Unit 05: Programming through quantum clouds

Tzu-Chieh Wei
C. N. Yang Institute for Theoretical Physics and Department of Physics and Astronomy, State University of New York at Stony Brook, Stony Brook, NY 11794-3840, USA
(Dated: September 27, 2023)

In this unit, we discuss quantum programming on IBM's quantum computers via the software framework: Qiskit. We will learning the variational quantum eigensolver (VQE), which can be applied to various topics: quantum chemistry of molecules, quantum approximate optimization algorithm (QAOA) for optimization, & hybrid classical-quantum neural network.

Learning outcomes: You'll be able to modify example Python Notebooks to run variational quantum eigensolvers for applications.

I. INTRODUCTION

This unit will bring us to the very frontier how near-term (usually referred to as NISQ [1]) quantum computers can be useful. We will learn about complexity classes related to problems that quantum computers can solve and variational methods for ground states and their enegies, optimization problems, quantum chemistry and even quantum neural networks.

II. COMPLEXITY

We will review the idea of Turing machine and a few complexity classes. Let us recap some of the definitions below.
Turing machine. It is a primitative model of a classical computer, with an infinite tape of memory, a head that can be moved to left or right depending on the state of a memory cell.

Here we introduce a few important complexity classes in a loose way, just to give the key ideas across.

P: a class of all decision problems that can be solved by a deterministic Turing machine in polynomial time.

NP (nondeterministic polynomial time): a class of decision problems for which the problem instances whose answer is "yes", have proofs verifiable in polynomial time by a deterministic Turing machine (classical computer). It is also formulated as problems that can be solved in polynomial time by a nondeterministic Turing machine (hence the name, NP). One millenmium problem "P=NP?" is still unsolved, but it is believed that **P** is not equal to **NP**.

BQP: a class of problems that can be solved in polynomial time of input size by quantum computers. More rigorously, BQP is a class of languages L in $(0,1)^*$, decidable with bounded error probability (say 1/3) by a uniform family of polynomial-size quantum circuit over some universal family of gates.

QMA: a class of problems that the solution can be verified in polynomial time by quantum computers.

III. QUANTUM ADVANTAGE AND NISQ DEVICES

Physicist John Preskill coined the term "quantum supremacy" [2] as the demonstration of quantum operation that classical computer cannot simulate efficiently. The quest for large-scale quantum computing may push physics into

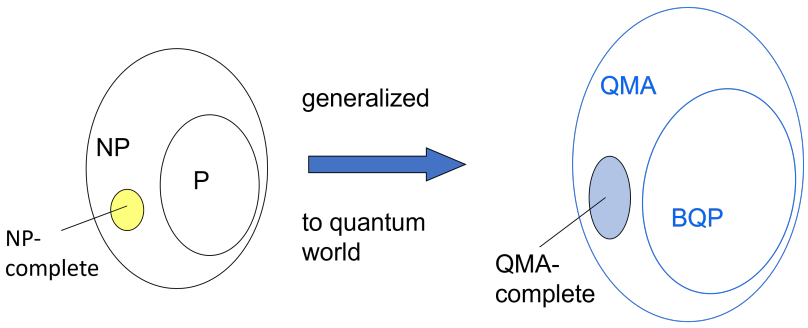


FIG. 1. Illustration of some classical complexity classes and their generalizations to the quantum regime.

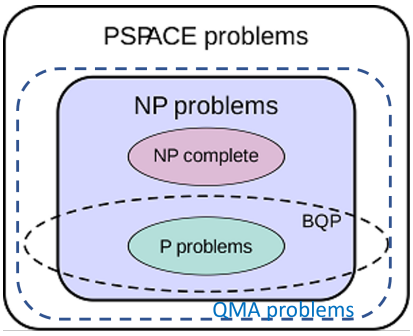


FIG. 2. Illustration of relations between several classical and quantum complexity classes.

a new regime never explored before. Some study suggested that about 50 qubits were the threshold where quantum advantage or supremacy might appear. This is one reason why companies like IBM, Intel & Google, etc. want to build quantum computers over 50 qubits. Current quantum processors are still noisy. Preskill also coined the term NISQ: Noisy Intermediate-Scale Quantum (NISQ) devices [1] to characterize current and near-term machines before full quantum error correction can be implemented. (We will discuss quantum error correction in the next unit.) Given, we cannot do QEC, what are near-term applications? Can they also display quantum advantage?

Google's Quantum team has performed random circuit experiments on their device of 53 qubits and they claimed that the experiments allow them to predict probabilities for random outcomes much more efficiently than any classical computers can calculate [3]. This gives some quantum supremacy; however, classical methods have also been improved (see, e.g., Ref. [4]), so the jury is still out whether quantum supremacy has actually been realized. Perhaps, Google just needs to ramp up the number of qubits and their circuit depth. Another type of tasks that yields 'quantum supremacy' is the so-called Boson Sampling Problem proposed by Aaronson and Arkhipov [5]. See a recent experiment carried out by the Xanadu group [6].

IV. VARIATIONAL APPROACH

This approach as adapted for quantum computers is now referred to as variational quantum eigensolver (VQE) [7]. It is based on the variational principle: for a Hamiltonian (i.e. a Hermitian matrix related to the energy of a system) the lowest energy can be approached from above by minimizing over some ansatz,

$$E_{\min} = \min_{\substack{\Psi \\ \text{unconstrained}}} \frac{\langle\Psi|H|\Psi\rangle}{\langle\Psi|\Psi\rangle} \leq \min_{\vec{\theta}} \frac{\langle\Psi(\vec{\theta})|H|\Psi(\vec{\theta})\rangle}{\langle\Psi(\vec{\theta})|\Psi(\vec{\theta})\rangle}.$$

VQE uses certain ansatz for the wavefunction Ψ parametrized by θ's i.e. constructed by a quantum circuit composed of gates with parameters θ's (thus Ψ is normalized, i.e. $|\Psi| = 1$).

It uses a quantum computer and measurement to evaluate the expectation (by repeatedly creating the state $\Psi(\{\theta\})$ and measuring H) $\langle\Psi(\vec{\theta})|H|\Psi(\vec{\theta})\rangle$. Then one applies a classical optimizer to find the next set of parameters $\{\theta^{(2)}\}$. The process is repeated until the expectation value converges within a prescribed accuracy.

A simple example. Suppose the Hamiltonian H is

$$H = -B_x\sigma_x - B_z\sigma_z = -\begin{pmatrix} B_z & B_x \\ B_x & -B_z \end{pmatrix},$$

and the wavefunction ansatz is

$$|\Psi(\theta,\phi,\lambda)\rangle = u3(\theta,\phi,\lambda)|0\rangle, \qquad |0\rangle \text{ —} \boxed{u3(\theta,\phi,\lambda)} \text{—}$$

To obtain the expectation value, we need to measure separately

$$\langle\Psi|\sigma_z|\Psi\rangle = P_0 - P_1, \quad \langle\Psi|\sigma_x|\Psi\rangle = P_+ - P_-.$$

The first one is given by the difference between the probabilities of obtaining 0 and 1. The second is similar, but in the +/- basis; can use first rotate to 0/1 using the Hadamard gate before measuring in 0/1 basis

$$|0\rangle \text{ —} \boxed{u3(\theta,\phi,\lambda)} \text{—} \boxed{\sigma_z} \text{═} \qquad |0\rangle \text{ —} \boxed{u3(\theta,\phi,\lambda)} \text{—} \boxed{H} \text{—} \boxed{\sigma_z} \text{═}$$

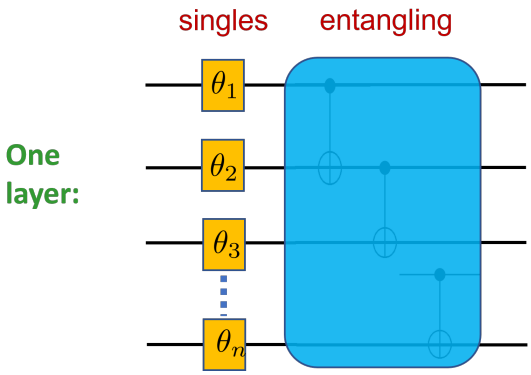


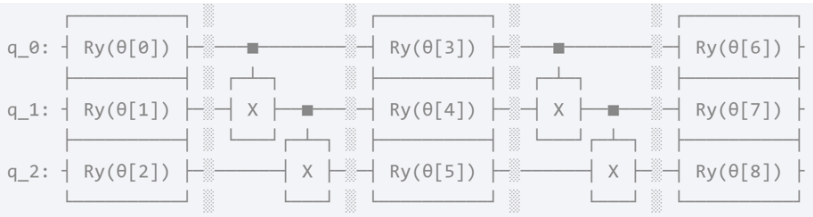
FIG. 3. Illustration of one layer in the variational quantum circuit.

Multi-qubit and more complicated ansatze. In the older version of Qiskit, there are several kinds of ansatze: (1) The RY trial wave function: layers of y rotations with entanglements; (2) The RYRZ trial wave function: layers of y plus z rotations with entanglements; (3) SwapRZ Variational Form: layers of swap plus z rotations with entanglements. These represent different types of single qubits and two-qubit gates used.

In terms of the entangling structure, one can have linear (e.g. nearest-neighbor) and full (i.e. all-to-all). This is implemented in Qiskit's circuit library,

```
>>> from qiskit.circuit.library import TwoLocal
>>> two = TwoLocal(3, 'ry', 'cx', 'linear', reps=2, insert_barriers=True)
>>> print(two)  # decompose the layers into standard gates
```

You will see the following circuit:



and the entanglement structure can be specified by a parameter

```
>>> entangler_map = [(0, 1), (1, 2), (2, 0)]
```

See `https://qiskit.org/documentation/locale/de_DE/stubs/qiskit.circuit.library.TwoLocal.html`.

One reason we have many types of variational circuits is that we do not know exactly what works and people have examined more types of circuits and collected them into our repertoire of ansatze.

The best summary of VQE is by Peruzzo et al. [7] as illustrated in Fig. 4, to find the lowest energy of a Hamiltonian

$$H = \sum_{i=1}^{N} c_i H_i.$$

V. SOME APPLICATIONS

A. MaxCut

The problem is to cut edges so that vertices are separated into two groups and the total weight is maximum. We can use $x_i = 0, 1$ to represent whether the vertex i is in group 0 or group 1,

$$\tilde{C}(\mathbf{x}) = \sum_{i,j} w_{ij} x_i (1 - x_j).$$

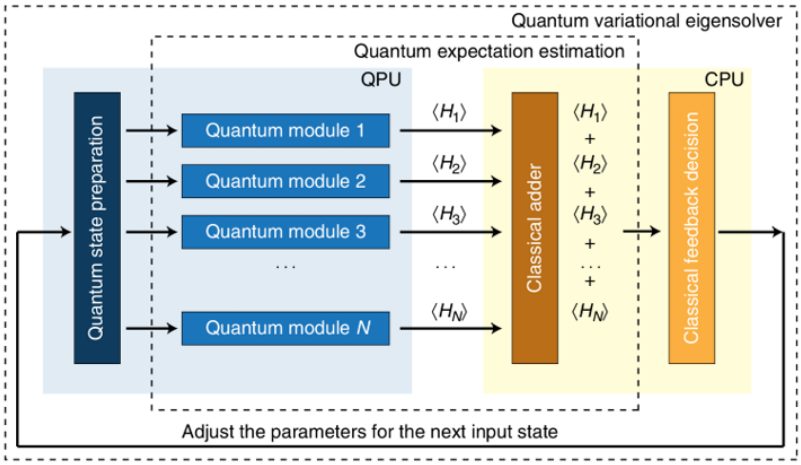


FIG. 4. Illustration of the variational quantum eigensolver; figure taken from Ref. [7].

We can map this classical cost function to a classical Ising-like Hamiltonian via: $x = 0/1$ is mapped to $z = \pm 1$ by $x = (1 - z)/2$,

$$C(\mathbf{Z}) = \sum_{i,j} \frac{w_{ij}}{4}(1 - Z_i)(1 + Z_j) + \sum_i \frac{w_i}{2}(1 - Z_i) = -\frac{1}{2}\left(\sum_{i<j} w_{ij} Z_i Z_j + \sum_i w_i Z_i\right) + \text{const.}$$

We thus minimize the Hamiltonian:

$$H = \sum_i w_i Z_i + \sum_{i<j} w_{ij} Z_i Z_j.$$

Then, we approximate universal quantum computing for optimization by use of variational wavefunctions to minimize the energy expectation value,

$$|\psi(\theta)\rangle = [U_{\text{single}}(\boldsymbol{\theta}) U_{\text{entangler}}]^m |+\rangle, \qquad C(\boldsymbol{\theta}) = \langle \psi(\boldsymbol{\theta}) \, |H| \, \psi(\boldsymbol{\theta})\rangle.$$

As an example, we create 4 nodes and add edges with weights. We use the python package `network`.

```
# Generating a graph of 4 nodes
n=4 # Number of nodes in graph
G=nx.Graph()
G.add_nodes_from(np.arange(0,n,1))
elist=[(0,1,1.0),(0,2,1.0),(0,3,1.0),(1,2,1.0),(2,3,1.0)]
# tuple is (i,j,weight) where (i,j) is the edge
G.add_weighted_edges_from(elist)
colors = ['r' for node in G.nodes()]
pos = nx.spring_layout(G)
default_axes = plt.axes(frameon=True)
nx.draw_networkx(G, node_color=colors, node_size=600, alpha=.8, ax=default_axes, pos=pos)
```

B. Traveling Salesman Problem

Find the shortest Hamiltonian cycle in a graph $G = (V, E)$ with $n = |V|$ nodes and distances, w_{ij} (distance from vertex i to vertex j). A Hamiltonian cycle is described by n^2 variables $x_{i,p}$, where i represents the node and p represents its order in a prospective cycle. The decision variable takes the value 1 if the solution occurs at node i at time order p. We require that every node can only appear once in the cycle, and for each time a node has to occur. This amounts to the two constraints (here and in the following, whenever not specified, the summands run over 0,1,...,n-1)

$$\sum_i x_{i,p} = 1 \quad \forall p, \qquad \sum_p x_{i,p} = 1 \quad \forall i.$$

For nodes in our prospective ordering, if $x_{i,p}$ and $x_{j,p+1}$ are both 1, then there should be an energy penalty if $(i,j) \notin E$ (not connected in the graph). The form of this penalty is

$$\sum_{i,j\notin E} \sum_{p} x_{i,p} x_{j,p+1} > 0,$$

where it is assumed the boundary condition of the Hamiltonian cycles $(p = n) \equiv (p = 0)$. However, here it will be assumed a fully connected graph and not include this term. The distance that needs to be minimized is

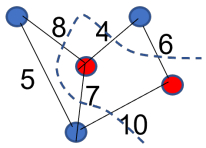
$$C(\mathbf{x}) = \sum_{i,j} w_{ij} \sum_{p} x_{i,p} x_{j,p+1}.$$

Putting everything in a single objective function to be minimized, we have

$$C(\mathbf{x}) = \sum_{i,j} w_{ij} \sum_{p} x_{i,p} x_{j,p+1} + A \sum_{p} \left(1 - \sum_{i} x_{i,p}\right)^2 + A \sum_{i} \left(1 - \sum_{p} x_{i,p}\right)^2.$$

VI. QUANTUM APPROXIMATE OPTIMIZATION ALGORITHM (QAOA)

This is an algorithm proposed by Farhi and collaborators [8]. The goal is to optimize a classical function, in particular, combinatorial problems, such as (weighted) MAXCUT and MAX-3SAT: $C_{\max} = \max_{\vec{x}} C(\vec{x})$,



$$(x_1 \vee \bar{x}_2 \vee \bar{x}_4) \wedge (x_2 \vee x_3 \vee x_5) \vee (x_1 \vee x_5 \vee \bar{x}_6)$$

It uses a specific variational form (with p layers) that reflects the classical problem,

$$|\psi_p(\vec{\gamma}, \vec{\beta})\rangle = e^{i\beta_p H_x} e^{i\gamma_p C} \cdots e^{i\beta_1 H_x} e^{i\gamma_1 C} |+\rangle^{\otimes n},$$

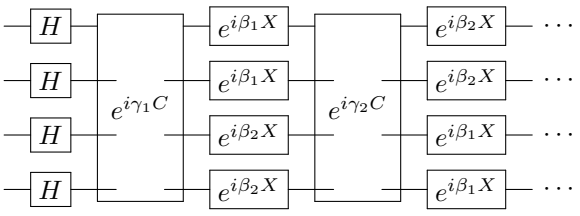
where

$$C = \sum_{\vec{z}} c_{\vec{z}} |\vec{z}\rangle\langle\vec{z}|, \quad H_x = \sum_{j=1}^{n} \sigma_x^j.$$

The goal is to maximze the expectation value of C:

$$\max_{\vec{\gamma},\vec{\beta}} \sum_{\vec{z}} c_{\vec{z}} |\langle\vec{z}|\psi(\vec{\gamma},\vec{\beta})\rangle|^2.$$

The circuit is illustrated below,



Note that optimal parameters β's and γ's are sometimes found using grid search (evaluating all grid points and identify the optimal ones) in a coarse way. Then one can choose a few different sets corresponding to lowest few C values for further optimization. These optimal parameters can also be optimized by using e.g. gradient descent, Nelder-Mead, etc. See Fig. 5 for illustration of the landscape for a pair of β and γ.

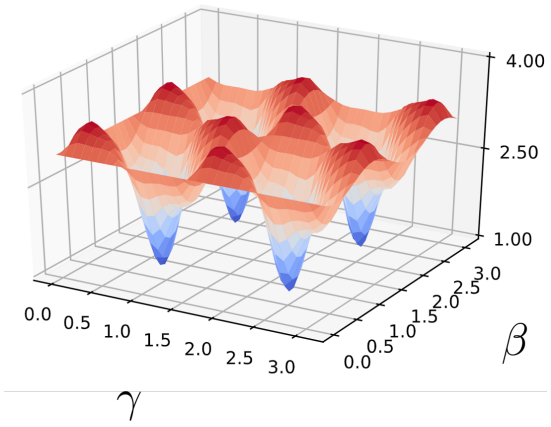


FIG. 5. Illustration of the landscape C for a pair of β and γ in QAOA.

VII. VQE FOR QUANTUM CHEMISTRY

We first obtain the Hamiltonian in terms of fermionic operators representing electrons for a specific molecule,

$$H = H_1 + H_2 = \sum_{\alpha,\beta} t_{\alpha\beta} c_\alpha^\dagger c_\beta + \frac{1}{2} \sum_{\alpha,\beta,\gamma,\delta} u_{\alpha\beta\gamma\delta} c_\alpha^\dagger c_\gamma^\dagger c_\delta c_\beta$$

Orbitals, of e.g. Li and H in HLi molecule, are labeled by $\alpha, \beta, \gamma, \delta$, and $t_{\alpha\beta}$ represents "hopping strength" of an electron from β-th orbital to α-th orbital and $u_{\alpha\beta\gamma\delta}$ represents interaction of two electrons that hop among four orbitals,

$$t_{\alpha\beta} = \int dx_1 \Psi_\alpha(x_1) \left(-\frac{\nabla_1^2}{2} + \sum_i \frac{Z_i}{|r_{1i}|} \right) \Psi_\beta(x_1),$$
$$u_{\alpha\beta\gamma\delta} = \iint dx_1 dx_2 \Psi_\alpha^*(x_1) \Psi_\beta(x_1) \frac{1}{|r_{12}|} \Psi_\gamma^*(x_2) \Psi_\delta(x_2).$$

Fortunately, there are existing packages, e.g. PyScf, that can take care of the calculation of these coefficients:

```
from qiskit_nature.drivers import PySCFDriver
inter_dist=1.6
driver = PySCFDriver(atom='Li .0 .0 .0; H .0 .0 ' + str(inter_dist), \
unit=UnitsType.ANGSTROM, charge=0, spin=0, basis='sto3g')
molecule = driver.run()
h1 = molecule.one_body_integrals
h2 = molecule.two_body_integrals
```

One can also obtain the second quantized operators, associated with the problem,

```
from qiskit_nature.problems.second_quantization.electronic import ElectronicStructureProblem
problem = ElectronicStructureProblem(driver)
second_q_ops = problem.second_q_ops()
main_op = second_q_ops[0]
```

The `main_op` is the Hamiltonian in fermionic operators. To convert to qubits, one has a few approaches: (1) Jordan-Wigner, (2) Parity, and (3) Bravyi-Kitaev. For Jordan-Wigner:

$$c_j = \Big(\prod_{k=1}^{j-1} \sigma_k^z\Big)\sigma_j^+ = \Big(\prod_{k=1}^{j-1} \sigma_k^z\Big)(\sigma_j^x + i\,\sigma_j^y), \qquad c_j^\dagger = \Big(\prod_{k=1}^{j-1} \sigma_k^z\Big)\sigma_j^- = \Big(\prod_{k=1}^{j-1} \sigma_k^z\Big)(\sigma_j^x - i\,\sigma_j^y).$$

The mapping from fermionic numbers to qubits is

$$|'\text{zero fermion}'\rangle \leftrightarrow |'\text{qubit} - \text{state}\,0'\rangle, \qquad |'\text{one fermion}'\rangle \leftrightarrow |'\text{qubit} - \text{state}\,1'\rangle.$$

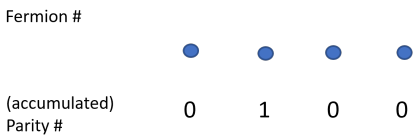


FIG. 6. Illustration of the parity mapping.

The parity mapping is illustrated in Fig. 6. To convert to the qubit Hamiltonian, one specifies the mapping method and uses 'QubitConverter'

```
from qiskit_nature.mappers.second_quantization import ParityMapper, JordanWignerMapper
from qiskit_nature.converters.second_quantization.qubit_converter import QubitConverter
converter = QubitConverter(mapper=JordanWignerMapper(), two_qubit_reduction=False)
num_particles = (problem.molecule_data_transformed.num_alpha,
             problem.molecule_data_transformed.num_beta)
qubit_op = converter.convert(main_op, num_particles=num_particles)
```

where `qubit_op` is the qubit-version Hamiltonian. To use the VQE, we need to construct (1) an initial state, and choose (2) a particular variational ansatz circuit, as well as (3) a classical optimizer. We omit the details but show a partial code,

```
algorithm = VQE(ansatz,
                optimizer=optimizer,
                quantum_instance=backend,
                callback=callback,
                initial_point=initial_point)

result = algorithm.compute_minimum_eigenvalue(qubit_op)
```

We note that for the Parity mapping, the Hamitonian usually possesses some symmetry and this and additional symmetry can be exploited to reduce the total number of qubits in the problem.

VIII. HYBRID CLASSICAL-QUANTUM NEURAL NETWORKS

Currently, hybrid classical-quantum neural networks employ variational quantum circuits. In order to load classical data, one needs to encode them into the quantum computer, and this is usually called quantum feature map. We will probably not be able to cover in detail. In fact, quantum machine learning was the subject of Qiskit Global Summer School 2021; see `https://qiskit.org/events/summer-school/`. To be updated...

IX. CONCLUDING REMARKS

In this unit, we have discussed several complexity classes and variational methods that quantum computers can be useful.

It is a good time to check whether you have achieved the following Learning Outcomes:
After this Unit, you'll be able to modify example Python Notebooks to run variational quantum eigensolvers for applications.

Suggested reading: Qb chap 4.

[1] J. Preskill, Quantum computing in the nisq era and beyond, Quantum **2**, 79 (2018).
[2] J. Preskill, Quantum computing and the entanglement frontier-rapporteur talk at the 25th solvay conference, arXiv preprint arXiv:1203.5813 (2012).

[3] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, Quantum supremacy using a programmable superconducting processor, Nature **574**, 505 (2019).
[4] F. Pan and P. Zhang, Simulation of quantum circuits using the big-batch tensor network method, Phys. Rev. Lett. **128**, 030501 (2022).
[5] S. Aaronson and A. Arkhipov, The computational complexity of linear optics, in *Proceedings of the forty-third annual ACM symposium on Theory of computing* (2011) pp. 333–342.
[6] L. S. Madsen, F. Laudenbach, M. F. Askarani, F. Rortais, T. Vincent, J. F. Bulmer, F. M. Miatto, L. Neuhaus, L. G. Helt, M. J. Collins, *et al.*, Quantum computational advantage with a programmable photonic processor, Nature **606**, 75 (2022).
[7] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, A variational eigenvalue solver on a photonic quantum processor, Nature communications **5**, 1 (2014).
[8] E. Farhi, J. Goldstone, and S. Gutmann, A quantum approximate optimization algorithm, arXiv preprint arXiv:1411.4028 (2014).