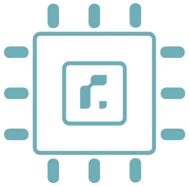
Unsupervised Machine Learning on a Hybrid Quantum Computer

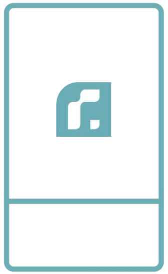
Johannes Otterbach

Bay Area Quantum Computing Meetup - YCombinator
February 1 , 2018

Full Stack **Quantum Computing**



Quantum Processor

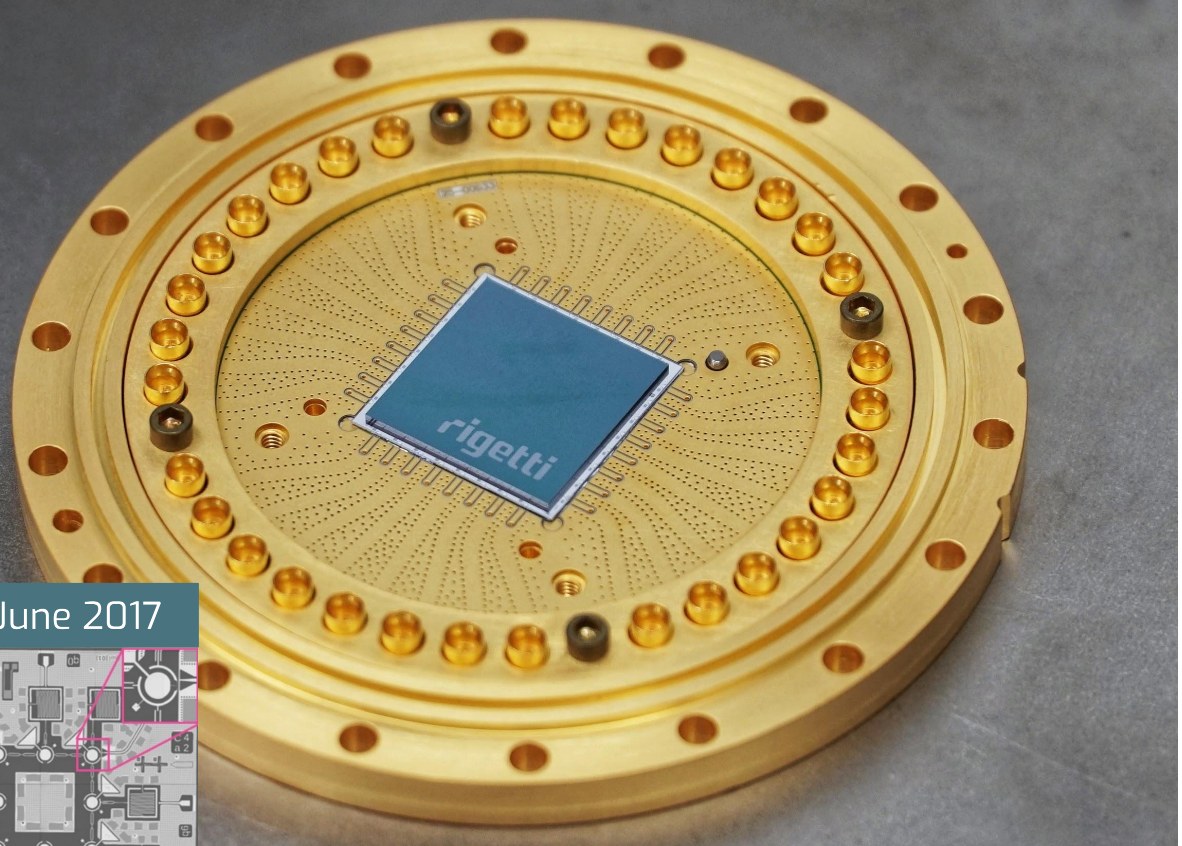


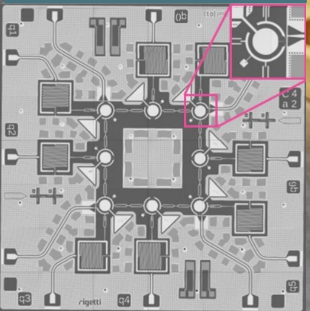
Hardware

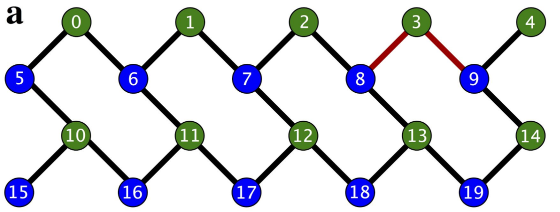
Cloud based Quantum Operating System

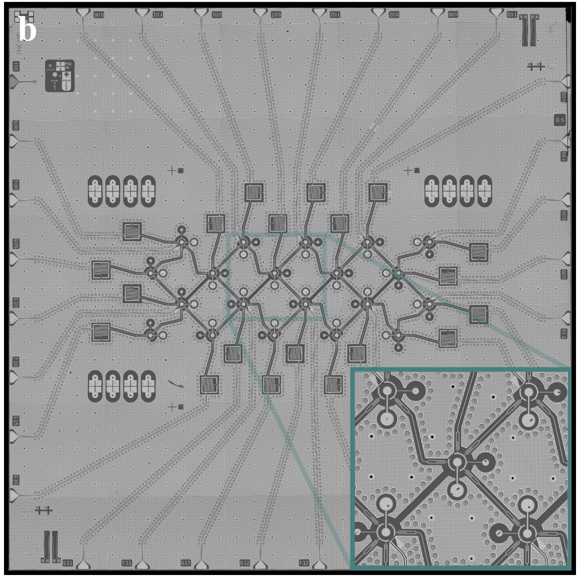
Applications

19Q-Acorn available now









pyquil.readthedocs.io

arXiv:1712.05771

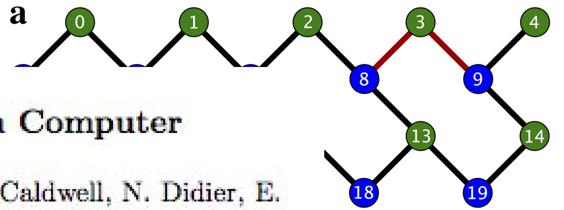
19Q-Acorn available now

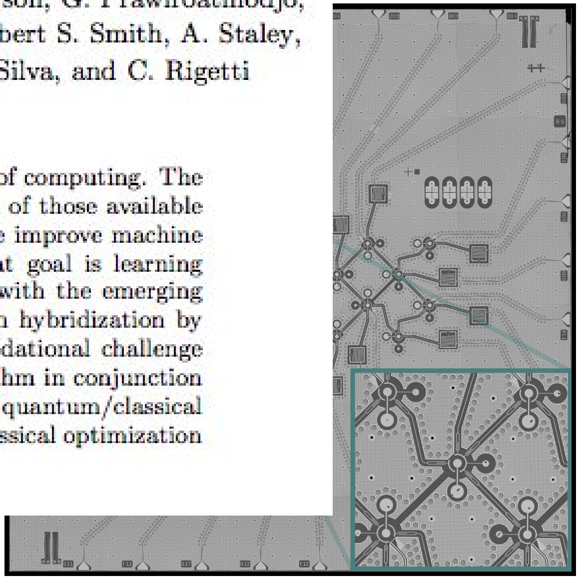
Unsupervised Machine Learning on a Hybrid Quantum Computer

J. S. Otterbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, E. Schuyler Fried, S. Hong, P. Karalekas, C. B. Osborn, A. Papageorge, E. C. Peterson, G. Prawiroatmodjo, N. Rubin, Colm A. Ryan, D. Scarabelli, M. Scheer, E. A. Sete, P. Sivarajah, Robert S. Smith, A. Staley, N. Tezak, W. J. Zeng, A. Hudson, Blake R. Johnson, M. Reagor, M. P. da Silva, and C. Rigetti
Rigetti Computing, Inc., Berkeley, CA

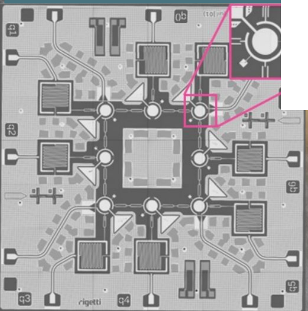
(Dated: December 18, 2017)

Machine learning techniques have led to broad adoption of a statistical model of computing. The statistical distributions natively available on quantum processors are a superset of those available classically. Harnessing this attribute has the potential to accelerate or otherwise improve machine learning relative to purely classical performance. A key challenge toward that goal is learning to hybridize classical computing resources and traditional learning techniques with the emerging capabilities of general purpose quantum processors. Here, we demonstrate such hybridization by training a 19-qubit gate model processor to solve a clustering problem, a foundational challenge in unsupervised learning. We use the quantum approximate optimization algorithm in conjunction with a gradient-free Bayesian optimization to train the quantum machine. This quantum/classical hybrid algorithm shows robustness to realistic noise, and we find evidence that classical optimization can be used to train around both coherent and incoherent imperfections.





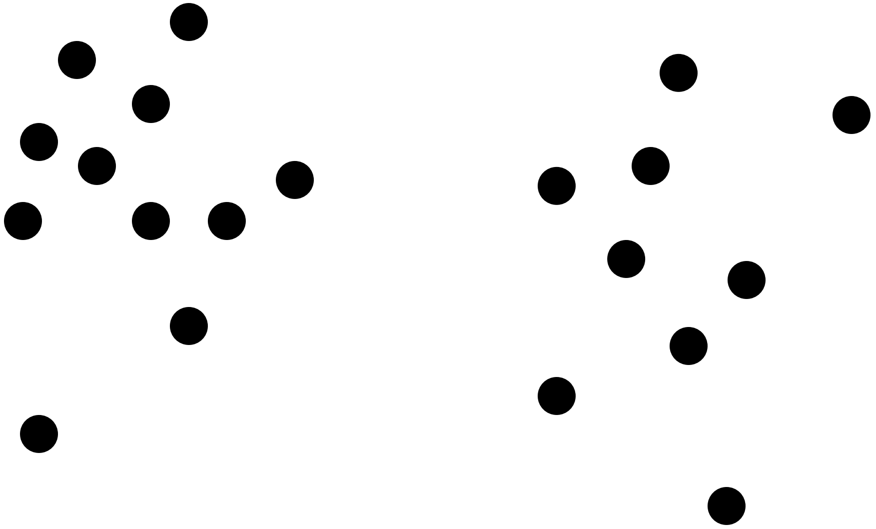
8Q June 2017



pyquil.readthedocs.io

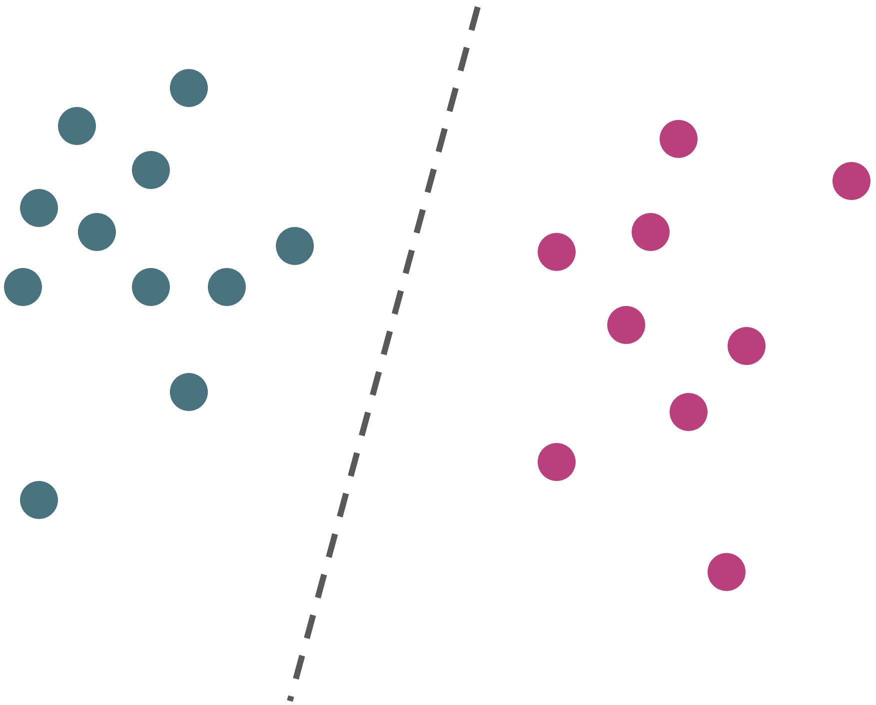
arXiv:1712.05771

Clustering



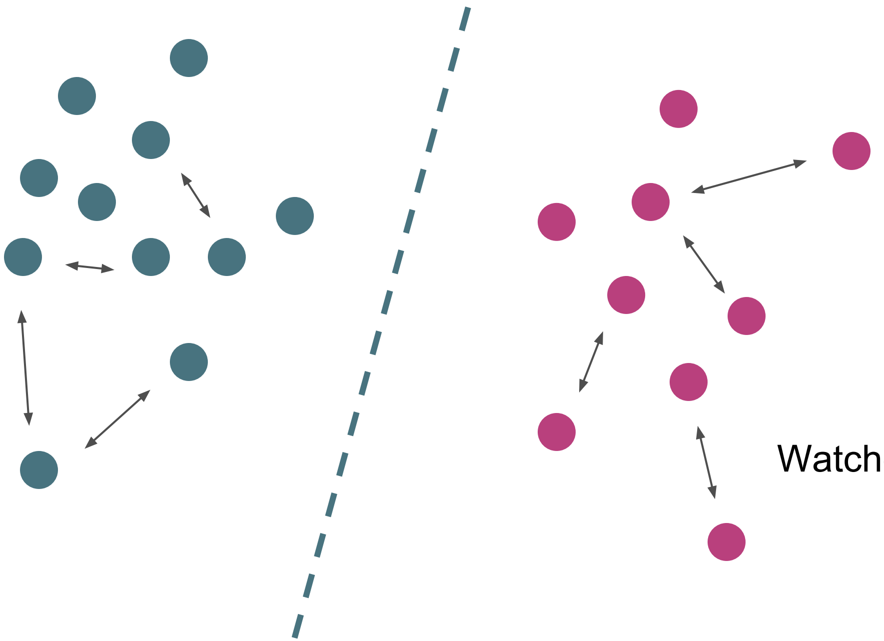
Given an unlabeled set of points

Clustering



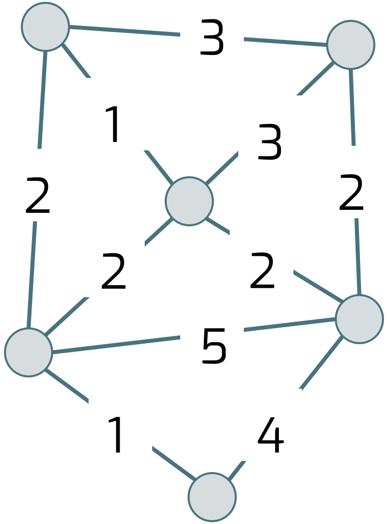
Given an unlabeled set of points, find labels based upon ***similarity*** metric (e.g. Euclidean distance).

Clustering Example - Recommender Systems



1. Recommend action movies
2. Recommend RomComs

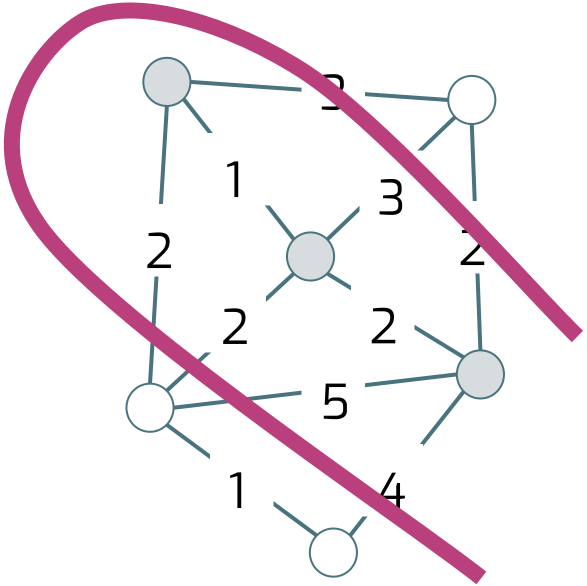
Clustering as MAXCUT



Construct a graph G=(V,E) where the edge weights $w_{i,j}$ are determined by the distance metric. Then, MAXCUT is a clustering algorithm for the original points.

$$\text{MAXCUT} = \max_{\text{cut } S \subset E} \sum_{(i,j) \in S} w_{ij}$$

Clustering as MAXCUT



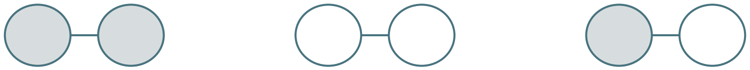
$$\text{MAXCUT} = \max_{\text{cut } S \subset E} \sum_{(i,j) \in S} w_{ij}$$

Construct a graph G=(V,E) where the edge weights $w_{i,j}$ are determined by the distance metric. Then, MAXCUT is a clustering algorithm for the original points.

Clustering transformed into an ***optimization*** problem.

MAXCUT as Energy Functional

"Maximize disagreement on a colored graph"



Score 0 Score 0 Score+1

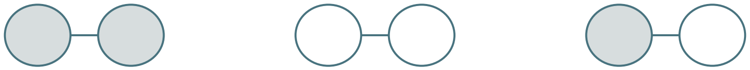
4-node "ring of disagrees"

Score 0 Score 2

Score 2 Score 4 (max)

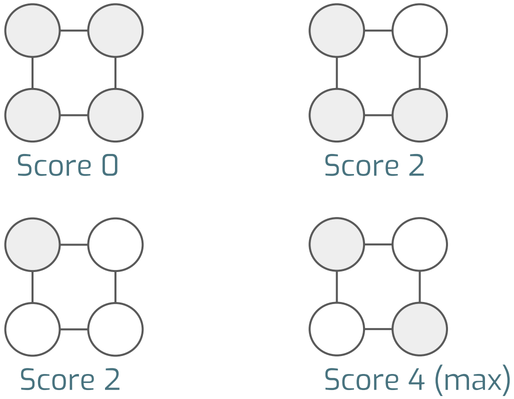
MAXCUT as Energy Functional

“Maximize disagreement on a colored graph”



Score 0 Score 0 Score+1

4-node “ring of disagrees”



Binary variable

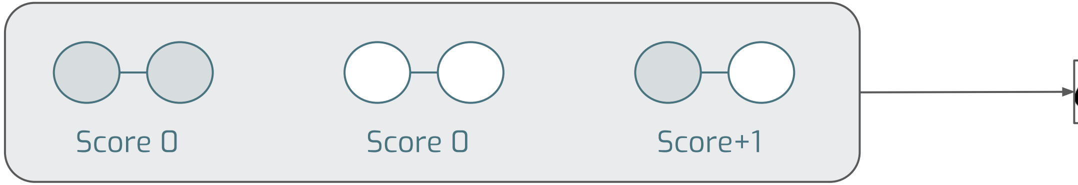
= 1

= 0

$$\sigma^z \in \{0, 1\}$$

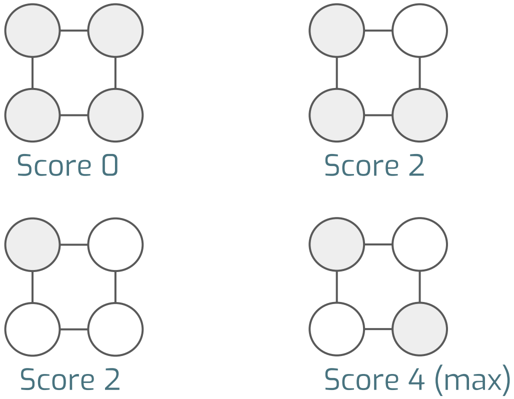
MAXCUT as Energy Functional

“Maximize disagreement on a colored graph”



$$\boxed{\sigma_1^z \sigma_2^z}$$

4-node “ring of disagrees”



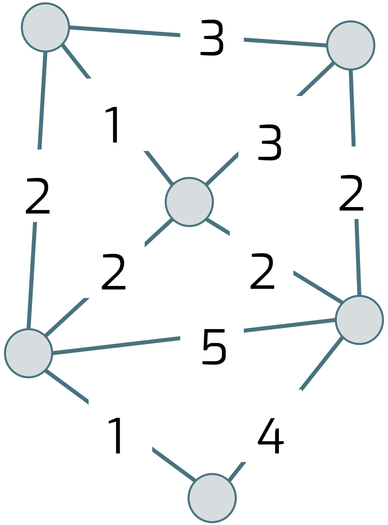
Binary variable

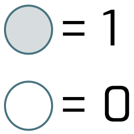
= 1

= 0

$$\sigma^z \in \{0, 1\}$$

MAXCUT as Energy Functional



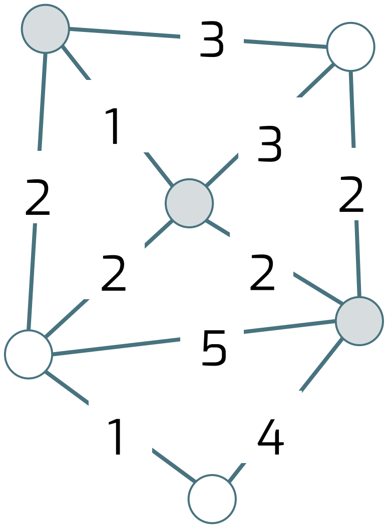


$\sigma^z \in \{0, 1\}$

$$\max_{\text{cut } S \subset E} \sum_{(i,j)\in S} w_{ij}$$

$$= \max_{\sigma_i^z \in \{0,1\}} \sum_{i,j\in V} w_{ij}\sigma_i^z \sigma_j^z$$

MAXCUT as Energy Functional



= 1

= 0

$$\sigma^z \in \{0, 1\}$$

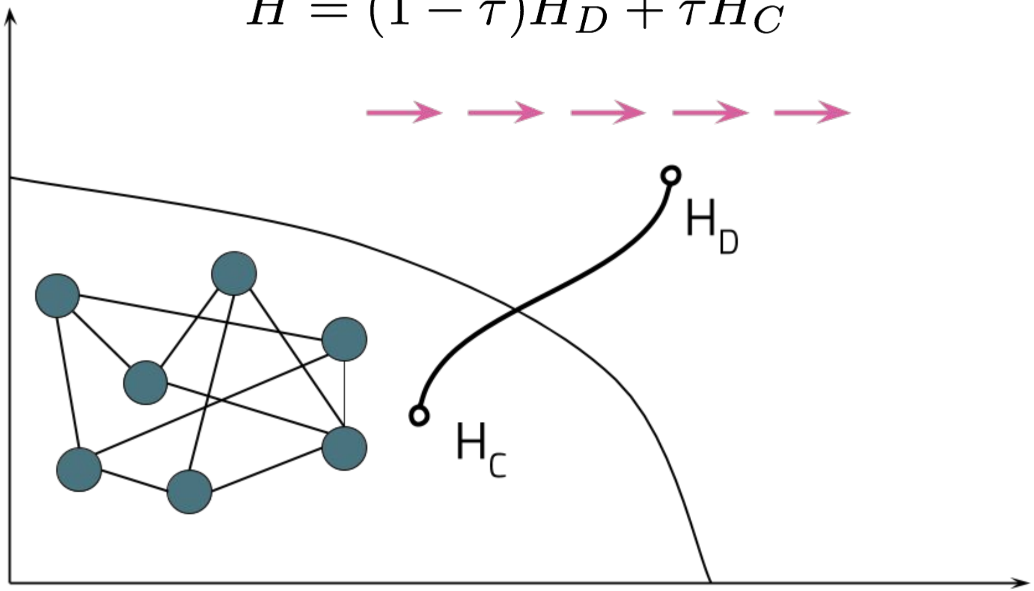
$$\max_{\text{cut } S \subset E} \sum_{(i,j) \in S} w_{ij}$$

$$= \max_{\sigma_i^z \in \{0,1\}} \sum_{i,j \in V} w_{ij} \sigma_i^z \sigma_j^z$$

Find the right bit-string assignment that maximizes the energy

QAOA - Quantum Approximate Optimization Algorithm

$$H = (1-\tau)H_D + \tau H_C$$



IDEA

Start at easy to prepare initial state of energy functional H_D

“Cool” the system until it freezes in the low energy state of H_C

Discretize the Cooling Protocol

- Gate model of Optimization (Farhi, Goldstone, Gutman, arxiv:1411.4028)

$$V(\beta) = \mathrm{e}^{-i\beta H_C}$$

$$U(\gamma) = \mathrm{e}^{-i\gamma H_D}$$

- Angles β, ɣ need not be small
- How to find optimal β, ɣ ?

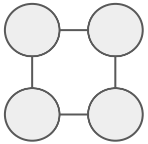
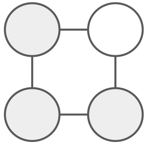
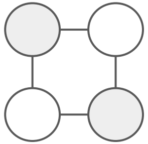
Back to MAXCUT

- Initial state is ground state of H_D : $$| \rightarrow\rangle = H^{\otimes n}|0\rangle$$

- Run the QAOA prescription:

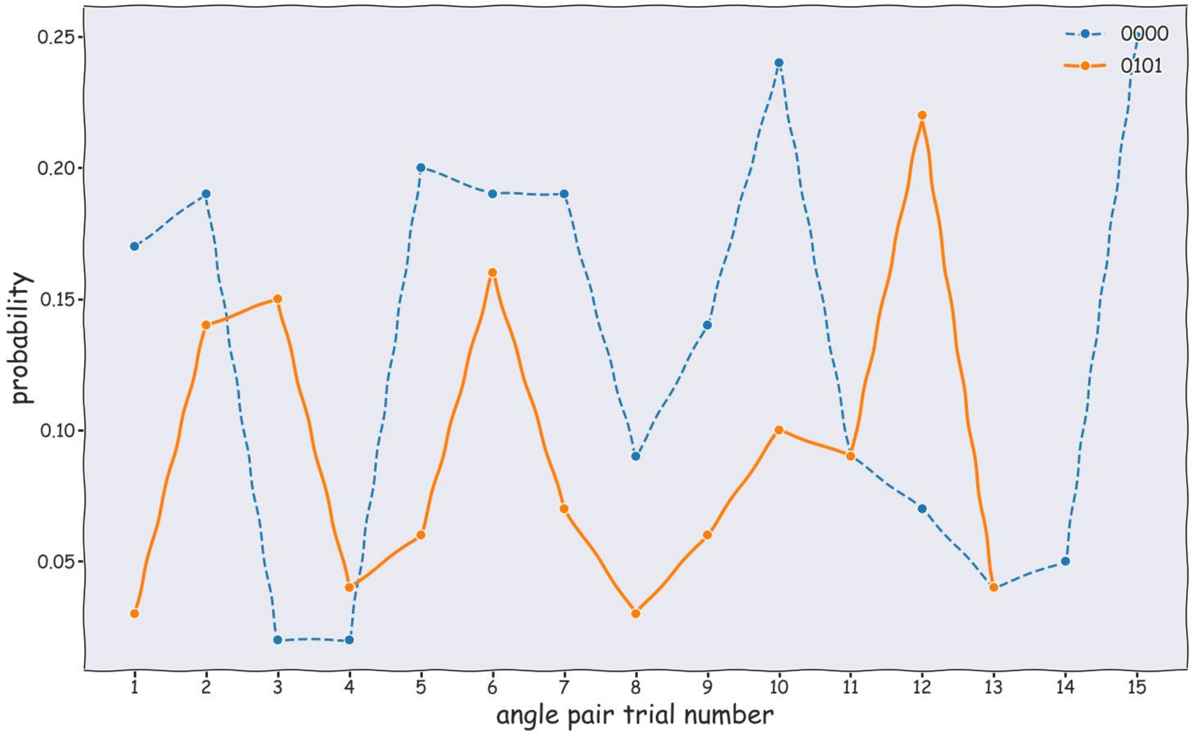
$$|\beta, \gamma\rangle = U(\gamma)V(\beta)H^{\otimes n}|0\rangle$$

- Intuitively: Superposition of bitstring configurations

$| \beta, \gamma\rangle = \sqrt{p_1}$  $+ \sqrt{p_2}$  $+ \ldots + \sqrt{p_{16}}$ 

Effects of the different Angles

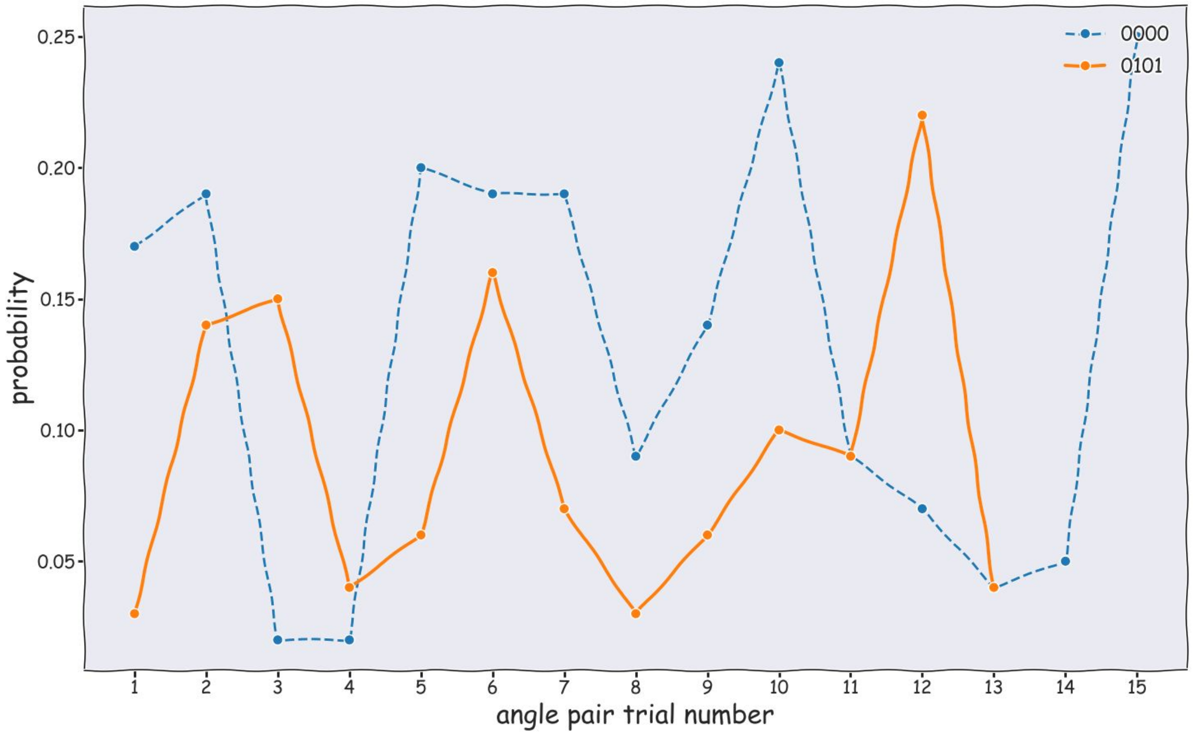
Angles change the probability to sample different bit strings



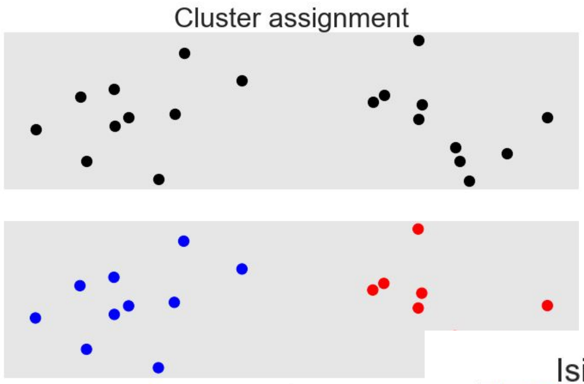
Effects of the different Angles

Angles change the probability to sample different bit strings

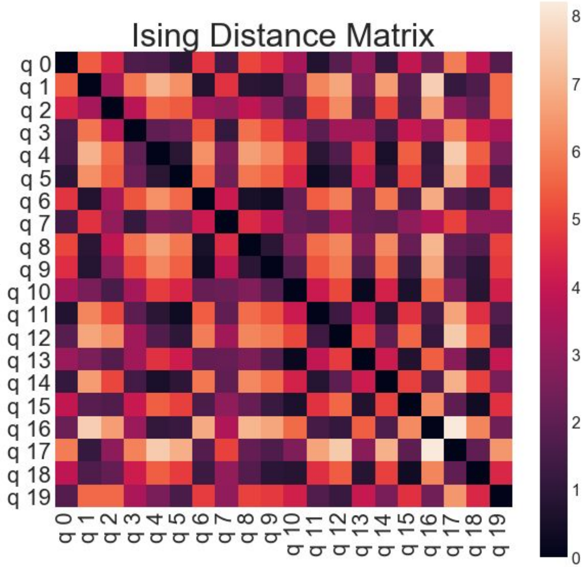
Want to maximize the probability to sample the "correct" bitstring



Clustering Procedure



Euclidean distance



Graph encoding

Hamiltonian

$$H_C = \sum_{ij} w_{ij} \sigma_i^z \sigma_j^z$$

QAOA

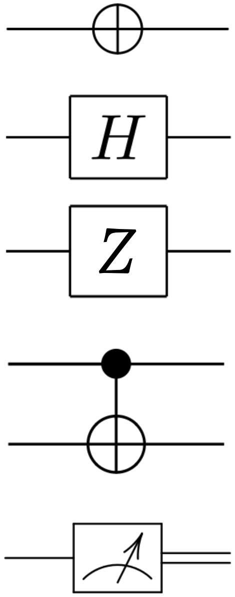
$$|\beta, \gamma\rangle = U(\gamma) V(\beta) H^{\otimes n} |0\rangle$$

Measurement in computational basis

$$[0, 1, 0, 0, 0, 1, \ldots, 1]$$

Forest - Quil

Quantum instructions



X

H

Z

CNOT

MEASURE

Quil [01⟩

Quantum Instruction Language

```
X q
H q
Z q
CNOT p q
MEASURE q [0]
MEASURE p [1]
```

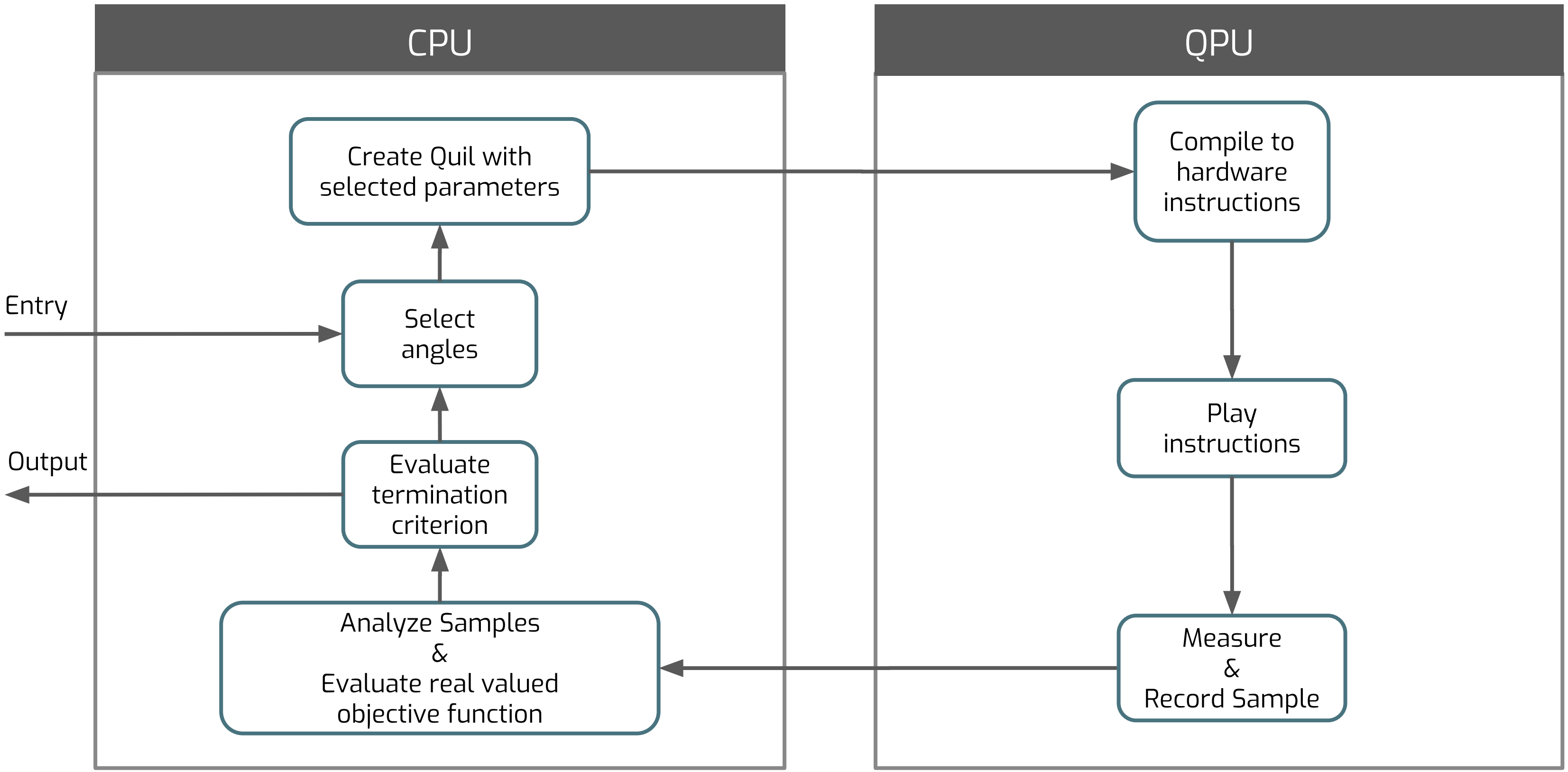
Forest - pyQuil

QAOA

In ***14*** lines of code

```
from pyquil.quil import Program
from pyquil.gates import H
from pyquil.paulis import sI, sX, sZ, exponentiate_commuting_pauli_sum
from pyquil.api import QVMConnection

graph = [(0, 1), (1, 2), (2, 3)]
nodes = range(4)

init_state_prog = sum([H(i) for i in nodes], Program())
h_cost = -0.5 * sum(sI(nodes[0]) - sZ(i) * sZ(j) for i, j in graph)
h_driver = -1. * sum(sX(i) for i in nodes)

def qaoa_ansatz(betas, gammas):
    return sum([exponentiate_commuting_pauli_sum(h_cost)(g) + exponentiate_commuting_pauli_sum(h_driver)(b) \
        for g, b in zip(gammas, betas)], Program())

program = init_state_prog + qaoa_ansatz([0., 0.5], [0.75, 1.])

qvm = QVMConnection()
qvm.run_and_measure(program, qubits=nodes, trials=10)
```

When are we done?



Objective Function

- Loss/Reward Function:

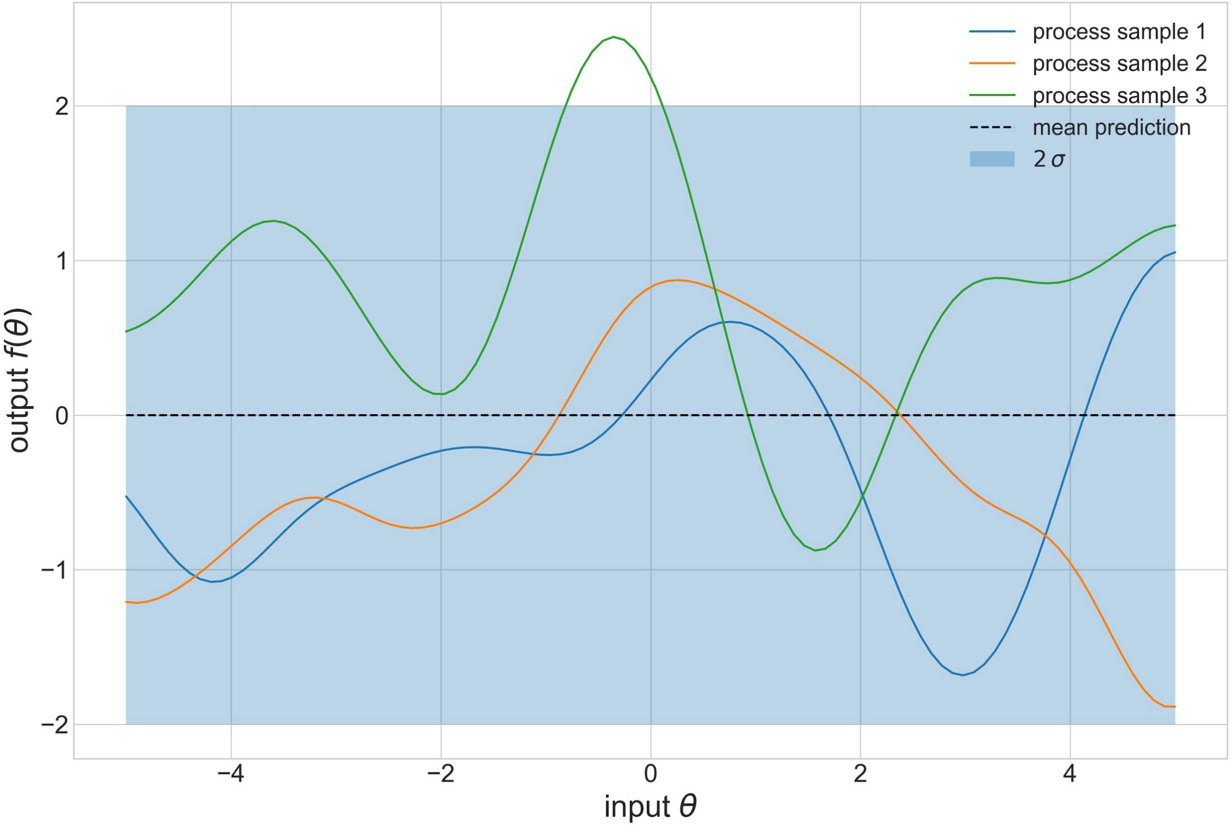
$$c_{\beta,\gamma} : \{0,1\}^n \mapsto \mathbb{R}$$

"quality of a sampled bit-string"

- Find the optimal value of the Reward function:
  - No easy access to gradients, need derivative free methods
  - E.g. Bayesian Methods

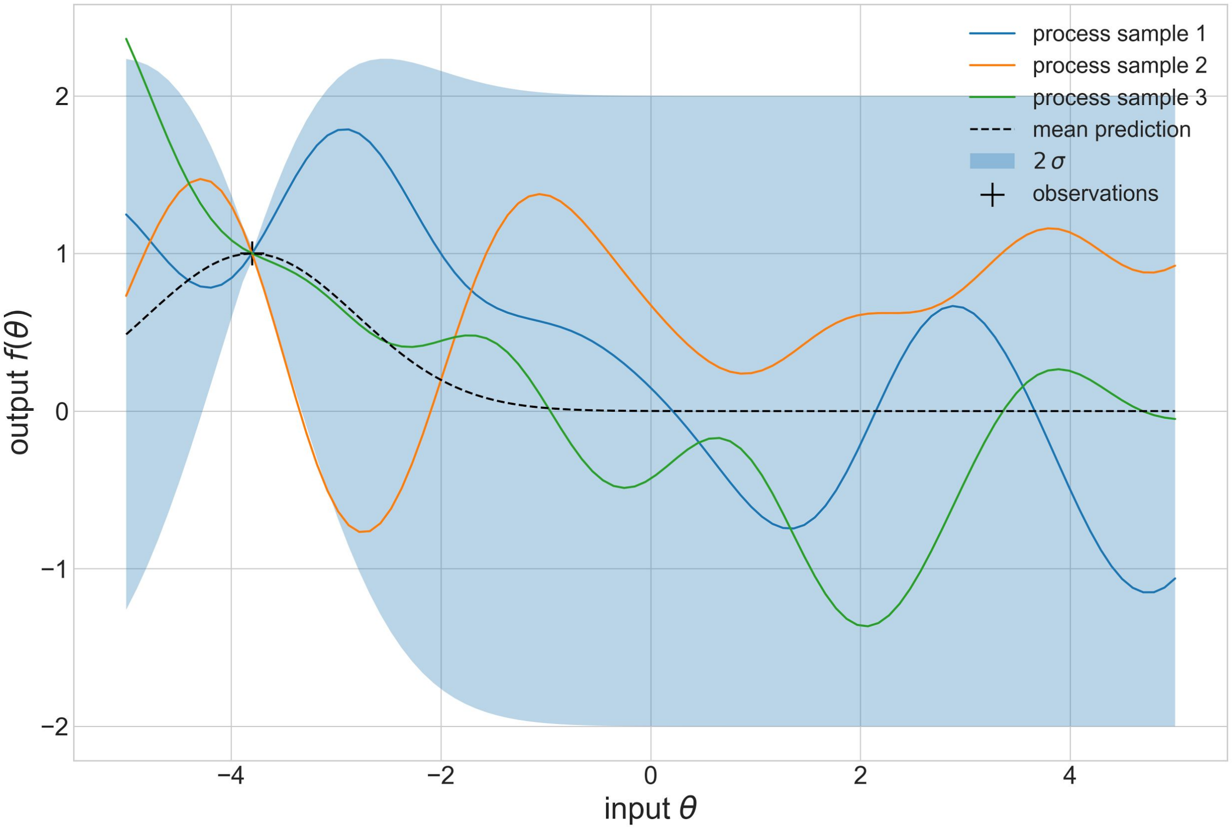
Bayesian Optimization

- Assume objective function is Gaussian



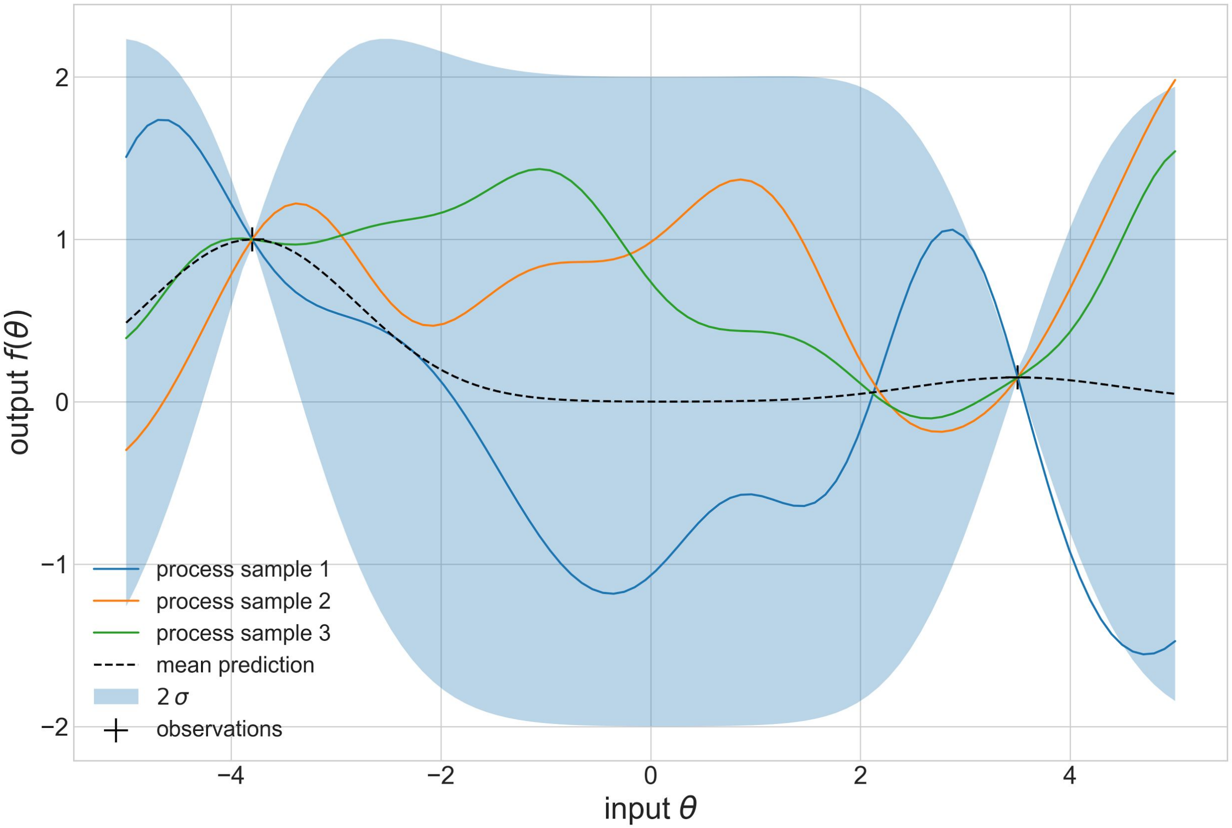
Bayesian Optimization

- Assume objective function is Gaussian
- Measure and update Prior



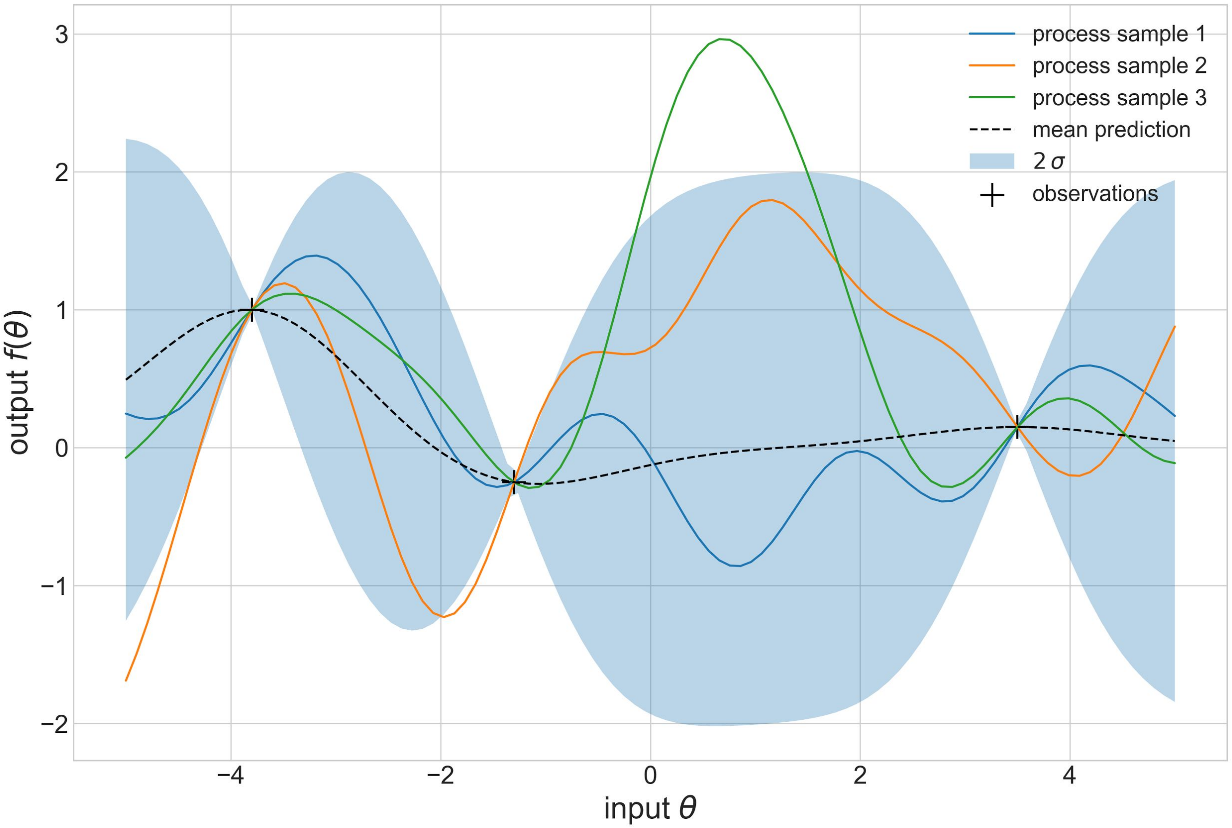
Bayesian Optimization

- Assume objective function is Gaussian
- Measure and update Prior
- Choose next point to measure and update



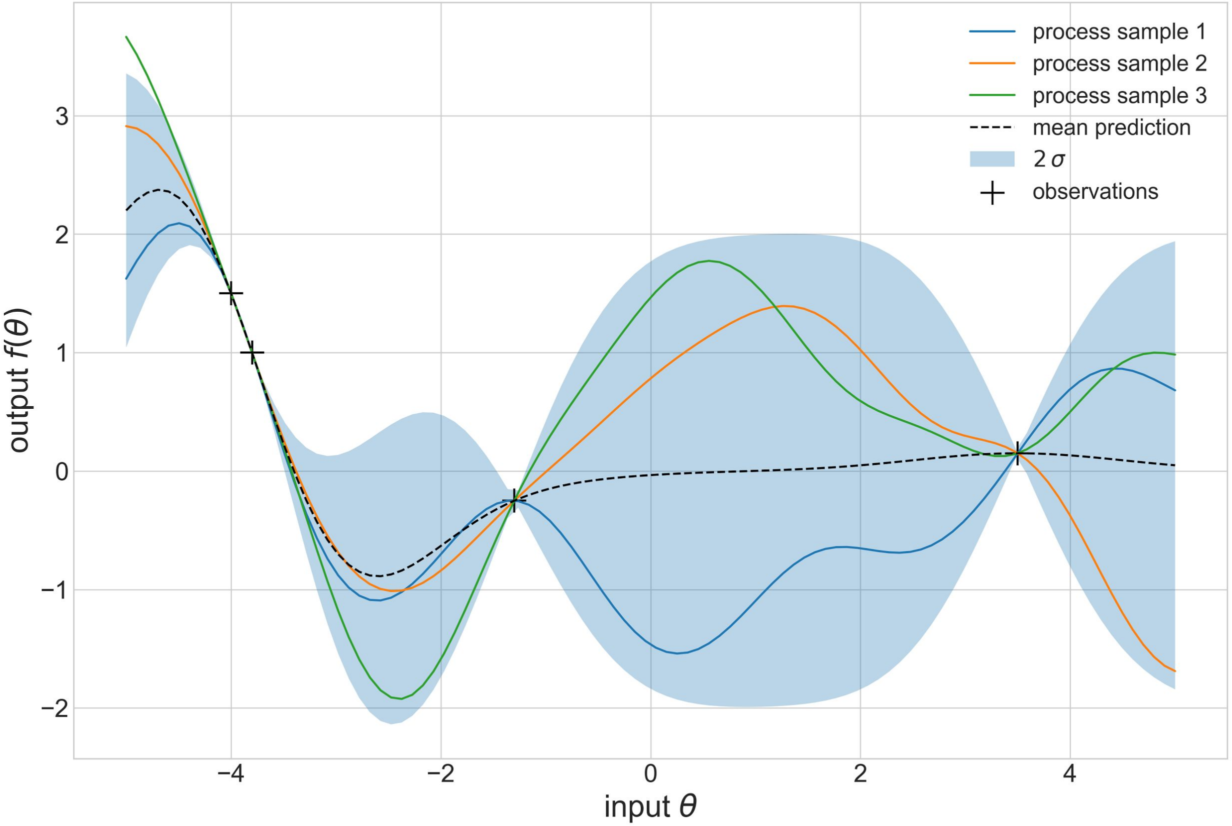
Bayesian Optimization

- Assume objective function is Gaussian
- Measure and update Prior
- Choose next point to measure and update
- Again



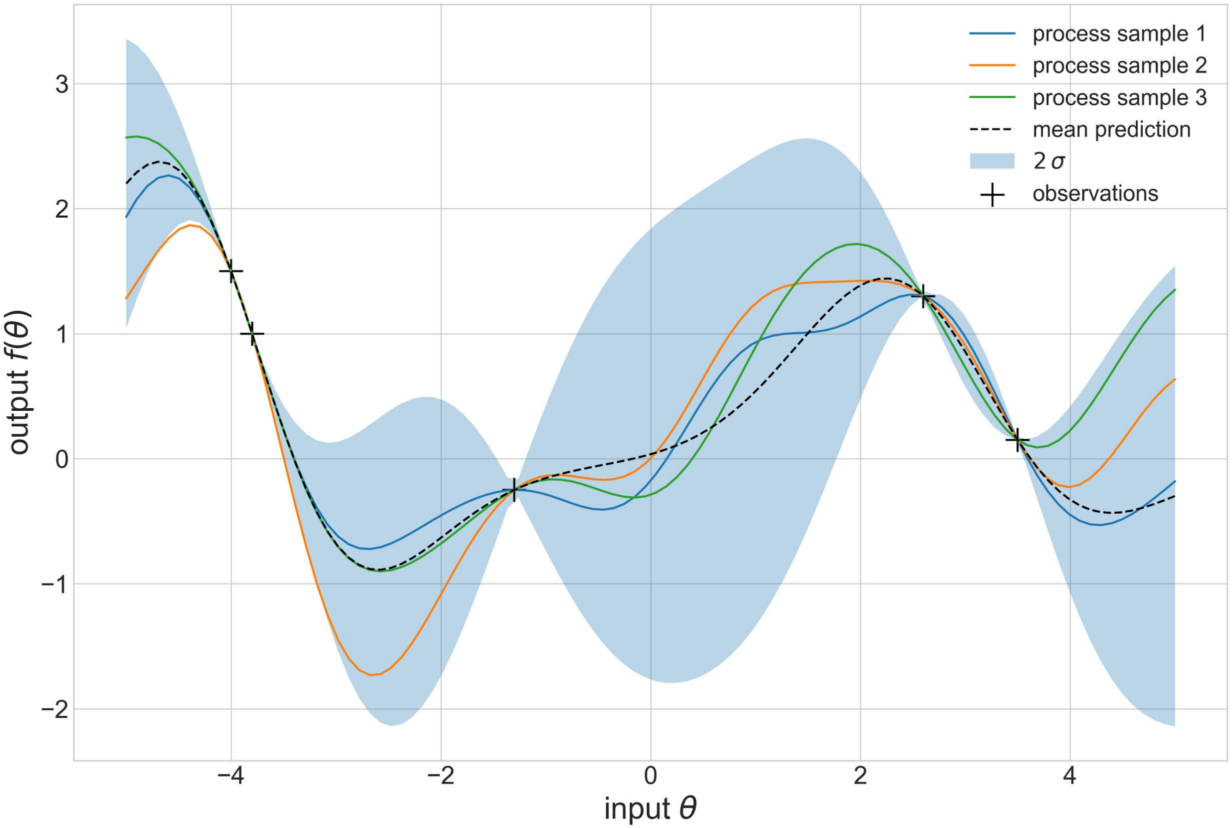
Bayesian Optimization

- Assume objective function is Gaussian
- Measure and update Prior
- Choose next point to measure and update
- Again
- ...



Bayesian Optimization

- Assume objective function is Gaussian
- Measure and update Prior
- Choose next point to measure and update
- Again
- ...
- ...



That's how it looks in practice

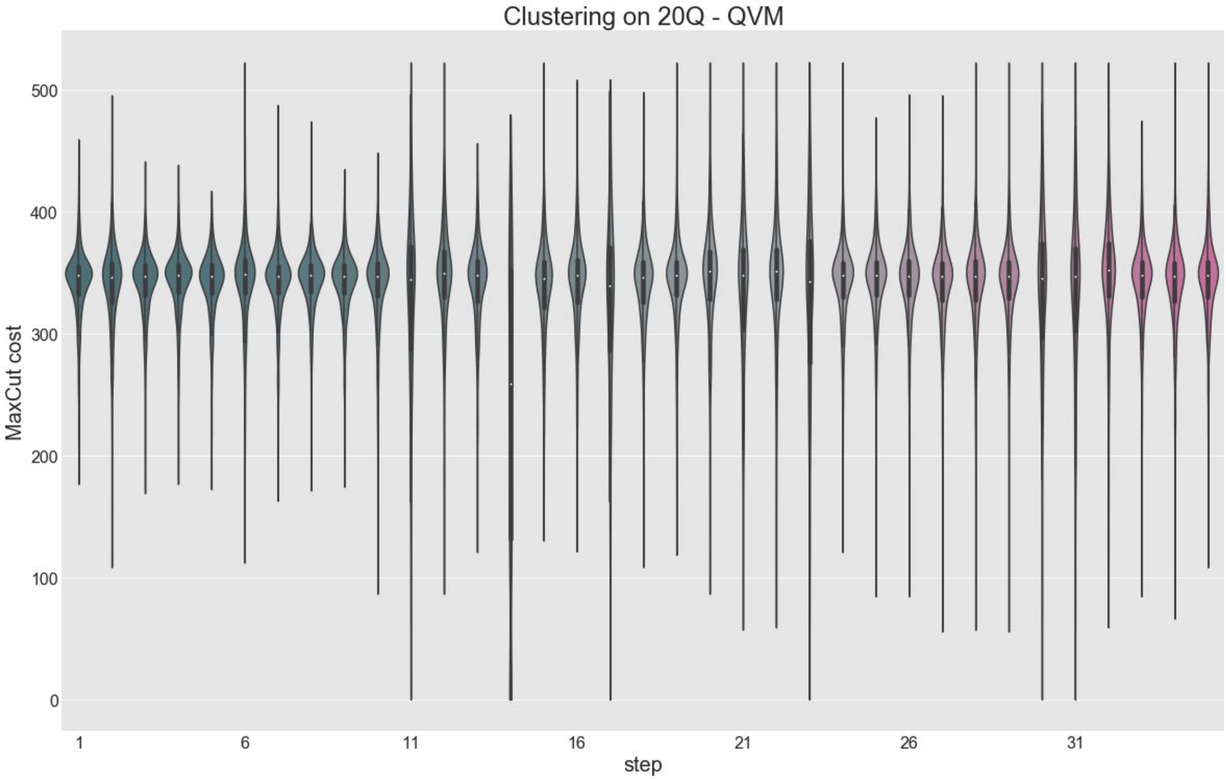
At each step

Sample a new angle pair from the Gaussian prior

Run the Quil program and sample several bitstrings

Evaluate the MAXCUT cost and return maximum as the value

Update Prior and repeat the process

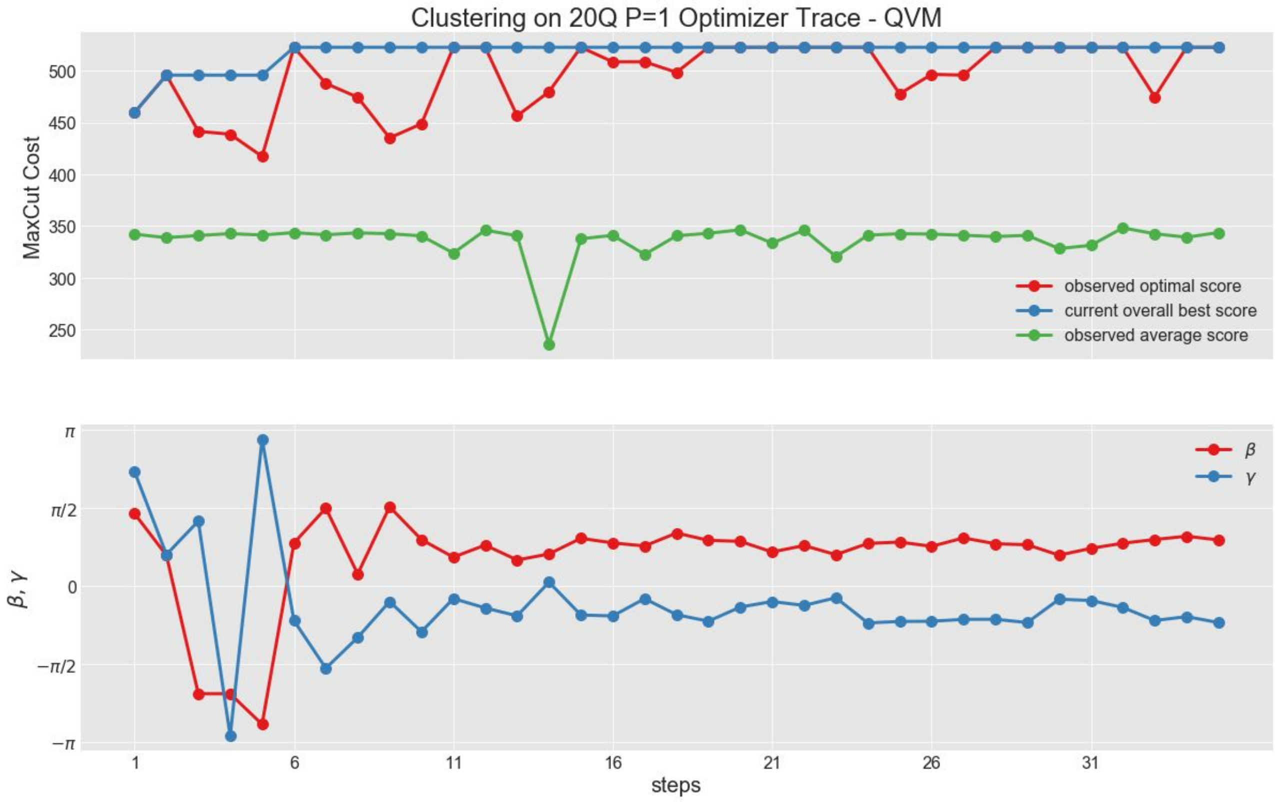


That's how it looks in practice

At each step

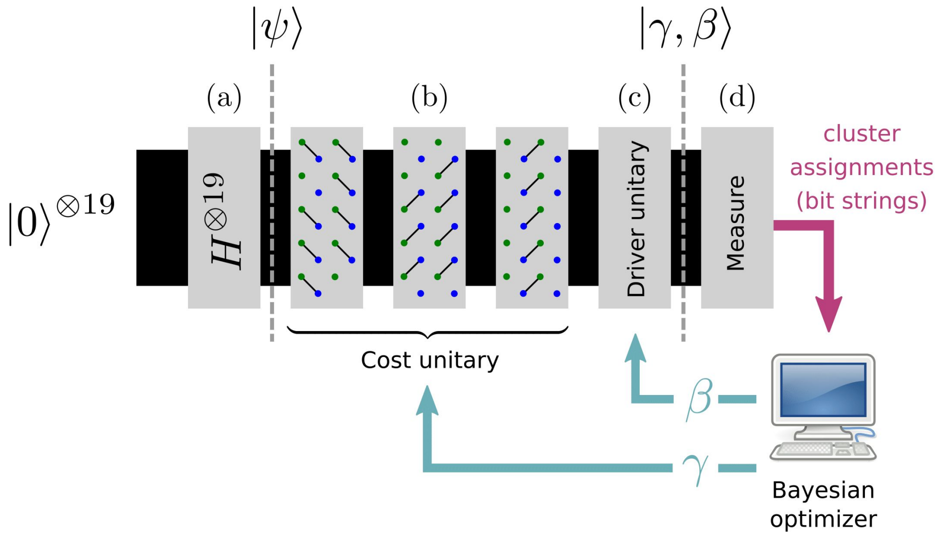
Keep the historic best value

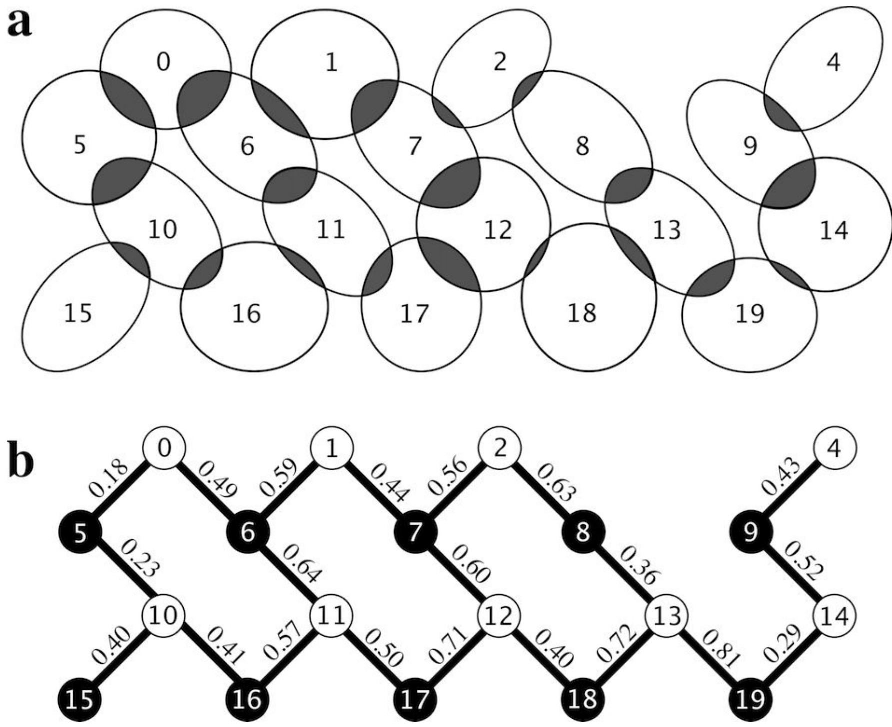
Angles start to converge for large steps numbers



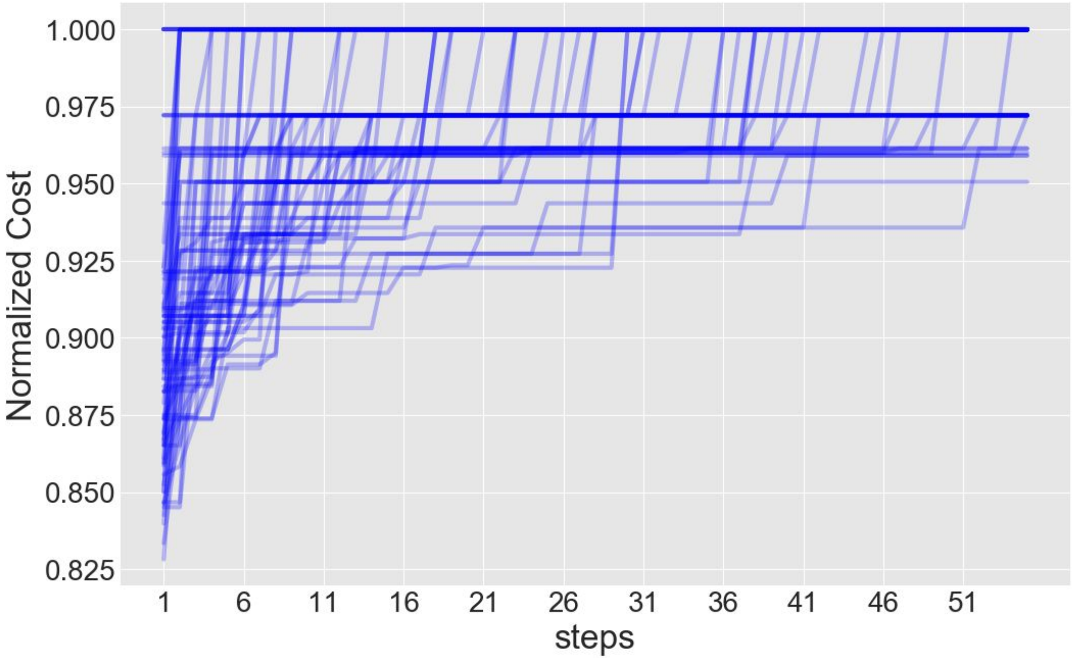
Clustering on a 19-Q Chip

For demonstration purposes chose problem instance to match chip topology



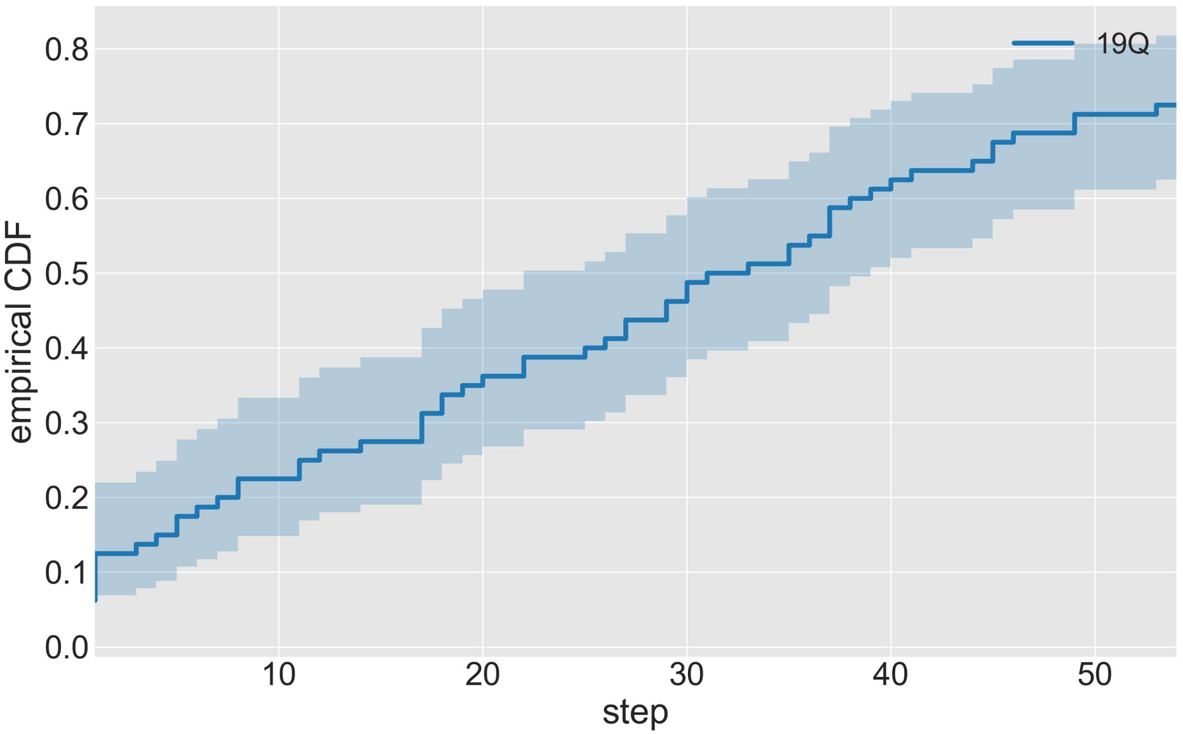


Putting it all together



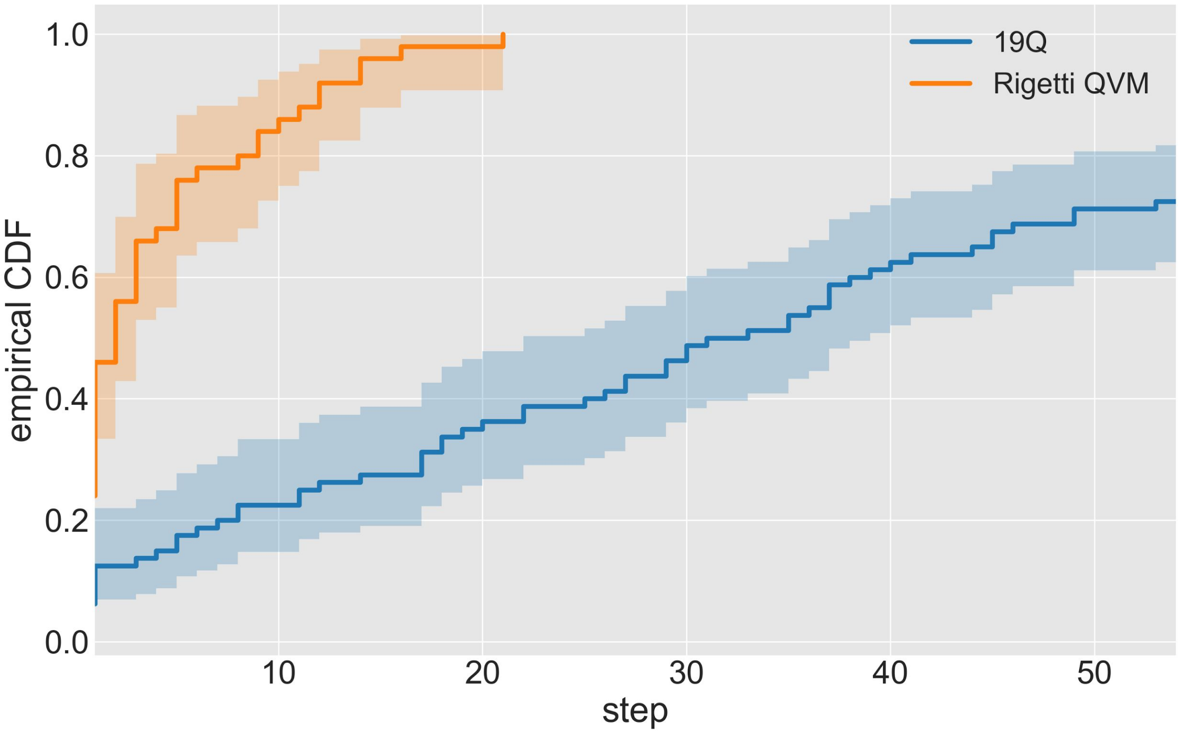
- 83 trial runs on the QPU
- Algorithm finds the optimum most of the time
- Calculate success probability from the traces

Empirical performance



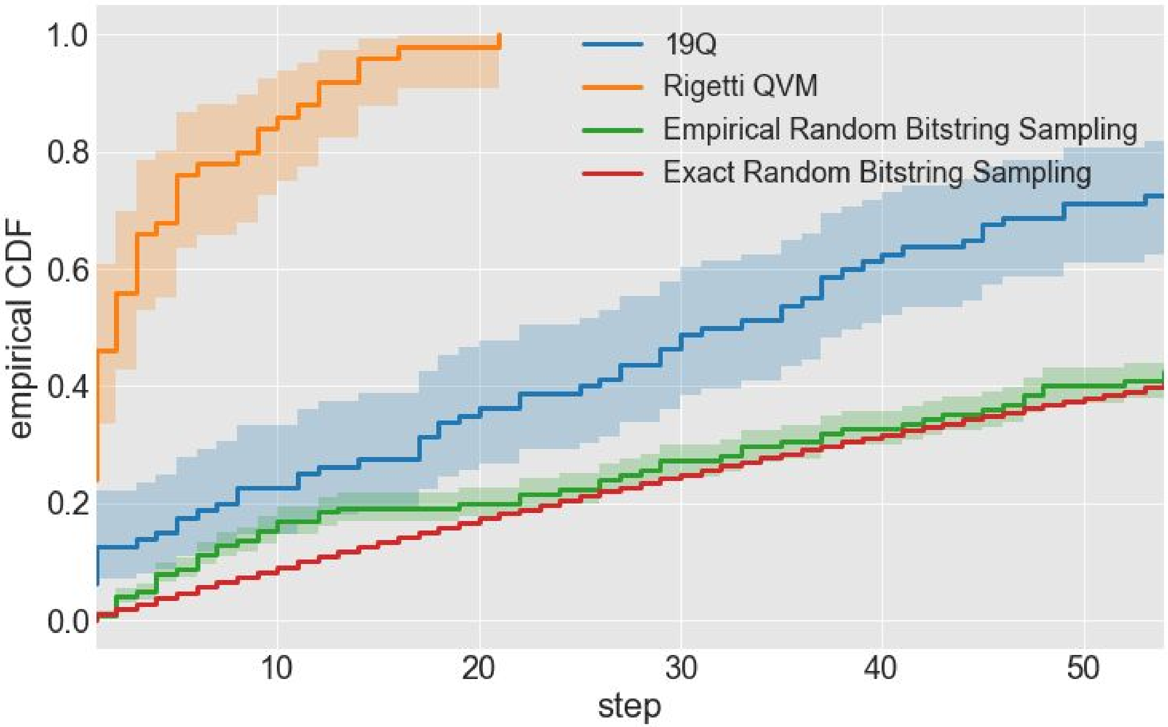
- Success probability monotonically increases with number of steps.

Empirical performance



- Success probability monotonically increases with number of steps.
- Noise in 19Q has a significant impact on performance.

Empirical performance



- Success probability monotonically increases with number of steps.
- Noise in 19Q has a significant impact on performance.
- Approach clearly outperforms random sampling.

Forest

Join our community Slack:

`slack.rigetti.com`

Find us on Github:

`github.com/rigetticomputing`

Sign-Up @ rigetti.com/forest

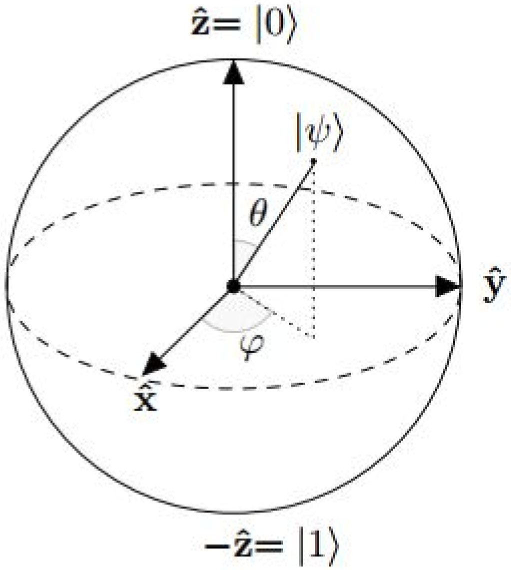
QPU access @ rigetti.com/qpu-request


Thank you
More details in our pre-print arXiv: 1712.05771

Spare slides

Bloch Sphere

Lives on the surface of the Bloch sphere

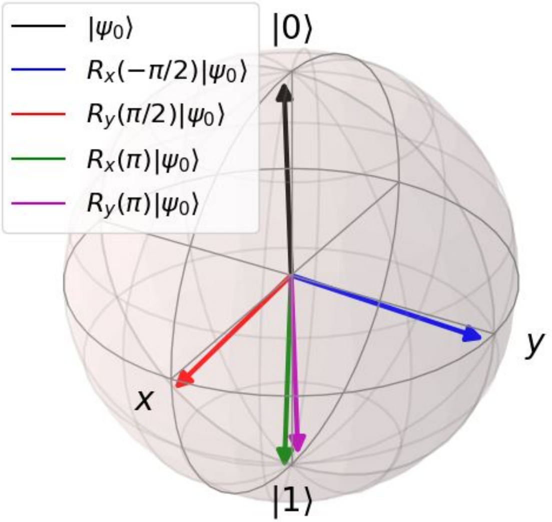


$$|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\varphi}\sin(\theta/2)|1\rangle$$

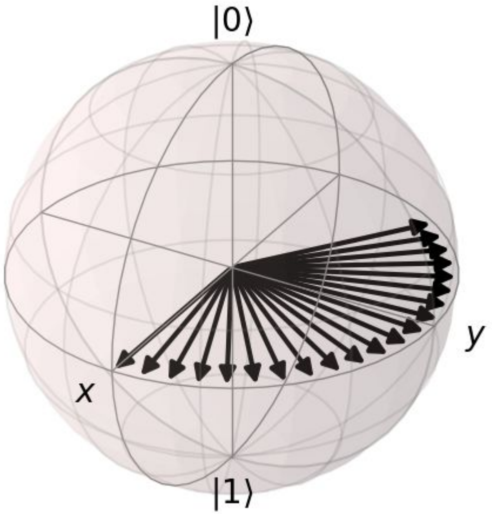
Quantum Control on the Bloch Sphere

"Machine that natively executes unitary operations on quantum systems"

Unitaries are rotations



X and Y rotations by driving



Z rotations by waiting