Quantinuum Challenge: InsertPresentationName

InsertTeamName

Grace J. Grace T. Jake M. Misheel O. Theo L.

“AAAAAAAAAA”

—InsertSomeoneFamous

01

Optimization

Improving Quantum Approximate Optimization Algorithm

Approaches

1. Understanding physical problem

   → generally successful

2. Proof of concept by FDSA or gradient descent

   → successful, got higher result

3. Used gradient descent and SPSA Optimization

   → generally successful, still improving understanding

4. Wrote our own SPSA implementation

   → successful, got substantially higher results

Quantum Approximate Optimization Algorithm (QAOA)

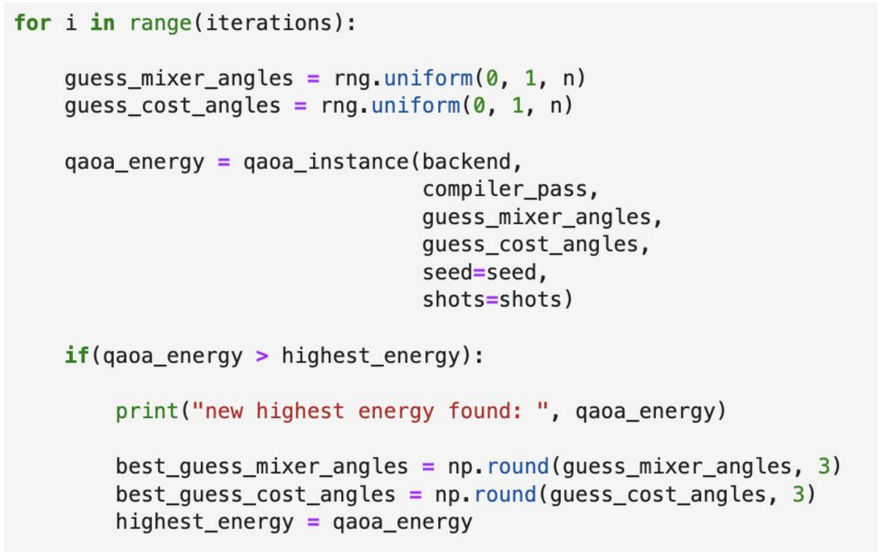
- Several layers of alternating cost and mixer Hamiltonians
- Layers are determined by the parameters

  γ_1, γ_2, … $\gamma_{\square}$, β_1, β_2, … $\beta_{\square}$, where 2p is the # of layers
- Certain combinations of parameters give good expected energy
- Maximize

Farhi, Edward, et al. "A Quantum Approximate Optimization Algorithm." ArXiv.org, 14 Nov. 2014, https://arxiv.org/abs/1411.4028.

Original Quantinuum Code

- Random number guess for angles from uniform distribution between 0 and 1
- Keeps choice of angles if it beats all previous choices



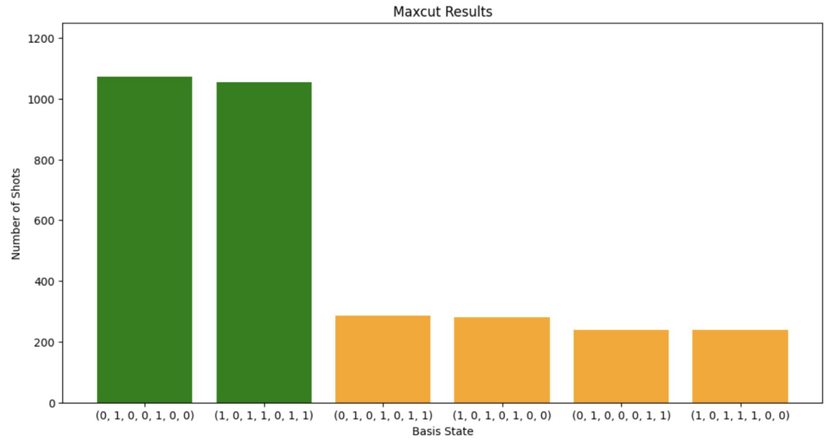
```python
for i in range(iterations):

    guess_mixer_angles = rng.uniform(0, 1, n)
    guess_cost_angles = rng.uniform(0, 1, n)

    qaoa_energy = qaoa_instance(backend,
                                compiler_pass,
                                guess_mixer_angles,
                                guess_cost_angles,
                                seed=seed,
                                shots=shots)

    if(qaoa_energy > highest_energy):

        print("new highest energy found: ", qaoa_energy)

        best_guess_mixer_angles = np.round(guess_mixer_angles, 3)
        best_guess_cost_angles = np.round(guess_cost_angles, 3)
        highest_energy = qaoa_energy
```

Graphs

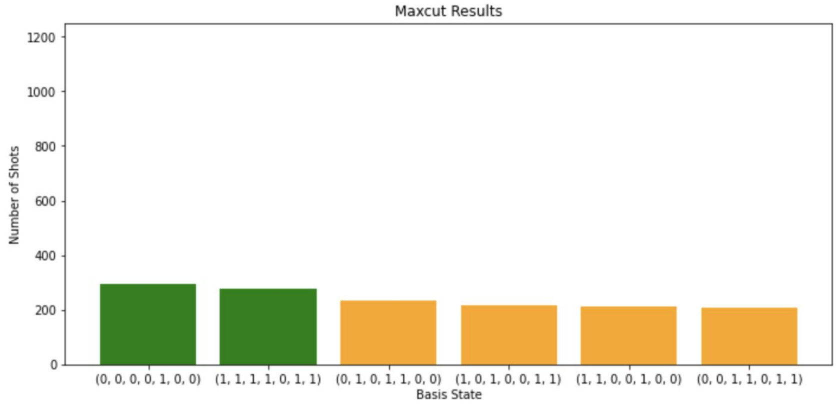
Original graph

A weighted graph

Success ratio 0.4252



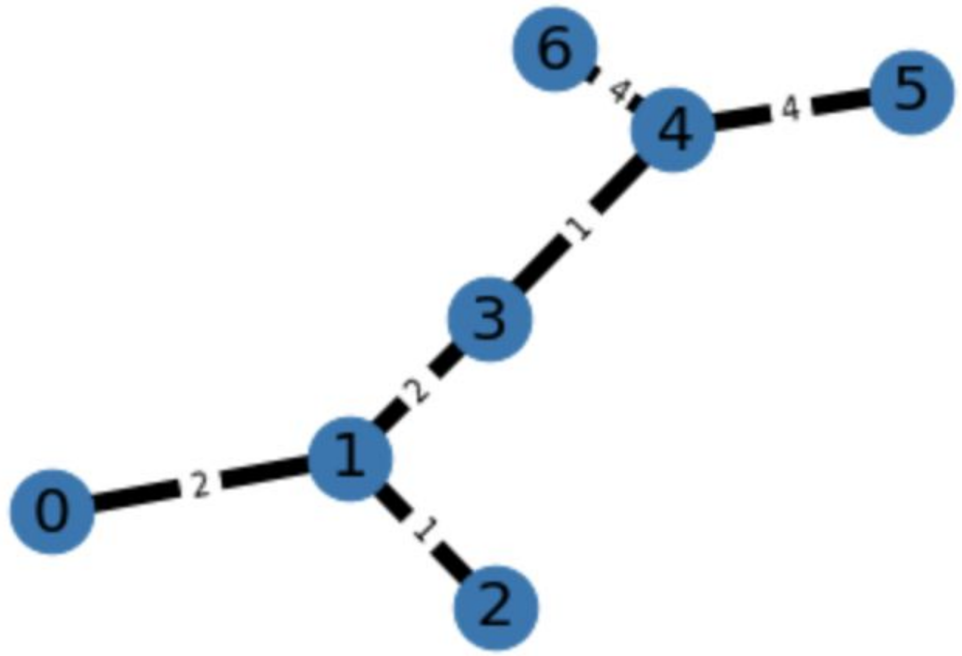
Success ratio 0.1142



(Kind of wrong)

Highest Expected Energy: 4.94

Highest Expected Energy: 9.4


0
1
2
3
4
5
6
2
1
2
1
4
4

Gradient Descent

- Estimate the gradient of the objective function by partial derivative in each variable
- Make jumps proportional to the estimated gradient

Pro:

- Gets better results than random guesses

Cons:

- Complexity increases with number of variables

The Math

For Objective function J(u), find

$$u^* = \arg\min_{u \in U} J(u).$$

Iterate by moving with gradient,

$$u_{n+1} = u_n - a_n \hat{g}_n(u_n),$$

Estimate gradient by,

$$(\hat{g_n}(u_n))_i = \frac{J(u_n + c_n e_i) - J(u_n - c_n e_i)}{2c_n}$$

“Simultaneous Perturbation Stochastic Approximation.” Wikipedia, Wikimedia Foundation, 3 Jan. 2023, https://en.wikipedia.org/wiki/Simultaneous_perturbation_stochastic_approximation.

Simultaneous Perturbation Stochastic Approximation (SPSA)

- Instead of testing each variable direction, tests a random direction twice.
- Estimates gradient
- Provable that gradient error approaches 0 over iterations

Pro:

- Complexity stays the same for more layers
- Usually does better than previous approaches
- Easy to implement a simple version

Cons:

- The pre-existing implementation was confusing and unpredictable

SPSA Math

We estimate the gradient by,

$$(\hat{g_n}(u_n))_i = \frac{J(u_n + c_n\Delta_n) - J(u_n - c_n\Delta_n)}{2c_n(\Delta_n)_i}$$

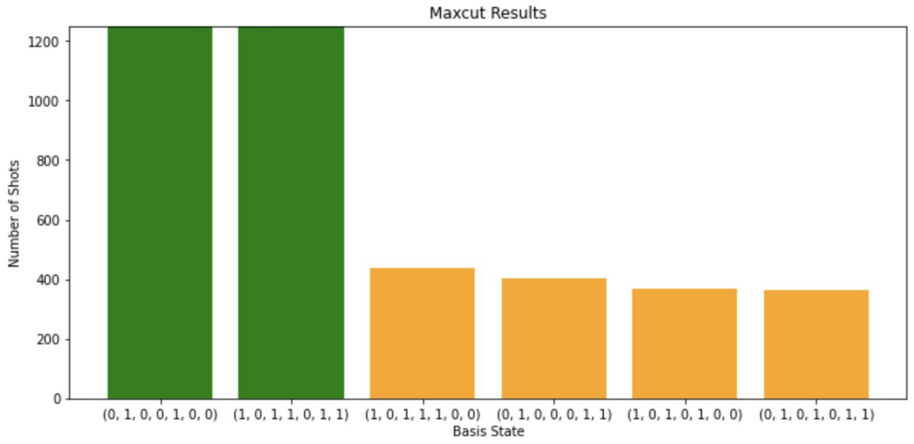
with the Δ□ drawn from a Rademacher distribution:
uniform discrete from {-1, +1}

Graphs

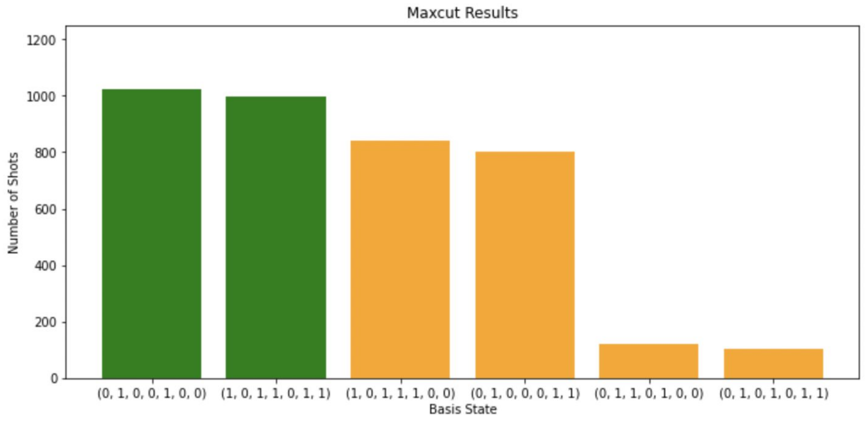
Original graph

A weighted graph

Success ratio 0.5962



Success ratio 0.4044



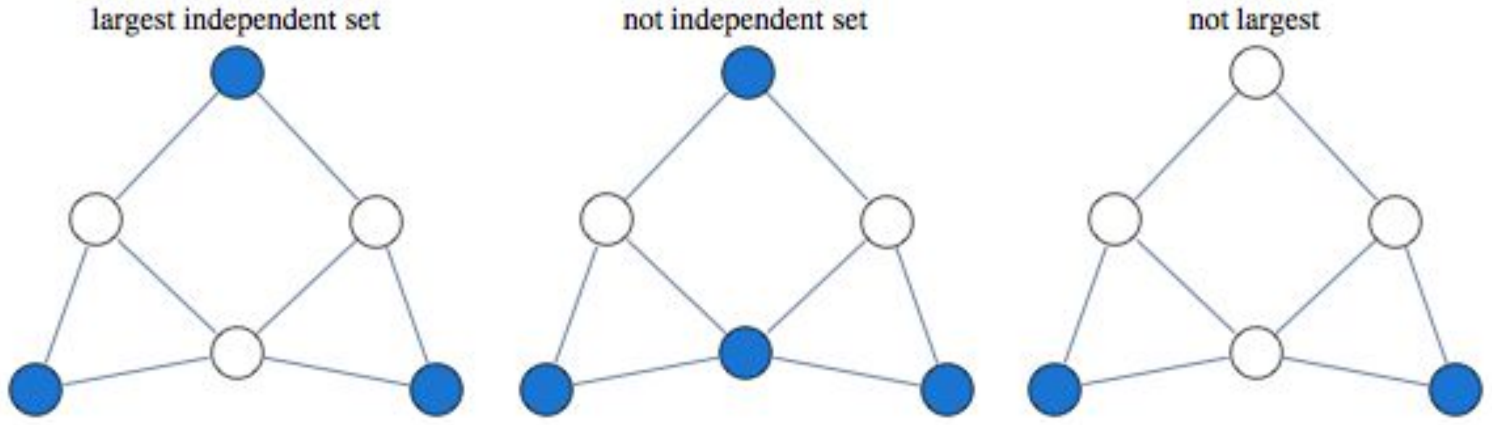
Highest Expected Energy: 5.46

Highest Expected Energy: 12.77

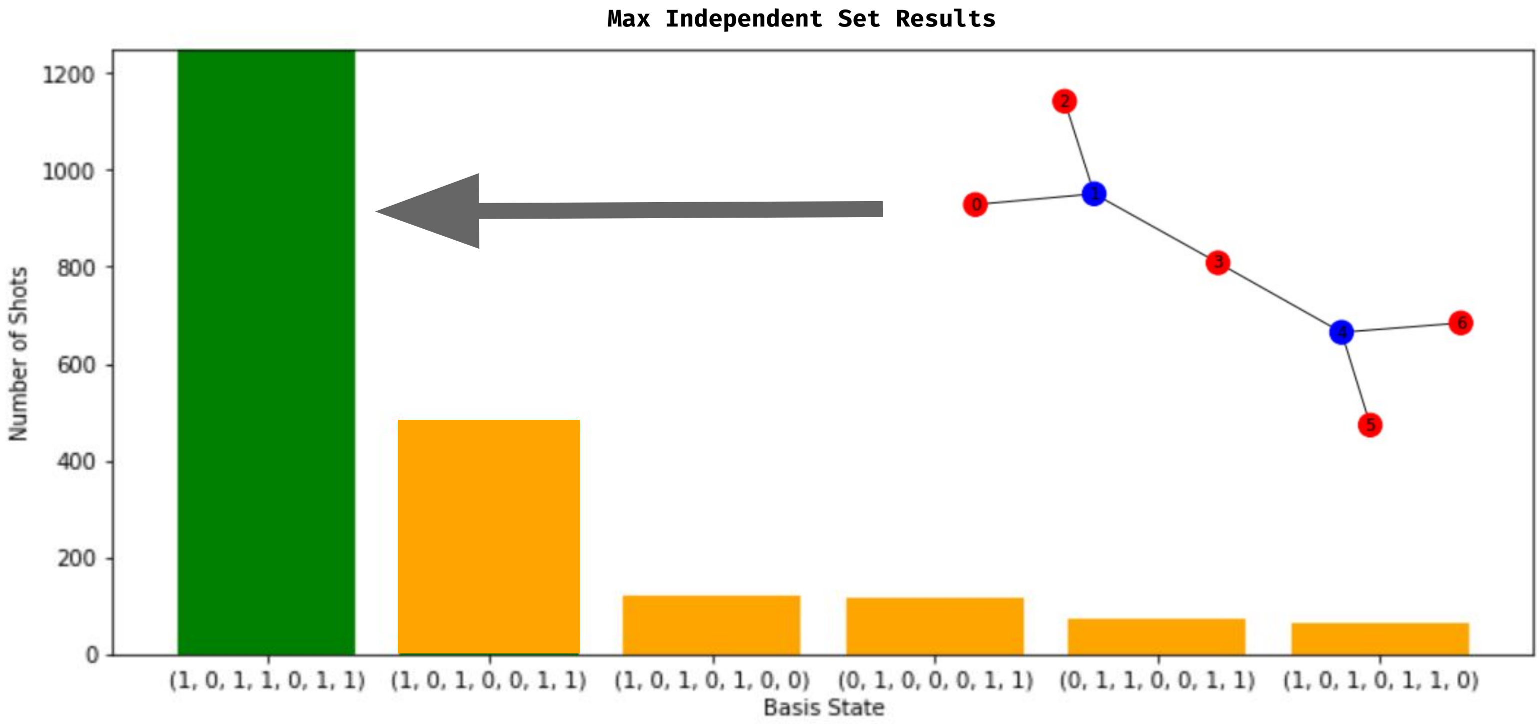
02

Further Applications

Max Independent Set



Results

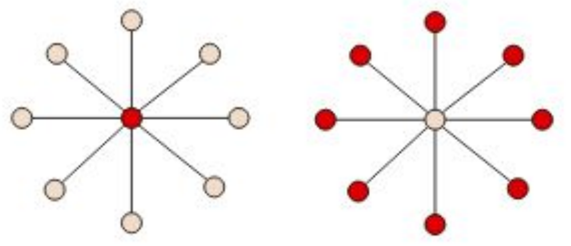


Max Independent Set Results
Number of Shots
1200
1000
800
600
400
200
0
(1, 0, 1, 1, 0, 1, 1)
(1, 0, 1, 0, 0, 1, 1)
(1, 0, 1, 0, 1, 0, 0)
(0, 1, 0, 0, 0, 1, 1)
(0, 1, 1, 0, 0, 1, 1)
(1, 0, 1, 0, 1, 1, 0)
Basis State
0
1
2
3
4
5
6

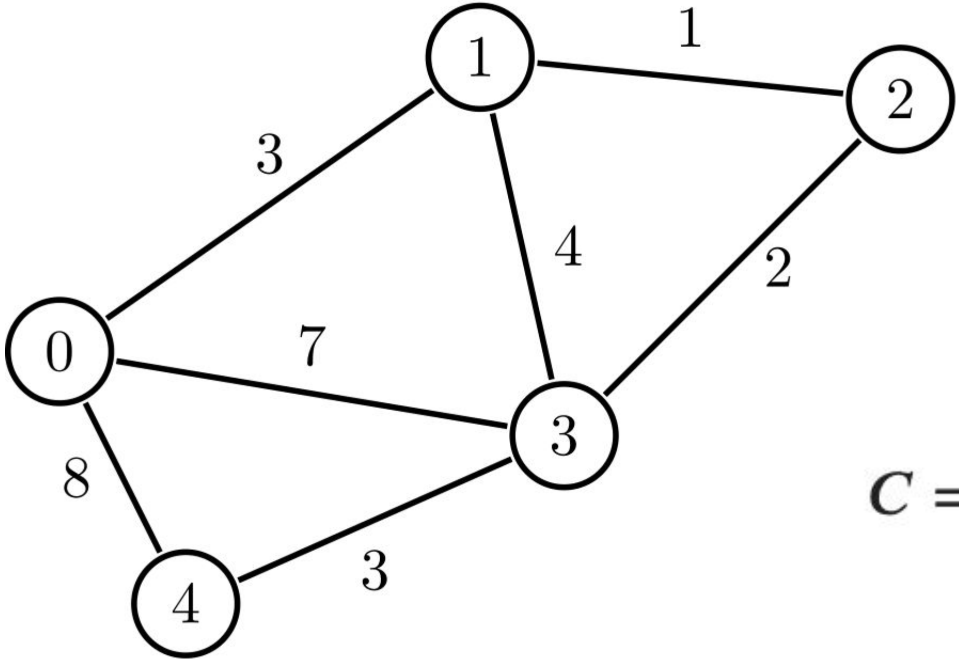
Real-World Applications

Max Independent Set

- Corresponds to maximum clique problem on complement graph
- Corresponds to minimum vertex cover problem



Max Cut for Weighted Graphs

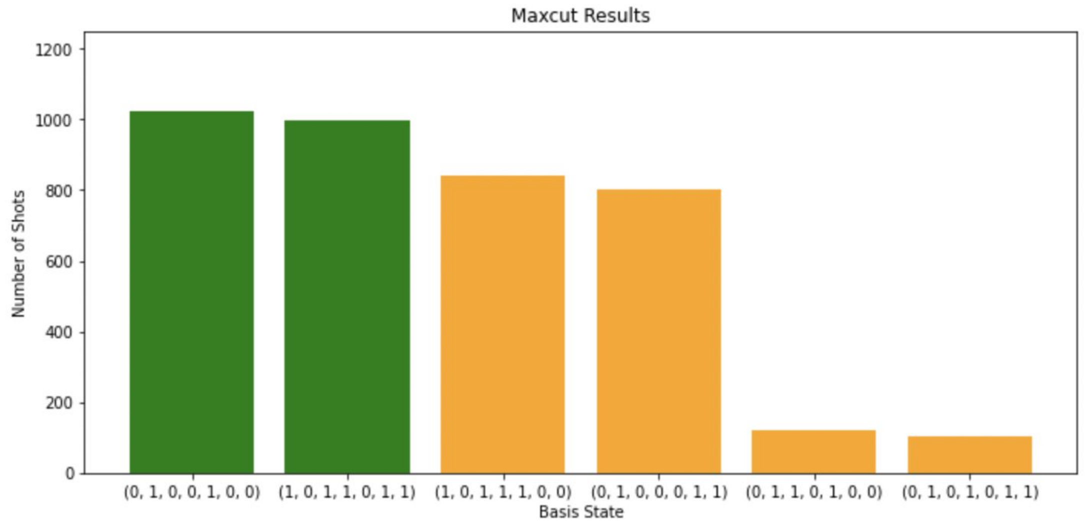


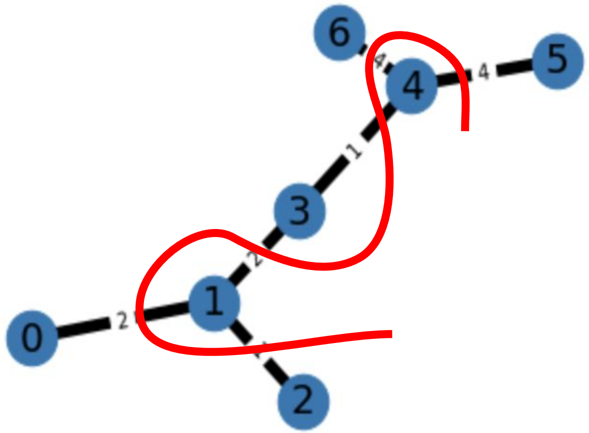
- Graph with edges that have weight
- Cut across an edge is now worth the edge's weight, not a default 1

$$C = \sum_{(i,j)} x_i(1 - x_j) \cdot w(x_i, x_j)$$

Results

```
Success ratio 0.4044
```





Real-World Applications

Max Cut for Weighted Graphs

- Grouping friends together, grouping those who don’t know each other well together
- Meal planning from foods in fridge (maximize nutrition)

	Seal	Monk	Liane
Seal	1	0.4	0.7
Monk	0.4	1	0.9
Liane	0.7	0.9	1