Date : 23/9/25

```
            TASK 10: Implement the QAOA algorithm
```

Aim: To implement the Quantum Approximate Optimization Algorithm (QAOA) using Qiskit and PyTorch to solve the Max-Cut problem, a classical NP-hard problem.

Algorithm - QAOA Algorithm

1. Graph Construction
2. Classical Baseline
3. QAOA Circuit Construction
4. Expectation Calculation
5. Hybrid Optimization
6. Circuit Visualization

```
!pip install qiskit qiskit-optimization torch networkx numpy
!pip install qiskit-aer
!pip install pylatexenc
import os
import numpy as np
import networkx as nx
import torch
from qiskit import QuantumCircuit
from qiskit_aer import Aer
from qiskit.quantum_info import Statevector
from qiskit_optimization.applications import Maxcut
from qiskit_optimization.problems import QuadraticProgram
import matplotlib
matplotlib.use(os.environ.get("MPLBACKEND", "Agg"))
import matplotlib.pyplot as plt
def make_graph():
    w = np.array([
        [0.0, 1.0, 1.0, 0.0],
        [1.0, 0.0, 1.0, 1.0],
        [1.0, 1.0, 0.0, 1.0],
        [0.0, 1.0, 1.0, 0.0]
    ])
    G = nx.from_numpy_array(w)
    return G, w
def objective_value(x, w):
    X = np.outer(x, (1 - x))
    w_01 = np.where(w != 0, 1, 0)
    return np.sum(w_01 * X)
def brute_force_maxcut(w):
    n = w.shape[0]
    best = -1
    best_x = None
    for i in range(2**n):
        x = np.array(list(map(int, np.binary_repr(i, width=n))))
        val = objective_value(x, w)
        if val > best:
            best = val
            best_x = x
    return best_x, best
def qaoa_circuit(n_qubits, edges, gammas, betas):
    p = len(gammas)
    qc = QuantumCircuit(n_qubits)
    qc.h(range(n_qubits))

    for layer in range(p):
        gamma = float(gammas[layer])
        for (i, j, w) in edges:
            if w == 0:
                continue
            theta = 2.0 * gamma * w
            qc.cx(i, j)
            qc.rz(theta, j)
            qc.cx(i, j)
        beta = float(betas[layer])
        for q in range(n_qubits):
            qc.rx(2.0 * beta, q)
    return qc
def expectation_from_statevector(statevector, w):
    n = w.shape[0]
    probs = Statevector(statevector).probabilities_dict()
    exp_val = 0.0
    for bitstr, p in probs.items():
        bits = np.array([int(b) for b in bitstr[::-1]])
        exp_val += objective_value(bits, w) * p
    return exp_val
def run_qaoa_with_pytorch(w, p=1, init_std=0.5, maxiter=100,
lr=0.1, finite_diff_eps=1e-3,
backend_name="aer_simulator_statevector"):
    n = w.shape[0]
    edges = [(i, j, w[i, j]) for i in range(n) for j in range(i)
if w[i, j] != 0]
    params = torch.randn(2 * p, dtype=torch.double) * init_std
    params.requires_grad = False
    optimizer = torch.optim.Adam([params], lr=lr)
    backend = Aer.get_backend(backend_name)
    best = {"val": -np.inf, "params": None, "bitstring": None}
    for it in range(maxiter):
        gammas = params.detach().numpy()[:p]
        betas = params.detach().numpy()[p:]
        qc = qaoa_circuit(n, edges, gammas, betas)
        qc.save_statevector()
        res = backend.run(qc).result()
        sv = res.get_statevector(qc)
        exp_val = expectation_from_statevector(sv, w)
        loss = -float(exp_val)
        if exp_val > best["val"]:
                probs = Statevector(sv).probabilities_dict()
                most = max(probs.items(), key=lambda kv: kv[1])[0]
                bits = np.array([int(b) for b in most[::-1]])
                best.update({"val": exp_val, "params":
params.detach().clone(), "bitstring": bits})
        grads = np.zeros_like(params.detach().numpy())
        base = params.detach().numpy()
        eps = finite_diff_eps
        for k in range(len(base)):
            plus = base.copy()
            minus = base.copy()
            plus[k] += eps
            minus[k] -= eps
            g_plus = _qaoa_expectation_with_params(plus, n,
edges, backend, w, p)
            g_minus = _qaoa_expectation_with_params(minus, n,
edges, backend, w, p)
            grad_k = (-(g_plus - g_minus) / (2 * eps))
            grads[k] = grad_k
        params_grad = \
torch.from_numpy(grads).to(dtype=torch.double)
        params.grad = params_grad
        optimizer.step()
        optimizer.zero_grad()
        if it % 10 == 0 or it == maxiter - 1:
            print(f"Iter {it:03d}: expected cut = {exp_val:.6f}, loss = {loss:.6f}")
    return best
def _qaoa_expectation_with_params(flat_params, n, edges,
backend, w, p):
    gammas = flat_params[:p]
    betas = flat_params[p:]
    qc = qaoa_circuit(n, edges, gammas, betas)
    qc.save_statevector()
    res = backend.run(qc).result()
    sv = res.get_statevector(qc)
    exp_val = expectation_from_statevector(sv, w)
    return exp_val
def show_circuit(qc: QuantumCircuit, filename: str = None,
style: str = "mpl"):
    print("\n--- Quantum Circuit ---")
    try:
        print(qc.draw(output="text"))
    except Exception as e:
        print("Failed to draw Quantum Circuit:", e)
    if style == "mpl":
        try:
            fig = qc.draw(output="mpl", interactive=False)
            fig.tight_layout()
            if filename:
                fig.savefig(filename, dpi=200,bbox_inches="tight")
                print(f"[Saved circuit figure to {filename}]")
            else:
                tempname = "qaoa_circuit.png"
                fig.savefig(tempname, dpi=200,bbox_inches="tight")
                print(f"[Saved circuit figure to {tempname}]")
            plt.close(fig)
        except Exception as e:
            print("Matplotlib drawing failed:", str(e))
            print("Fallback: Quantum Circuit diagram above.")
def demo_display_initial_circuit(w, p=1,
filename="qaoa_initial_circuit.png"):
    n = w.shape[0]
    gammas = np.random.randn(p) * 0.8
    betas = np.random.randn(p) * 0.8
    edges = [(i, j, w[i, j]) for i in range(n) for j in range(i)
if w[i, j] != 0]
    qc = qaoa_circuit(n, edges, gammas, betas)
    show_circuit(qc, filename=filename, style="mpl")

def demo_display_best_circuit(w, best_params, p=1,
filename="qaoa_best_circuit.png"):
    n = w.shape[0]
    if isinstance(best_params, torch.Tensor):
        flat = best_params.detach().cpu().numpy()
    else:
        flat = np.array(best_params)
    gammas = flat[:p]
    betas = flat[p:]
    edges = [(i, j, w[i, j]) for i in range(n) for j in range(i)
if w[i, j] != 0]
    qc = qaoa_circuit(n, edges, gammas, betas)
    show_circuit(qc, filename=filename, style="mpl")
if __name__ == "__main__":
    G, w = make_graph()
    print("Graph edges:", list(G.edges()))
    bf_x, bf_val = brute_force_maxcut(w)
    print("Brute-force best:", bf_x, "value:", bf_val)
    demo_display_initial_circuit(w, p=1,
filename="qaoa_initial_circuit.png")
    best = run_qaoa_with_pytorch(w, p=1, init_std=0.8,
maxiter=80, lr=0.2, finite_diff_eps=1e-3)
    print("QAOA best expected value:", best["val"])
    print("Most-likely bitstring found:", best["bitstring"])
    exact_val = objective_value(best["bitstring"], w)
    print("Exact value of that bitstring:", exact_val)
    if best["params"] is not None:
        demo_display_best_circuit(w, best["params"], p=1,
filename="qaoa_best_circuit.png")
    else:
        print("No best params found to display.")
```

```
«q_2: ┤ Rz(-2.2245) ├┤ X ├┤ Rx(1.9293) ├
«     └─────────────┘└─┬─┘├────────────┤
«q_3: ─────────────────■──┤ Rx(1.9293) ├
«                         └────────────┘
[Saved circuit figure to qaoa_initial_circuit.png]
Iter 000: expected cut = 2.477099, loss = -2.477099
Iter 010: expected cut = 3.182933, loss = -3.182933
Iter 020: expected cut = 3.113887, loss = -3.113887
Iter 030: expected cut = 3.202149, loss = -3.202149
Iter 040: expected cut = 3.228003, loss = -3.228003
Iter 050: expected cut = 3.234326, loss = -3.234326
Iter 060: expected cut = 3.235256, loss = -3.235256
Iter 070: expected cut = 3.236372, loss = -3.236372
Iter 079: expected cut = 3.236769, loss = -3.236769
QAOA best expected value: 3.237060074331573
Most-likely bitstring found: [1 0 0 1]
Exact value of that bitstring: 4

--- Quantum Circuit ---
     ┌───┐┌───┐┌─────────────┐┌───┐┌───┐┌─────────────┐┌───┐┌────────────┐»
q_0: ┤ H ├┤ X ├┤ Rz(0.57618) ├┤ X ├┤ X ├┤ Rz(0.57618) ├┤ X ├┤ Rx(2.5219) ├»
     ├───┤└─┬─┘└─────────────┘└─┬─┘└─┬─┘└─────────────┘└─┬─┘└──┬───┬─────┘»
q_1: ┤ H ├──■───────────────────■────┼───────────────────┼─────┤ X ├──────»
     ├───┤                           │                   │     └─┬─┘      »
q_2: ┤ H ├───────────────────────────■───────────────────■───────■────────»
     ├───┤                                                                »
q_3: ┤ H ├────────────────────────────────────────────────────────────────»
     └───┘                                                                »
«                                                                »
«q_0: ───────────────────────────────────────────────────────────»
«     ┌─────────────┐┌───┐┌───┐┌─────────────┐┌───┐┌────────────┐»
«q_1: ┤ Rz(0.57618) ├┤ X ├┤ X ├┤ Rz(0.57618) ├┤ X ├┤ Rx(2.5219) ├»
«     └─────────────┘└─┬─┘└─┬─┘└─────────────┘└─┬─┘└──┬───┬─────┘»
«q_2: ─────────────────■────┼───────────────────┼─────┤ X ├──────»
«                           │                   │     └─┬─┘      »
«q_3: ──────────────────────■───────────────────■───────■────────»
«                                                                »
«
«q_0: ──────────────────────────────────
«
«q_1: ──────────────────────────────────
«     ┌─────────────┐┌───┐┌────────────┐
«q_2: ┤ Rz(0.57618) ├┤ X ├┤ Rx(2.5219) ├
«     └─────────────┘└─┬─┘├────────────┤
«q_3: ─────────────────■──┤ Rx(2.5219) ├
«                         └────────────┘
[Saved circuit figure to qaoa_best_circuit.png]
```