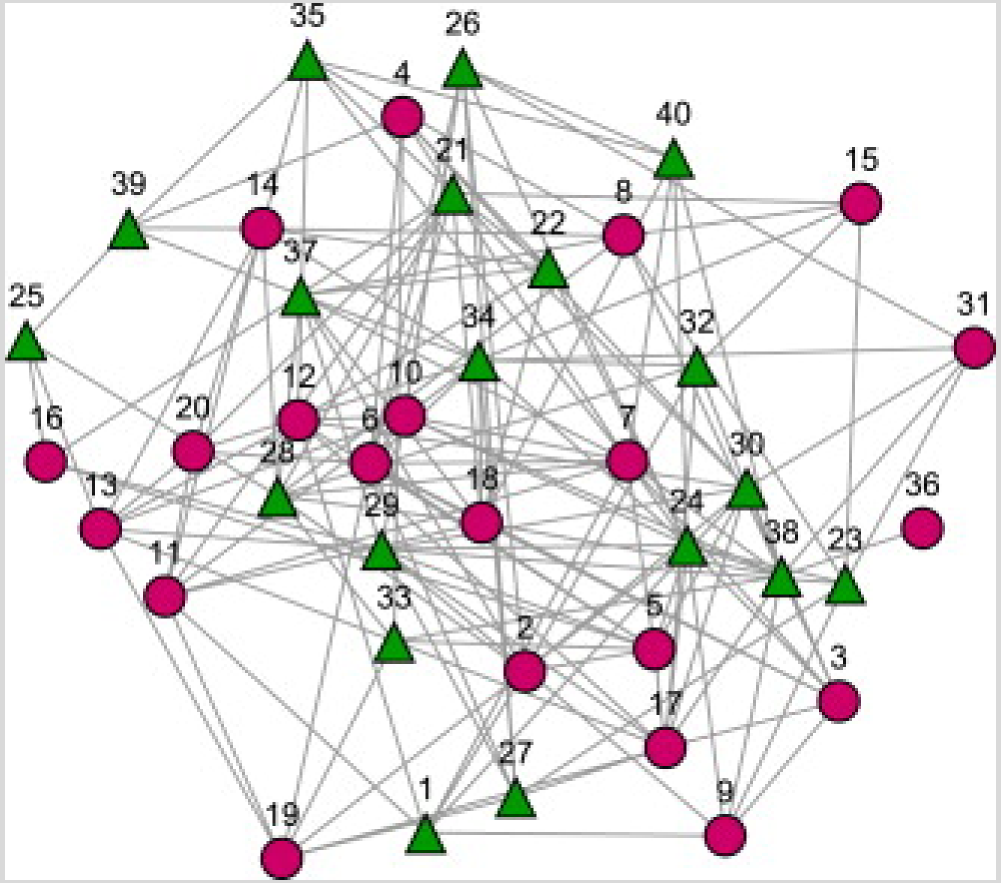


Wang, Wang, *JCAM*, May 2010.

QAOA

Combinatorial Optimization

ECE 592/CSC 591 – Fall 2019

Variational Quantum Algorithms

Based on **variational method** of quantum mechanics

- finding approximations to ground state - lowest energy eigenstate

General idea

  Prepare trial state based on parameter(s)

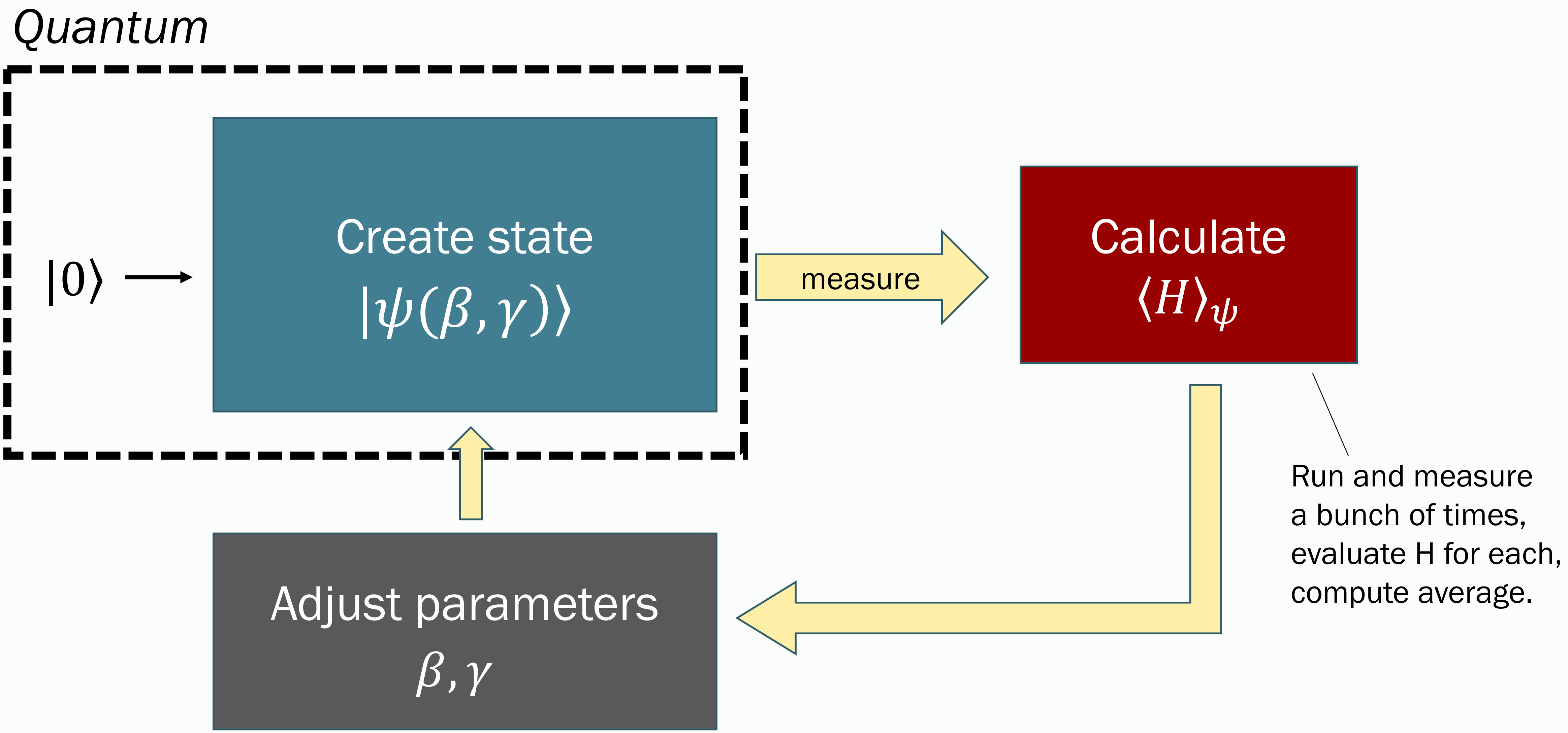
  Evaluate expected value of observable (Hamiltonian) for that state

    Minimum expected value = lowest eigenvalue

    (Our examples will actually try to maximize the expected value.)

  Adjust parameter(s) -- usually with classical optimizer

  Repeat to find minimum (or maximum)



Quantum
$|0\rangle$
Create state
$|\psi(\beta,\gamma)\rangle$
measure
Calculate
$\langle H\rangle_\psi$
Run and measure
a bunch of times,
evaluate H for each,
compute average.
Adjust parameters
β,γ

QAOA and VQE

Quantum Approximate Optimization Algorithm (QAOA)

State = alternating circuits (operators).
based on **cost** function and **mixing** function

$$|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle = U(B, \beta_p)\, U(C, \gamma_p) \cdots U(B, \beta_1)\, U(C, \gamma_1)\ |s\rangle$$

Depth (p): number of alternating operators

Farhi, Golstone, Gutman, arXiv:1411.4028v1, 2014.

Variational Quantum Eigensolver (VQE)

Based on fixed “variational forms”

Peruzzo, et al. *Nature Comm*, 2014.
https://community.qiskit.org/textbook/ch-applications/vqe-molecules.html

QAOA

Objective Function

$$C(z) = \sum_{\alpha=1}^{m} C_\alpha(z)$$

z is an n-bit string.

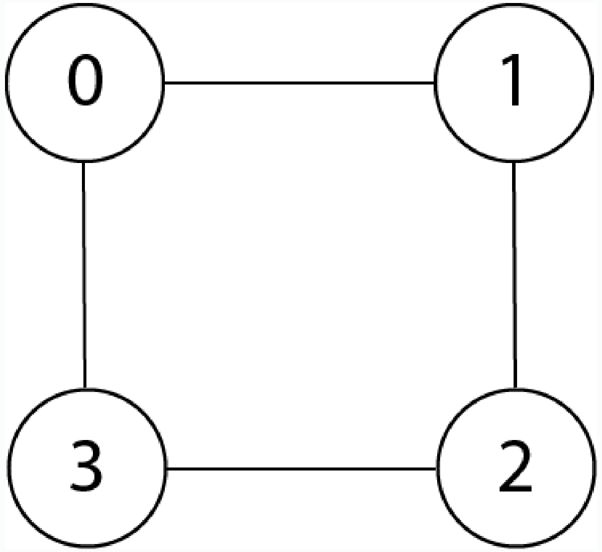
There are m clauses in the objective.
Each clause typically involves only a few bits.

C_a(z) is 1 if z satisfies the clause.

Satisfiability: Is there a string (z) that satisfies all clauses?
MaxSat: Which string (z) satisfies the most clauses?
Approximate Optimization: Find a string (z) for which C(z) is close to the maximum.

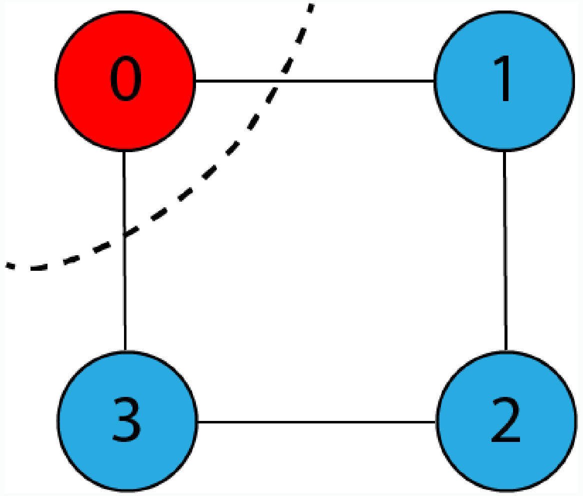
Example: MaxCut

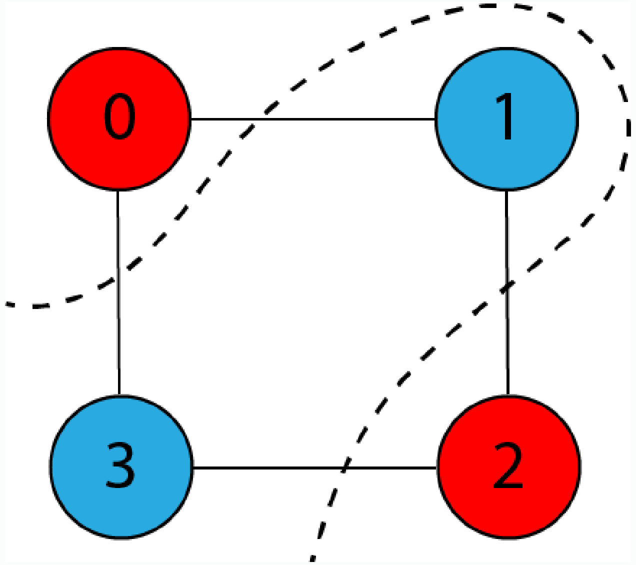
Divide a graph into two partitions, such that the number of edges that connect between partitions is maximized.



$$C = \sum_{\langle jk \rangle} C_{\langle jk \rangle}$$

$$C_{\langle jk \rangle} = \frac{1}{2}\left(-\sigma_j^z \sigma_k^z + 1\right)$$





Define a Unitary Operator

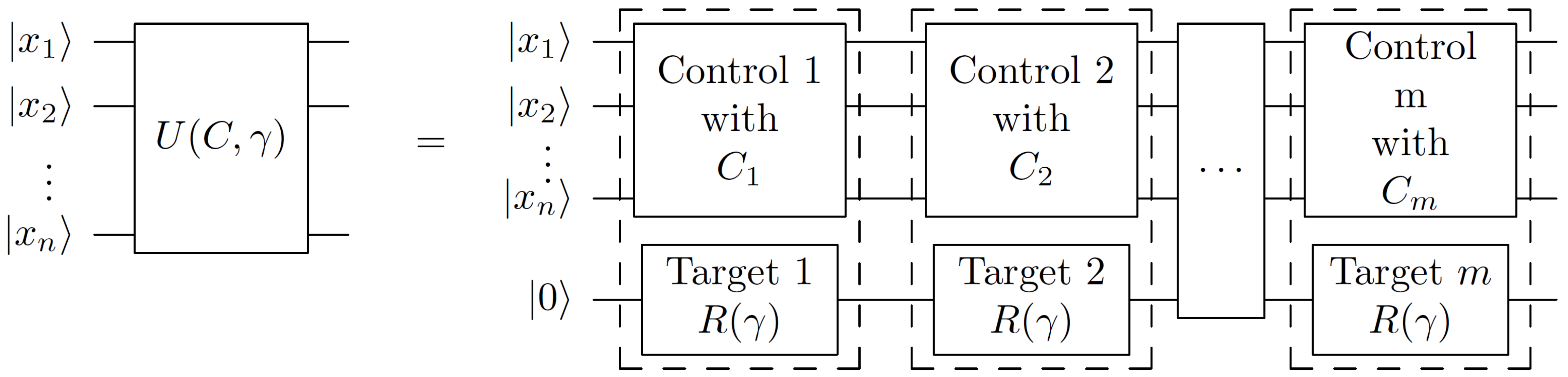
$$U(C,\gamma) = e^{-i\gamma C} = \prod_{\alpha=1}^{m} e^{-i\gamma C_\alpha}$$

For a given state z, each clause that is satisfied applies a **phase** rotation of γ. So state(s) that maximize objective function will be rotated most.

Range of γ = 0 to 2π

$$\boxed{e^{i\gamma \boldsymbol{A}} = \cos(\gamma)\boldsymbol{I} + i\sin(\gamma)\boldsymbol{A}}$$

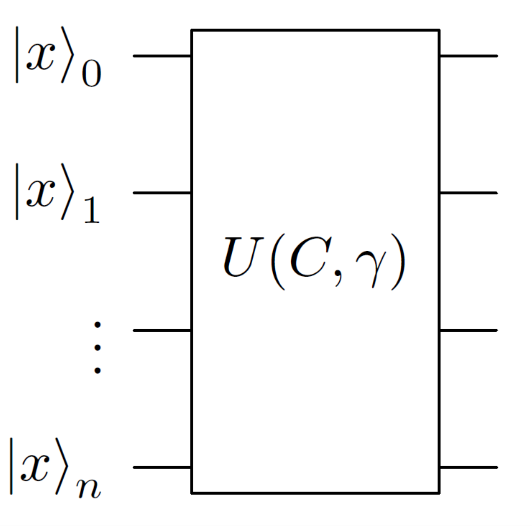
Implementation

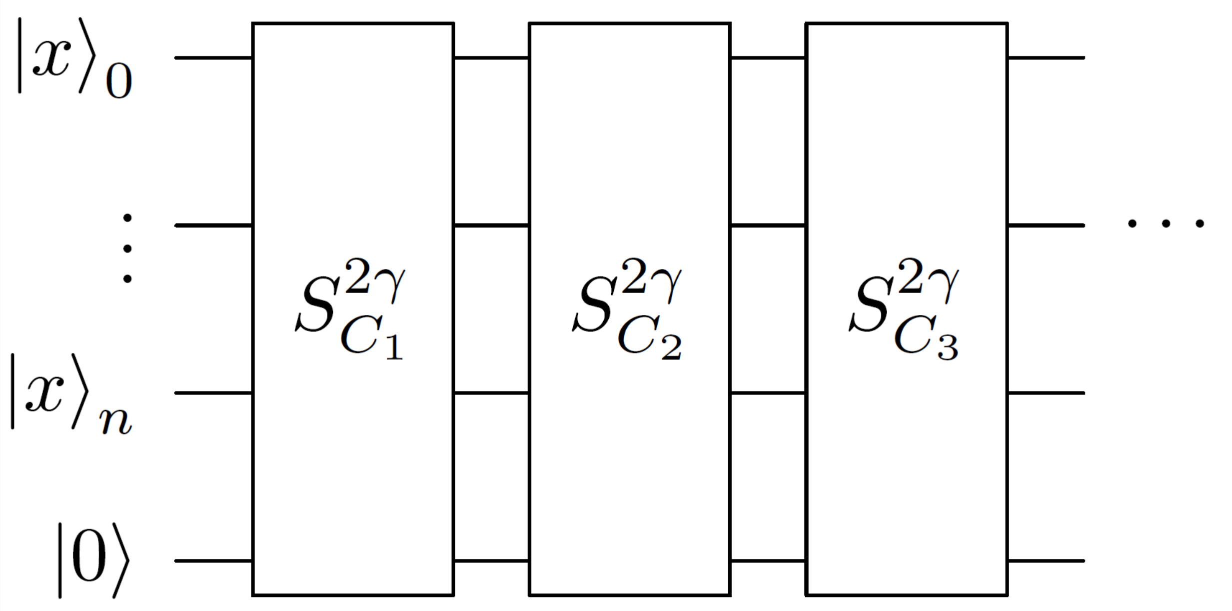


[Quirk](#)

https://www.cs.umd.edu/class/fall2018/cmsc657/projects/group_16.pdf

Implementation





[Quirk]

Selective phase change

So far…

What if we apply $U(C,\gamma)$ to input $|s\rangle$?

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_z |z\rangle$$

Rotating phase does not change amplitude,
so measuring will not give any advantage.

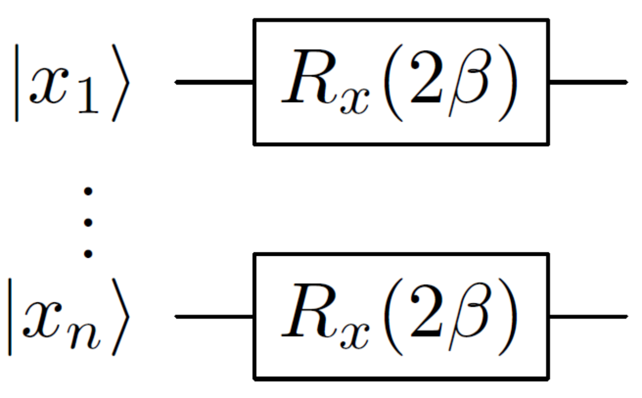
Each z has a probability of ${}^1/_{\sqrt{2^n}}$ so it's no better than guessing.

Need to convert phase into amplitude…

Define a Mixing Operator

$$B = \sum_{j=1}^{n} \sigma_j^x$$

$$U(B, \beta) = e^{-i\beta B} = \prod_{j=1}^{n} e^{-i\beta\sigma_j^x}$$



Quirk

Create Trial State

Choose 2p angles and…

Circuit depth is *mp* + *p*

$$|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle = U(B, \beta_p)\, U(C, \gamma_p) \cdots U(B, \beta_1)\, U(C, \gamma_1)\ |s\rangle$$

Get expected value: $F_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \langle \boldsymbol{\gamma}, \boldsymbol{\beta}|\, C\, |\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$

$$M_p = \max_{\boldsymbol{\gamma}, \boldsymbol{\beta}} F_p(\boldsymbol{\gamma}, \boldsymbol{\beta})$$

$$M_p \geq M_{p-1}$$

Maximization at p-1 can be considered constrained maximization at p.

$$\lim_{p \to \infty} M_p = \max_z C(z)$$

Challenge: Finding Good Angles

Classical preprocessing (original paper)

With p = 1, approximation ratio is 0.6924. (Choosing random partition = 2/3)

Create grid over $\gamma = [0,2\pi]$ and $\beta = [0,\pi]$ and explore.

Gradient descent -- can get stuck at local minima/maxima, expensive in terms of evaluations required. (not recommended)

For noisy:

SPSA = Simultaneous Perturbation Stochastic Approximation

For simulator:

From IBM textbook

SLSP = Sequential Least Squares Programming

COBYLA = Constrained Optimization by Linear Approximation

Discussion

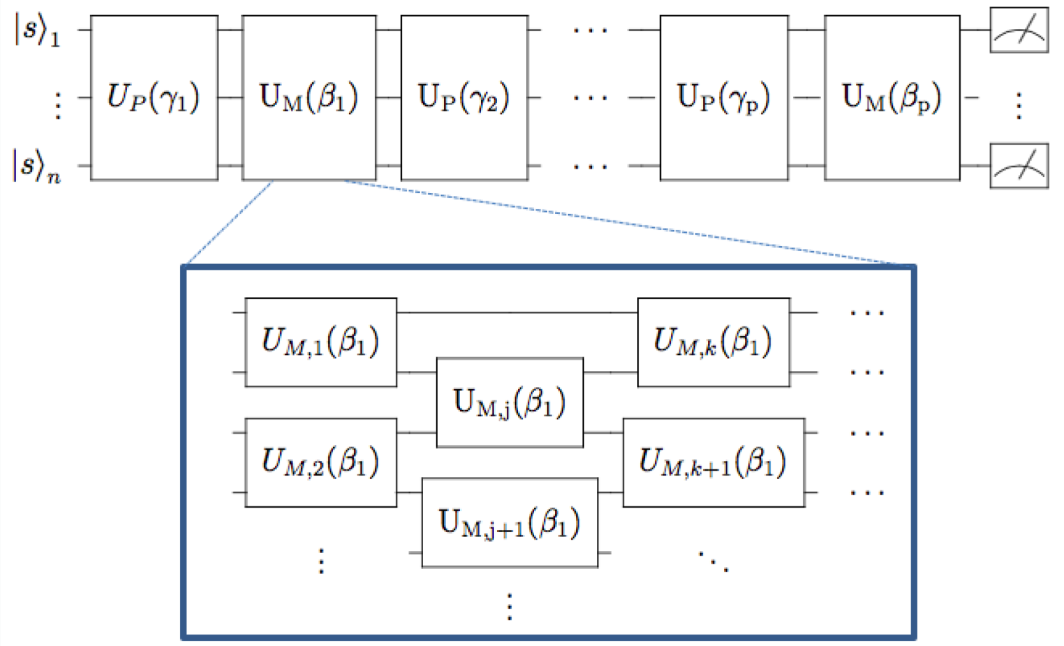
QAOA has not been demonstrated to perform better than classical.

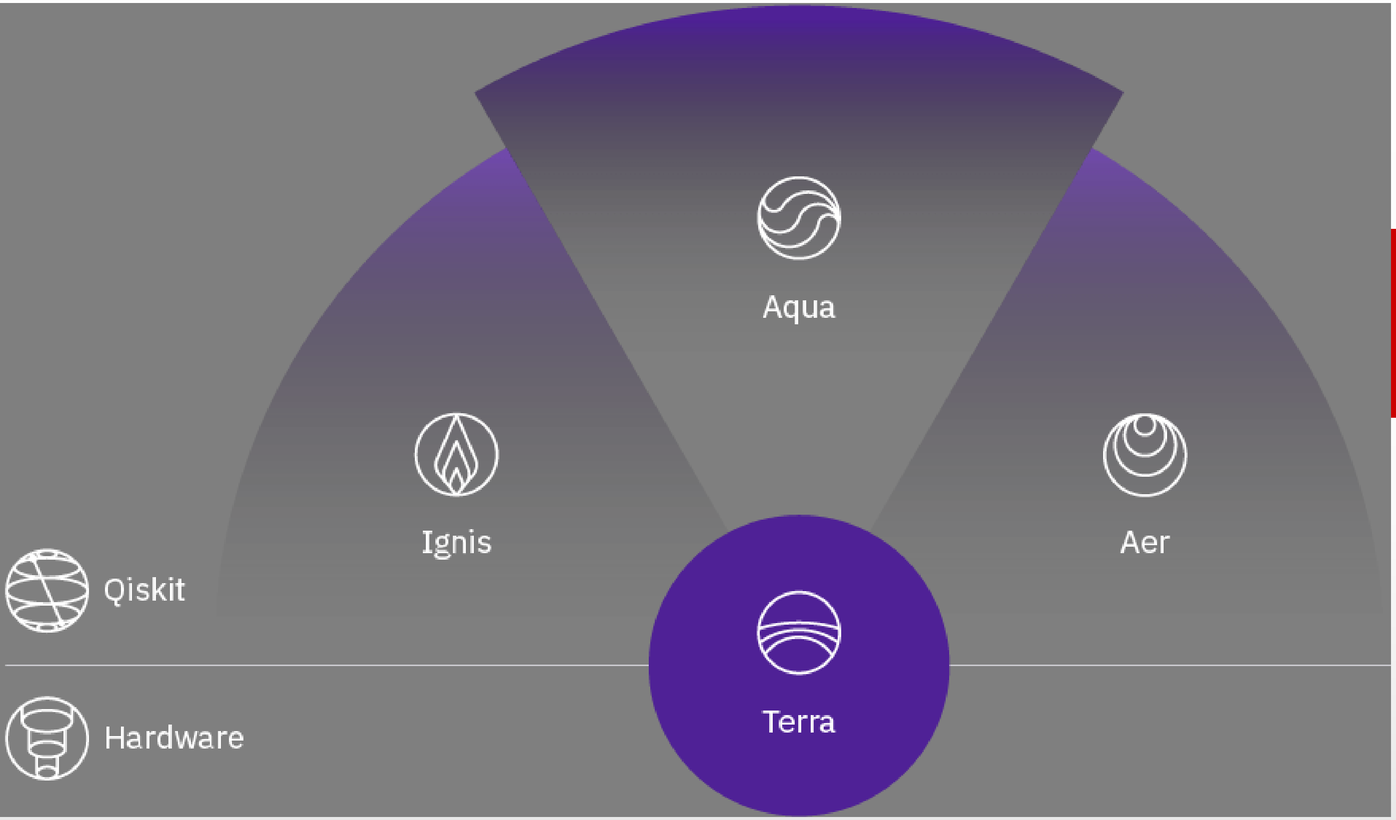
Better approximation? Faster?

Can apply different mixing operators

Hadfield, et al., arXiv:1709.03489
Quantum Alternating Operator Ansatz (QAOA)

Dealing with more general
cost functions and constraints





Qiskit Aqua

Aqua Components

Pluggable algorithm classes

- Ising model operators
- QAOA, VQE, QGAN, ...

Circuits

- Boolean logic functions, QFT, phase estimation...

Optimizers

- COBYLA, SPSA, ...

Abstractions above circuits and gates

API Documentation

https://qiskit.org/documentation/index.html

class QAOA(…)¶

Parameters

operator (BaseOperator) – Qubit operator

operator_mode (str) – operator mode, used for eval of operator

p (int) – the integer parameter p as specified in https://arxiv.org/abs/1411.4028

initial_state (InitialState) – the initial state to prepend the QAOA circuit with

mixer (BaseOperator) – the mixer Hamiltonian to evolve with.

optimizer (Optimizer) – the classical optimization algorithm.

Cost Operator

```
num_nodes = weight_matrix.shape[0]
pauli_list = []
shift = 0
for i in range(num_nodes):
    for j in range(i):
        if weight_matrix[i, j] != 0:
            x_p = np.zeros(num_nodes, dtype=np.bool)
            z_p = np.zeros(num_nodes, dtype=np.bool)
            z_p[i] = True
            z_p[j] = True
            pauli_list.append([0.5 * weight_matrix[i, j],
                                    Pauli(z_p, x_p)])
            shift -= 0.5 * weight_matrix[i, j]
return WeightedPauliOperator(paulis=pauli_list), shift
```

qiskit/optimization/ising/max_cut.py

Using Operator to Build Circuit

```
circuit.u2(0, np.pi, q)
for idx in range(self._p):
    beta, gamma = angles[idx], angles[idx + self._p]
    circuit += self._cost_operator.evolve(
        evo_time=gamma, num_time_slices=1, quantum_registers=q
    )
    circuit += self._mixer_operator.evolve(
        evo_time=beta, num_time_slices=1, quantum_registers=q
    )
```

qiskit/aqua/algorithms/adaptive/qaoa/var_forms.py

Useful Tutorials

qiskit-iqx-tutorials/qiskit/advanced/aqua/optimization/

max_cut_and_tsp.ipynb

vehicle_routing.ipynb