Enhancing the Quantum Approximation Optimization Algorithm using Graph Convolutional Networks for Ising Models

Rebecca Tsekanovskiy, Micheal Halpern
Rensselaer Polytechnic Institute
Troy, NY
`Tsekar@rpi.edu, Halpem2@rpi.edu`

Abstract

The Ising Spin Model is a fundamental concept in the study of Spin Glass systems, which explores particle interactions. Our research focuses on solving the Ising Model using the Quantum Approximation Optimization Algorithm. The Quantum Approximation Optimization Algorithm approximates Hamiltonian minimization, allowing it to find the ground state efficiently. We implemented a graph convolutional network to predict optimal γ and β values alongside a custom created loss function to analyze the expected value for a max cut based off the predictions. Traditional evaluation metrics do not account for expected value requirements and using a custom loss function allows for a direct comparison between known γ and β values. Ultimately, by utilizing a graph convolutional network, we aim to improve the efficiency of the Quantum Approximation Optimization Algorithm.

1 Background

1.1 Ground State of the Ising Model

The Ising Spin Model is a critical component in the study of Spin Glass systems, which are concerned with the interactions of particles. In this model, each spin, denoted as σ_i, can take one of two values: +1 or -1, where σ_i represents the i-th spin. Particles with a spin quantum number of -1 have a spin orientation that is opposite to that of particles with a spin quantum number of +1 [2]. The term w_{ij} signifies the interaction between neighboring spins and their strength. It holds a value of zero for non-neighboring spins and -1 if and only if spins i and j are connected [2]. Each spin σ_i contributes its energy to the Hamiltonian of the spin system. This contribution is defined as follows [1]:

$$H(\sigma) = -\sum_{i<j} w_{ij}\sigma_i\sigma_j \tag{1}$$

This equation represents the Hamiltonian of the system, which is a measure of the total energy of the system. The ground state of the Ising model corresponds to the configuration of spins that minimizes this Hamiltonian and thus minimizes the total energy of the system. Finding the ground state of an Ising spin model can be mapped to a two-dimensional graph, where each particle can be represented as a node. Particles with spin quantum numbers of +1 and -1, which are neighbors, share an edge in the graph. Minimizing the energy of the physical system and finding it's ground state is equivalent to finding the max-cut of this graph. An example of mapping the Ising spin model to a two-dimensional graph and finding the graph's max-cut is 1.

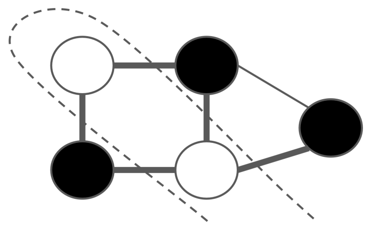

Figure 1: Example: A graph with five nodes and six edges. The dashed line represents the max-cut of this graph: the empty nodes are in set V_1 and the colored nodes are in set V_2. There are other max-cuts, but in this graph the max-cut is five.

1.2 Quantum Approximation Optimization Algorithm

The Quantum Approximation Optimization Algorithm (QAOA) is used for many different types of optimization problems in Quantum Computing, more specifically, the QAOA can generate an approximate Hamiltonian minimization and thus in turn approximately minimize the system's energy and find the approximate ground state of the Ising Model [2]. This is represented as 1 where $\mathbf{b} \in \{0,1\}^n$ [2].

$$C(\mathbf{b}) = \sum_{\alpha=1}^{m} C_\alpha(\mathbf{b}) \tag{2}$$

Equations 1 and 2 are closely related, as 1 represents the Hamiltonian that 2 is approximately minimizing. By mapping the Ising system model to the max-cut problem, we can define the QAOA by the following steps: first, a qubit is allocated for each node in the graph. All qubits are initialized to a state that is an equal superposition. Then, by applying the $U(H_c, \gamma)$, otherwise known as the Hamiltonian cost 3, we are able to calculate the cut value for the given graph for an angle γ. By applying the $U(H_c, \beta)$, otherwise known as the Hamiltonian mixer 4, we are able to modify the current states of the qubits based on the cost calculated in step 3 for an angle β. We will repeat the previous two steps, for a total of p times with different parameters γ_i and β_i, ultimately, creating new states. p is defined as the depth chosen for the algorithm. Based off the function of the depth, $F_p(\gamma, \beta)$, we calculate the expected Hamiltonian cost after running the QAOA algorithm for a given depth p.

1.2.1 Hamiltonian Cost Function and Hamiltonian Mixer Function

The Hamiltonian Cost Function 3 acts on quibits i, j which are the edges in our graph and w_{ij}. This function applies Pauli-z operators on both nodes i and j rotating the quibits about the Z-axis for a given rotation γ. This function changes the quibits states based on their edge connections to other quibits [1].

$$H_C = \frac{1}{2} \sum_{\langle i,j \rangle} w_{ij} \left(I - \sigma_z^{(i)} \sigma_z^{(j)} \right) \tag{3}$$

The Hamiltonian mixer function 4 acts on all quibits in the circuit. A Pauli-X gate is applied, rotating the quibits about the X-axis for a given rotation β. This rotation results in mixing the computational state, changing the probability of receiving a positive or negative state when measuring the quibit [1].

$$H_B = \sum_{j=1}^{n} \sigma_x^{(j)} \tag{4}$$

1.2.2 Finding Gamma and Beta Values

To find suitable γ and β values, we can search the two-dimensional solution space of those values. To search the solution space, we can repeatedly calculate the expected value produced by running the QAOA algorithm at depth one for a given set of γ and β values 2. Searching the solution space this way allows us to find γ and β values which produce low amounts of energy after running QAOA at depth one. These low energy values are likely to lead to lower states of energy when the QAOA algorithm is run at a higher depth, producing a low energy state [1].

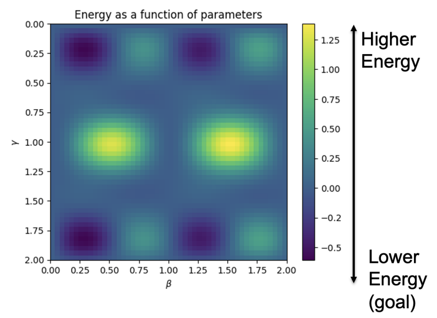

Figure 2: The visual representation of the γ and β value search space for a given graph. In the example, 25 beta and 25 gamma values are calculated making the size of the search space 2500. The darker regions represent γ and β values associated with lower energy states and therefore more promising γ and β values, while the lighter regions represent γ and β values associated with higher energy states [4].

1.3 Graph Convolutional Networks

Graph Convolutional Networks (GCNs), a type of deep learning model operates specifically in a non-Euclidean space, meaning these models best understand data in a graph like structure, like transportation and social networks [5]. GCNs cause each node to continuously update its representation based off the neighboring nodes around. This iterative process makes a GCN an effective choice for capturing complex relationships in Ising spin models. We use GCNs to help find the most optimal γ and β values for the QAOA algorithm. By using a GCN model, hidden patterns in the graph structure are better understood and analyzed [6].

2 Motivation

Quantum computing deals with optimizing problems like Ising Model, a known NP-complete problem. Solving the Ising model enables the solution of all other NP-complete problems [3]. A solution to The Ising Model can be found by using QAOA algorithms in linear time, but finding optimal β and γ parameters requires sampling a solution space. This research aims to optimize solving the Ising Model on the Rensselaer Polytechnic Institute (RPI)-IBM Quantum System One, to understand the challenges and limitations, and use machine learning algorithms to increase the efficiency of finding β and γ parameters for a given graph.

Calculating the starting Gamma and Beta values usually require running the QAOA algorithm at depth one for a range of β and γ parameters. This process either requires a large scale simulation, which is very computationally expensive on classical hardware. This simulation scales linearly with the number of edges and nodes for a given graph. On the other hand, running on real quantum hardware is computationally very slow and expensive due to the repeated transpilation and long wait times for API calls, which are both required to search the β and γ parameter space. Both options are computationally expensive and slow when compared to a neural network, which should give results in a faster time.

3 Methodology

3.1 Data Creation

We created our own dataset containing a graph ID, adjacency list, and optimal γ and β values, which were used to train the graph convolutional network model. To generate this data, we defined four input parameters: minimum number of vertices, maximum number of vertices, minimum fill rate of a graph, and maximum fill rate of a graph. We define the minimum and maximum fill rates as the percentages of edges that are filled in when compared to a complete graph. A fill rate of 1 represents a complete graph. As shown in figure 3, the dataset generated includes graphs with varying densities and structures, as evidenced by the wide range of node degrees.

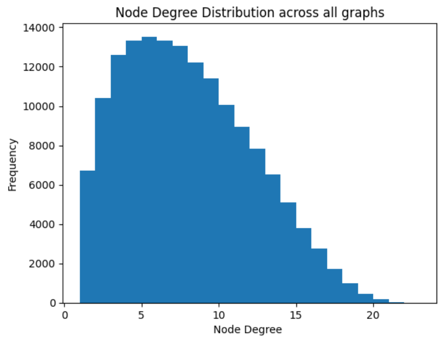

Figure 3: The histogram demonstrates the diversity of node degrees resulting from the predefined minimum and maximum fill rates and vertex counts.

The number of vertices, defined as V, was randomly selected between the predefined minimum and maximum vertices. We then calculated the number of edges for V vertices. We set the minimum number of edges to be equal to the minimum fill rate times the number of edges in a complete graph. Similarly, we set the maximum number of edges to be equal to the maximum fill rate times the number of edges in a complete graph. We then chose a random number between the minimum and maximum number of edges, which is how the adjacency list was formed.

To calculate the optimal gamma and beta values, we ran the QAOA at depth one. We used COBYLA, a SciPy method for constrained optimization, to find the best gamma and beta values that provide a good expected value from the QAOA function. This computation was run on RPI's Aimos's DSC Cluster.

3.2 Evaluation Metrics

We introduce a custom loss function specifically designed for evaluating the γ_i and β_i prediction values in our model. Traditional evaluation metrics such as mean absolute error, root mean squared error, training loss, and testing loss are inadequate for our purposes because they do not account for the requirements focusing on the expected value.

Our custom loss function addresses these requirements by focusing on the expected values produced from a given γ and β value. Here, we define expected value as the average cut of a graph after QAOA at depth 1 for the given γ and β value. This approach allows us to directly compare the expected value calculated with the predicted γ and β values against the expected value calculated from known good γ and β values from our created dataset. This will help ensure a more relevant and precise assessment of prediction accuracy. Traditional metrics result in high error rates because they compare our predicted values with dataset values we assume to be good, which may not reflect the best possible γ and β values. We need a custom loss function because the model could predict a better γ and β value, but because we provided the model with our known values, a normal loss function will indicate bad model performance. This will cause a high error rate if the γ and β values do not match our dataset. By directly comparing to the expected value, our custom loss function provides a more accurate and meaningful evaluation of our model's performance.

3.3 Model Architecture and Implementation

The GCN model implementation is done in PyTorch. We implemented multiple convolutional layers to allow the model to grasp different representations of the graph structures and understand the relationships between nodes. We use batch normalization between the convolutional layers to have higher accuracy, lower loss, and faster convergence. We employed a dropout layer to avoid over fitting and ensure the model is not memorizing specific graph structures. We developed a custom class using PyTorch to implement our Graph Convolutional Network (GCN) for predicting optimal γ and β values. Our dataset, which currently includes 10,000 graph structures, was split into training, validation, and testing sets in a 70/20/10 ratio to ensure proper evaluation. We employed the Adam optimizer with combined weight decay to improve generalization and convergence, penalizing large weights to prevent overfitting . Additionally, we utilized a "reduce on plateau" feature to decrease the learning rate if the validation loss plateaued, enhancing the model's adaptability. To further ensure the model's performance, we implemented "early stopping" with a minimum delta of 0.001, defining this threshold as the criterion for improvement in validation loss. This approach helped prevent overfitting and ensured the model's continuous improvement.

3.3.1 QAOA Qiskit

Later in our code implementation, after finding optimal γ and β values, we generate a circuit that can be run on either RPI-IBM Quantum System One or a quantum simulator. In this context, the GCN model predicts the γ and β parameters, which are then used by the QAOA circuit function to construct, optimize, and run the quantum circuit. Our circuit generation is defined as follows:

```
def QAOA_circuit(beta:float, gamma:float, depth:int, adjacencyList: list[list[int]]):
    num_qubits = len(adjacencyList)
    qc = QuantumCircuit(num_qubits, num_qubits)
    qc.h(range(num_qubits))
    qc.barrier()
    for _ in range(depth):
        for edgeIndex, neighbors in enumerate(adjacencyList):
            for edge in neighbors:
                if edge > edgeIndex:
                    qc.cz(edgeIndex, edge)
                    qc.rz(2 * gamma, edgeIndex)
                    qc.cz(edge, edgeIndex)
        qc.barrier()
        for qubit in range(num_qubits):
            qc.rx(2 * beta, qubit)
        qc.barrier()
    qc.measure(range(num_qubits), range(num_qubits))
    return qc
```

4 Tentative Execution Plan and Expectations

We have implemented the QAOA algorithm and will execute it on graphs in Qiskit SDK v1.1 using the RPI IBM Quantum System One to benchmark its performance. A dataset of β and γ parameters for a set of respective graphs has been created employing traditional techniques. We will continue training the GNN model and evaluate its performance. Subsequently, we will execute the QAOA with model-generated β and γ parameters in Qiskit SDK v1.1 using the RPI IBM Quantum System One and benchmark its performance. Finally, we will evaluate the performance of the model versus traditional methods, focusing on speed and accuracy. We expect a reduction in computation time by 5 $\%$ compared to traditional QAOA parameter selection methods. We expect an average reduction in time taken to find β and γ values. We also expect improvement in the expected value due to the custom loss function.

5 Conclusion

The solving of the Ising Spin System will map over to any other NP-complete problem. Ultimately, by improving the computation of this problem will help the scientific community by improving the solutions to problems like polymer folding, memory, and decision-making in social sciences and economics[3]. Using the benchmarks of QAOA with traditional methods of finding β and γ parameters, we aim to develop a GCN model that finds better β and γ parameters in a shorter amount of time. By performing a comparative analysis, we will attempt to beat the results of benchmarked traditional methods of finding β and γ parameters and understand the specific strengths and weakness in our GCN model.

6 References

[1] Jin, Allison, and Xiao-Yang Liu. 2023. “A Fast Machine Learning Algorithm for the MaxCut Problem.” 2023 IEEE151 MIT Undergraduate Research Technology Conference (URTC), October, 1–5. https://doi.org/10.1109/urtc60662.2023.10534996.

[2] Lu, Yicheng, and Xiao- Yang Liu. 2023. “Reinforcement Learning for Ising Model.” NeurIPS .

[3] Lucas, Andrew. 2014. “Ising Formulations of Many NP Problems.” Frontiers in Physics 2. https://doi.org/10.3389/fphy.2014.00005.

[4] “Quantum Approximate Optimization Algorithm for the Ising Model.” 2024. Google Quantum AI. 2024. https://quantumai.google/cirq/experiments/qaoa.

[5] Uzair Aslam Bhatti, Hao Tang, Guilu Wu, Shah Marjan, and Aamir Hussain. 2023. “Deep Learning with Graph Convolutional Networks: An Overview and Latest Applications in Computational Intelligence” 2023 (February): 1–28. https://doi.org/10.1155/2023/8342104.

[6] Zhang, Si, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. 2019. “Graph Convolutional Networks: A Comprehensive Review.” Computational Social Networks 6 (1). https://doi.org/10.1186/s40649-019-0069-y.