QAOA Applications
University of Delaware
Spring 2024
Gabriel Cofre

Graph 3-Coloring Problem

1 Introduction

The advent of quantum computing has ushered in a new era of computational capabilities, promising significant advancements in solving some of the most challenging problems in computer science, optimization, and materials science. Among the various quantum algorithms developed, the Quantum Approximate Optimization Algorithm (QAOA) stands out as a pivotal hybrid quantum-classical approach, designed primarily for tackling combinatorial optimization problems. This report delves into the conceptual foundation, practical applications, and the transformative potential of QAOA, with a special focus on its application to the Graph 3-Coloring Problem and other relevant domains.
The Quantum Approximate Optimization Algorithm is engineered to exploit the principles of quantum superposition and entanglement to approximate solutions to combinatorial problems. QAOA operates by encoding the solution to the problem into the ground state of a Hamiltonian, and then uses a variational technique to approximate this ground state. The process involves constructing a parameterized quantum circuit that evolves an initial state through a sequence of unitary transformations characterized by a set of parameters that are optimized classically.
One of the problems that can be solved by using QAOA is the Graph 3-Coloring Problem, an NP-complete problem crucial in theoretical computer science and various practical applications, poses a significant challenge for classical algorithms, especially as the size of the graph scales and the required resources do exponentially too. QAOA presents a novel approach to this problem by leveraging quantum mechanics to explore a vast space of potential solutions more efficiently than classical counterparts. The following report will explore a formal definition of the problem, the solution approach using QAOA, and analysis of the results.

2 Formal Definition

The Graph 3-coloring problem can be succinctly stated as that given an undirected graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. The question is to *determine whether the vertices of a graph can be colored using no more than three colors such that no two adjacent vertices share the same color.* Mathematically, it can be stated as if there exists a coloring function $c : V \to \{1, 2, 3\}$ such that for every edge $(u, v) \in E$ of two adjacent vertices $c(u)$ and $c(v)$, there should receive the same color represented here by the numbers 1, 2, and 3, $c(u) \neq c(v)$.
The Graph 3-Coloring is essentially a constraint satisfaction problem where each edge forms a constraint that its two vertices must be colored differently. This makes the problem computationally intensive, especially as the number of edges and vertices grows. The problem is NP-complete, it is at least as hard as the hardest problems in NP (Nondeterministic Polynomial time). There is no known polynomial-time algorithm that can solve all instances of this problem efficiently.

3 Solution Approach

For a graph $G = (V, E)$, each vertex $v \in V$ has three qubits: $x_{v,1}, x_{v,2}, x_{v,3}$, where $x_{v,c} = 1$ if vertex v is colored with color c, and $x_{v,c} = 0$ otherwise. We must ensure that each vertex is colored with exactly one color (vertex color validity term) and that no two adjacent vertices are colored the same (edge constraint term).
1. Vertex Color Validity Term. For each vertex v, we want that $x_{v,1} + x_{v,2} + x_{v,3} = 1$, meaning exactly one of the qubits should be '1' (indicating the vertex is colored with one color). We can rewrite this condition as:

$$C(x)_{single} = C\sum_{v=1}^{n}\left(1 - \sum_{c=1}^{3} x_{v,c}\right)^2$$

we can ensure that $\left(1 - \sum_{c=1}^{3} x_{v,c}\right)$ is 0 when one color is assigned, and positive when more than one color was assigned (this is why we place a power of two), thus adding a penalty to our cost function, where C and D can be arbitrary positive numbers.
2. Edge Constraint Term. For adjacent vertexes u and v, they should not share the same color. For an edge (u, v), we want that $x_{u,1}x_{v,1}+x_{u,2}x_{v,2}+x_{u,3}x_{v,3} = 0$. If this is not accomplished we can add a penalty to our cost function. It can be written as:

$$C(x)_{adjacent} = D\sum_{(u,v)\in E}^{n}\sum_{c=1}^{3} A_{uv}x_{u,c}x_{v,c}$$

where A_{uv} is the Adjency matrix, $x_{u,c}x_{v,c} = 1$ if both vertices u and v are assigned the same color c; otherwise, it is 0. C and D can be arbitrary positive numbers. The corresponding Ising model is thus:
The final cost function can be expressed as:

$$C(x) = C\sum_{v=1}^{n}\left(1 - \sum_{c=1}^{3} x_{v,c}\right)^2 + D\sum_{(u,v)\in E}^{n}\sum_{c=1}^{3} A_{uv}x_{u,c}x_{v,c}$$

Then, we need to find the Hamiltonian Operator H_C that encodes our cost function, such that $H_C\left|x\right\rangle = C(x)\left|x\right\rangle$. We will use $x_{u,c} = \frac{\mathbf{I} - Z_{v,c}}{2}$. Thus, our ising model is given by:

$$H = C\sum_{v=1}^{n}\left(2I - \sum_{c=1}^{3}(1 - Z_{v,c})\right)^2 + D\sum_{(u,v)=1}^{n}\sum_{c=1}^{3} A_{uv}(1 - Z_{u,c})(1 - Z_{v,c}).$$

This Hamiltonian will be implemented into the QAOA.

3.1 Qiskit Implementation

We start by initializing our state by a characterization of an equal superposition state given by:

$$\left|+\right\rangle^{\otimes n} = \sum_{x\in\{0,1\}^n}\frac{1}{\sqrt{2^n}}\left|x\right\rangle$$

Then, we implement the Hamiltonian belonging to our system by taking advantage of the Matrix Expontential definition in a potential series, $e^M = \sum_{k=0}^{\infty}\frac{1}{k!}M^k$. Pauli matrices are particularly benefited as $e^{-i\theta X} = R_X(2\theta)$, $e^{-i\theta Y} = R_Y(2\theta)$, and $e^{-i\theta Z} = R_Z(2\theta)$ where R is a rotation matrix in function of θ. In our case, we know that:
The H_{single} component aims to achieve a constraint where each vertex has exactly one of three possible colors, expressed via the Pauli Z matrices $Z_{v,c}$. Each term $(1 - Z_{v,c})$ can be realized by a combination of identity and Pauli-Z operations.

The term $(1 - Z_{v,c})$ can be expressed in a quantum circuit as RZ rotations because it is diagonal in the computational basis:

$$U_{single}(\gamma) = e^{-i\gamma H_{single}} = \prod_{v=1}^{n} e^{-i\gamma\left(2I - \sum_{c=1}^{3}(1-Z_{v,c})\right)^2}$$

Each RZ rotation is applied based on the eigenvalues of Z which are ± 1, and thus the RZ rotation angle for each term can be calculated and applied appropriately.
The $H_{adjacent}$ component penalizes adjacent nodes having the same color. This can be handled using CNOT and RZ gates. For each term $(1 - Z_{u,i})(1 - Z_{v,j})$, you can apply controlled phase rotations:

$$U_{adjacent}(\beta) = e^{-i\beta H_{adjacent}} = \prod_{(u,v)} \prod_{i=1}^{3} e^{-i\beta A_{uv}(1-Z_{u,i})(1-Z_{v,i})}$$

This can be implemented using CNOT gates to entangle u and v followed by an RZ gate to add the phase conditionally based on the adjacency matrix A_{uv}. Both qiskit implementations can be found in Figuew 1 and Figure 2.

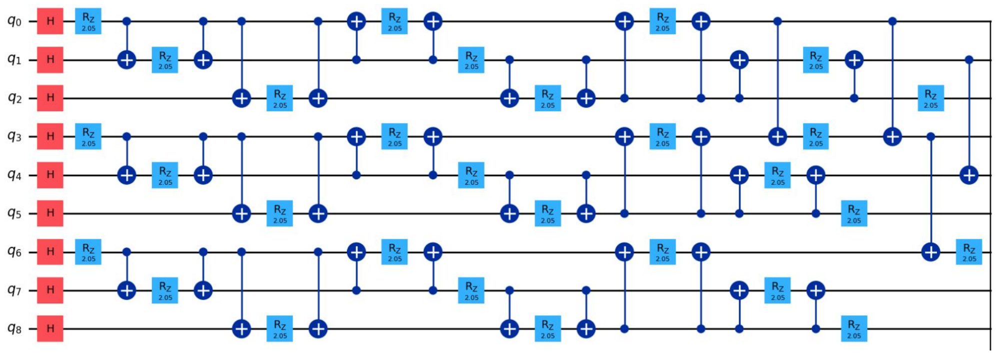


Figure 1: Gate circuit: initialization and single evolution operator

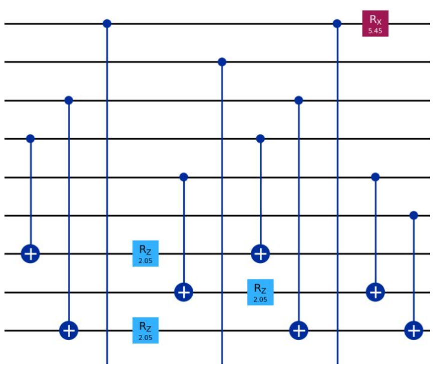


Figure 2: Gate circuit: adjacent evolution operator

The next step is to implement the mixer unitary, which is given by:

$$B = \sum_j X_j$$

where X_j denotes the Pauli-X operator acting on the j-th qubit. The implementation of the mixing unitary, which corresponds to a rotation around the x-axis for each qubit, is mathematically represented by:

$$e^{-i\beta X_j} = \cos(\beta)I - i\sin(\beta)X_j$$

This results in the quantum gate:

$$RX(2\beta) = \begin{bmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{bmatrix}$$

We can repeat p times the set of the hamiltonian evolution encoding and the mixing unitary operator. Finally, we proceed to measure and get into classical data. For the case of 5 qubits our graph looks like Figure 3.

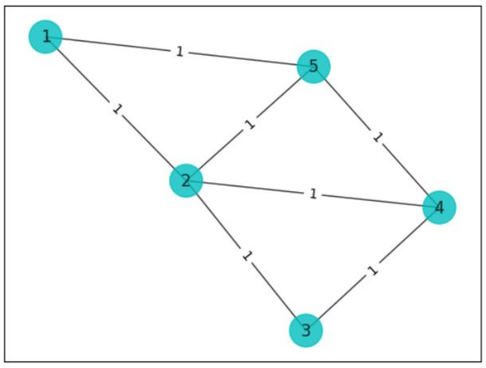


Figure 3: Initial Graph: 5 nodes, 7 edges

4 Results and Conclusion

The 3-coloring graph problem was implemented using the graph in Figure 3 as a baseline. The implementation was executed on the QASM simulator, with an optimization layer to find the optimal parameters of β and γ. The cost function was developed based on the formula in point 3, with all constants and weights set to 1. Table 1 shows the variation in counts as a function of the circuit depth (the number of times the Hamiltonian and mixer were repeatedly applied). The simulation was run for a depth of 60.
Due to the number of qubits used (5) and the 3-color assignment, the states associated with each count were nearly indistinguishable. However, some peaks in certain states suggested possible color associations, though the results were not definitive. The cost function was used as a metric to measure the correlation between error and depth. It was found that after a depth of 40, the cost function began to converge, aligning with the expectation that increased evolution time (depth) brings the circuit closer to its ground state due to Trotterization.
It is important to note that the results from this project are not conclusive in determining whether QAOA can effectively solve the 3-coloring problem. While theoretically feasible, the practical implementation faces challenges due to the exponential scaling of computational resources required as the number of nodes and colors increases.

5 References

- Introduction to the Quantum Approximate Optimization Algorithm and Applications
- Tabi, Z., El-Safty, K. H., Kallus, Z., Hága, P., Kozsik, T., Glos, A., Zimborás, Z. (2020, October). Quantum optimization for the graph coloring problem with space-efficient embedding. In 2020 IEEE international conference on quantum computing and engineering (QCE) (pp. 56-62). IEEE.

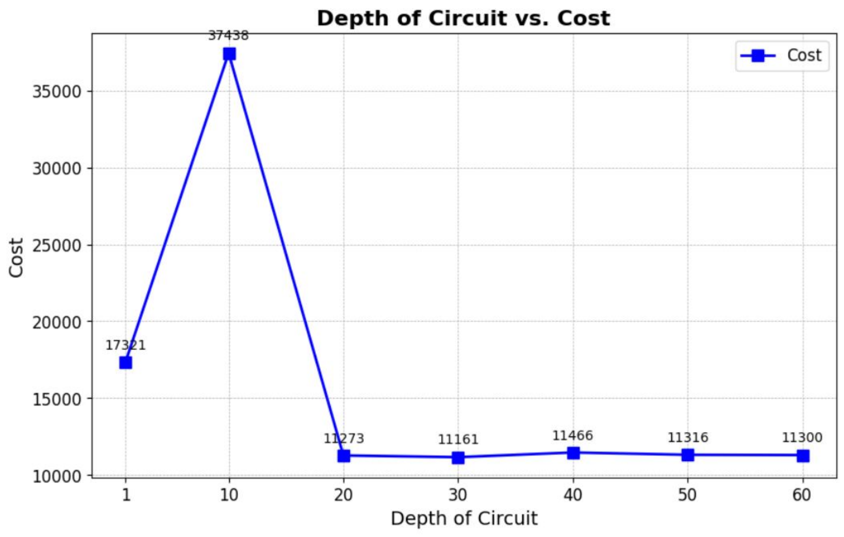


Figure 4: Cost function convergence in function of the depth

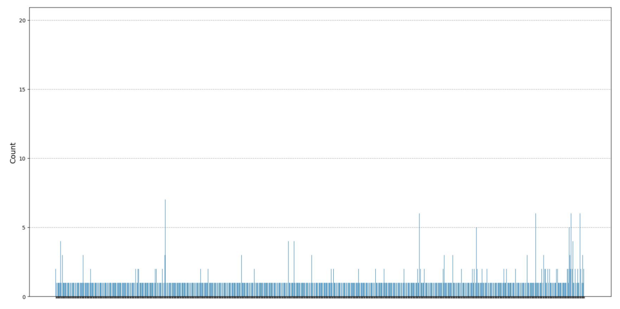
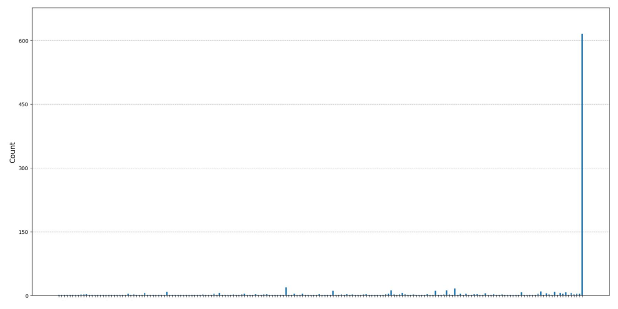
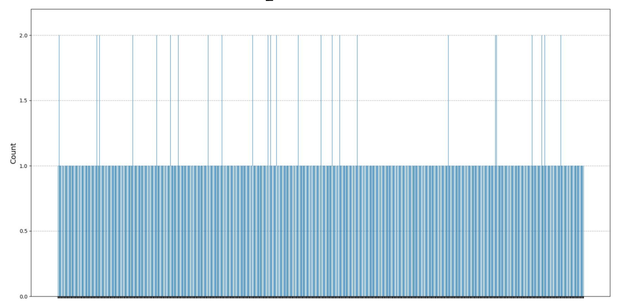
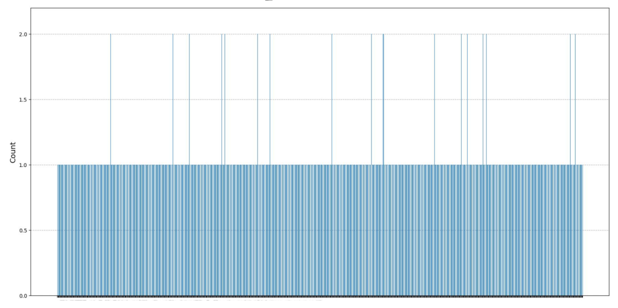
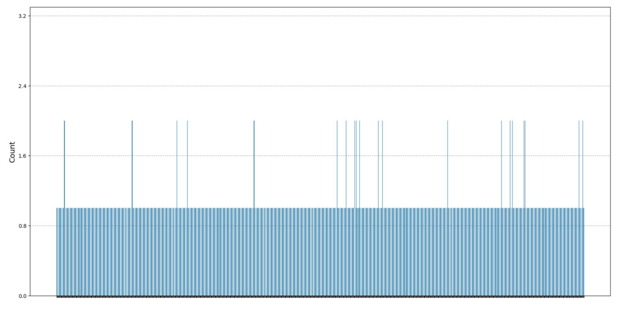
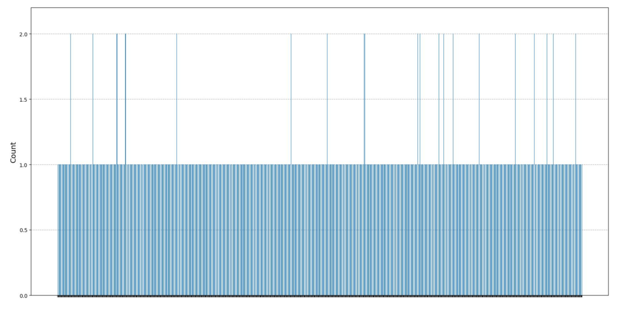
Depth = 1	**Depth = 10**
	
Depth = 20	**Depth = 30**
	
Depth = 40	**Depth = 50**
	

Table 1: Counts of states of 3 Graph Coloring Circuit at different circuit depths