Solving the MaxCut Problem Using the Quantum Approximate Optimization Algorithm

CHENG, HSU-KAI (D13K46009)

Email: d13k46009@ntu.edu.tw

***Abstract*—This report presents an implementation and empirical analysis of the Quantum Approximate Optimization Algorithm (QAOA) for solving the MaxCut problem, a representative NP-hard combinatorial optimization problem. The QAOA framework encodes the MaxCut objective function into a quantum circuit and optimizes a parametrized quantum state to approximate the optimal partition of a graph. QAOA's performance on unweighted graphs of varying sizes and circuit depths was evaluated and its ability to approximate optimal solutions in small instances was demonstrated using both PennyLane and Qiskit frameworks.**

***Index Terms*—Quantum computing, QAOA, MaxCut, combinatorial optimization, variational algorithms.**

I. Introduction

The MaxCut problem is a fundamental NP-hard problem in graph theory and combinatorial optimization. Given an undirected graph $G = (V, E)$, the objective is to partition the vertex set V into two disjoint subsets such that the number of edges connecting vertices across the subsets is maximized. (See Figure 1)

Due to the computational hardness of MaxCut, approximate algorithms are of practical importance. The Quantum Approximate Optimization Algorithm (QAOA), proposed by Farhi et al. [1], provides a hybrid quantum-classical method for approximating solutions to such problems. QAOA alternates between applying problem-specific and mixer Hamiltonians to drive a quantum state toward an optimal solution.

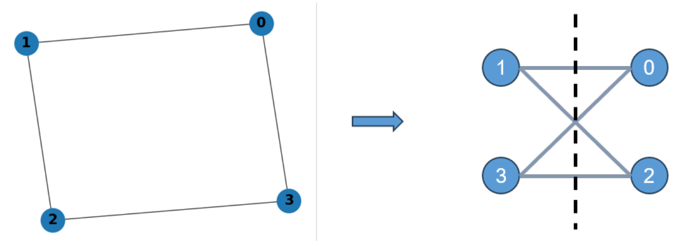

Fig. 1: unweighted MaxCut problem sample

II. Problem Definition

In the unweighted MaxCut problem, each edge contributes 1 to the cut value if it connects nodes in different subsets. The cost function can be defined as:

$$C(z) = \sum_{(j,k)\in E} \frac{1}{2}(1 - z_j z_k), \tag{1}$$

where $z_j \in \{-1, 1\}$ represents the subset assignment of vertex j.

This objective can be encoded into a Hamiltonian suitable for a quantum algorithm by representing the graph structure and using Pauli-Z operators.

III. QAOA Framework

QAOA prepares a parametrized quantum state that is evolved under alternating unitaries derived from two Hamiltonians:

A. Cost Hamiltonian

The cost Hamiltonian encodes the objective function as:

$$H_C = \sum_{(j,k)\in E} \frac{1}{2}(I - Z_j Z_k), \tag{2}$$

where Z_j is the Pauli-Z operator acting on qubit j.

The cost Hamiltonian is designed such that its ground state encodes the optimal solution to the MaxCut problem.

B. Mixer Hamiltonian

To allow transitions between computational basis states, a mixer Hamiltonian is used:

$$H_M = \sum_{j=1}^{n} X_j, \tag{3}$$

with X_j being the Pauli-X operator on qubit j, effectively flipping the qubit state.

The mixer Hamiltonian facilitates transitions between computational basis states, thereby allowing the quantum state to explore the solution space more effectively.

C. Quantum Circuit Structure

The QAOA ansatz of depth p alternates between these two Hamiltonians:

$$U(\vec{\gamma}, \vec{\beta}) = \prod_{l=1}^{p} e^{-i\beta_l H_M} e^{-i\gamma_l H_C}, \tag{4}$$

starting from the uniform superposition:

$$|\psi_0\rangle = |+\rangle^{\otimes n}. \tag{5}$$

The final state is:

$$\left|\psi(\vec{\gamma}, \vec{\beta})\right\rangle = U(\vec{\gamma}, \vec{\beta})\,|\psi_0\rangle. \tag{6}$$

The parameters $\vec{\gamma}$ and $\vec{\beta}$ are optimized using a classical optimizer to maximize $\langle\psi(\vec{\gamma}, \vec{\beta})|H_C|\psi(\vec{\gamma}, \vec{\beta})\rangle$.

IV. Implementation

The test cases were chosen primarily for easy verification. That is, the MaxCut solutions for these small graphs are intuitive and can be manually checked or derived with straightforward logic. This avoids the scenario where the problem is too complex to know whether the algorithm has converged to a correct or optimal solution. In other words, problem instances that are too complicated to verify was intentionally avoided, especially in early-stage experiments.

I implemented QAOA using the PennyLane (primary) and Qiskit quantum machine learning framework **[Note: the OpenQASM issue (as described in the Discussion section) was not fully addressed by the final presentation day so the OpenQASM part (as well as the Qiskit part) was not demonstrated at that time]**. The key steps include:

- Encoding graphs with 4, 6, and 8 vertices.
- Constructing corresponding Hamiltonians.
- Initializing qubits in $|+\rangle$ states via Hadamard gates.
- Applying cost and mixer unitaries alternately.
- Performing basis measurements to estimate the objective value.
- Using classical optimization to tune parameters.

The number of QAOA layers p, which corresponds to the number of alternating applications of the cost and mixer Hamiltonians (i.e., the number of parameter pairs (γ, β)), was varied from $p = 1$ to $p = 8$. This allowed investigating how increasing circuit depth influences solution quality and convergence behavior.

V. Circuit Visualization

This section illustrates the structure of the QAOA circuits constructed for MaxCut. Both PennyLane-generated and IBM OpenQASM circuit visualizations were included, which depict the gates and qubit interactions for a selected graph instance. For simplicity and clarity, the circuit visualizations shown correspond to the case with 4 nodes and a QAOA depth of $p = 1$ (See Figure 2 and Figure 3). This configuration provides an intuitive understanding of the quantum gate structure while keeping the circuit depth minimal.

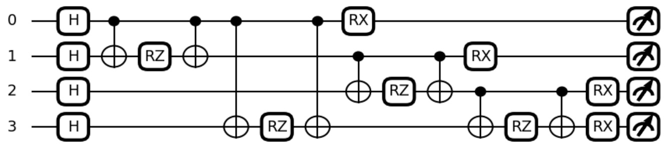

Fig. 2: QAOA Circuit for 4-node MaxCut generated using PennyLane

VI. Experimental Results

A. 4-Vertex Graph

- Number of edges: 4. (See Figure 4 for the original graph problem and an intuitive optimal partition)
- Most frequently sampled states: `0101`, `1010`. (See Figure 5 for the computational basis state distribution from the optimized circuits.)

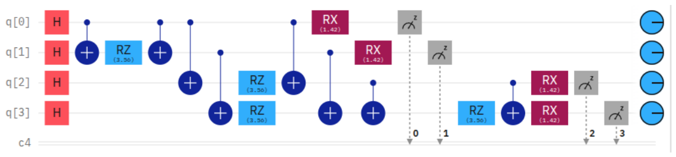

Fig. 3: QAOA Circuit visualized via IBM Quantum Composer (OpenQASM)

- Maximum achievable cut value: 4. (See Figure 6 for objective values regarding different p's and steps.)

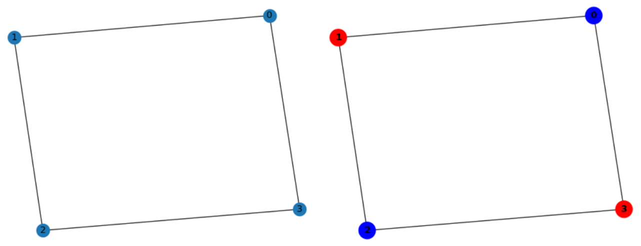

Fig. 4: 4-vertices graph problem and an intuitive optimal partition

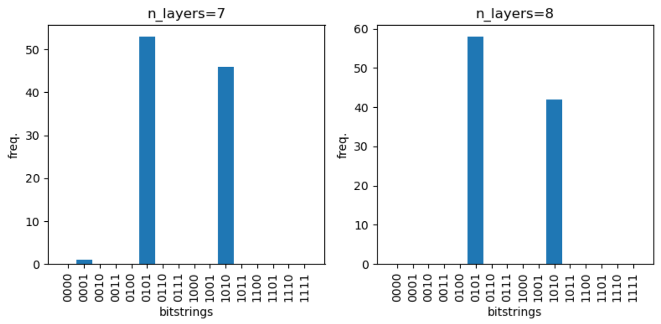

Fig. 5: basis distribution for a 4-wire optimized circuit (p=7 and p=8)

B. 6-Vertex Graph

- Number of edges: 7. (See Figure 7 for the original graph problem and an intuitive optimal partition)
- Most frequently sampled states: `010101`, `101010`. (See Figure 8 for the computational basis state distribution from the optimized circuits.)
- Maximum achievable cut value: 7.(See Figure 9 for objective values regarding different p's and steps.)

C. 8-Vertex Graph

- Number of edges: 10. (See Figure 10 for the original graph problem and an intuitive optimal partition)
- Frequently sampled partitions: `01010101`, `10101010`. (See Figure 11 for the computational basis state distribution from the optimized circuits.)
- Maximum achievable cut value: 10. (See Figure 12 for objective values regarding different p's and steps.)

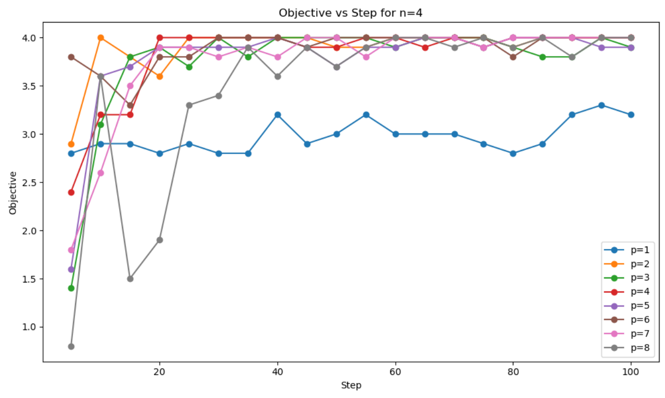

Fig. 6: Convergence of QAOA objective for 4-node graph

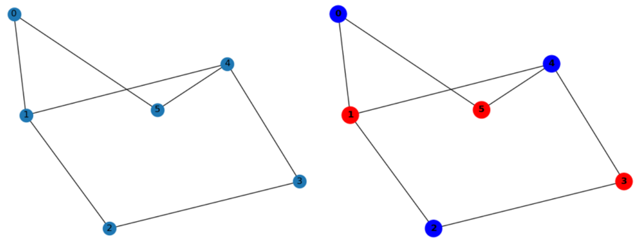

Fig. 7: 6-vertices graph problem and an intuitive optimal partition

VII. Discussion

As the number of qubits (i.e., nodes) increases, the size of the computational basis grows exponentially, following 2^n where n is the number of qubits. This rapid growth can be clearly observed in the frequency distribution plots shown in Figure 5, Figure 8, and Figure 11, where the number of distinct basis states sampled expands significantly from 4 to 6 to 8 qubits. As the number of nodes increases, the quantum state becomes more delocalized, and a wider range of computational basis states begin to appear in the output distribution—reducing the dominance of a few optimal bit strings that are otherwise prominent in smaller graphs. This exponential state space highlights the inherent complexity of quantum systems and underscores the importance of scalable optimization strategies for QAOA.

The results indicate that QAOA can recover optimal or near-optimal MaxCut partitions in small unweighted graphs. The performance improves with increasing circuit depth p, but at the cost of circuit complexity and training time. Notably, the choice of this hyperparameter p is non-trivial, as it directly affects both expressivity and trainability of the quantum circuit. However, its influence can be better understood by visualizing the QAOA energy landscape, which illustrates how the optimization landscape evolves with different values of p (refer to Figure 13 for an example.) Furthermore, as the number of qubits increases, the state space grows exponentially, emphasizing the need for efficient parameter optimization and noise mitigation in real hardware implementations.

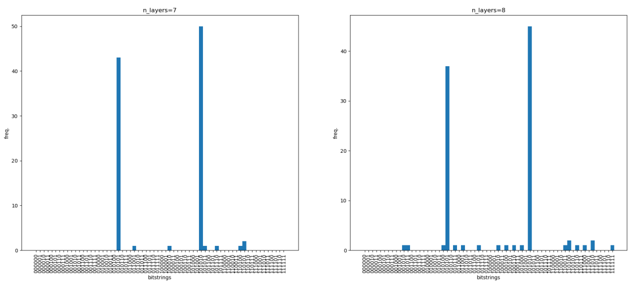

Fig. 8: basis distribution for a 4-wire optimized circuit (p=7 and p=8)

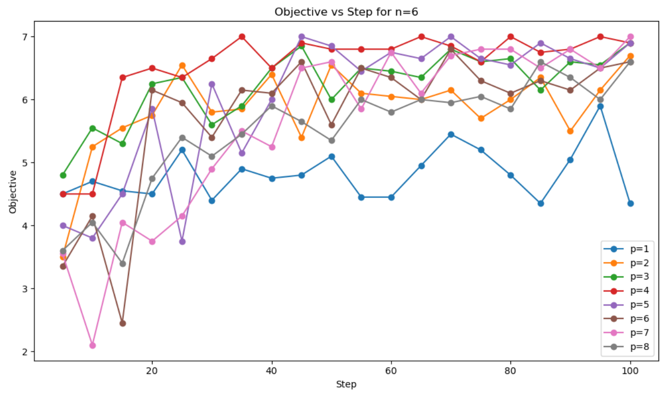

Fig. 9: Objective convergence for 6-node graph

In this project, I also tried to reproduce the results of the PennyLane QAOA MaxCut demo using OpenQASM 2.0. For example, in Figure 2 and 3, the quantum circuits implemented in both frameworks shared the same structure, corresponding to the MaxCut problem on a four-node graph defined by the edge set $(0, 1), (0, 3), (1, 2), (2, 3)$. Additionally, I directly applied the optimal QAOA parameters from the PennyLane demo:

- $\gamma = [-0.91594698, -1.12814301]$, used as 2γ in the cost Hamiltonian.
- $\beta = [0.55739812, -1.12663788]$, used as 2β in the mixer layers.

Although the circuit structure and parameters were matched, the output distributions from the two implementations diverged significantly (Figure 14). In the PennyLane demo, the most probable basis states were `0101` and `1010`, which correspond to the correct MaxCut solutions with cut size 4. However, the OpenQASM circuit executed on IBM Quantum Composer predominantly returned `0000` and `1111`, which are not valid MaxCut solutions for the intended graph.

This discrepancy was traced to a subtle but critical detail: the mapping between graph nodes and quantum registers (qubits). In PennyLane, the correspondence between graph nodes and qubit indices is handled implicitly and can differ from a straightforward assignment such as node $i \rightarrow$ qubit

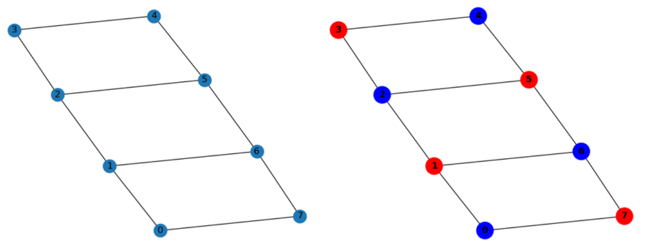

Fig. 10: 8-vertices graph problem and an intuitive optimal partition

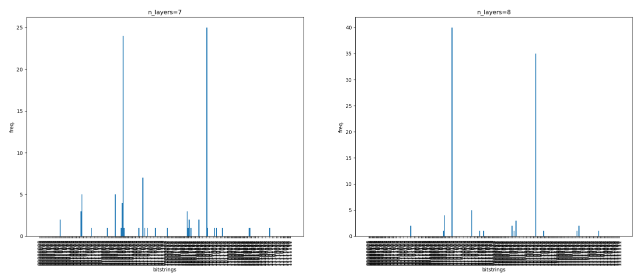

Fig. 11: basis distribution for a 4-wire optimized circuit (p=7 and p=8)

$q[i]$. In contrast, OpenQASM circuits explicitly assume such a mapping unless otherwise modified. As a result, even though the entangling structure (i.e., the cost layer implementing ZZ interactions) was correct, the semantic correspondence between qubits and graph nodes was misaligned. This mismatch led to the application of PennyLane-optimized angles on a logically different graph layout, invalidating the original optimization.

To resolve this inconsistency and ensure accurate MaxCut results, one must either reverse-engineer and match the qubit-to-node mapping used in PennyLane or perform a fresh parameter optimization using the specific qubit layout employed in the OpenQASM circuit.

In light of this, I proceed to perform parameter optimization directly within Qiskit for the given MaxCut graph structure and qubit assignment. This approach ensures that the optimized angles are consistent with the circuit semantics, allowing for faithful reproduction of the MaxCut solution on the target quantum platform (Figure 15 and 16).

While the cut value of a bitstring produced by QAOA can be efficiently evaluated by counting the number of edges it cuts, verifying whether it corresponds to the *optimal* MaxCut remains computationally intractable for arbitrary graphs due to the NP-hard nature of the problem. For large graphs, where the number of vertices increases, not only does classical verification of optimality become impractical, but the QAOA circuit itself requires more qubits and deeper circuits to encode the larger solution space, thereby posing scalability challenges

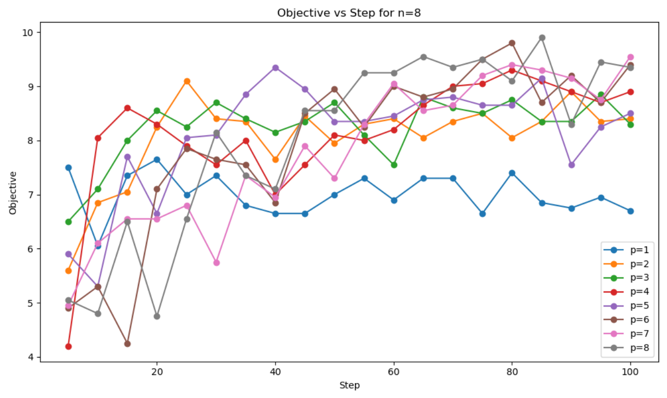

Fig. 12: QAOA performance on 8-node MaxCut

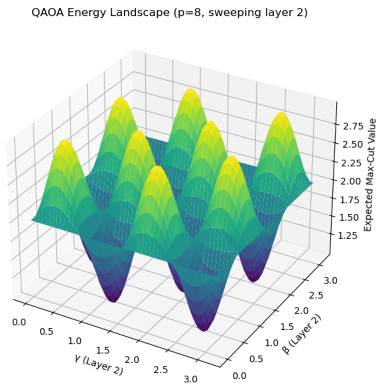

Fig. 13: QAOA energy landscape example

on current quantum hardware. Hereby, I tried to approach the Maxcut problem from a different point of view.

A. Quantum Kernel Approach ***[Note: this section was not demonstrated on the final presentation day due to time limitation]***

Kernel methods are widely used in machine learning to project non-linearly separable data into higher-dimensional feature spaces. The kernel trick allows inner products to be computed implicitly via a kernel function $K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$.

In the quantum case, a kernel is constructed using a parameterized quantum circuit (ansatz) that encodes classical data into quantum states [2]. The kernel value between two inputs x and x' is computed as:

$$K(x, x') = |\langle \psi(x) | \psi(x') \rangle|^2$$

A circuit with Hadamard, RZ (data encoding), and RY (trainable parameters) gates is adopted to build quantum feature maps. These kernels are then used with a classical SVM to learn the partition boundaries of graphs, i.e., which vertices belong to which side of the MaxCut.

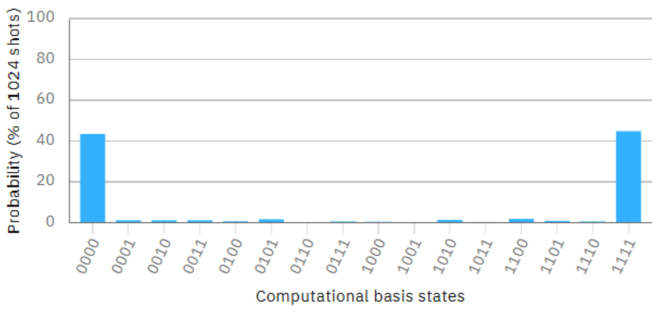

Fig. 14: basis distribution for a 4-wire circuit (p=2) from OpenQASM sampling

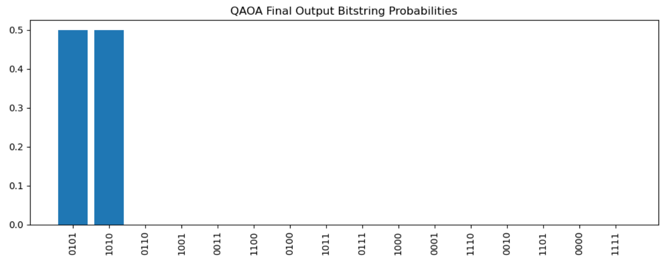

Fig. 15: basis distribution for a 4-wire circuit (p=8) from Qiskit Optimization

B. Quantum Kernel Results ***[Note: this section was not demonstrated on the final presentation day due to time limitation]***

While the MaxCut problem is traditionally framed as finding an optimal partition of vertices that maximizes the number of edges crossing between two disjoint subsets (as shown in Figure 4, where the vertical line crossing each edge represents a “cut” or boundary), this quantum kernel approach offers a complementary perspective. Instead of starting from the graph and seeking the optimal partition, the idea here is to reverse the problem: *given a partition, can we learn a generalizable boundary structure on the original graph?*

That is, by treating the assignment of vertices to different subsets as class labels, the quantum kernel acts as a non-classical feature mapping that encodes the graph structure. A classical SVM then attempts to learn a boundary that captures this partitioning rule, effectively translating the combinatorial cut into a supervised learning task on the graph topology.

This reinterpretation opens a new avenue for analyzing MaxCut through the lens of classification, potentially enabling more generalizable insights into graph partitioning problems beyond exact optimization.

- Kernel matrices were computed from pairwise quantum state overlaps.
- Ideal labels were used to train the kernel via alignment loss. [3]
- A classical SVM trained on the quantum kernel achieved perfect classification for the tested graphs. (Figure 17 and 18 and 19.)

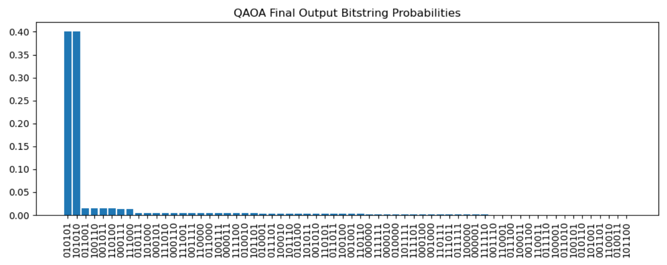

Fig. 16: basis distribution for a 4-wire circuit (p=8) from Qiskit Optimization

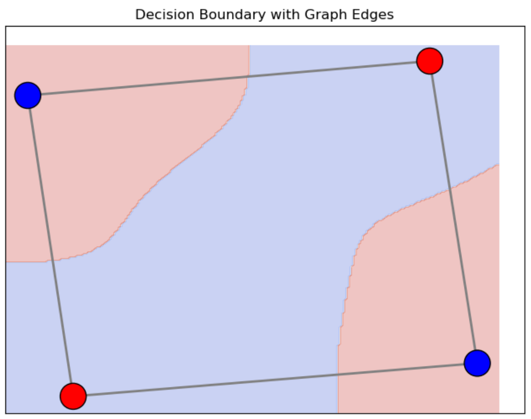

Fig. 17: 4-vertices graph with partition boundary

VIII. Conclusion

QAOA offers a promising quantum approach to solving the MaxCut problem. This study demonstrates its viability for small graph instances and illustrates the impact of circuit depth on performance. QEK offers an alternative by viewing MaxCut as a classification problem. The quantum kernel method can sometimes require fewer qubits than QAOA if only structural features are encoded. Moreover, it opens opportunities to explore other classification-based combinatorial problems. Future work includes applying QAOA to weighted graphs, scaling to larger graphs, benchmarking on actual quantum hardware [4]–[7] and exploring hybrid approaches that combine both QAOA and kernel-based strategies.

Code Availability

The complete implementation and data used in this study are available at: https://github.com/Hsu-Kai/QAOA-MaxCut-Project

Acknowledgments

The author would like to acknowledge the development teams and open-source contributors of PennyLane [8], Qiskit [9], and OpenQASM [10] for providing the tools and frameworks that enabled this research.

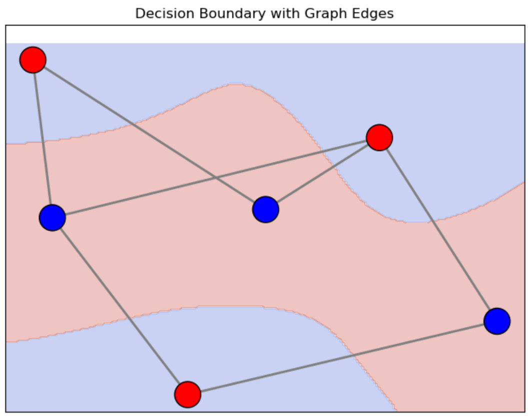

Fig. 18: 6-vertices graph with partition boundary

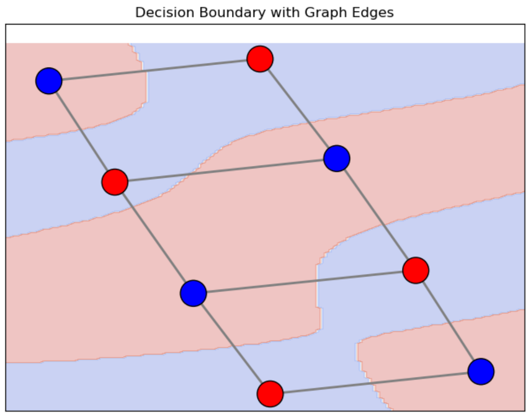

Fig. 19: 8-vertices graph with partition boundary

References

[1] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," 2014. [Online]. Available: https://arxiv.org/abs/1411.4028

[2] T. Hubregtsen, D. Wierichs, E. Gil-Fuster, P.-J. H. S. Derks, P. K. Faehrmann, and J. J. Meyer, "Training quantum embedding kernels on near-term quantum computers," *Phys. Rev. A*, vol. 106, p. 042431, Oct 2022. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.106.042431

[3] T. Wang, D. Zhao, and S. Tian, "An overview of kernel alignment and its applications," *Artificial Intelligence Review*, vol. 43, pp. 179–192, 2015.

[4] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *J. ACM*, vol. 42, no. 6, p. 1115–1145, Nov. 1995. [Online]. Available: https://doi.org/10.1145/227683.227684

[5] E. Boros and P. L. Hammer, "Pseudo-boolean optimization," *Discrete applied mathematics*, vol. 123, no. 1-3, pp. 155–225, 2002.

[6] G. E. Crooks, "Performance of the quantum approximate optimization algorithm on the maximum cut problem," 2018. [Online]. Available: https://arxiv.org/abs/1811.08419

[7] N. Kuete Meli, F. Mannel, and J. Lellmann, "A universal quantum algorithm for weighted maximum cut and ising problems," *Quantum Information Processing*, vol. 22, no. 7, p. 279, 2023.

[8] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin *et al.*, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv preprint arXiv:1811.04968*, 2022. [Online]. Available: https://arxiv.org/abs/1811.04968

[9] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello *et al.*, "Qiskit: An open-source framework for quantum computing," *Zenodo*, 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2562111

[10] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," 2017. [Online]. Available: https://arxiv.org/abs/1707.03429