VQE and QAOA to solve MAXCUT

Valeran MAYTIE

1 Coding graphs

1.1 Cuts in graphs

We need to cut every links from a vertex in V_0 to a vertex in V_1

```
def costOfCut(g,s):
    r = 0
    l = len(s) - 1
    for n, e in g.items():
        for d in e:
            if s[l - d] != s[l - n]:
                r += 1
    return r
```

Listing 1: Code that computes the cost of a cut s in a graph g

1.2 Scalar product

```
def scalprod(g,d):
    r = 0
    for k, p in d.items():
        r += costOfCut(g, k) * p
    return -r
```

Listing 2: Code that computes the scalar product who whant to optimize

2 MAXCUT with VQE

```
def probDistVQE(a):
    q = QuantumRegister(4)
    c = ClassicalRegister(4)
    qc = QuantumCircuit(q,c)

    assert(len(a) % 12 == 0)

    for i in range(len(a) // 12):
        for qi in range(4):
            j = qi * 3 + i * 12
            qc.u(a[j], a[j + 1], a[j + 2], q[qi])
        for qi in range(4):
            qc.cnot(q[qi], q[(qi+1) % 4])
    qc.measure(q, c)

    backend = BasicAer.get_backend('qasm_simulator')
    job = execute(qc, backend, shots=1000)
    res = dict(job.result().get_counts(qc))

    for i in res:
        res[i] = res[i] / 1000
    return res
```

Listing 3: code that creates and simulates the VQE circuit

- What are the various proposed cuts (V_0, V_1) ?

cuts	probabilities
1001	0.699
0110	0.292
1010	0.003
0011	0.005
1100	0.001

- Change the graph and check that it does not work "just by chance" (for instance, use g_2, g_3, or your own).

  It also work for g_2 and g_3.

- Are all possible cuts there ?
- The problem is symmetric, in the sense that if '0110' is an answer, so is '1001'. Is this reflected in the resulting probabilities ?

  There's symmetry, but you don't get it every time

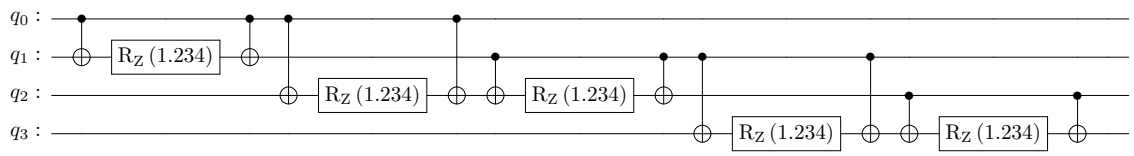
- If we had access to a real quantum co-processor, how would the code change ?

3 MAXCUT with QAOA

```
def V(qc,q,angle,g):
    for s in g:
        for e in g[s]:
            qc.cnot(q[s], q[e])
            qc.rz(angle, q[e])
            qc.cnot(q[s], q[e])
    return
```

Listing 4: code that add V gate in a quantum circuit

If we exectue V on an empty circuit with graph g_1 and angle = 1.234 we have this circuit :



```python
def probDistQAOA(a,g):
    p = int(len(a)/2)
    beta = a[:p]
    gamma = a[p:]

    q = QuantumRegister(4)
    c = ClassicalRegister(4)
    qc = QuantumCircuit(q,c)


    qc.h(q)


    for i in range(p):
        V(qc,q,gamma[i],g)
        for qb in q:
            qc.rx(beta[i], qb)
    qc.measure(q, c)

    backend = BasicAer.get_backend('qasm_simulator')
    job = execute(qc, backend, shots=1000)
    res = dict(job.result().get_counts(qc))

    for i in res:
        res[i] = res[i] / 1000
    return res
```

Listing 5: code that creates and simulates the QAOA circuit

- Does it still work with other graphs ?

  Yes, it still work.

- Does it find all of the possibilities ? How about the symmetry of the results ?

  The symmetry is much more common than VQE.

- Play with the length of the array a ($p = 2, 4, ...$ – make sure to keep it even). Do you see any loss/increase in precision ?

  Yes if we increase the length of the array a we can an improvement of the precision but the exectution is slower.

- Remark how the number of necessary parameters is way smaller than for VQE.

  The number of parameters is smaller for QAOA than VQE and better precision is obtained earlier for QAOA.