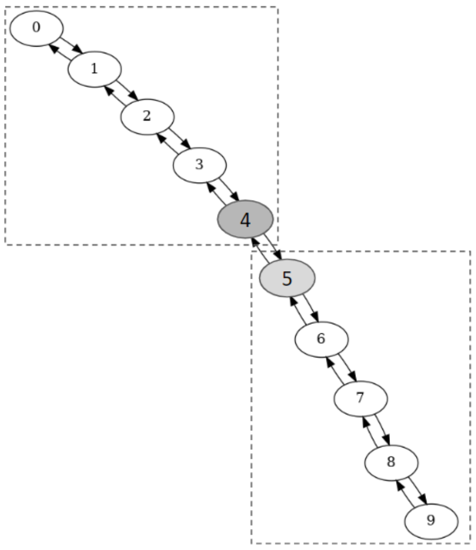
[19] M. S. Rudolph, J. Chen, J. Miller, A. Acharya, and A. Perdomo-Ortiz, "Decomposition of matrix product states into shallow quantum circuits," *Quantum Science and Technology*, vol. 9, no. 1, p. 015 012, Jan. 2024, ISSN: 2058-9565. DOI: 10.1088/2058-9565/ad04e6.

A | Source Code

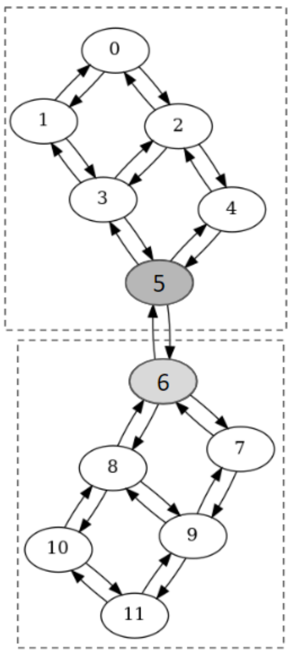
Source code for all of the methods implemented in Chap. 2 for the project can be found in the GitHub repository:

`https://github.com/natashaval/qubit-mapping-distributed-qc`.

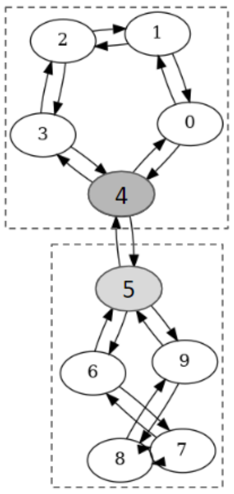
B | Coupling Graph by Group



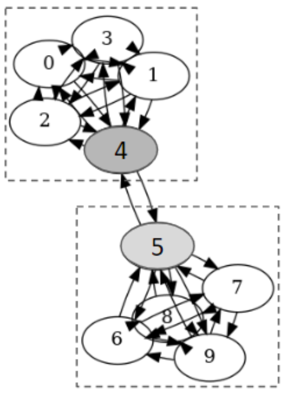
(a) Coupling graph line_10_2



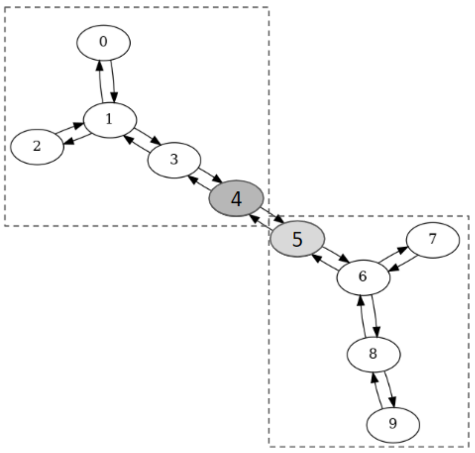
(b) Coupling graph grid_6_2



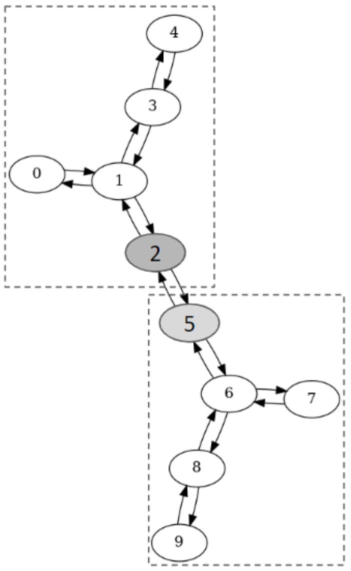
(c) Coupling graph ring_5_2



(d) Coupling graph full_5_2



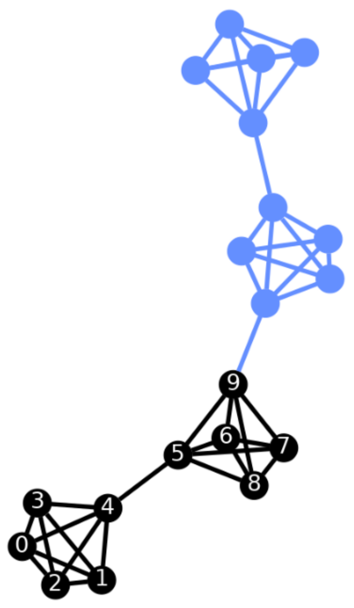
(e) Coupling graph t_horizontal_5_2



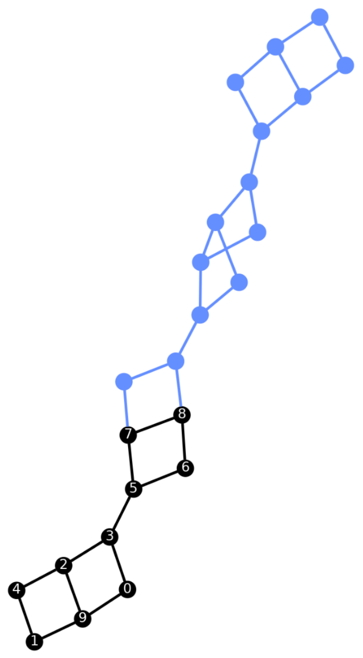
(f) Coupling graph t_vertical_5_2

Figure B.1: Generate group coupling graph

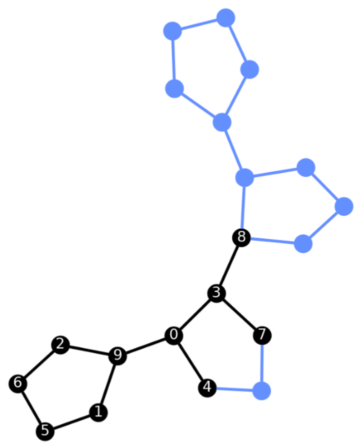
C | Qubit Mapping Illustration



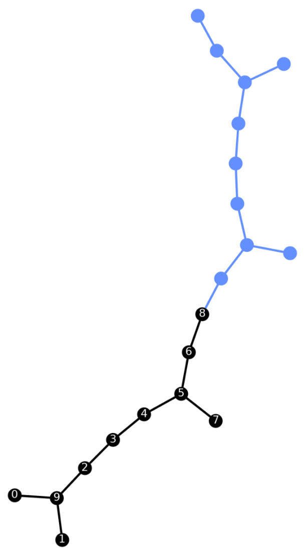
(a) full_5_4 layout



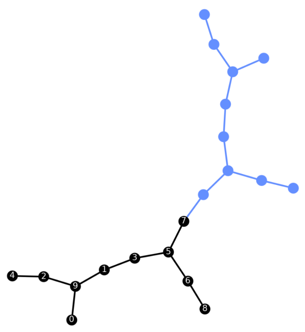
(b) grid_6_4 layout



(c) ring_5_4 layout



(d) t_horizontal_5_4 layout



(e) t_horizontal_5_4 layout

Figure C.1: Deutsch-Jozsa n = 10 in coupling map. Black nodes with numbers illustrate the location of logical qubits, and blue nodes illustrate the available physical qubits.

D | Failed Algorithms by Layout and Group

Table D.1 lists failed algorithms of 10 and 15 qubits, showing the layouts where timeout occured.

Table D.1: Failed algorithms grouped by circuit size

Algorithms	**n = 10**		**n = 15**	
	mapping timeout	swap timeout	mapping timeout	swap timeout
ghz			full_5_4	
qft		ring_7_3 ring_5_4	full_5_4	
wstate			full_5_4	
qftentangled		ring_7_3	full_5_4	ring_7_3 ring_5_4
vqe		ring_7_3 ring_5_4	full_5_4	
twolocalrandom		ring_5_4	full_5_4	full_7_3 ring_5_4
su2random		ring_5_4	full_5_4	full_7_3 ring_5_4
qnn		ring_7_3 ring_5_4	full_5_4	full_7_3 ring_7_3 ring_5_4
portfolioqaoa		ring_7_3 ring_5_4	full_5_4	full_7_3 ring_7_3 ring_5_4
random		ring_7_3 ring_5_4	full_5_4	full_7_3 ring_7_3 ring_5_4
portfoliovqe		ring_7_3 ring_5_4	full_5_4	full_7_3 ring_7_3 ring_5_4

E | Benchmark Result Table by Group

These tables present the benchmark results for various layouts running different algorithms on circuit sizes of 5, 10, and 15 qubits.
layout: layout configuration, where the first number indicates the number of qubits and the second number represents the number of groups.
benchmark: benchmark algorithms used for testing
g: total number of gates
d: circuit depth
s_B: total additional swap gates for "swap basic"
s_S: total additional swap gates for "swap sabre"
s_L: total additional swap gates for "swap lookahead"
Δs_B: gate difference between "swap basic-lookahead" (%)
Δs_S: gate difference between "swap sabre-lookahead" (%)
d_B: circuit depth for "swap basic"
d_S: circuit depth for "swap sabre"
d_L: circuit depth for "swap lookahead"
Δd_B: depth difference between "swap basic-lookahead" (%)
Δd_S: depth difference between "swap sabre-lookahead" (%)