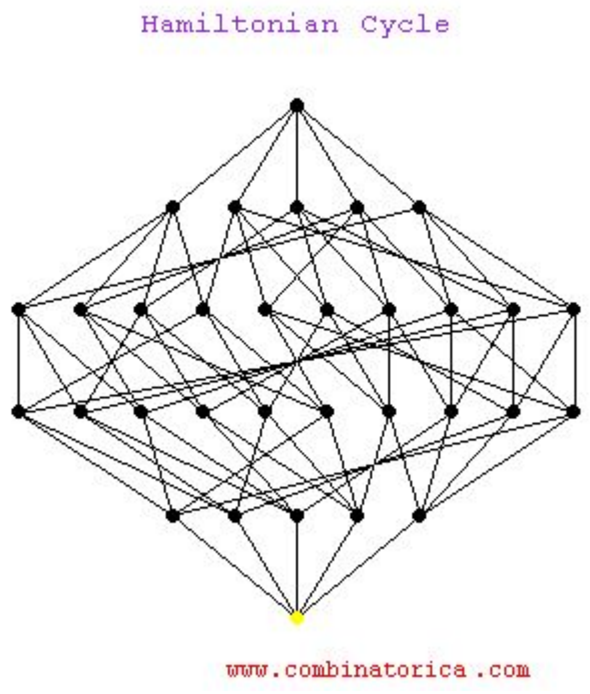
Finding an Efficient Solution for Hamiltonian Cycle #22

[Solvers]

What is Hamiltonian Cycle

- A cycle visits each node exactly once.
- Useful to generate Gray code.
- NP-complete problem.
- No worst-case efficient algorithm.
- Backtracking works only in small-sized graphs..



https://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/ham.html

Solution

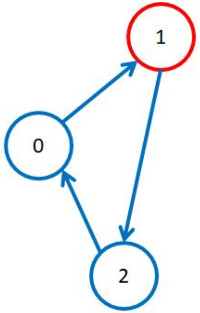
1. with VQE
2. with Grover’s algorithm

A. Mahasinghe et al., 2019

1. Solution with VQE

Define the problem and the objective function

$$x_{i,j} = \begin{cases} 1 & \text{if vertex } i \text{ is at position } j \text{ of the sequence} \\ 0 & \text{otherwise} \end{cases}$$



- Vertex = {0, 1, 2}
- Edge = {(0,1), (0,2), (1,2)}

Right result

$$\mathbf{x} = [0, 0, 1, \quad 1, 0, 0, \quad 0, 1, 0]$$

Wrong result

$$\mathbf{x} = [0, 1, 1, \quad 1, 0, 0, \quad 0, 1, 0]$$

$$\mathbf{x} = [0, 0, 0, \quad 1, 0, 0, \quad 0, 1, 0]$$

$$\mathbf{x} = [1, 0, 0, \quad 1, 0, 0, \quad 0, 1, 0]$$

Objective function

$$F(\mathbf{x}) = H(\mathbf{x}) + P_1(\mathbf{x}) + P_2(\mathbf{x})$$

where,

$$H(\mathbf{x}) = \sum_{(i_1, i_2) \in V \times V - E(G)} \left(x_{i_1,0} x_{i_2,n-1} + \sum_{j=0}^{n-2} x_{i_1,j}\, x_{i_2,j+1} \right),$$

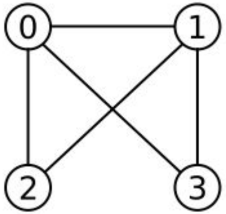
$$P_1(\mathbf{x}) = \sum_{i=0}^{n-1} \left(1 - \sum_{j=0}^{n-1} x_{i,j} \right)^2$$

and

$$P_2(\mathbf{x}) = \sum_{j=0}^{n-1} \left(1 - \sum_{i=0}^{n-1} x_{i,j} \right)^2.$$

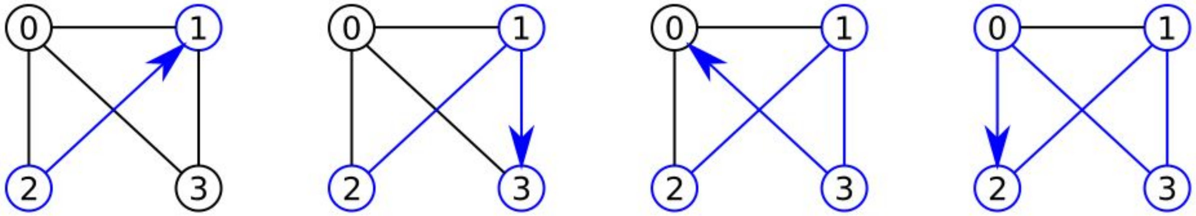
1. Solution with VQE

Problem



Optimal function value : 0.0 #This graph is truly a Hamiltonian Cycle

Solution



1. Solution with VQE

```
Vertex = {0, 1, 2, 3}
Edge   = {(0,1), (1,2), (0,3), (1,3), (0,2)}

n      = len(Vertex)
```

Achievement – Run with exact solver

```
# solving Quadratic Program using exact classical eigensolver
exact = MinimumEigenOptimizer(NumPyMinimumEigensolver())
result = exact.solve(qp)
print(result)
```

```
optimal function value: 0.0
optimal value: [0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0.]
status: SUCCESS
```

1. Solution with VQE

Achievement – Run with qasm simulator

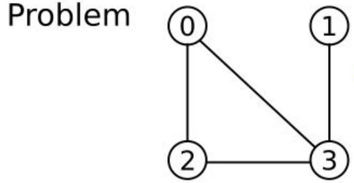
```
# provider = IBMQ.load_account()
aqua_globals.random_seed = np.random.default_rng(123)
seed = 10598
# simulator_backend = provider.get_backend('ibmq_qasm_simulator')
backend = Aer.get_backend('qasm_simulator')
quantum_instance = QuantumInstance(backend, seed_simulator=seed, seed_transpiler=seed)
```

```
counts = []
values = []
def store_intermediate_result(eval_count, parameters, mean, std):
    counts.append(eval_count)
    values.append(mean)

# construct VQE
spsa = SPSA(maxiter=1000)
ry = TwoLocal(qubitOp.num_qubits, 'ry', 'cz', reps=2, entanglement='linear')
vqe = VQE(qubitOp, ry, spsa, quantum_instance=quantum_instance, callback=store_intermediate_result)

# run VQE
result = vqe.run(quantum_instance)

# print results
x = sample_most_likely(result.eigenstate)
print('energy:', result.eigenvalue.real)
print('time:', result.optimizer_time)
print('hamiltonian-cycle objective:', result.eigenvalue.real + offset)
print('solution:', x)
```

```
energy: -422949.21875
time: 4681.928060054779
hamiltonian-cycle objective: 17050.78125
solution: [0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0]
```

1. Solution with VQE

With Exact solver, we can check whether the graph is feasible or not.



This graph is not a Hamiltonian Cycle

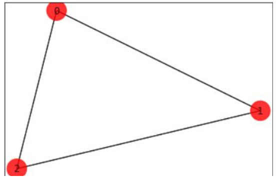
```
# solving Quadratic Program using exact classical eigensolver
exact = MinimumEigenOptimizer(NumPyMinimumEigensolver())
result = exact.solve(qp)
print(result)
```

```
optimal function value: 1.0
optimal value: [0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0.]
status: SUCCESS
```

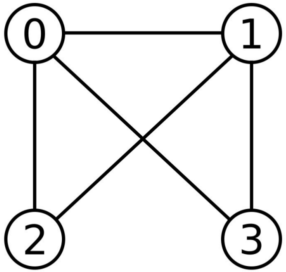
1. Solution with VQE

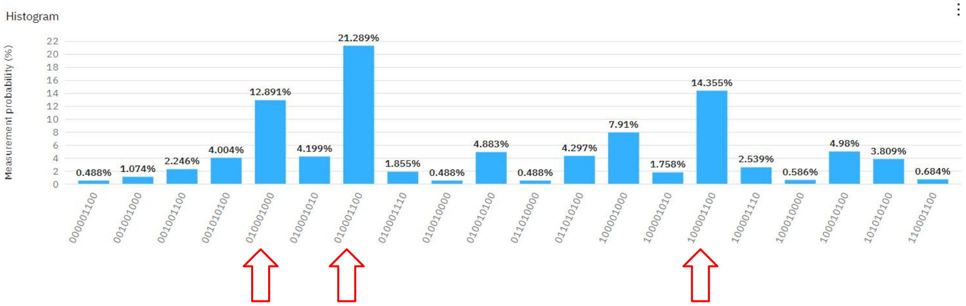
Achievement - Run with QASM Simulator

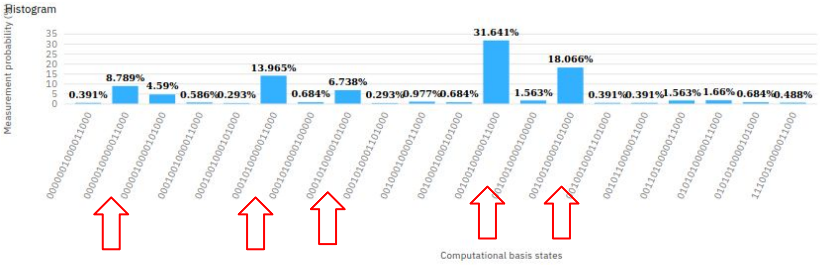
1. Triangle



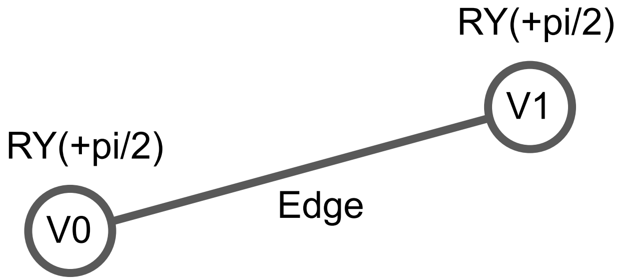
2. Rectangle



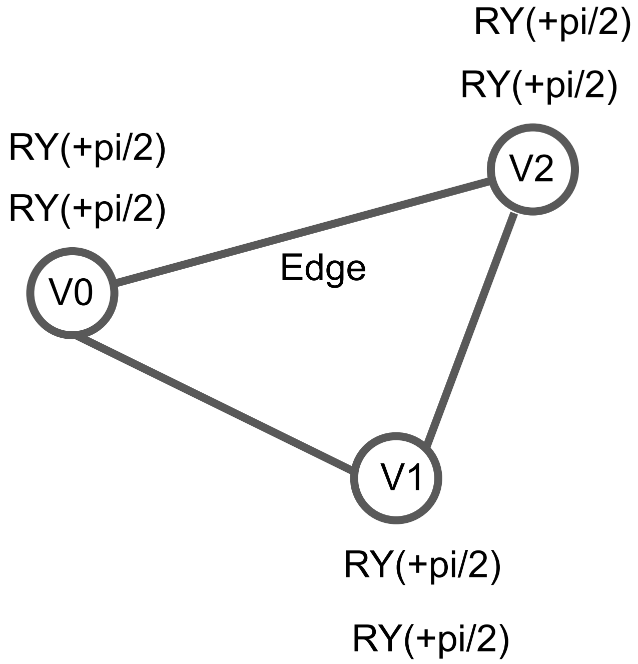




2. Solution with Grover’s algorithm



When we choose an edge,
the qubits representing connected vertices
with the edge will rotate by +pi/2

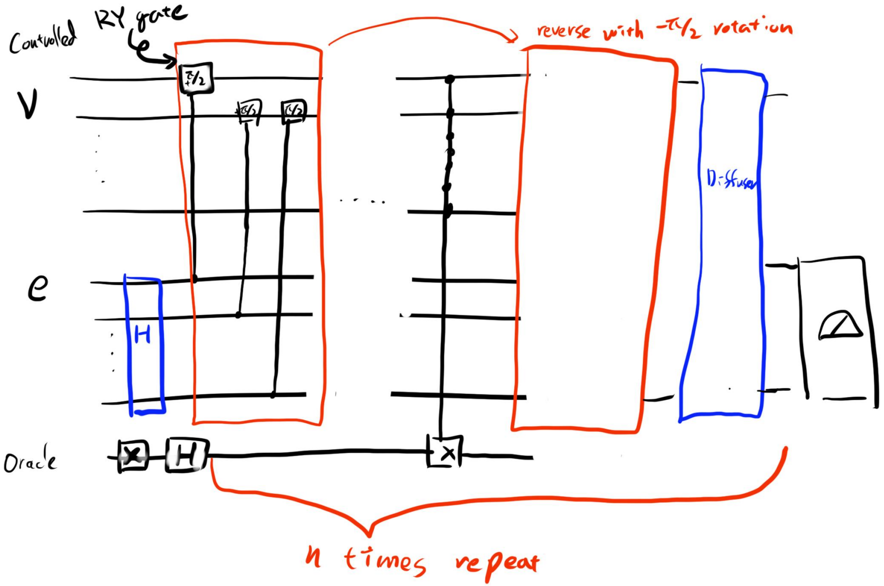


If the chosen path is the Hamiltonian cycle,
the qubits representing vertices will be |1>.

2. Solution with Grover's algorithm

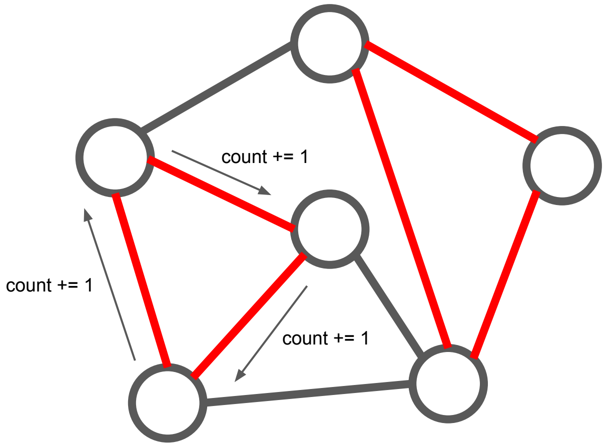
Required qubits = n(V) + n(E) + 1

iteration = sqrt(n(E))*pi/4



2. Solution with Grover's algorithm

It needs post-process to filter out wrong answers (sub-cycles).

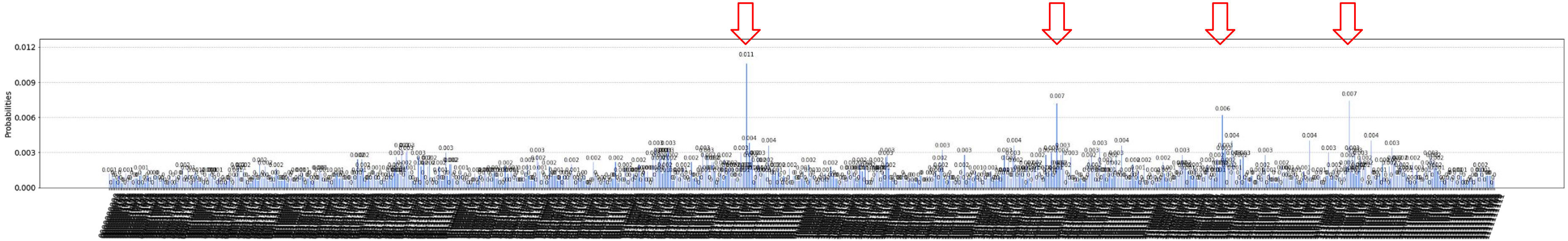


We filtered out them in a way of classical computing

```
if Count == the number of vertices:
    return True
```

2. Solution with Grover's algorithm

Result from QASM Simulator



After Postprocessing

We successfully found correct cycles.

```
In [9]: postprocess(count, g_dict, n_vertices)

Out[9]: ['0111010101', '1010111001', '1100110101']
```