Report

on

Implement Quantum-inspired GEO on a Smaller Problem with Classical and Quantum Generative Models

By

Priyadharshan R

Overview:

Quantum Inspired Generator Enhanced Optimization (GEO) algorithms are one of the most efficient algorithm in Quantum Computing. Quantum-inspired GEO optimization algorithms is a combination of classical algorithms that draw inspiration from principles or techniques found in quantum computing. The goal of these algorithms is to use the certain features of quantum computing such as superposition, entanglement, or quantum annealing to enhance their performance in solving complex optimization problem. These methods often combine classical and probabilistic techniques with elements that copy quantum behavior.

The results show that TN-GEO is among the best compared to these state-of-the-art algorithms; a remarkable outcome given the solvers used in the comparison have been fine-tuned for decades in this real-world industrial application. We see this as an important step toward a practical advantage with quantum-inspired models and, subsequently, with quantum generative models.

Problem:

If Given a undirected graph with weighed edges, the objective of the Max-cut problem is to partition the graph node into two disjoint sets such that the total weight of the edges crossing the partition is maximized.

In other words, we want to divide the graphs nodes into two groups such that the sum of weights of edges between the two groups is as large as possible. This cut is called Max-cut.

Our goal is to find the Optimal cut value and Max-cut value of a given graph using Goemans-Williamson Max-Cut Algorithm using Quantum Approximation Optimization Algorithm (QAOA).

Algorithm:

Goemans-Williamson Max-Cut Algorithm using Quantum Approximation Optimization Algorithm (QAOA).

- **Creating a Graph:**
  - Simple undirected graph with 4 nodes and 4 edges.
  - Each edge connects 2 nodes and goal is to find a cut of a graph that maximizes the total weight of cut edges.
- **Building a QAOA circuit:**
  - Here QAOA circuit is of Max-cut problem is created.
  - The circuit consists of alternating parametrized quantum gates like Hadamard gate and controlled-Z rotation gates.
  - Parameter p refers to the number of alternating gates i.e., the depth of the circuit.
- **Executing and measuring the QAOA circuit:**
  - This circuit is simulated on a 'qasm_simulator' using Qiskit Aer Backend .
  - This circuit is runned for a specified number of times i.e., shots and the measured outcomes are collected.
- **Finding Optimal solution:**
  - The measured results are obtained as different classical bit strings corresponding to qubits states
  - This algorithm searches for the outcome with highest frequency which results in value of optimal cut value.
- **Display the result:**
  - In the end the value of the optimal cut is printed
  - And also the ratio of number of occurrences of the optimal cut to the total number of shots, providing an estimate of Max-cut value.

Program:

```
from qiskit import Aer, QuantumCircuit, transpile, assemble
from qiskit import execute, ClassicalRegister, QuantumRegister
import numpy as np
'''This code is a demonstration of Goemans-Williamson Max-Cut Algorithm using
Quantum Approximation Optimization algorithm.
This is one of the Generative Enchaned Optimization(GEO) algorithms which the
application of Quantum Computing'''

# creating a graph
def create_max_cut_graph():
    # The graph with four nodes and the following edges: (0, 1), (0, 2), (1,
3), (2, 3).
    num_nodes = 4
    edges = [(0, 1), (0, 2), (1, 3), (2, 3)]
    graph = {
        'num_nodes': num_nodes,
        'edges': edges
    }
    return graph

# QAOA circuit
def build_qaoa_circuit(graph, p):
    num_nodes = graph['num_nodes']
    edges = graph['edges']

    q = QuantumRegister(num_nodes, name='q')
    c = ClassicalRegister(num_nodes, name='c')
    qc = QuantumCircuit(q, c)

    qc.h(q)
    for _ in range(p):
        for edge in edges:
            qc.cx(q[edge[0]], q[edge[1]])
            qc.rz(np.pi, q[edge[1]])
            qc.rz(np.pi, q[edge[0]])
    qc.measure(q, c)

    return qc
```

```python
if __name__ == "__main__":
    p = 1
    max_cut_graph = create_max_cut_graph()
    qaoa_circuit = build_qaoa_circuit(max_cut_graph, p)

    # Running QAOA circuit
    backend = Aer.get_backend('qasm_simulator')
    shots = 1000
    qaoa_job = execute(qaoa_circuit, backend=backend, shots=shots)
    result = qaoa_job.result().get_counts(qaoa_circuit)

    # finding optimal solution
    max_cut_value = 0
    optimal_cut = ''
    for cut in result:
        if result[cut] > max_cut_value:
            max_cut_value = result[cut]
            optimal_cut = cut

    print(f"Optimal cut: {optimal_cut}, Max-Cut value: {max_cut_value / shots}")
```

Output:

```
[Running] python -u "d:\Quantum\Qkrishi\geo.py"
Optimal cut: 1111, Max-Cut value: 0.079
```

Conclusion:

This is the demonstration of how the QAOA works with classical optimization techniques to solve the Max-Cut problem using Generator Enhanced Optimization.