Qiskit Optimization: Quantum algorithms for applications of optimization

Takashi Imamichi and Atsushi Matsuo
IBM Quantum, IBM Research – Tokyo
imamichi@jp.ibm.com
matsuoa@jp.ibm.com

Workshop 1.3
Nov 30, 2021
10:30 AM - 1:30 PM

EQTC 2021
2nd European Quantum Technologies *Virtual* Conference
29 November - 2 December 2021

Qiskit

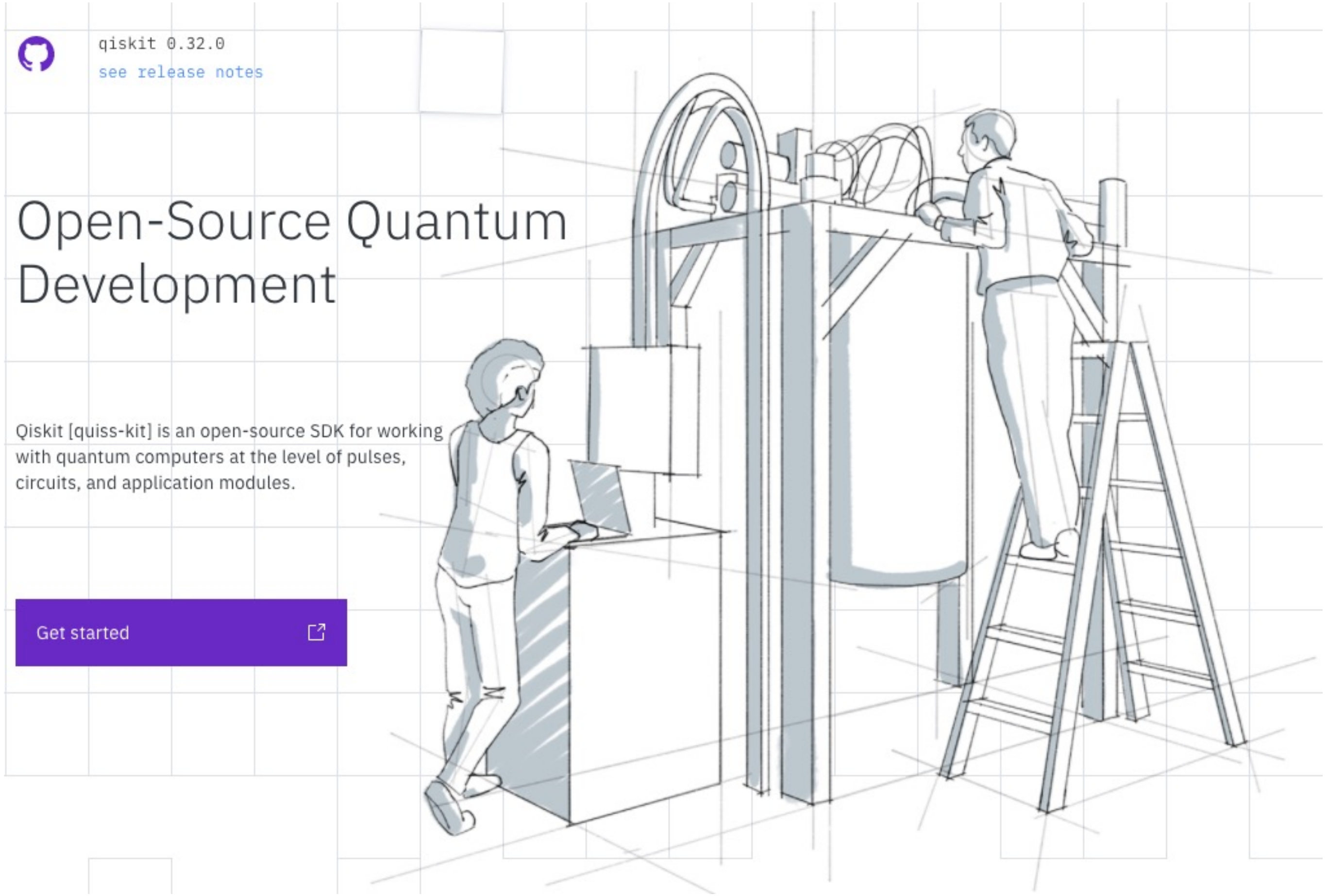
IBM **Quantum**

Schedule

- **9:30 – 10:55 (85 min): Introduction of combinatorial optimization and quantum algorithms (Takashi Imamichi)**
  - Introduction of this tutorial
  - Overview of combinatorial optimization
  - Classical-quantum hybrid scheme
  - Noisy quantum computers
  - How to convert optimization problems to Hamiltonians
  - Quantum algorithms for optimization
    - Variational quantum algorithms (VQE)
    - Quantum approximate optimization algorithm (QAOA)
  - Overview of Qiskit Optimization
  - Q&A
- **10 min break**
- **11:05 – 12:30 (85 min): Hands-on: Qiskit Optimization workflow, how to define an optimization model and solve it (Atsushi Matsuo)**
  - Overview of Qiskit Optimization and Qiskit Terra
  - Workflow to define and solve an optimization model
    - Basics of Docplex to define optimization models
    - Data flow of how a problem is converted into QUBO and executed on quantum computers
  - Example: Maxcut
    - Define a Maxcut problem instance
    - Solve the problem instance with VQE and QAOA
    - Solve the problem with a simulator and a noise model
  - Exercise: TSP
  - Q&A

Introduction of Qiskit

- Qiskit is an open-source SDK for working with quantum computers at the level of pulses, circuits, and application modules
- Qiskit consists of various modules to develop programs and applications, execute and simulate programs
    - Qiskit Terra
    - Qiskit Aer
    - Qiskit Optimization
    - Qiskit Machine learning
    - Qiskit Nature
    - Qiskit Finance
    - Etc.



qiskit 0.32.0
see release notes

Open-Source Quantum Development

Qiskit [quiss-kit] is an open-source SDK for working with quantum computers at the level of pulses, circuits, and application modules.

https://qiskit.org/

Quantum Computer and Optimization

- You may have heard news that some optimization problems are solved by quantum computer more efficiently than classical computer?
  - Traveling salesman problem (TSP)
  - Maximum cut problem (Maxcut)

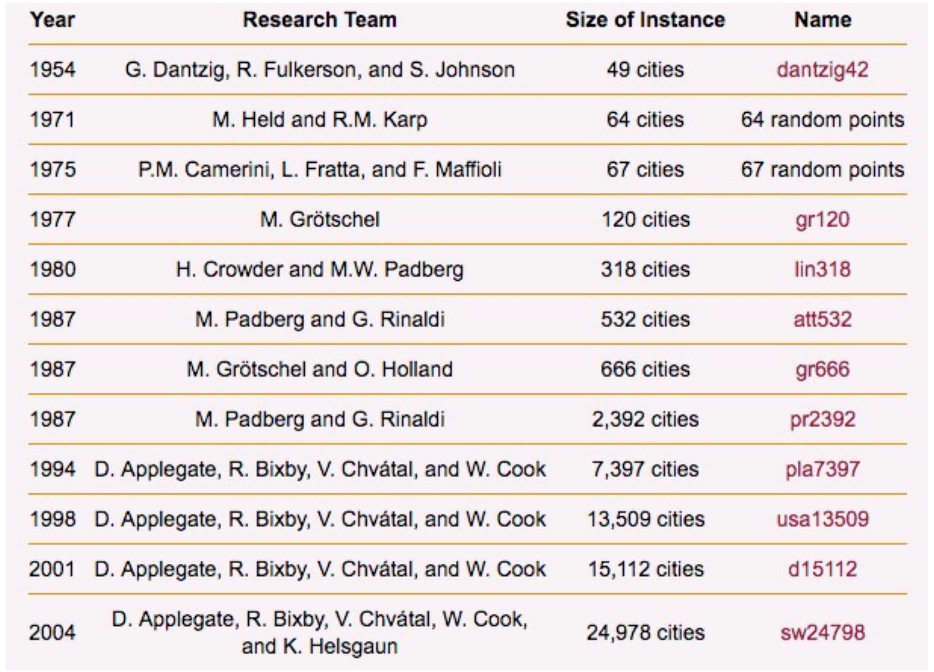
- Some news also explain that superposition can deal with combinatorial optimization problem efficiently because it can compute f(x) of all possible x
  - Is it true?

- It is true that you can computer superposition of f(x) of all possible x.
- But you cannot get the minimum/maximum easily…
  - Just applying measurement to f(x) results in f(x) of a random x
    - No guarantee of optimality
  - You may know some quantum algorithms such as Grover search and notice that you need some techniques to obtain relevant solutions.

Traveling Salesman Problem 1/2

- Definition: Given a set of n cities, you are asked to find the shortest route that visits each city and returns to the start.
  - TSP is known to be NP-hard
  - Naïve approach: Check all permutations of the cities. O(n!) time
  - Claim: If you have several tens of cities, it takes millions of years to solve it with conventional computers. Is it true?
    - E.g., 50! ≒ 3 x 10^{64}
- History
  - An instance with 49 cities was solved in 1954(!)
  - An instance with 85900 cities was solved in 2006
- Reference
  - http://www.math.uwaterloo.ca/tsp/index.html

Source: https://physics.aps.org/articles/v10/s32



Year	Research Team	Size of Instance	Name
1954	G. Dantzig, R. Fulkerson, and S. Johnson	49 cities	dantzig42
1971	M. Held and R.M. Karp	64 cities	64 random points
1975	P.M. Camerini, L. Fratta, and F. Maffioli	67 cities	67 random points
1977	M. Grötschel	120 cities	gr120
1980	H. Crowder and M.W. Padberg	318 cities	lin318
1987	M. Padberg and G. Rinaldi	532 cities	att532
1987	M. Grötschel and O. Holland	666 cities	gr666
1987	M. Padberg and G. Rinaldi	2,392 cities	pr2392
1994	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	7,397 cities	pla7397
1998	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	13,509 cities	usa13509
2001	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	15,112 cities	d15112
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, and K. Helsgaun	24,978 cities	sw24798

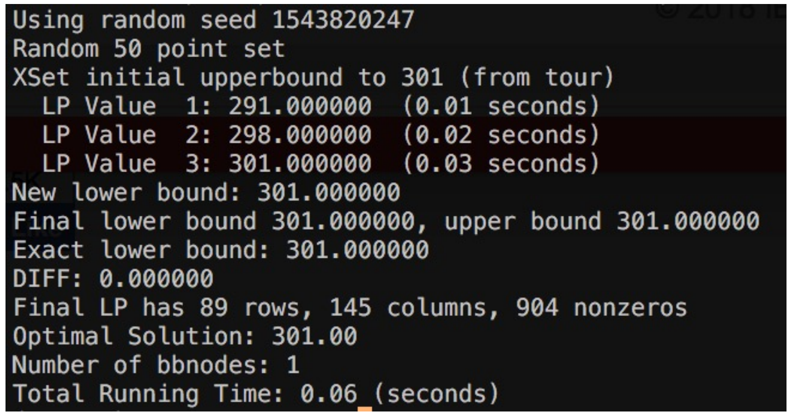
Milestones in the solutions of TSP instances

Source: https://www.math.uwaterloo.ca/tsp/history/milestone.html

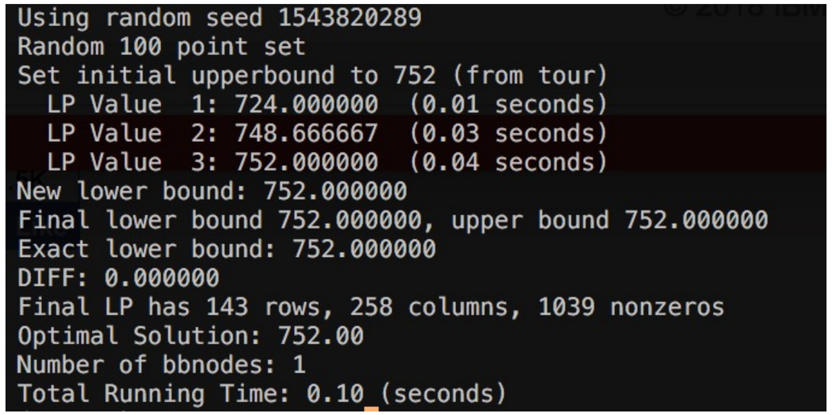
Traveling Salesman Problem 2/2

- Concorde is the state-of-the-art TSP solver
- I solved a random instance with 50 cities **exactly** within 0.1 second
  - Macbook Pro (13-inch, Early 2015, Core i5 2.7GHz)

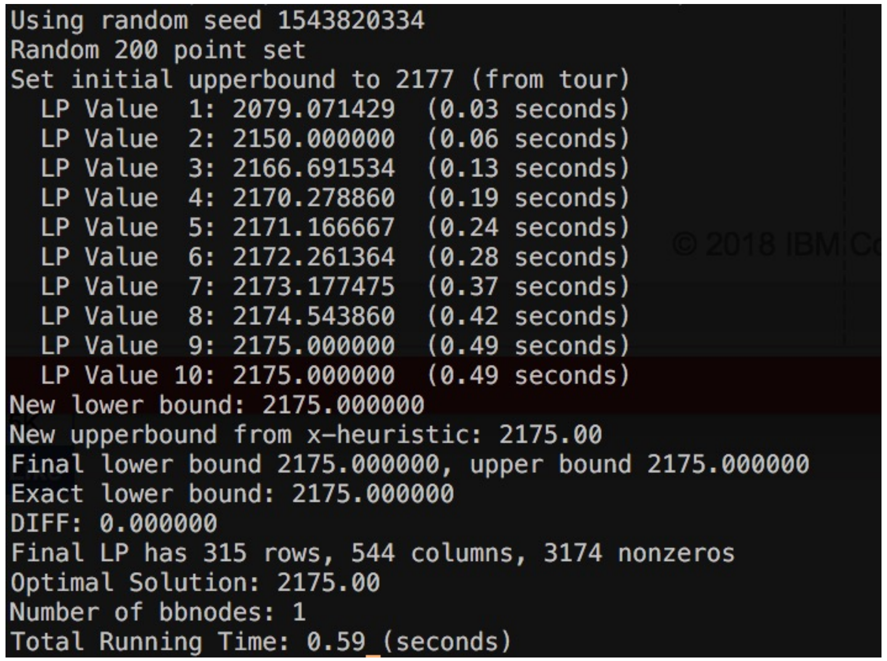
- 50 cities 0.06 sec 100 cities 0.10 sec 200 cities 0.59 sec



```
Using random seed 1543820247
Random 50 point set
XSet initial upperbound to 301 (from tour)
  LP Value  1: 291.000000  (0.01 seconds)
  LP Value  2: 298.000000  (0.02 seconds)
  LP Value  3: 301.000000  (0.03 seconds)
New lower bound: 301.000000
Final lower bound 301.000000, upper bound 301.000000
Exact lower bound: 301.000000
DIFF: 0.000000
Final LP has 89 rows, 145 columns, 904 nonzeros
Optimal Solution: 301.00
Number of bbnodes: 1
Total Running Time: 0.06 (seconds)
```



```
Using random seed 1543820289
Random 100 point set
Set initial upperbound to 752 (from tour)
  LP Value  1: 724.000000  (0.01 seconds)
  LP Value  2: 748.666667  (0.03 seconds)
  LP Value  3: 752.000000  (0.04 seconds)
New lower bound: 752.000000
Final lower bound 752.000000, upper bound 752.000000
Exact lower bound: 752.000000
DIFF: 0.000000
Final LP has 143 rows, 258 columns, 1039 nonzeros
Optimal Solution: 752.00
Number of bbnodes: 1
Total Running Time: 0.10 (seconds)
```



```
Using random seed 1543820334
Random 200 point set
Set initial upperbound to 2177 (from tour)
  LP Value  1: 2079.071429  (0.03 seconds)
  LP Value  2: 2150.000000  (0.06 seconds)
  LP Value  3: 2166.691534  (0.13 seconds)
  LP Value  4: 2170.278860  (0.19 seconds)
  LP Value  5: 2171.166667  (0.24 seconds)
  LP Value  6: 2172.261364  (0.28 seconds)
  LP Value  7: 2173.177475  (0.37 seconds)
  LP Value  8: 2174.543860  (0.42 seconds)
  LP Value  9: 2175.000000  (0.49 seconds)
  LP Value 10: 2175.000000  (0.49 seconds)
New lower bound: 2175.000000
New upperbound from x-heuristic: 2175.00
Final lower bound 2175.000000, upper bound 2175.000000
Exact lower bound: 2175.000000
DIFF: 0.000000
Final LP has 315 rows, 544 columns, 3174 nonzeros
Optimal Solution: 2175.00
Number of bbnodes: 1
Total Running Time: 0.59 (seconds)
```

- How about quantum computers?

- Reference: http://www.math.uwaterloo.ca/tsp/concorde/index.html

Brief Introduction of Optimization

- Optimization problem consists of three components
  - Decision variables
    - Discrete: binary / integer
    - Continuous
  - Objective function: to be minimized or maximized
  - Constraints
    - None / Equality / Inequality

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{Subject to} & g_i(x) = b_i \\ & h_i(x) \leq c_i \\ & x = (x_1, \dots, x_n) \\ & l_i \leq x_i \leq u_i \\ \multicolumn{2}{l}{x_i\text{: binary / integer / continuous}} \end{array}$$

- Task: You are asked to find the best solution among all candidates
  - A candidate that satisfied the constraints is called a feasible solution
- Various types of optimization problems
  - Linear programming (LP)
  - Integer linear Programming (ILP)
  - Mixed integer linear programming (MILP)
  - (Quadratically constrained) Quadratic programming (QP, QCQP)
  - Quadratic unconstrained binary optimization (QUBO)
  - Semidefinite programming (SDP)
  - Nonlinear programming

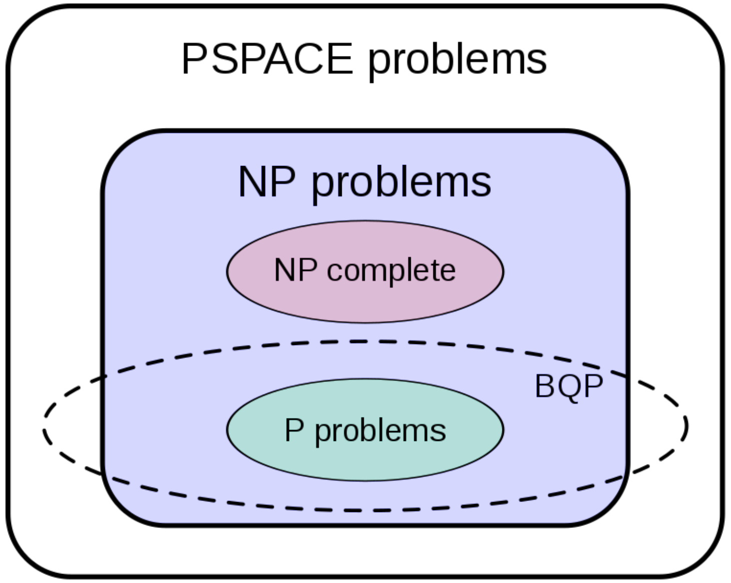
Algorithms to Optimization Problems

- What does “solve” mean?

- Exact algorithm
  - Guarantee of the optimality of the solution
  - E.g., Exhaustive search, Dynamic programming, Branch-and-bound
- Approximation algorithm
  - Solution is not always optimal, but there is a guarantee of distance from the optimal solution
  - E.g., Christofides algorithm (1.5 approximation algorithm to TSP, the solution is 1.5 times longer than the optimal solution at most)
- Heuristics
  - No theoretical guarantee of the quality of solutions, but works well practically
  - E.g., Greedy search, Local search
  - Meta-heuristics
    - Higher level of design of heuristics
    - Often inspired from nature
    - E.g., Genetic algorithm, Simulated annealing, Ant colony optimization

Can Quantum Computers Solve Optimization Problems Efficiently?

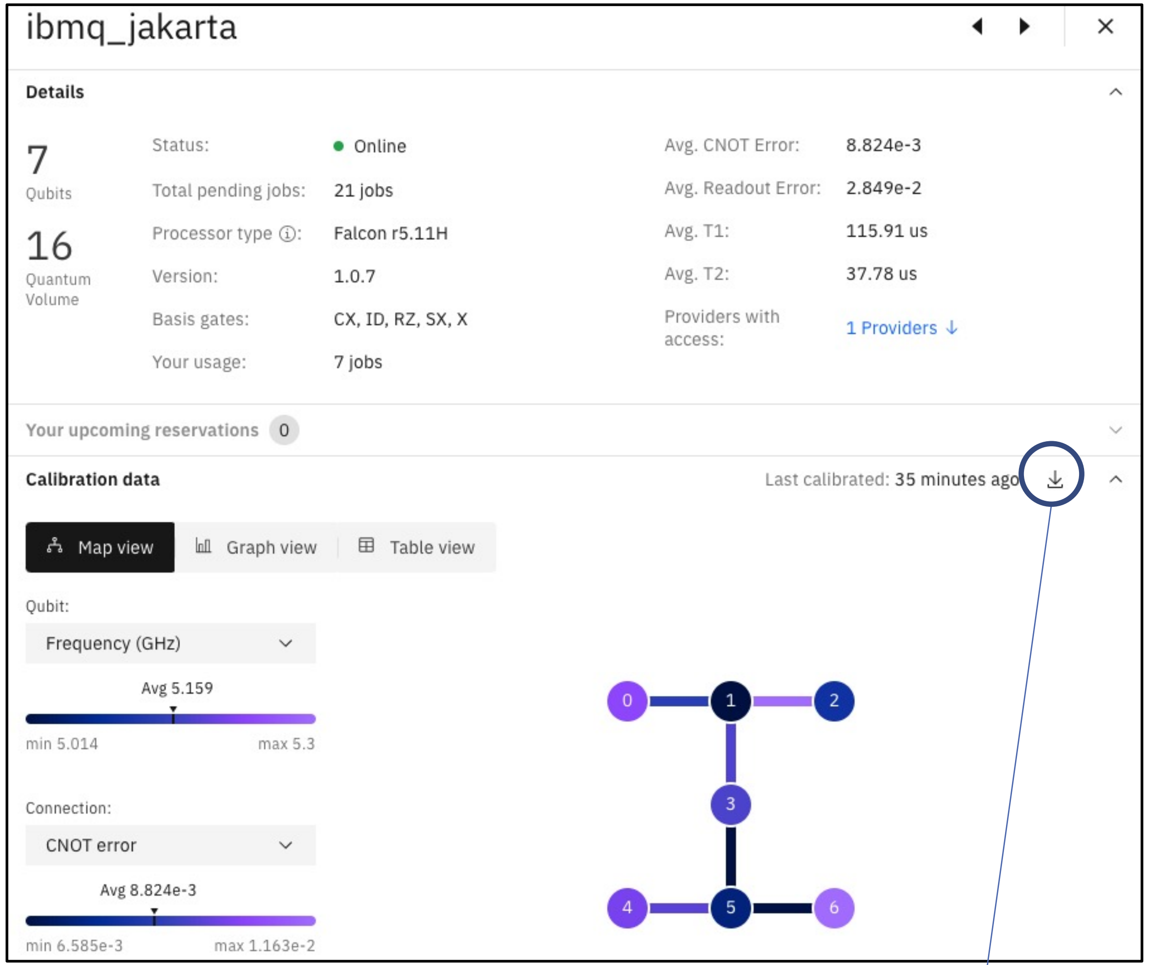
- “Efficiently” usually means that a problem is solvable exactly in polynomial time
  - We assume decision problems, which is a yes-no type problem
- Classes of decision problems
  - P: conventional computers can solve it in polynomial time
  - BQP: quantum computers can solve it in polynomial time
    - Bounded-error quantum polynomial time
  - NP-complete: very hard to solve it with conventional computers in polynomial time
- Some problems are thought not to be in P but in BQP
  - Integer factorization
  - Some problems were proven to be in P suddenly → Quantum-inspired algorithms by Ewin Tang
- Does BQP include NP-complete?
  - Likely to be No
- From practical perspective, it is important to find applications in BQP and produce new quantum-based heuristics



Source: https://en.wikipedia.org/wiki/BQP

Noisy Quantum Devices

- Various limitations of the current noisy quantum devices
  - T1: qubit relaxation
  - T2: qubit dephasing
  - Single qubit error
  - CNOT error
  - Readout error
  - Limited connectivity of qubits
    - More CNOT gates are necessary for qubit mapping
- You cannot expect that the current devices execute millions of gates
  - What if you execute k gates whose error rate is p?
  - Fidelity of the result will be $F = (1-p)^k$
  - E.g., $p = 1e\text{-}2$, $k=100 \rightarrow F = 0.366$



ibmq_jakarta

Details

7 Qubits

16 Quantum Volume

Status:	● Online	Avg. CNOT Error:	8.824e-3
Total pending jobs:	21 jobs	Avg. Readout Error:	2.849e-2
Processor type ⓘ:	Falcon r5.11H	Avg. T1:	115.91 us
Version:	1.0.7	Avg. T2:	37.78 us
Basis gates:	CX, ID, RZ, SX, X	Providers with access:	1 Providers ↓
Your usage:	7 jobs		

Your upcoming reservations 0

Calibration data

Last calibrated: 35 minutes ago

Qubit:

Frequency (GHz)

Avg 5.159

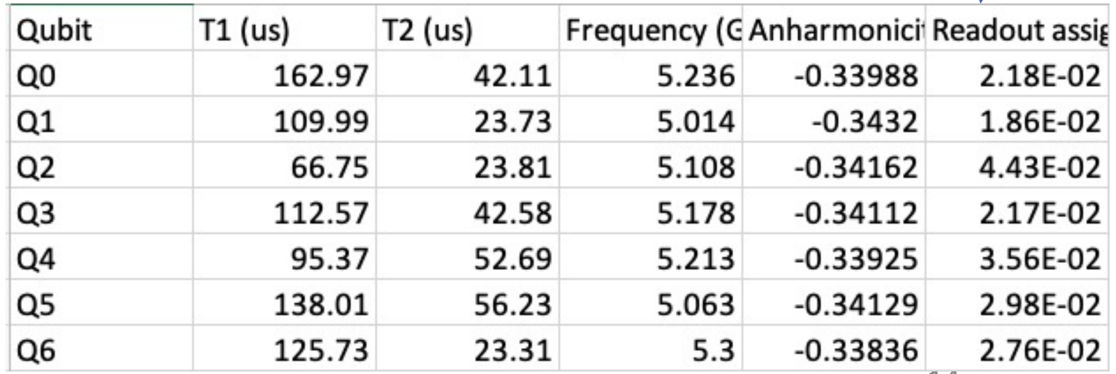
min 5.014 max 5.3

Connection:

CNOT error

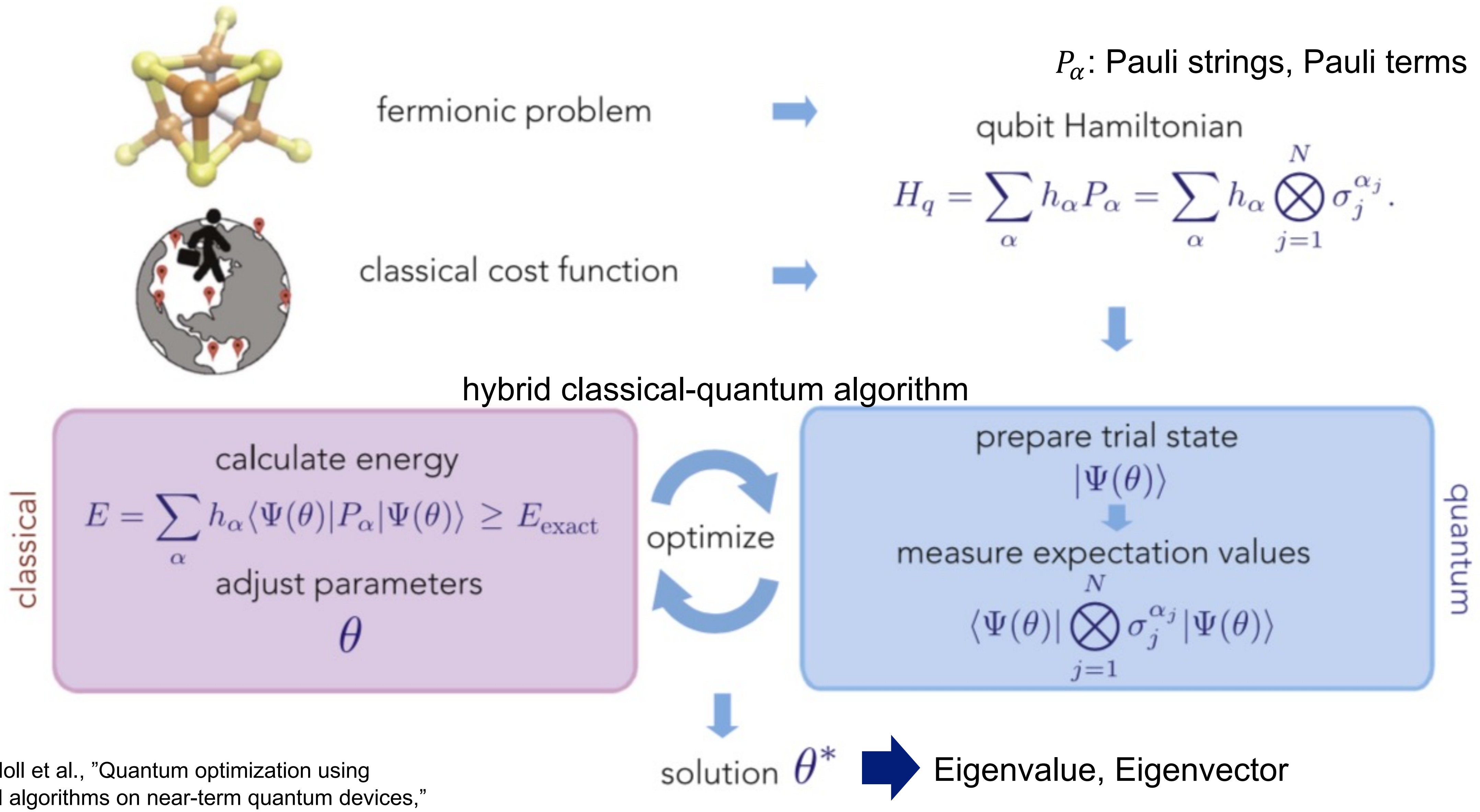
Avg 8.824e-3

min 6.585e-3 max 1.163e-2



Qubit	T1 (us)	T2 (us)	Frequency (G	Anharmonici	Readout assig
Q0	162.97	42.11	5.236	-0.33988	2.18E-02
Q1	109.99	23.73	5.014	-0.3432	1.86E-02
Q2	66.75	23.81	5.108	-0.34162	4.43E-02
Q3	112.57	42.58	5.178	-0.34112	2.17E-02
Q4	95.37	52.69	5.213	-0.33925	3.56E-02
Q5	138.01	56.23	5.063	-0.34129	2.98E-02
Q6	125.73	23.31	5.3	-0.33836	2.76E-02

Overview of Hybrid Classical-Quantum Algorithm


P_α: Pauli strings, Pauli terms
fermionic problem
classical cost function
qubit Hamiltonian
$H_q = \sum_\alpha h_\alpha P_\alpha = \sum_\alpha h_\alpha \bigotimes_{j=1}^{N} \sigma_j^{\alpha_j}.$
hybrid classical-quantum algorithm
classical
calculate energy
$E = \sum_\alpha h_\alpha \langle \Psi(\theta) | P_\alpha | \Psi(\theta) \rangle \geq E_{\text{exact}}$
adjust parameters
θ
optimize
prepare trial state
$|\Psi(\theta)\rangle$
measure expectation values
$\langle \Psi(\theta) | \bigotimes_{j=1}^{N} \sigma_j^{\alpha_j} | \Psi(\theta) \rangle$
quantum
solution θ^*
Eigenvalue, Eigenvector

Source: Moll et al., "Quantum optimization using variational algorithms on near-term quantum devices," Quantum Sci. Technol. 3 (2018) 030503

Convert Optimization problem to Hamiltonian

- Flow: Quadratic program → QUBO → Hamiltonian
  - Assume variables are only binary or integer
  - Represent both the objective function and constraints as a Hamiltonian
- Convert inequality constraints into equality constraints by introducing slack variables
  - $x \leq b \ \rightarrow x + s = b, s \geq 0$: variable
- Convert equality constraints into penalties and add them to the objective function
  - $x = b \ \rightarrow \lambda(x-b)^2$, λ: positive constant
- Encode integer variables with binary variables
- Replace binary variables with Pauli Z matrices
  - $x_i \ \rightarrow (1 - Z_i)/2$ where Z_i is a Pauli Z on *i*-th qubit
    - $x_i = 0 \leftrightarrow |0\rangle,\ x_i = 1 \leftrightarrow |1\rangle$
    - $(1 - Z_i)|0\rangle/2 = 0,\ (1 - Z_i)|1\rangle/2 = |1\rangle$

Quadratic programming problem

$$\begin{array}{ll} \text{Minimize} & x^T A x + B x \\ \text{Subject to} & c_i^T x \leq d_i \\ & x = (x_1, \dots, x_n) \\ & l_i \leq x_i \leq u_i \\ & x_i\text{: binary / integer} \end{array}$$

QUBO

$$\begin{array}{ll} \text{Minimize} & x'^T A' x' + B' x' \\ \multicolumn{2}{l}{+ \lambda \Sigma_i \left(c'^T_i x' + s_i - d'_i\right)^2} \\ \text{Subject to} & x' = (x'_1, \dots, x'_m) \\ & x'_i\text{: binary} \end{array}$$

Hamiltonian

$$H = \Sigma_i w_i P_i$$

P_i: Pauli string (e.g., $ZI = Z \otimes I$)
= tensor product of Pauli matrices

Encode integer variables with binary variables

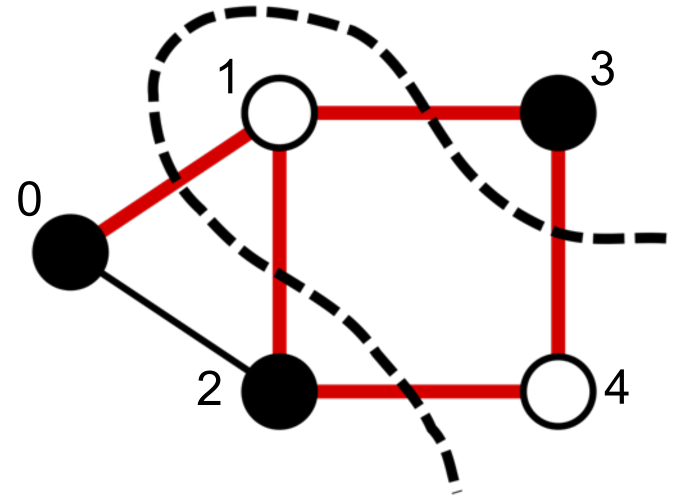
- Two major approaches
- Assume that x is an integer variables with both lower bound and upper bound
  - $l \leq x \leq u$
  - Represent x with a set of binary variables y_i
  - Example: $2 \leq x \leq 10$
- One-hot representation
  - $x = l - 1 + \sum_{i=1} i \cdot y_i$
    - Needs a constraint $\sum_{i=1} y_i = 1$ so that only 1 variable is enabled
  - Example: $x = 1 + \sum_{i=1}^{9} i \cdot y_i$
  - Requires $O(u - l)$ binary variables and 1 additional constraint
- Log representation
  - $x = l + \sum_{i=0} 2^i \cdot y_i + k \cdot y$
  - Example: $x = 2 + y_0 + 2y_1 + 4y_2 + y$
  - Requires $O(\log(u - l))$ binary variables and no additional constraint
  - Qiskit optimization uses this representation

Example 1: Maxcut problem

- Given a graph G = (V, E) and weights w_{ij} of edges (i, j)
  - $w_{ij} > 0$ and $w_{ij}=w_{ji}$
- Separate nodes into two groups such that you maximize the sum of weights between the groups
- Formulation

Maximize $\sum_{(i,j)\in E}\left[w_{ij}x_i(1-x_j)+w_{ij}(1-x_i)x_j\right]$
Subject to $x_i \in \{0,1\}, i \in V$

Take a sum of the cases $(x_i, x_j) = (0,1), (1,0)$



Source: https://en.wikipedia.org/wiki/Maximum_cut

- Conversions

Minimize $\sum_{(i,j)\in E}[-\frac{w_{ij}}{4}(1-Z_i)(1+Z_j)-\frac{w_{ij}}{4}(1+Z_i)(1-Z_j)] = \sum_{(i,j)\in E}[-\frac{w_{ij}}{2}(1-Z_iZ_j)]$
Subject to $Z_i \in \{-1,1\}, i \in V$

Note: multiply -1 to change maximization into minimization

$H = \sum_{(i,j)\in E}\frac{w_{ij}}{2}Z_iZ_j - (\sum_{(i,j)\in E}\frac{w_{ij}}{2}$: constant)
where $Z_i = I^{\otimes n-i-1} \otimes Z \otimes I^{\otimes i-1}, i \in V$ (*i*-th position is Z, otherwise I)

Example 2: Knapsack problem

- Given a set *I* of *n* items, each with a weight and a value, and a capacity of a knapsack
  - w_i : weight of item i, v_i : value of item i, W : capacity of the knapsack
- Determine a collection of items so that the total weight is less than or equal to the capacity and maximize the total value
- Formulation and conversions

$$\begin{aligned}&\text{Maximize} \;\; \textstyle\sum_{i\in I} v_i x_i \\ &\text{Subject to} \;\; \textstyle\sum_{i\in I} w_i x_i \le W \\ &\qquad\qquad x_i \in \{0,1\}, i \in I\end{aligned}$$

- Add a slack variable s to convert the inequality constraint into an equality constraint

$$\begin{aligned}&\text{Maximize} \;\; \textstyle\sum_{i\in I} v_i x_i \\ &\text{Subject to} \;\; \textstyle\sum_{i\in I} w_i x_i + s = W \\ &\qquad x_i \in \{0,1\}, i \in I, s \in \{0,\dots,W\}\end{aligned}$$

- Convert the objective function from maximization to minimization
- Convert the equality constraint into a penalty of the objective function

$$\begin{aligned}&\text{Minimize} \;\; -\textstyle\sum_{i\in I} v_i x_i + \lambda(\sum_{i\in I} w_i x_i + s - W)^2 \\ &\text{Subject to} \;\; x_i \in \{0,1\}, i \in I, s \in \{0,\dots,W\}\end{aligned}$$

- Convert integer variables into a set of binary variables
- Translate binary variable into Pauli Z

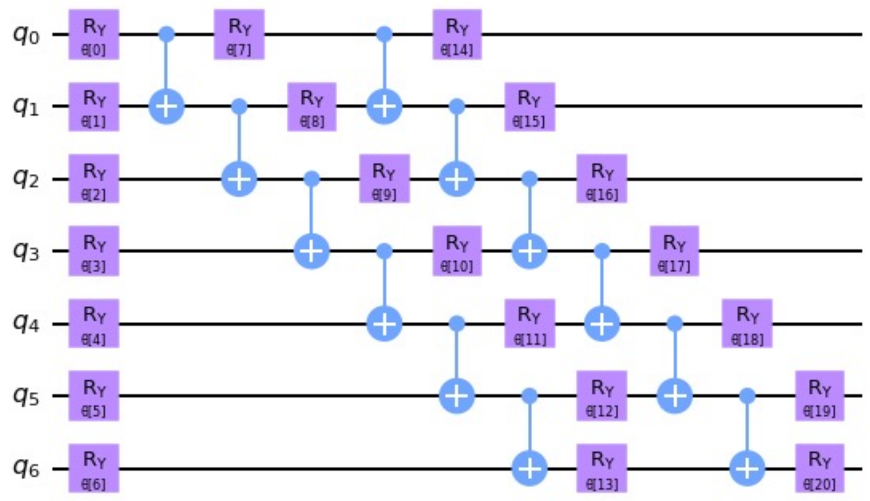
$$\begin{aligned}&\text{Minimize} \;\; \textstyle\sum w_{ij} Z_i Z_j + \sum w_i Z_i \\ &\text{Subject to} \;\; Z_i \in \{-1,1\}\end{aligned}$$

Hamiltonian and Variational Method

- Basic strategy: represent problems with a matrix (Hamiltonian) and find the minimum eigenvalue and eigenvector
  - Eigenvector (eigenvalue) is often called ground state (energy)
- Describe a Hamiltonian as Hermitian matrix
  - Self-adjoint matrix, i.e., $H^\dagger = H$
  - Eigenvalues are real
    - Suppose $|\psi_i\rangle$ and λ_i are an eigenvector (eigenvalue) of H, it holds $\langle\psi_i|H|\psi_i\rangle=\lambda_i\langle\psi_i|\psi_i\rangle=\lambda_i$
    - $\langle\psi_i|H|\psi_i\rangle^\dagger(=\lambda_i^*)=\langle\psi_i|H^\dagger|\psi_i\rangle = \langle\psi_i|H|\psi_i\rangle=\lambda_i \rightarrow \lambda_i = \lambda_i^*$
  - H can be expressed using eigenvalues and eigenvectors as follows
    - $H=\sum_{i=1}^{N}\lambda_i|\psi_i\rangle\langle\psi_i|$
- Let $\langle H\rangle_\phi\equiv\langle\phi|H|\phi\rangle$ denote the expectation value of H on a quantum state or a wave function $|\phi\rangle$
  - $\langle H\rangle_\phi=\langle\phi|H|\phi\rangle=\sum_{i=1}^{N}\lambda_i|\langle\psi_i|\phi\rangle|^2$
  - $|\langle\psi_i|\phi\rangle|^2 \geq 0 \rightarrow \langle H\rangle_\phi \geq \lambda_{\min}$, $\langle H\rangle_{\psi_{\min}} = \lambda_{\min}$ ($|\psi_{\min}\rangle$ is the ground state)
  - Variational method (variational principle)

Variational Quantum Eigensolver

- Task: Given a Hamiltonian H, guess a quantum state or wave function $|\psi\rangle$ that minimizes the expectation value $\langle H\rangle_{\psi}$
  - Wave function $|\psi\rangle$ is often called ansatz
- Variational forms or ansatz
  - Parametrized quantum circuit (PQC) with a fixed form
  - $|\psi(\theta)\rangle \equiv U(\theta)|\psi_0\rangle$
    - $|\psi_0\rangle$: initial state, e.g., $|0\rangle^{\otimes n}$ and $|+\rangle^{\otimes n}$
    - $U(\theta)$: parametrized unitary transformation
- Variational quantum eigensolver (VQE)
  - Quantum part
    - Execute the quantum circuits and get counts associated with bitstrings
  - Classical part
    - Calculate the expectation value $\langle H\rangle_{\psi(\theta)}$ by aggregating counts associated with bitstrings
    - Adjust (optimize) θ and generate quantum circuits of variational forms and Hamiltonian
  - Return $\mathrm{argmin}_{\psi(\theta)} \langle H\rangle_{\psi(\theta)}$ as an approximate ground state

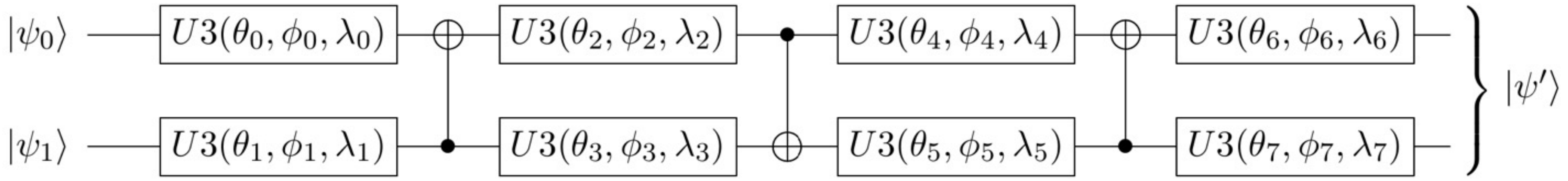


More on Variational Forms

- We want n-qubit variational form that generates any possible state $|\psi(\theta)\rangle$ ideally
  - To guarantee that the variational form covers the ground state
- We construct n-qubit variational form in general
  - 1 qubit rotation gate: Ry, Rz, U3
  - 2 qubit gate: CX, CZ

$$U3(\theta, \phi, \lambda) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -e^{i\lambda}\sin(\frac{\theta}{2}) \\ e^{i\phi}\sin(\frac{\theta}{2}) & e^{i\lambda+i\phi}\cos(\frac{\theta}{2}) \end{pmatrix}$$

- 1 qubit case: U3 gate can generate any possible 1-qubit state
- 2 qubit case: Following circuit generates any 2-qubit state (Shende et al.)



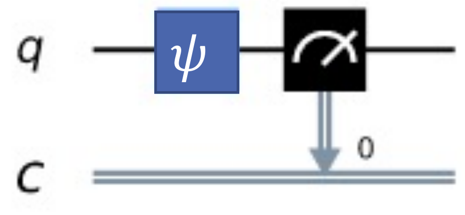
$|\psi_0\rangle$
$|\psi_1\rangle$
$U3(\theta_0, \phi_0, \lambda_0)$
$U3(\theta_1, \phi_1, \lambda_1)$
$U3(\theta_2, \phi_2, \lambda_2)$
$U3(\theta_3, \phi_3, \lambda_3)$
$U3(\theta_4, \phi_4, \lambda_4)$
$U3(\theta_5, \phi_5, \lambda_5)$
$U3(\theta_6, \phi_6, \lambda_6)$
$U3(\theta_7, \phi_7, \lambda_7)$
$|\psi'\rangle$

- Trade-off: more gates, more coverage, more error

- Reference
  - Shende, Vivek V., Igor L. Markov, and Stephen S. Bullock. "Minimal universal two-qubit cnot-based circuits." arXiv preprint quant-ph/0308033 (2003).

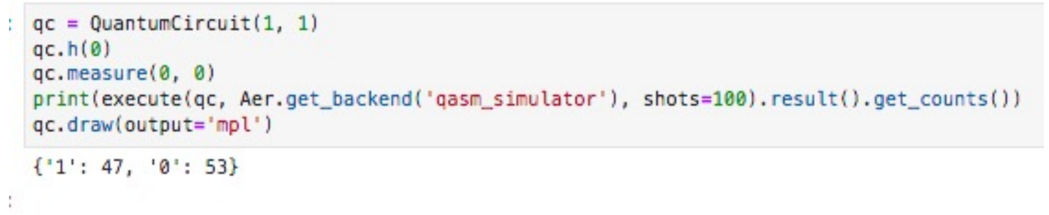
Quantum Part of VQE

- Hamiltonian is usually expressed as a sum of Pauli strings
  - n-qubit Pauli string is a tensor product of n Pauli matrices (including I)
    - E.g., $H = 2\times I \otimes X \otimes Y + 3\times X \otimes Z \otimes Z$ (2 Pauli strings with 3 qubits)
  - We usually use only Z and I for optimization problems
  - Recall that Pauli matrices and I are Hermitian (and unitary too)
    - $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
- Example: $H = I \text{ or } Z$ and $|\psi\rangle = a|0\rangle + b|1\rangle$
  - If you execute the following circuit, you will get counts with probability {'0': $|a|^2$, '1': $|b|^2$} because of Z-measurement
  - $\langle I \rangle_\psi$ = $\langle\psi|\psi\rangle$ = $|a|^2 + |b|^2$
  - $\langle Z \rangle_\psi$= $\langle\psi|Z|\psi\rangle$ = $|a|^2 - |b|^2$
  - What if $\langle Z \otimes I \rangle_\psi$?
    - You construct circuits each qubit independently
  - What is $\langle X \rangle_\psi$?
    - You need a trick to diagonalize a matrix



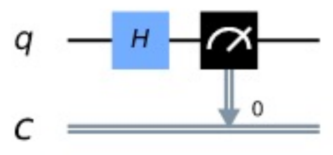
Sampling Error of Quantum Computer

- If you execute a circuit on a quantum computer, you get counts of bit strings.
- Counts always have sampling error
  - E.g., $\psi = (|0\rangle + |1\rangle)/\sqrt{2}$
    → 100 shots → {'1': 47, '0': 53}
  - Trade-off: more shots, more precise, more time
- Even fault-tolerant quantum computers have this type of error

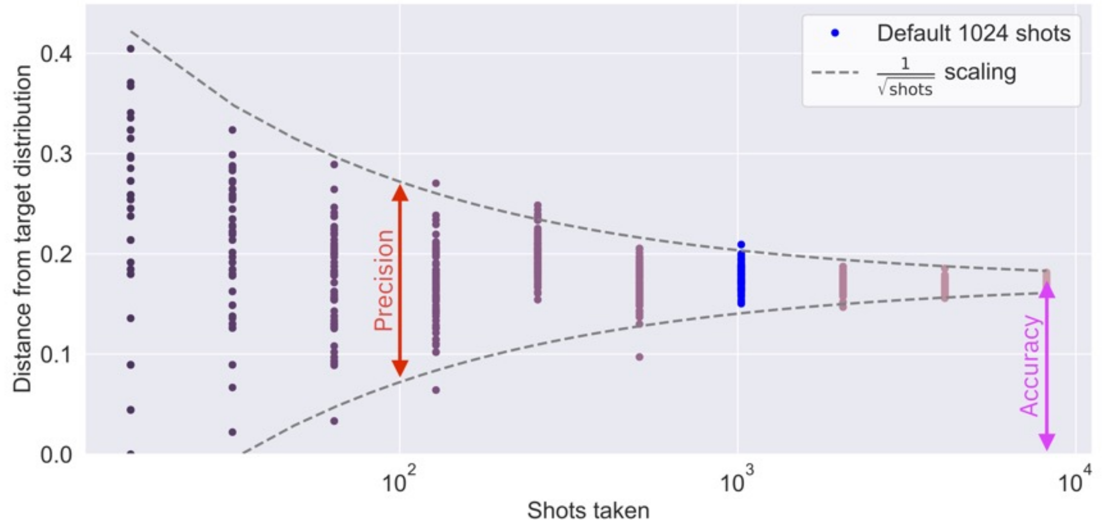


```
qc = QuantumCircuit(1, 1)
qc.h(0)
qc.measure(0, 0)
print(execute(qc, Aer.get_backend('qasm_simulator'), shots=100).result().get_counts())
qc.draw(output='mpl')
```

```
{'1': 47, '0': 53}
```



- “Distance (in terms of Hellinger distance) for a Bell state run on the IBM Quantum Boeblingen system from the theoretical answer as a function of the number of shots taken. For each value of the shots, the experiment is repeated 100 times.”



Source: https://quantum-computing.ibm.com/docs/cloud/backends/configuration (link lost)

Classical Part of VQE

- You need to solve an optimization problem
  - $\min_\theta f(\theta)$, where $f(\theta) = \langle H \rangle_{\psi(\theta)}$
  - Concern: Is it as difficult as the original optimization problem?

- If you optimize $f(\theta)$ directly, it is a black-box optimization
  - It is hard to solve a large-scale problem
  - Nelder-Mead method, Powell method, COBYLA

- By taking advantage of gradients $\nabla f(\theta)$, you can deal with a larger-scale problem
  - Two types of gradients: Numerical gradient, Analytical gradient
  - Gradient-based algorithm: Simultaneous Perturbation Stochastic Approximation optimizer (SPSA), quasi-Newton method, trust region method
    - Mari, et al. 2020. “Estimating the gradient and higher-order derivatives on quantum hardware,” arXiv:2008.06517.

- By taking advantage of some properties of $f(\theta)$ (unitary), you can directly find out local minima w.r.t. specific variables
  - “NFT”: Nakanishi, et al. 2019. “Sequential Minimal Optimization for Quantum-Classical Hybrid Algorithms,” arXiv:1903.12166.
  - “Flaxis”: Watanabe, et al. 2021. “Optimizing Parameterized Quantum Circuits with Free-Axis Selection,” arXiv:2104.14875.
  - “Rotoselect”: Ostaszewski, et al. 2021. “Structure optimization for parameterized quantum circuits”, Quantum 5, p. 391.

Quantum Approximate Optimization Algorithm (QAOA)

- Focus on finding a good approximation solution to combinatorial problems (page 10 [1])
  - Approximation algorithm to max-cut of 3-regular graph (0.6924, with p=1)
  - C.f., SDP-based 0.87856-approximation algorithm [2]
- Ansatz (parameter p; *n* qubits)
  - $|\psi_p(\gamma, \beta)\rangle = e^{-i\beta_p B} e^{-i\gamma_p H} \ldots e^{-i\beta_1 B} e^{-i\gamma_1 H} |+\rangle^{\otimes n}$
  - H: Hamiltonian, $B = \sum_{i=1}{}^{n} X_i$: Mixer
  - $\beta = (\beta_1, \ldots, \beta_p), \gamma = (\gamma_1, \ldots, \gamma_p)$
  - (Can be seen as Trotter decomposition of quantum adiabatic algorithm)
- Expectation value and minimization
  - $F_p(\gamma, \beta) = \langle \psi_p(\gamma, \beta) | H | \psi_p(\gamma, \beta) \rangle$
  - $\gamma^*, \beta^* = \mathrm{argmin}_{\gamma,\beta} F_p(\gamma, \beta)$ gives an approximate ground state $|\psi_p(\gamma^*, \beta^*)\rangle$
    - Apply optimizers as same as VQE does
  - If *p* is large, F_p will be better, but it will be harder to optimize due to more variables. It will be harder to execute the corresponding circuits with higher *p*
- References
  - [1] Farhi et al. 2014. “A Quantum Approximate Optimization Algorithm,” arxiv:1411.4028.
  - [2] Goemans and Williamson. 1995. “Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming.” JACM 42 (6): 1115–45.

Quantum Circuits of QAOA

- Ansatz (parameter p; n qubits)
  - $|\psi_p(\gamma,\beta)\rangle = e^{-i\beta_p B} e^{-i\gamma_p H} \dots e^{-i\beta_1 B} e^{-i\gamma_1 H} |+\rangle^{\otimes n}$
  - H: Hamiltonian, $B = \sum_{i=1}{}^{n} X_i$: Mixer
- Matrix exponential
  - $e^{A+B} = e^A e^B$ if matrices A and B commute, i.e., $AB = BA$
  - (Otherwise, Trotter decomposition is often used)
  - $H = \sum w_i P_i$: P_i, P_j commute because they are tensor products of Pauli Z and I
  - $B = \sum_{i=1}{}^{n} X_i$: X_i, X_j commute because they are tensor products of Pauli X and I
  - $e^{-i\frac{w}{2}Z_j} \rightarrow$ RZ gate
  - $e^{-i\frac{w}{2}Z_j Z_k} \rightarrow$ RZZ gate
  - $e^{-i\frac{w}{2}X_j} \rightarrow$ RX gate

$$RZ(\lambda) = exp(-i\frac{\lambda}{2}Z) = \begin{pmatrix} e^{-i\frac{\lambda}{2}} & 0 \\ 0 & e^{i\frac{\lambda}{2}} \end{pmatrix}$$

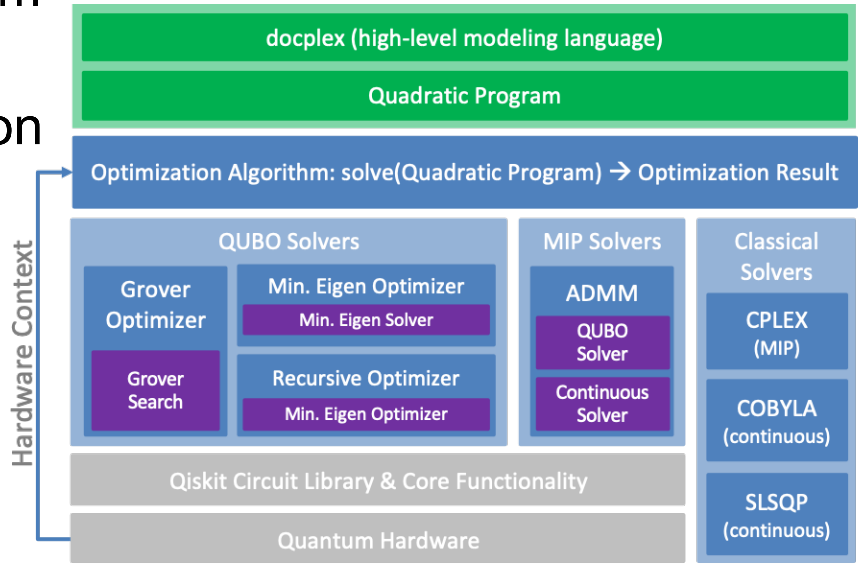
$$RX(\theta) = exp(-i\frac{\theta}{2}X) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

$$R_{ZZ}(\theta) = exp(-i\frac{\theta}{2}Z\otimes Z) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 & 0 & 0 \\ 0 & e^{i\frac{\theta}{2}} & 0 & 0 \\ 0 & 0 & e^{i\frac{\theta}{2}} & 0 \\ 0 & 0 & 0 & e^{-i\frac{\theta}{2}} \end{pmatrix}$$

- Reference
  - Qiskit API reference: https://qiskit.org/documentation/apidoc/circuit_library.html

Introduction of Qiskit Optimization

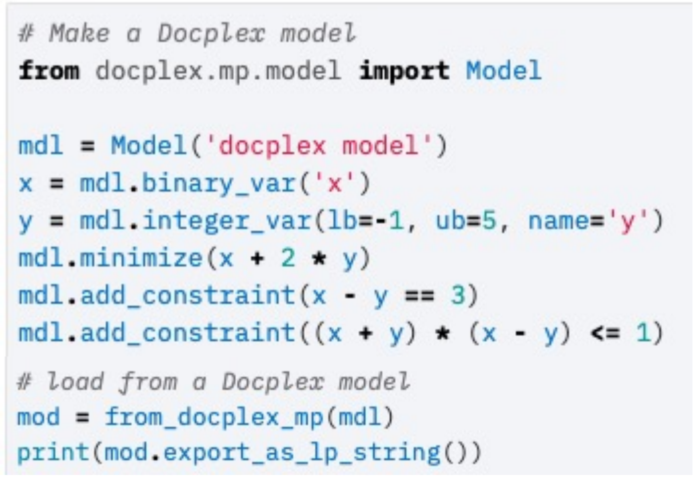
- **Qiskit Optimization** is an open-source framework that covers the whole range from high-level modeling of optimization problems, with automatic conversion of problems to different required representations, to a suite of easy-to-use quantum optimization algorithms that are ready to run on classical simulators, as well as on real quantum devices via Qiskit.
- Qiskit Optimization consists of the following components
  - Problem: `QuadraticProgram` (QP) is the core object to represent an optimization problem
    - Quadratically constrained quadratic programming problem
  - Converters: converts QP into QP in another form
    - E.g., penalizes constraints as part of the objective function
  - Algorithms: solves QP
    - There are both classical solver and quantum solver
  - Applications: pre-defined optimization problems
    - E.g., Knapsack problem and graph partitioning problem



Problem representation

- Qiskit Optimization has `` `QuadraticProgram` `` (QP): the core object to represent an optimization problem
  - We can directly define their own optimization problem with QP
  - We can import and export problems from and to Docplex and Gurobipy
    - Docplex: mathematical modeling library for CPLEX
    - Gurobipy: Python API of Gurobi
  - Note: QP can handle quadratically constrained quadratic programming problem, but we don't have converters to deal with quadratic constraints at this moment

$$
\begin{array}{ll}
\text{minimize} & x^\top Q_0 x + c^\top x \\
\text{subject to} & Ax \leq b \\
 & x^\top Q_i x + a_i^\top x \leq r_i, \quad 1, \dots, i, \dots, q \\
 & l_i \leq x_i \leq u_i, \quad 1, \dots, i, \dots, n,
\end{array}
$$



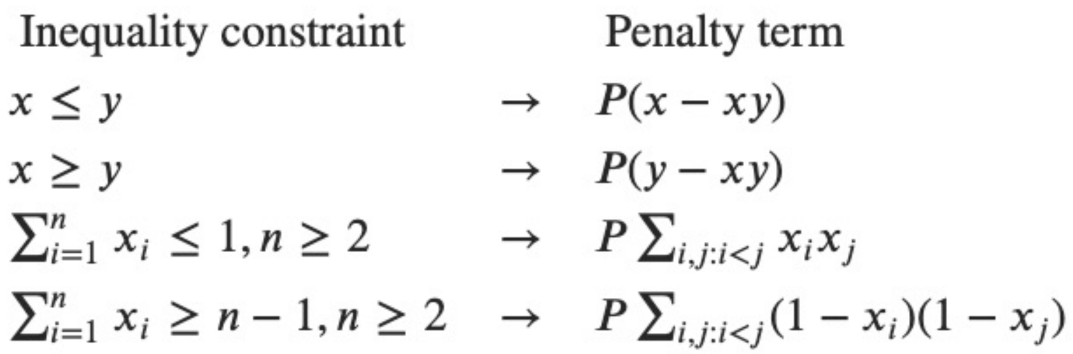
```python
# Make a Docplex model
from docplex.mp.model import Model

mdl = Model('docplex model')
x = mdl.binary_var('x')
y = mdl.integer_var(lb=-1, ub=5, name='y')
mdl.minimize(x + 2 * y)
mdl.add_constraint(x - y == 3)
mdl.add_constraint((x + y) * (x - y) <= 1)
```

```python
# load from a Docplex model
mod = from_docplex_mp(mdl)
print(mod.export_as_lp_string())
```

```
\ This file has been generated by DOcplex
\ ENCODING=ISO-8859-1
\Problem name: docplex model

Minimize
 obj: x + 2 y
Subject To
 c0: x - y = 3
 q0: [ x^2 - y^2 ] <= 1

Bounds
 0 <= x <= 1
 -1 <= y <= 5

Binaries
 x

Generals
 y
End
```

Converters for Quadratic Programs

- We can convert QP to QUBO with converters
  - InequalityToEquality
  - IntegerToBinary
  - LinearEqualityToPenalty
  - MaximizeToMinimize / MinimizeToMaximize
  - QuadraticProgramToQubo
    - a wrapper of a set of converters
  - LinearInequalityToPenalty
    - Some special patterns of inequality constraints that can be converted into penalty terms directly.



Inequality constraint		Penalty term
$x \leq y$	$\rightarrow$	$P(x - xy)$
$x \geq y$	$\rightarrow$	$P(y - xy)$
$\sum_{i=1}^{n} x_i \leq 1, n \geq 2$	$\rightarrow$	$P \sum_{i,j:i<j} x_i x_j$
$\sum_{i=1}^{n} x_i \geq n-1, n \geq 2$	$\rightarrow$	$P \sum_{i,j:i<j} (1 - x_i)(1 - x_j)$

P: penalty coefficient

Quadratic programming problem

Minimize $x^T A x + B x$
Subject to $c_i^T x \leq d_i$
$x = (x_1, \ldots, x_n)$
$l_i \leq x_i \leq u_i$
x_i: binary / integer

$\Downarrow$

QUBO

Minimize $x'^T A' x' + B' x'$
$+ \lambda \Sigma_i \left(c'^T_i x' + s_i - d'_i \right)^2$
Subject to $x' = (x'_1, \ldots, x'_m)$
x'_i: binary

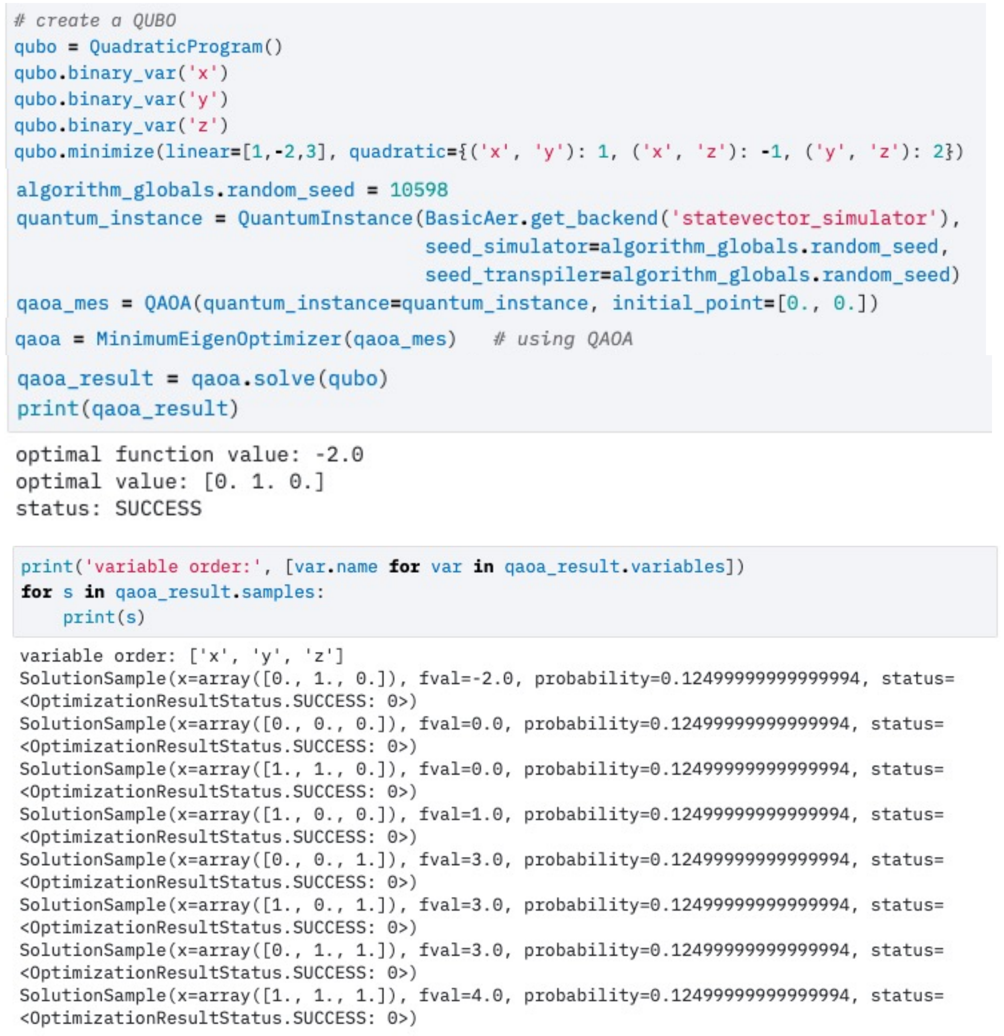
$\Downarrow$

Hamiltonian

$H = \Sigma_i w_i P_i$
P_i: Pauli string (e.g., $ZI = Z \otimes I$)
= tensor product of Pauli matrices

Optimization Algorithms

- OptimizationAlgorithms solves QP
  - MinimumEigenOptimizer: an algorithm that uses the minimum eigensolver of Terra (VQE, QAOA)
  - CplexSolver / GurobiSolver: an algorithm that uses CPLEX / Gurobi
  - RecursiveMinimumEigenOptimizer: Recursive QAOA [1]
  - WarmStartQAOAOptimizer: Warm start QAOA [2]
- Results of VQE and QAOA consist of pairs of a bitstring or a state vector and probability associated with it
  - The algorithms interprets the bitstrings into solutions of the original problem with the converters
- Reference
  - [1] S. Bravyi, et al. (2019), Obstacles to State Preparation and Variational Optimization from Symmetry Protection. arxiv:1910.08980
  - [2] D. Egger, et al. (2020), Warm-starting quantum optimization, arxiv:2009.10095



```
# create a QUBO
qubo = QuadraticProgram()
qubo.binary_var('x')
qubo.binary_var('y')
qubo.binary_var('z')
qubo.minimize(linear=[1,-2,3], quadratic={('x', 'y'): 1, ('x', 'z'): -1, ('y', 'z'): 2})

algorithm_globals.random_seed = 10598
quantum_instance = QuantumInstance(BasicAer.get_backend('statevector_simulator'),
                                   seed_simulator=algorithm_globals.random_seed,
                                   seed_transpiler=algorithm_globals.random_seed)
qaoa_mes = QAOA(quantum_instance=quantum_instance, initial_point=[0., 0.])
qaoa = MinimumEigenOptimizer(qaoa_mes)   # using QAOA
qaoa_result = qaoa.solve(qubo)
print(qaoa_result)
```

```
optimal function value: -2.0
optimal value: [0. 1. 0.]
status: SUCCESS
```

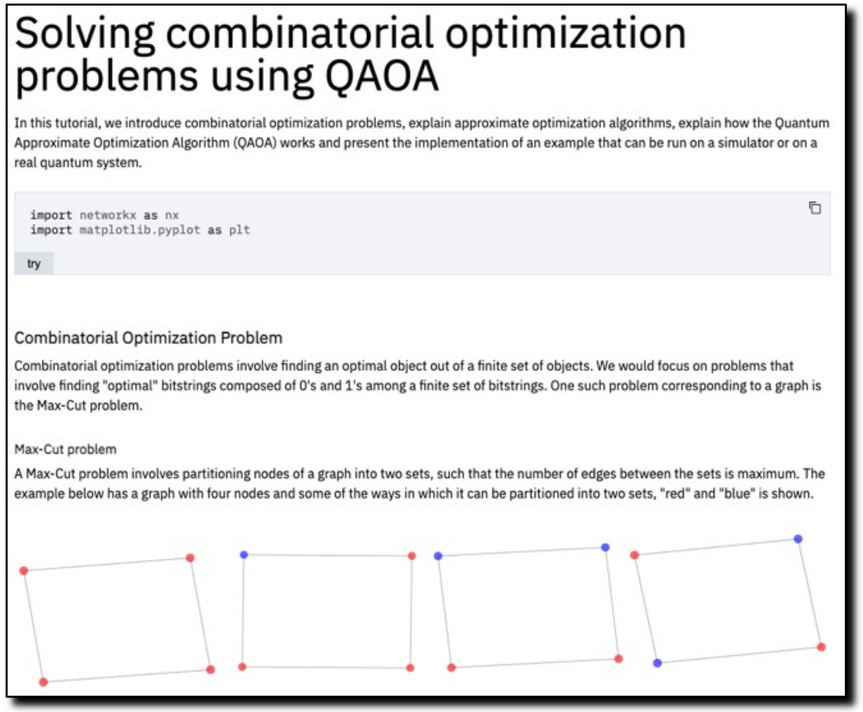
```
print('variable order:', [var.name for var in qaoa_result.variables])
for s in qaoa_result.samples:
    print(s)
```

```
variable order: ['x', 'y', 'z']
SolutionSample(x=array([0., 1., 0.]), fval=-2.0, probability=0.12499999999999994, status=
<OptimizationResultStatus.SUCCESS: 0>)
SolutionSample(x=array([0., 0., 0.]), fval=0.0, probability=0.12499999999999994, status=
<OptimizationResultStatus.SUCCESS: 0>)
SolutionSample(x=array([1., 1., 0.]), fval=0.0, probability=0.12499999999999994, status=
<OptimizationResultStatus.SUCCESS: 0>)
SolutionSample(x=array([1., 0., 0.]), fval=1.0, probability=0.12499999999999994, status=
<OptimizationResultStatus.SUCCESS: 0>)
SolutionSample(x=array([0., 0., 1.]), fval=3.0, probability=0.12499999999999994, status=
<OptimizationResultStatus.SUCCESS: 0>)
SolutionSample(x=array([1., 0., 1.]), fval=3.0, probability=0.12499999999999994, status=
<OptimizationResultStatus.SUCCESS: 0>)
SolutionSample(x=array([0., 1., 1.]), fval=3.0, probability=0.12499999999999994, status=
<OptimizationResultStatus.SUCCESS: 0>)
SolutionSample(x=array([1., 1., 1.]), fval=4.0, probability=0.12499999999999994, status=
<OptimizationResultStatus.SUCCESS: 0>)
```

Resources of Qiskit Optimization

- Documentation
  - https://qiskit.org/documentation/optimization/
- Tutorials
  - https://qiskit.org/documentation/optimization/tutorials/
- GitHub
  - https://github.com/Qiskit/qiskit-optimization
- Qiskit Textbook: Solving combinatorial optimization problems using QAOA
  - https://qiskit.org/textbook/ch-applications/qaoa.html
- YouTube Qiskit channel
  - Qiskit Optimization & Machine Learning Demo Session with Atsushi Matsuo & Anton Dekusar
    - https://www.youtube.com/watch?v=claoY57eVIc
  - Circuits for Optimization Problems - Circuit Sessions with Stefan Woerner
    - https://www.youtube.com/watch?v=LnNaOEqyB7o

Qiskit Textbook



Solving combinatorial optimization problems using QAOA

In this tutorial, we introduce combinatorial optimization problems, explain approximate optimization algorithms, explain how the Quantum Approximate Optimization Algorithm (QAOA) works and present the implementation of an example that can be run on a simulator or on a real quantum system.

```
import networkx as nx
import matplotlib.pyplot as plt
```

try

Combinatorial Optimization Problem

Combinatorial optimization problems involve finding an optimal object out of a finite set of objects. We would focus on problems that involve finding "optimal" bitstrings composed of 0's and 1's among a finite set of bitstrings. One such problem corresponding to a graph is the Max-Cut problem.

Max-Cut problem

A Max-Cut problem involves partitioning nodes of a graph into two sets, such that the number of edges between the sets is maximum. The example below has a graph with four nodes and some of the ways in which it can be partitioned into two sets, "red" and "blue" is shown.