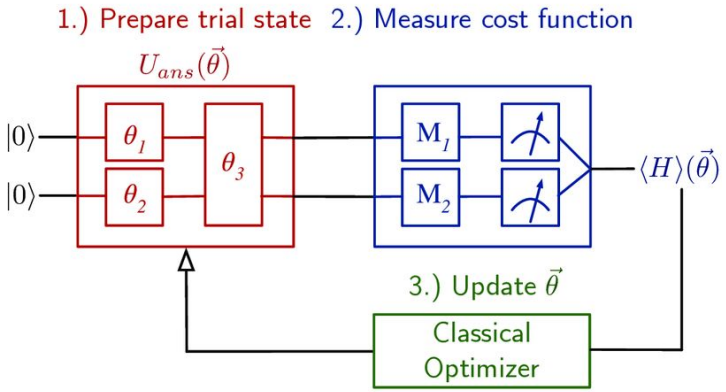
Variational Quantum Eigensolvers using the Hubbard Model of Interacting Fermions

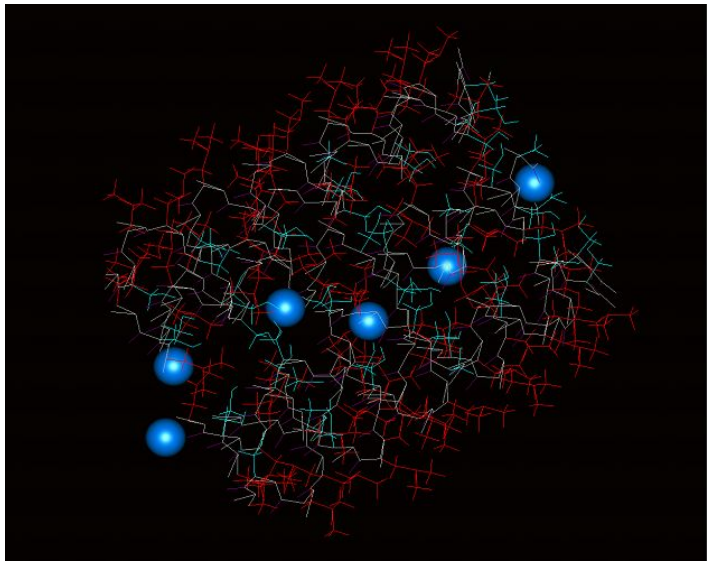
Danny Duke

What is a VQE

- A Variational Quantum Eigensolver is a quantum computing algorithm used to find the lowest possible eigenvalue of a hamiltonian in a given quantum system.
- Hamiltonian describes the total energy of a system.
- Eigenvalues are the possible energies of the hamiltonian given certain parameters.



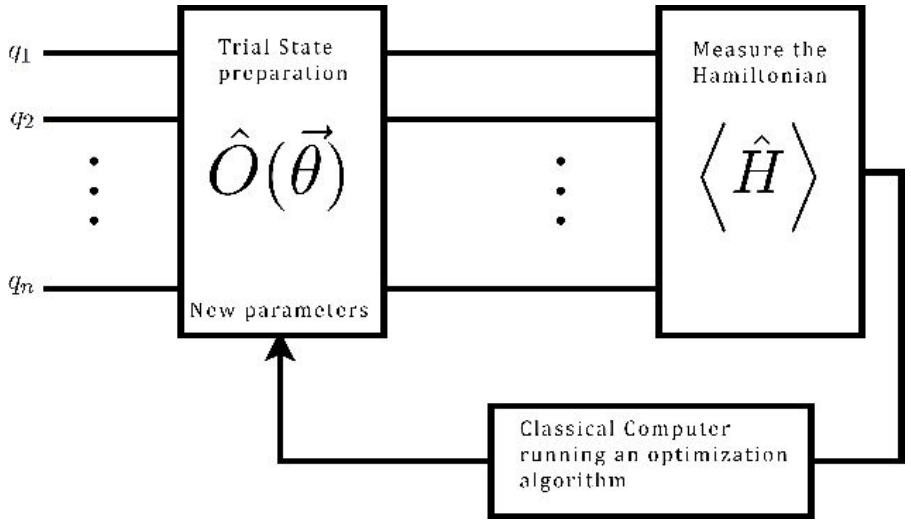
Why VQE



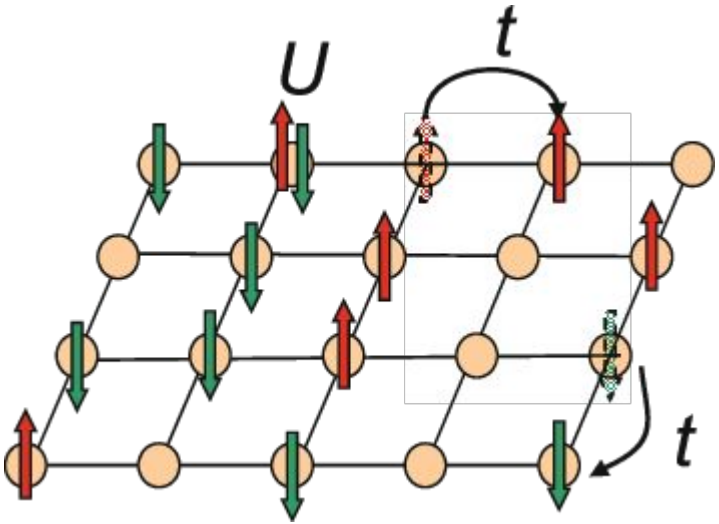
- When you map the properties of a molecule onto a qubit hamiltonian, you can use a VQE to find the lowest possible Eigenvalue of the hamiltonian, which in this case would represent the ground state of the molecule.
- This principle can be especially useful when simulating a molecule that is not viable to physically obtain.
- Once you know the ground state of a molecule is known, it is easy to derive many other properties such as electromagnetic dipoles, polarizability, and other forces. Additionally, properties of the reaction between two molecules can be determined by the difference in their energies.
- Materials science requires the testing of many different materials of uses such as advanced medicine, research into superconductivity, and creating more durable and sustainable materials that meet the ever growing needs of society. Not having to physically obtain and test every material we use is a necessary step to designing the bricks with with we build the future.

How does a VQE Work?

- Starts by applying a applying a parameterized transformation to the hamiltonian which puts it in a state that the energy can be measured.
- Measures the energy based on the parameter theta.
- Classical computer uses this information to prepare a new guess for parameter theta



Hubbard Model of Interacting Fermions



- A lattice of sites that represent electron orbitals.
- Electrons are stored with their spin in the lattice, and are able to move from one orbital to a neighboring orbital,
- Pauli exclusion principle applies.
- t is the hopping energy for the electron
- U is the “Hubbard term” which describes the energy it takes for any two electrons in the same orbital to have opposite spins (Coulomb energy).

Hamiltonian of the System Described by the Hubbard Model

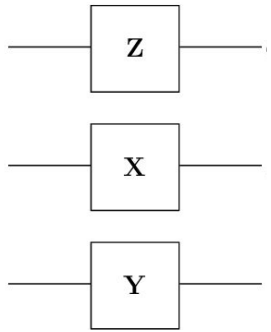
$$H = -t \sum_{\sigma} \sum_{\langle ij \rangle} a^{\dagger}_{j,\sigma} a_{i,\sigma} + U \sum_{i} \hat{n}_{i,\uparrow} \hat{n}_{i,\downarrow},$$

- The sum of the hopping energy and the coulomb energy.
- t is the hopping integral.
- There is one t term for each pair of neighboring sights.
- $a^{\dagger}_{j,\sigma} a_{i,\sigma}$ Is the term for creation and annihilation at neighboring indices i and j.
- U represents the Coulomb energy for each orbital that contains 2 electrons, in a higher energy state due to their necessity to have opposite spins. This is repeated for each index of the lattice

Mapping Hamiltonian to Qubits: Jordan-Wigner

- To interact with the Hamiltonian in the Quantum Computer, it must be mapped onto gates that can be applied to qubits.

$$H = -t \sum_{\sigma} \sum_{\langle ij \rangle} a^{\dagger}_{j,\sigma} a_{i,\sigma} + U \sum_{i} \hat{n}_{i,\uparrow} \hat{n}_{i,\downarrow},$$



Pauli Matrices

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Jordan Wigner Cont'd

- The Jordan Wigner approach to mapping the Hamiltonian works by defining the creation and annihilation operators of the Hubbard model in terms of basic matrices/quantum gates.

$$a_{i+1}^{\dagger} a_i = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$X - iY = \begin{pmatrix} 0 & 0 \\ 2 & 0 \end{pmatrix}$$

$$X + iY = \begin{pmatrix} 0 & 2 \\ 0 & 0 \end{pmatrix}$$

$$a_{i+1}^{\dagger} a_i = \left(\frac{X - iY}{2} \right) \left(\frac{X + iY}{2} \right) = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

Developing the Hamiltonian (non-interacting)

$$a_{i+1}^{\dagger}a_i = (\frac{X_{i+1} - iY_{i+1}}{2})Z_i \cdots Z_0(\frac{X_i + iY_i}{2})Z_{i-1} \cdots Z_0$$

$$= (\frac{X_{i+1} - iY_{i+1}}{2})(\frac{Z_iX_i + iZ_iY_i}{2}) = (\frac{X_{i+1} - iY_{i+1}}{2})(\frac{X_i + iY_i}{2})$$

$$= \frac{1}{4}(iX_{i+1}Y_i + X_{i+1}X_i + Y_{i+1}Y_i - iY_{i+1}X_i)\cdot$$

$$a_i^{\dagger}a_{i+1} = \frac{1}{4}(iX_iY_{i+1} + X_iX_{i+1} + Y_iY_{i+1} - iY_iX_{i+1})\cdot$$

$$a_{i+1}^{\dagger}a_i + a_i^{\dagger}a_{i+1} = \frac{1}{2}(X_iX_{i+1} + Y_iY_{i+1})\cdot$$

Developing the Hamiltonian (Interacting)

$$a_i^\dagger a_i = (\frac{X_i - iY_i}{2})Z_{i-1}\cdots Z_0(\frac{X_i + iY_i}{2})Z_{i-1}\cdots Z_0 = \frac{1}{2}(I - Z_i),$$

$$a_{i\uparrow}^\dagger a_{i\uparrow} a_{i\downarrow}^\dagger a_{i\downarrow} = \frac{1}{4}(I - Z_{i\uparrow} - Z_{i\downarrow} + Z_{i\uparrow}Z_{i\downarrow}).$$

Time Evolution in terms of Basic Gates

$$U(\theta_j) = \exp(i\theta_j(H_0 + H_U)) = \exp(i\theta_{j,xe}H_{xe})\exp(i\theta_{j,xo}H_{xo})\exp(i\theta_{j,ye}H_{ye})\exp(i\theta_{j,yo}H_{yo})\exp(i\theta_{j,z}H_U),$$

$$\exp(i\theta_{j,xe}H_{xe}) = \Pi_\sigma \Pi_{i=2k} \exp(-\frac{t}{2} i\theta_{j,xe} X_{i\sigma}X_{i+1\sigma}),$$
$$\exp(i\theta_{j,ye}H_{ye}) = \Pi_\sigma \Pi_{i=2k} \exp(-\frac{t}{2} i\theta_{j,ye} Y_{i\sigma}Y_{i+1\sigma}),$$
$$\exp(i\theta_{j,z}H_U) = \Pi_i \exp(\frac{U}{4} i\theta_{j,z}(I - Z_{i\uparrow} - Z_{i\downarrow} + Z_{i\uparrow}Z_{i\downarrow}))\cdot$$

Time Evolution in terms of Basic Gates

$$e^{i\theta X_i X_j} = \begin{pmatrix} \cos(\theta) & 0 & 0 & i\sin(\theta) \\ 0 & \cos(\theta) & i\sin(\theta) & 0 \\ 0 & i\sin(\theta) & \cos(\theta) & 0 \\ i\sin(\theta) & 0 & 0 & \cos(\theta) \end{pmatrix}.$$

$$e^{i\theta(I - Z_{i\uparrow} - Z_{i\downarrow} + Z_{i\uparrow} Z_{i\downarrow})} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix}$$

$$e^{i\theta Y_i Y_j} = \begin{pmatrix} \cos(\theta) & 0 & 0 & -i\sin(\theta) \\ 0 & \cos(\theta) & i\sin(\theta) & 0 \\ 0 & i\sin(\theta) & \cos(\theta) & 0 \\ -i\sin(\theta) & 0 & 0 & \cos(\theta) \end{pmatrix}$$

Time Evolution in terms of Basic Gates

$$e^{i\theta X_i X_j} = \mathrm{CX}_i \cdot \mathrm{Rx}_i(\theta) \otimes \mathrm{I}_j \cdot \mathrm{CX}_i,$$

$$e^{i\theta(I - Z_{i\uparrow} - Z_{i\downarrow} + Z_{i\uparrow} Z_{i\downarrow})} = \mathrm{cRz}(\theta),$$

$$e^{i\theta Y_i Y_j} = \mathrm{CX}_i \cdot I_i \otimes H_j \cdot \mathrm{CX}_j \cdot I_i \otimes \mathrm{Rx}_j(-\theta) \cdot \mathrm{CX}_j \cdot I_i \otimes H_j \cdot \mathrm{CX}_i \cdot$$

VQE - Initializing Circuit

- Define molecule you are describing with properties such as number of spin up and down electrons, lattice sites, hopping energy, and Coulomb energy.
- Circuit is in initialized using twice as many qubits as there are lattice sites to accommodate both up and down spins.

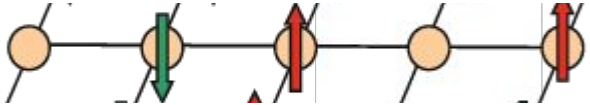
```
import numpy as np
from copy import deepcopy

from qiskit import *

M  = 5    # number of lattice sites
t  = 1.0  # hopping energy
U  = 0.1  # Coulomb energy
Nu = 3    # number of spin-up electrons
Nd = 3    # number of spin-down electrons
```

Creating the Molecule in terms of Qubits

```
# compose quantum circuit

def compose_initial_circuit(circuit, _M, _Nu, _Nd):
    for i in range(_Nu):
        circuit.x(i)
    for i in range(_Nd):
        circuit.x(_M+i)
    return circuit

circ_initial = compose_initial_circuit(deepcopy(circ_empty), M, Nu, Nd)
circ_initial.draw()
```

```
q_0: |0>─┤ X ├
q_1: |0>─┤ X ├
q_2: |0>─┤ X ├
q_3: |0>──────
q_4: |0>──────
q_5: |0>─┤ X ├
q_6: |0>─┤ X ├
q_7: |0>─┤ X ├
q_8: |0>──────
q_9: |0>──────
```

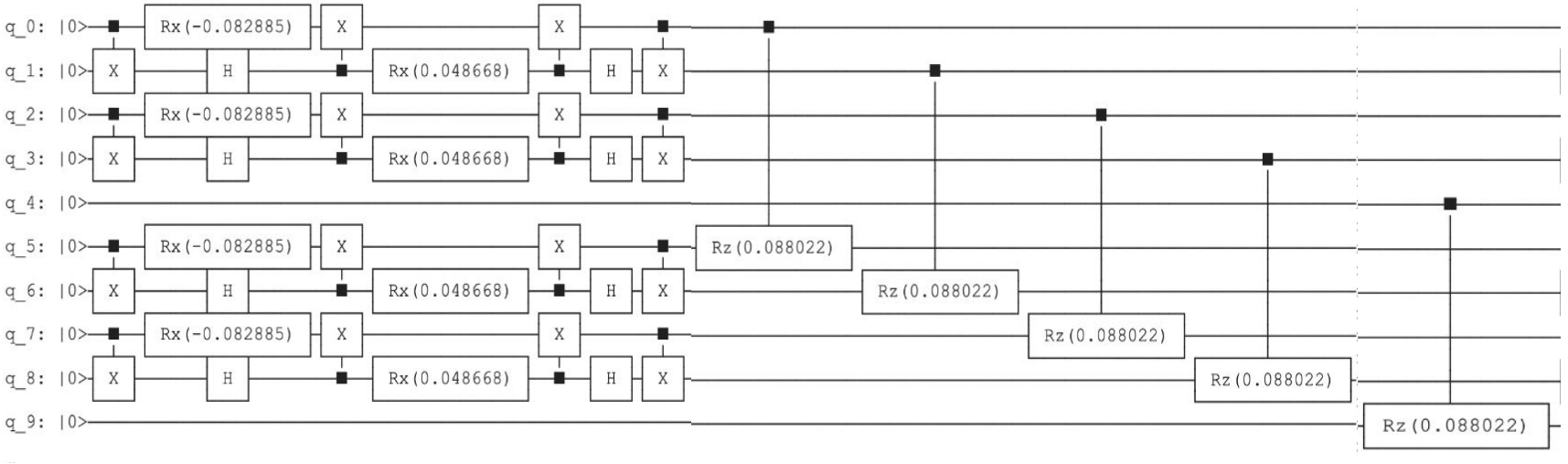
- Linear Hubbard model
- X gates to initialize electrons in at different indices
- Top half of Qubits represent spin up, bottom half represent spin down.



VQE - Creating Ansatz Function

```
def compose_ansatz_circuit(circuit, _M, _S, _theta):
    # scale theta based on t and U
    _theta[:, :4] = -t/2.0*_theta[:, :4]
    _theta[:, 4] = U*_theta[:, 4]
    # loop over S Trotter-Suzuki steps
    for j in range(_S):
        # XX and YY interactions: loop over even and odd terms
        for parity in [0]:
            # loop over orbitals
            for i in range(_M-1):
                if i%2 == parity:
                    # loop over spin up and down qubits
                    for spin_index in [0, _M]:
                        circuit.cx(spin_index+i, spin_index+i+1)
                        circuit.rx(_theta[j, 2*parity+0], spin_index+i)    # theta_j,xe(o)
                        circuit.h(spin_index+i+1)
                        circuit.cx(spin_index+i+1, spin_index+i)
                        circuit.rx(-_theta[j, 2*parity+1], spin_index+i+1) # theta_j,ye(o)
                        circuit.cx(spin_index+i+1, spin_index+i)
                        circuit.h(spin_index+i+1)
                        circuit.cx(spin_index+i, spin_index+i+1)
        # barrier
        circuit.barrier()
        # ZZ interactions
        # loop over orbitals
        for i in range(_M):
            circuit.crz(_theta[j, 4], i, _M+i)
    return circuit

circ_ansatz = compose_ansatz_circuit(deepcopy(circ_empty), M, S, theta)
circ_ansatz.draw()
```

- Quantum aspect of the algorithm.
- Implement the series of gates and rotations necessary to mimic the hamiltonian given an input parameter θ.
- Gates are determined by time evolution.


q_0: |0>
q_1: |0>
q_2: |0>
q_3: |0>
q_4: |0>
q_5: |0>
q_6: |0>
q_7: |0>
q_8: |0>
q_9: |0>
Rx(-0.082885)
X
H
Rx(0.048668)
Rz(0.088022)

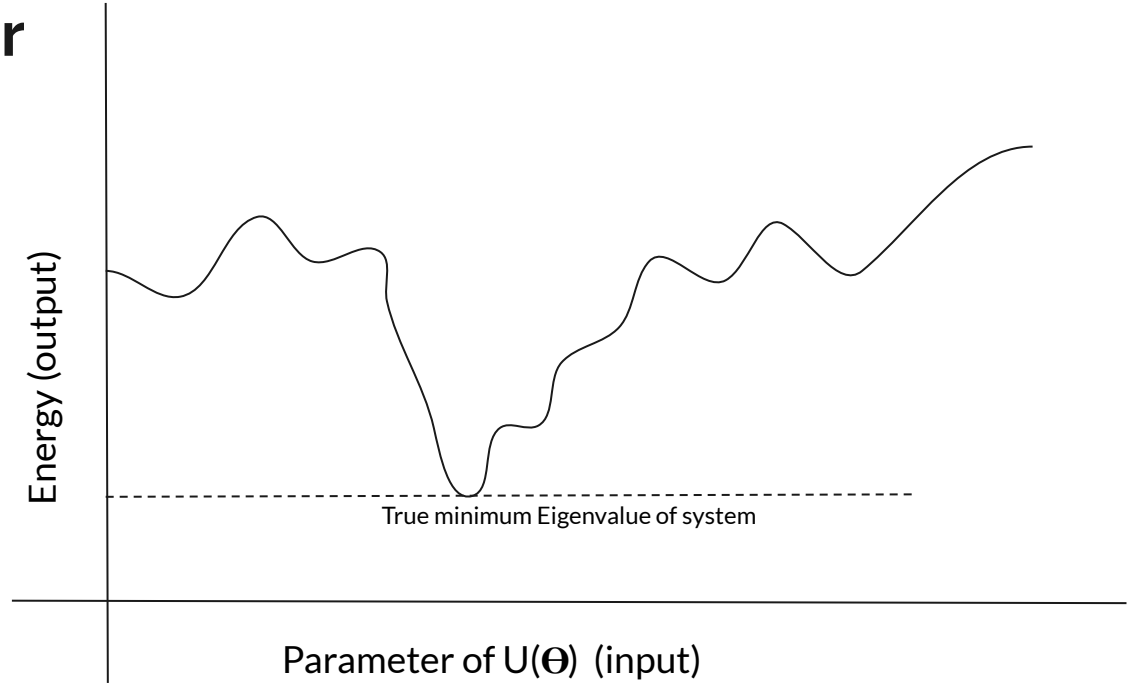
Measurement Function

- Circuit for measuring the energy of the ansatz, i.e. the eigenvalue of the hamiltonian.

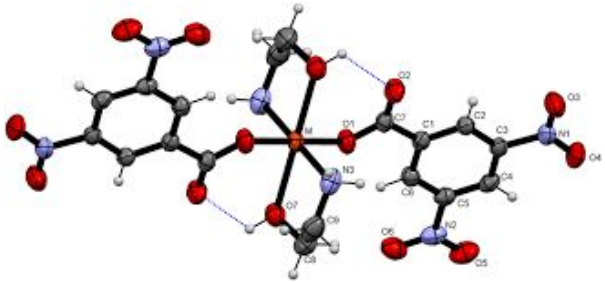
```
# compose quantum circuit

def compose_measurement_circuit(circuit, _M, which_term):
    if which_term == "xe":
        for i in range(_M-1):
            circuit.x(i)
            circuit.x(_M+i)
    elif which_term == "xo":
        for i in range(_M-1):
            circuit.x(i+1)
            circuit.x(_M+i+1)
    elif which_term == "ye":
        for i in range(_M-1):
            circuit.y(i)
            circuit.y(_M+i)
    elif which_term == "yo":
        for i in range(_M-1):
            circuit.y(i+1)
            circuit.y(_M+i+1)
    else:
        pass
    return circuit

circ_measurement = compose_measurement_circuit(deepcopy(circ_empty), M, "xe")
circ_measurement.draw()
```

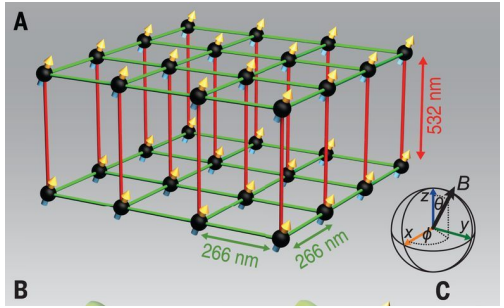
Classical Optimizer

- VQE is a hybrid algorithm (relies on both classical and quantum computers.
- Basic machine learning
- Takes the energy output of the measurement circuit and the parameter theta that resulted in that energy, and then produces an alternate guess for theta that will result in a lower energy until it approaches the absolute minima.



Future Plans





- This presentation covered an overview of the topic and explained code written to develop the hamiltonian into a time evolution applied as quantum gates.
- Plans are to develop code that works as one whole executable program and to test this software on a real molecule.
- Expand the concepts to 3d and 3d hubbard model lattices.

Any Questions?

References

Smfarzaneh. “Vqe-Tutorials/Hubbard-Circuit.ipynb at Master · Smfarzaneh/Vqe-Tutorials.” GitHub, github.com/smfarzaneh/vqe-tutorials/blob/master/hubbard-circuit.ipynb.

arXiv:2111.05176 [quant-ph]

arXiv:2103.12097 [cond-mat.str-el]

Bradben. “Jordan-Wigner Representation - Azure Quantum.” Azure Quantum | Microsoft Docs, docs.microsoft.com/en-us/azure/quantum/user-guide/libraries/chemistry/concepts/jordan-wigner.

Hal Tasaki 1998 J. Phys.: Condens. Matter 10 4353

“Taylor Series.” From Wolfram MathWorld, mathworld.wolfram.com/TaylorSeries.html.

“SuzukiTrotter¶.” SuzukiTrotter - Qiskit 0.37.2 Documentation, qiskit.org/documentation/stubs/qiskit.synthesis.SuzukiTrotter.html.

arXiv:2206.08798 [quant-ph]

Quantum Approximate Optimization Algorithms - Knapsack Problem

Esther Wang

AEOP Undergraduate Apprentice

Knapsack Problem - Background

- Combinatorial optimization problem
  - Finding an optimal object from a finite set of objects
- Given a knapsack with a limited capacity
- Given a finite set of items
  - Weight
  - Value

https://medium.com/algobox/knapsack-problems-4ccee7c295ea

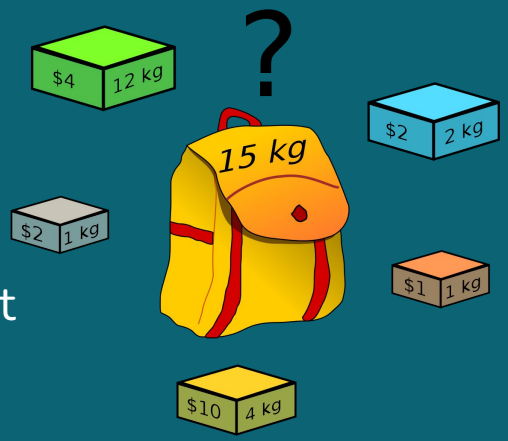
Knapsack Problem - Background

- Determine a subset of items to include in the knapsack
  - Total weight is less than or equal to the capacity of the knapsack
  - Total value of the items is as large as possible
  - Example
    - 4kg ($10) + 2kg ($2) + 1kg ($1) + 1kg ($1)
    - = 8kg < 15kg
    - Total value is $15, which is the max value

https://medium.com/algobox/knapsack-problems-4ccee7c295ea

Knapsack Problem - Background

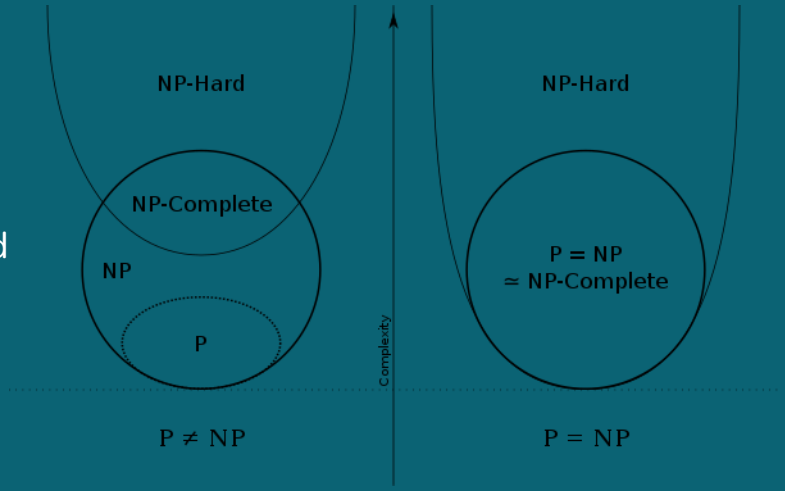
- Derived from a commonplace problem of packing the most valuable items without overloading the luggage
- Often appears in real-word decision-making processes
  - Example
    - Resource allocation problem
    - Given a set of non-divisible tasks
      - Under a fixed budget or time constraint



https://en.wikipedia.org/wiki/Knapsack_problem

Knapsack Problem - Computational Complexity

- Decision problem form of knapsack
  - NP (non-deterministic polynomial-time) Complete
    - Brute-force algorithm by testing all possible cases to find the solution
    - Correctness of the solution can be verified in polynomial time
- Optimization problem form of knapsack
  - Not NP Complete
  - No known polynomial algorithm that can tell whether a given solution is optimal



https://commons.wikimedia.org/wiki/File:P_np_np-complete_np-hard.svg

Classical Solutions

- Classical Algorithms
  - Brute force recursion
  - Dynamic programming
    - Optimization method to solve a class of problems that have overlapping subproblems
  - Branch and bound
    - Algorithm that explores the entire search space to find the optimal solution
  - Hybridization of dynamic programming and branch and bound

Classical Solution - Dynamic Programming

- Dynamic programming solution

```
def knapSack(w, wt, val, n):
    K = [[0 for x in range(w+1)] for y in range(2)]

    # We know we are always using the  current row or
    # the previous row of the array/vector . Thereby we can
    # improve it further by using a 2D array but with only
    # 2 rows i%2 will be giving the index inside the bounds
    # of 2d array K
    for i in range(n + 1):
        for w in range(w + 1):
            if (i == 0 or w == 0):
                K[i % 2][w] = 0
            elif (wt[i - 1] <= w):
                K[i % 2][w] = max(
                    val[i - 1]
                    + K[(i - 1) % 2][w - wt[i - 1]],
                    K[(i - 1) % 2][w])

            else:
                K[i % 2][w] = K[(i - 1) % 2][w]

    return K[n % 2][w]
```

Quantum Approximate Optimization Algorithm

- Algorithm that finds approximate solutions to combinatorial optimization problems
- Performs better than classical computers
- Use the quantum superposition states to compute solutions faster
    - Apply to many possible inputs simultaneously

Unitary Quantum Operators

$$|\beta, \gamma\rangle \equiv U(B, \beta_p)U(C, \gamma_p).....U(B, \beta_1)U(C, \gamma_1)\,|s\rangle$$

where $|\psi_0\rangle$ is a suitable initial state

- $U(\beta,\gamma)$ Unitary is characterized by its parameters to prepare a quantum state
- Composed of $U(\beta)=e^{-i\beta H_B}$ and $U(\gamma)=e^{-i\gamma H_P}$ where H_B is the mixing Hamiltonian and H_P is the problem Hamiltonian
- Quantum state is prepared by applying the unitaries as alternating blocks applied p times
- Find the optimal parameters so that the quantum state encodes the solution

https://arxiv.org/pdf/1908.02210.pdf

Knapsack Cost Function

$$C(X') = M(W_{max} - sum_{i=0}^{n-1} x_i w_i - S)^2 - sum_i^{n-1} x_i v_i$$

where S = sum(2^j * y[j]), j goes from n to n + log(W_{max}). M is a number large enough to dominate the sum of values.

The minimum value will be where the constraint is respected and the sum of the values are maximized.

https://qiskit.org/documentation/stable/0.24/stubs/qiskit.optimization.applications.ising.knapsack.html

Hamiltonian Operator

- Hamiltonian mapped from the cost function

General function form: $$H = H_A + H_B,$$

Formula: $$H_A = A\left(1 - \sum_{n=1}^{W} y_n\right)^2 + A\left(\sum_{n=1}^{W} n y_n - \sum_{\alpha} w_\alpha x_\alpha\right)^2$$

$$H_B = -B \sum_{\alpha} c_\alpha x_\alpha.$$

https://www.frontiersin.org/articles/10.3389/fphy.2014.00005/full

```
prob = Knapsack(values=[3,4,5,6,7], weights=[2,3,4,5,6], max_weight=10)
qp = prob.to_quadratic_program()
print(qp.prettyprint())
```

```
Problem name: Knapsack

Maximize
  3*x_0 + 4*x_1 + 5*x_2 + 6*x_3 + 7*x_4

Subject to
  Linear constraints (1)
    2*x_0 + 3*x_1 + 4*x_2 + 5*x_3 + 6*x_4 <= 10  'c0'

  Binary variables (5)
    x_0 x_1 x_2 x_3 x_4
```

```
# QAOA
qins = QuantumInstance(backend=Aer.get_backend("aer_simulator"), shots=100)
meo = MinimumEigenOptimizer(min_eigen_solver=QAOA(reps=1, quantum_instance=qins))
result = meo.solve(qp)
print(result.prettyprint())
print("\nsolution:", prob.interpret(result))
print("\ntime:", result.min_eigen_solver_result.optimizer_time)
```

```
objective function value: 13.0
variable values: x_0=1.0, x_1=1.0, x_2=0.0, x_3=1.0, x_4=0.0
status: SUCCESS

solution: [0, 1, 3]

time: 1.8710551261901855
```

The Quantum Approximate Optimization Algorithm simulated on a specific knapsack problem with weights and values.

```
# intermediate QUBO form of the optimization problem
conv = QuadraticProgramToQubo()
qubo = conv.convert(qp)
print(qubo.prettyprint())
```

```
Problem name: Knapsack

Minimize
  26*c0@int_slack@0^2 + 104*c0@int_slack@0*c0@int_slack@1
  + 208*c0@int_slack@0*c0@int_slack@2 + 156*c0@int_slack@0*c0@int_slack@3
  + 104*c0@int_slack@1^2 + 416*c0@int_slack@1*c0@int_slack@2
  + 312*c0@int_slack@1*c0@int_slack@3 + 416*c0@int_slack@2^2
  + 624*c0@int_slack@2*c0@int_slack@3 + 234*c0@int_slack@3^2
```

```
# qubit Hamiltonian and offset
op, offset = qubo.to_ising()
print(f"num qubits: {op.num_qubits}, offset: {offset}\n")
print(op)
```

```
num qubits: 9, offset: 1417.5

-258.5 * IIIIIIIIZ
- 388.0 * IIIIIIIZI
- 517.5 * IIIIIIZII
- 647.0 * IIIIIZIII
```

Generating Hamiltonian and the result printing Pauli operators.

```
# op is our hamiltonian
# multiplying by parameter theta and finding the exponentiation
evo_time = Parameter('θ')
evolution_op = (evo_time*op).exp_i()
print(evolution_op) # Note, EvolvedOps print as exponentiations
print(repr(evolution_op))
```

```
e^(-i*1.0*θ * (
  -258.5 * IIIIIIIIZ
```

```
# approximate e^-iHt using two-qubit gates
trotterized_op = PauliTrotterEvolution(trotter_mode=Suzuki(order=2, reps=1)).convert(evo_and_meas)
# We can also set trotter_mode='suzuki' or leave it empty to default to first order Trotterization.
print(trotterized_op)
```

```
ComposedOp([
  OperatorMeasurement(-258.5 * IIIIIIIIZ
  - 388.0 * IIIIIIIZI
  - 517.5 * IIIIIIZII
  - 647.0 * IIIIIZIII
  - 776.5 * IIIIZIIII
  - 130.0 * IIIZIIIII
  - 260.0 * IIZIIIIII
```

Here are the expectation values $\langle \Phi_+ | e^{iHt} H e^{-iHt} | \Phi_+ \rangle$ corresponding to the different values of the parameter

```
h_trotter_expectations.eval()
```

```
[(-0.5+0j),
 (-0.4999999999980105+2.8422e-14j),
 (-0.5000000000000568+5.6843e-14j),
 (-0.4999999999993747+0j),
 (-0.49999999999920425+5.6843e-14j),
 (-0.4999999999997726+2.8422e-14j),
 (-0.5000000000000284+0j),
 (-0.5000000000009948+0j)]
```

```
sampler = CircuitSampler(backend=Aer.get_backend('aer_simulator'))
# sampler.quantum_instance.run_config.shots = 1000
sampled_trotter_exp_op = sampler.convert(h_trotter_expectations)
sampled_trotter_energies = sampled_trotter_exp_op.eval()
print('Sampled Trotterized energies:\n {}'.format(np.real(sampled_trotter_energies)))
```

```
Sampled Trotterized energies:
 [ 27.2265625   31.84765625  31.84765625 -55.953125    64.1953125
 -18.984375   -62.88476563 -30.53710937]
```

Analyzing Results

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

```
variable order: ['x_0', 'x_1', 'x_2', 'x_3', 'x_4']
SolutionSample(x=array([1., 1., 0., 1., 0.]), fval=13.0, probability=0.029900000000000003, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 0., 1., 0., 1.]), fval=12.0, probability=0.03630000000000001, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([1., 1., 1., 0., 0.]), fval=12.0, probability=0.0338, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([0., 0., 1., 1., 0.]), fval=11.0, probability=0.0417, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([0., 1., 0., 0., 1.]), fval=11.0, probability=0.025500000000000002, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 1., 0., 1., 0.]), fval=10.0, probability=0.0324, status=<OptimizationResultStatus.SUCCES
```

```
SolutionSample(x=array([1., 1., 1., 1., 1.]), fval=25.0, probability=0.0221, status=<OptimizationResultStatus.INFEASI
BLE: 2>)
SolutionSample(x=array([0., 1., 1., 1., 1.]), fval=22.0, probability=0.0252, status=<OptimizationResultStatus.INFEASI
BLE: 2>)
SolutionSample(x=array([1., 0., 1., 1., 1.]), fval=21.0, probability=0.03919999999999999, status=<OptimizationResultS
tatus.INFEASIBLE: 2>)
SolutionSample(x=array([1., 1., 0., 1., 1.]), fval=20.0, probability=0.0358, status=<OptimizationResultStatus.INFEASI
BLE: 2>)
SolutionSample(x=array([1., 1., 1., 0., 1.]), fval=19.0, probability=0.028099999999999993, status=<OptimizationResult
Status.INFEASIBLE: 2>)
SolutionSample(x=array([1., 1., 1., 1., 0.]), fval=18.0, probability=0.0331, status=<OptimizationResultStatus.INFEASI
BLE: 2>)
SolutionSample(x=array([0., 0., 1., 1., 1.]), fval=18.0, probability=0.0218, status=<OptimizationResultStatus.INFEASI
```

Analyzing Results

```python
def get_filtered_samples(
    samples: List[SolutionSample],
    threshold: float = 0,
    allowed_status: Tuple[OptimizationResultStatus] = (OptimizationResultStatus.SUCCESS,),
):
    res = []
    for s in samples:
        if s.status in allowed_status and s.probability > threshold:
            res.append(s)

    return res
```

```python
filtered_samples = get_filtered_samples(
    result.samples, threshold=0.005, allowed_status=(OptimizationResultStatus.SUCCESS,)
)
for s in filtered_samples:
    print(s) # val for each configurations; this filters out low prob
```

```
SolutionSample(x=array([1., 1., 0., 1., 0.]), fval=13.0, probability=0.029900000000000003, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 0., 1., 0., 1.]), fval=12.0, probability=0.03630000000000001, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([1., 1., 1., 0., 0.]), fval=12.0, probability=0.0338, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([0., 0., 1., 1., 0.]), fval=11.0, probability=0.0417, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([0., 1., 0., 0., 1.]), fval=11.0, probability=0.025500000000000002, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 1., 0., 1., 0.]), fval=10.0, probability=0.0324, status=<OptimizationResultStatus.SUCCES
```

Analyzing Results

```
SolutionSample(x=array([1., 1., 0., 1., 0.]), fval=13.0, probability=0.029900000000000003, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 0., 1., 0., 1.]), fval=12.0, probability=0.03630000000000001, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([1., 1., 1., 0., 0.]), fval=12.0, probability=0.0338, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([0., 0., 1., 1., 0.]), fval=11.0, probability=0.0417, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([0., 1., 0., 0., 1.]), fval=11.0, probability=0.025500000000000002, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 1., 0., 1., 0.]), fval=10.0, probability=0.0324, status=<OptimizationResultStatus.SUCCES
S: 0>)
SolutionSample(x=array([1., 0., 0., 0., 1.]), fval=10.0, probability=0.026300000000000004, status=<OptimizationResult
Status.SUCCESS: 0>)
SolutionSample(x=array([0., 1., 1., 0., 0.]), fval=9.0, probability=0.027200000000000002, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([1., 0., 0., 1., 0.]), fval=9.0, probability=0.0194, status=<OptimizationResultStatus.SUCCESS:
0>)
SolutionSample(x=array([1., 0., 1., 0., 0.]), fval=8.0, probability=0.019399999999999994, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([0., 0., 0., 0., 1.]), fval=7.0, probability=0.026200000000000005, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([1., 1., 0., 0., 0.]), fval=7.0, probability=0.0339, status=<OptimizationResultStatus.SUCCESS:
0>)
SolutionSample(x=array([0., 0., 0., 1., 0.]), fval=6.0, probability=0.024300000000000002, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([0., 0., 1., 0., 0.]), fval=5.0, probability=0.025300000000000003, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([0., 1., 0., 0., 0.]), fval=4.0, probability=0.025699999999999997, status=<OptimizationResultS
tatus.SUCCESS: 0>)
SolutionSample(x=array([1., 0., 0., 0., 0.]), fval=3.0, probability=0.0375, status=<OptimizationResultStatus.SUCCESS:
0>)
SolutionSample(x=array([0., 0., 0., 0., 0.]), fval=0.0, probability=0.04229999999999999, status=<OptimizationResultSt
atus.SUCCESS: 0>)
```

References

https://en.wikipedia.org/wiki/Knapsack_problem#:~:text=The%20knapsack%20problem%20is%20a,is%20as%20large%20as%20possible.

https://en.wikipedia.org/wiki/NP-completeness

https://qiskit.org/documentation/stable/0.24/stubs/qiskit.optimization.applications.ising.knapsack.html

https://qiskit.org/documentation/optimization/tutorials/09_application_classes.html#Knapsack-problem

https://www.frontiersin.org/articles/10.3389/fphy.2014.00005/full

https://qiskit.org/documentation/tutorials/operators/01_operator_flow.html

https://arxiv.org/pdf/1908.02210.pdf

https://arxiv.org/pdf/1411.4028.pdf

Shor's Algorithm

By Riya Raina

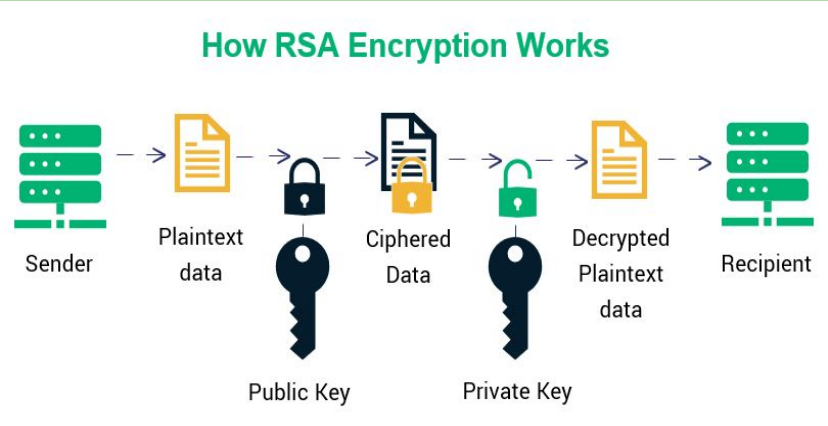
The Problem

Find the prime factorization of a large integer in an efficient manner

Why is it important to solve?

Some applications of prime factorization include:

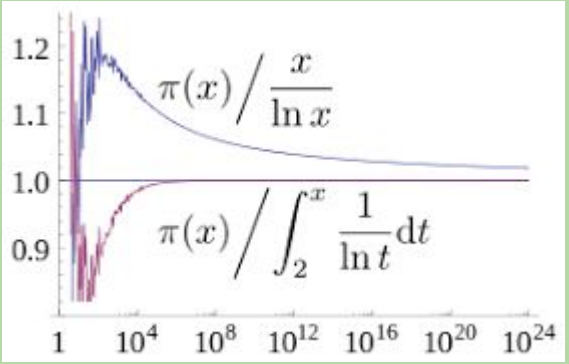
- Cryptography and Cybersecurity
  - RSA Algorithm used to encrypt messages
  - Public and private keys are large integers, multiplied to encrypt data
  - Relies on large numbers being hard to factor
  - Potential Cybersecurity Threat



Why is it important to solve?

Some applications of prime factorization include:

- Prime Number Theorem
  - Estimates frequency of prime numbers
- Discovery of New Largest Prime Numbers
  - Current Largest Known: 2^82,589,933-1



Classical Solution - Method I

- Loop through numbers 0 - √num,
- Add current number to factors array if perfectly divisible

```python
# Method 1: time complexity - O(n^(1/2)log(n))
import math


def prime_factors(num):
    factors = []
    while num % 2 == 0:
        factors.append(2)
        num = num / 2

    for i in range(3, int(math.sqrt(num)) + 1, 2):

        while num % i == 0:
            factors.append(i)
            num = num / i

    if num > 2:
        factors.append(int(num))

    print(factors)


# example to test
n = 9023724723
prime_factors(n)
```

```
[3, 13, 113, 983, 2083]
```

Classical Solution - Method II

- Variable i = 2
- Continuous loop while num > 1
- If num is divisible by i, add i to factors and divide num by i
- Else, increment i by 1

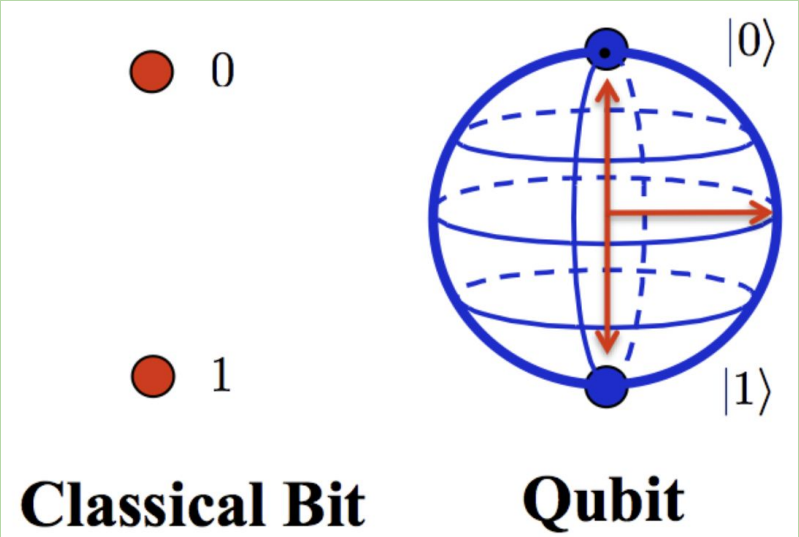
```python
# Method 2: time complexity - O(n)

def prime_factors(num):
    factors = []
    i = 2
    while num > 1:
        if num % i == 0:
            factors.append(i)
            num = num / i
        else:
            i = i + 1
    print(factors)


# example to test
n = 9023724723
prime_factors(n)
```

```
[3, 13, 113, 983, 2083]
```

Quantum Solution

Background Info about Quantum Computing

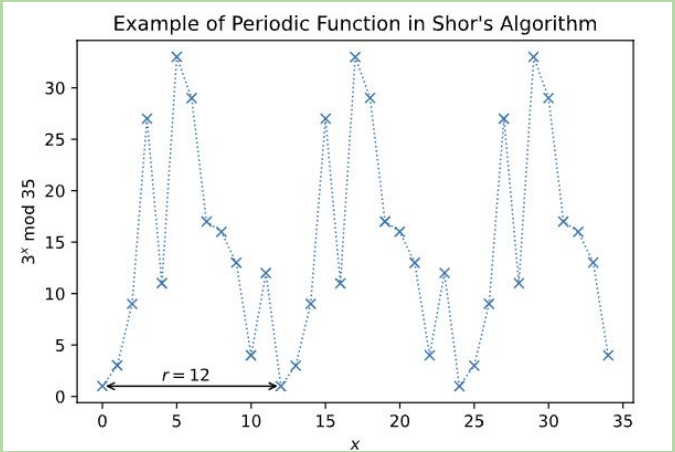
- Classical Bit - Unit of info with 2 distinct states, 1 and 0
- Quantum Qubit - Unit of info that can be in superposition of both 0 and 1 states simultaneously
- Qubits can be represented as 3D vector states that are rotated by gates and operations



Quantum Solution

- A Factoring problem can be turned into the problem of finding the period of a function
- The period (r) is the smallest non-zero integer that satisfies the following equation:

$$a^r \bmod N = 1$$



Quantum Solution

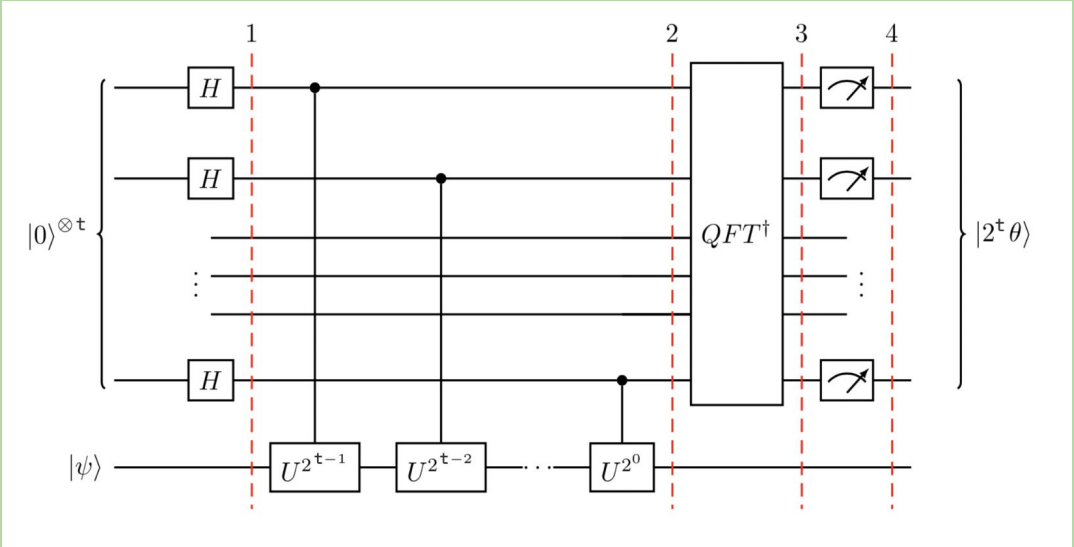
Example of Shor's Algorithm with N = 15

- Generate a random number less than N, such as a = 2
- Calculate gcd $gcd(a, N) = gcd(2, 15) = 1$
- Find period of $a^r \bmod N$ using quantum phase estimation

Quantum Solution

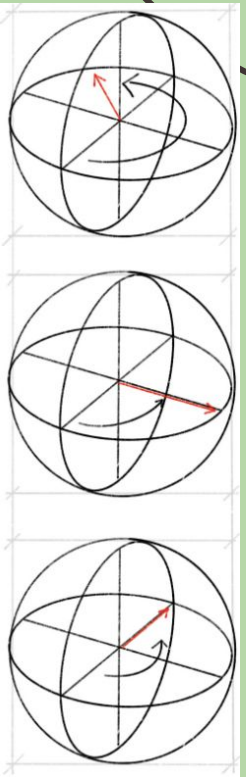
A look into Quantum Phase Estimation

- Top register: t counting qubits that control unitary operations
- Bottom register: qubits in state of $|\psi\rangle$ which operations are applied to



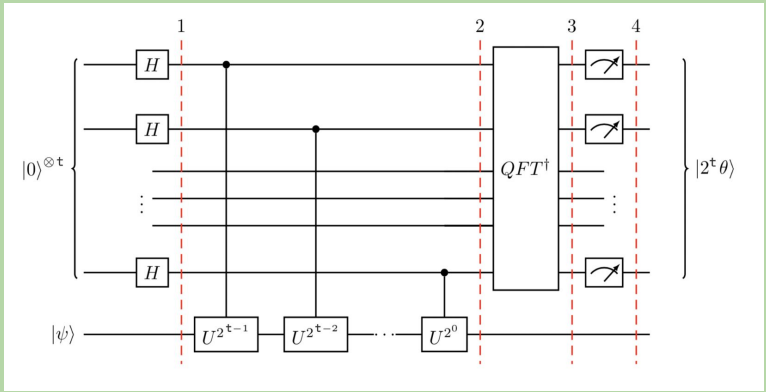
Quantum Solution

- Hadamard gates put the counting qubits into superposition
- Apply unitary operations on bottom qubits
  - Controlled phase rotations of a specific angle - the phase we wish to estimate
  - First qubit does 1 rotation
  - Second qubit does 2 rotations
  - Third qubit does 4 rotations and so on



Quantum Solution

- Apply an inverse Quantum Fourier Transformation (QFT) to the counting qubits and measure them
- The inverse QFT transforms the qubit state from the Fourier basis to the computational basis so it can be measured as a binary value
- Convert binary to decimal



Quantum Solution

- To estimate the phase we use the formula:
- Θ is the estimated phase
- M is the measured value
- n is the number of counting qubits

$$\theta_{estimated} = \frac{M}{2^n}$$

Now that we have calculated the phase, it can be used in the previous calculation

Quantum Solution

$$\theta_{estimated} = \frac{M}{2^n}$$

Back to example with N = 15 and a = 2

- Quantum Phase Estimation yields 10000000 in binary
- Convert to 128 in decimal
- 5 counting qubits used, so n = 5

Quantum Phase Estimation has calculated that the phase (r) = 4

Quantum Solution

$$a^r \bmod N = 1$$

$$(a^r - 1) \bmod N = 0$$

$$a^r - 1 = (a^{r/2} - 1)(a^{r/2} + 1)$$

- Rewrite original equation
- Substitute in values of a and r

$$gcd(a^{\frac{r}{2}} + 1, N) = gcd(5, 15) = 5$$
$$gcd(a^{\frac{r}{2}} - 1, N) = gcd(3, 15) = 5$$

```
guesses = [gcd(a**(r//2) - 1, N), gcd(a**(r//2) + 1, N)]
```

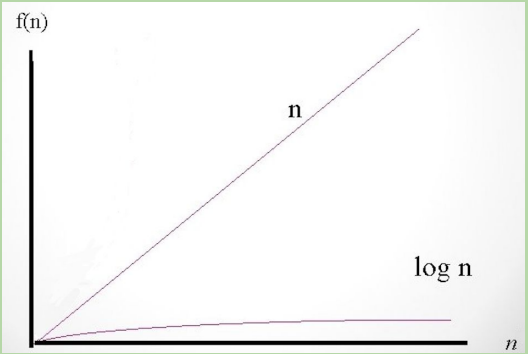
- For N = 15, the two decomposed prime numbers were 3 and 5

```
Guessed Factors: 3 and 5
*** Non-trivial factor found: 3 ***
*** Non-trivial factor found: 5 ***
Done!
```

Conclusion

Classical Solution

Time complexity:
O(n)



Quantum Solution

Time complexity:
O(logN)

When the classical solution takes 1 billion seconds, the quantum solution takes 9 seconds!

Sources

Shor's Algorithm Diagrams:
https://qiskit.org/textbook/ch-algorithms/shor.html

Time Complexity Diagram:
https://slidetodoc.com/algorithm-analysis-with-big-oh-data-structures-and/

Quantum Phase Estimation Diagrams:
https://qiskit.org/textbook/ch-algorithms/quantum-phase-estimation.html

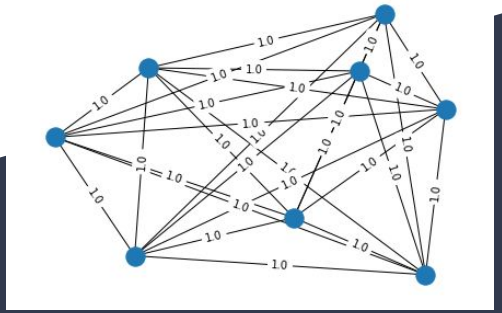
RSA Algorithm Diagram:
https://sectigostore.com/blog/ecdsa-vs-rsa-everything-you-need-to-know/

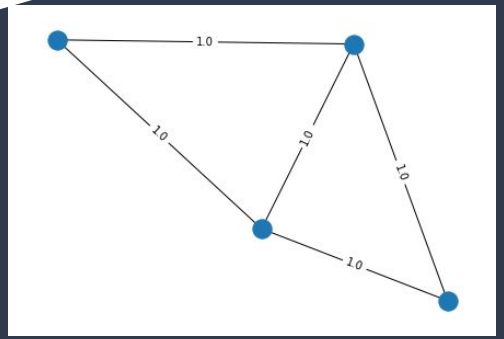
Prime Number Theory Diagram:
https://en.wikipedia.org/wiki/Prime_number_theorem

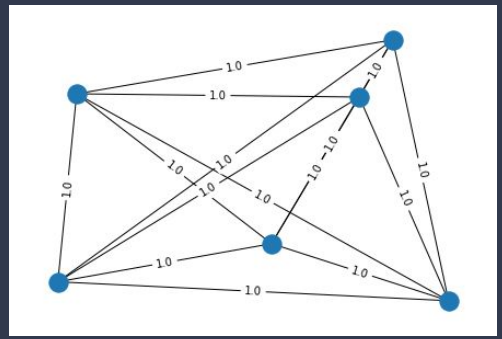
Questions?

Implementing Quantum Approximation Optimization Algorithm (QAOA) with the Max Cut Problem

Anna Hsu

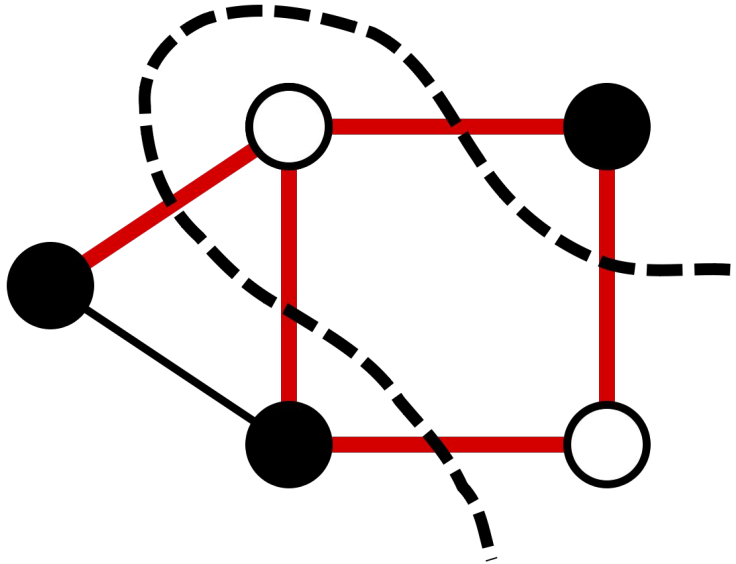






The Max Cut Problem

→ Given an undirected graph with n nodes and weighted edges connecting the nodes, find the cut that divides the nodes into two groups and maximizes the number of edges across groups



Example of the max cut of a graph
https://en.wikipedia.org/wiki/Maximum_cut

The Max Cut Problem

Can be used to model:

- Circuits with different flow levels
- Marketing model, using weighted edges to represent the potential influence of an interaction between two people

A similar QAOA approach can solve problems such as the Traveling Salesman, Graph Coloring, and Knapsack problems

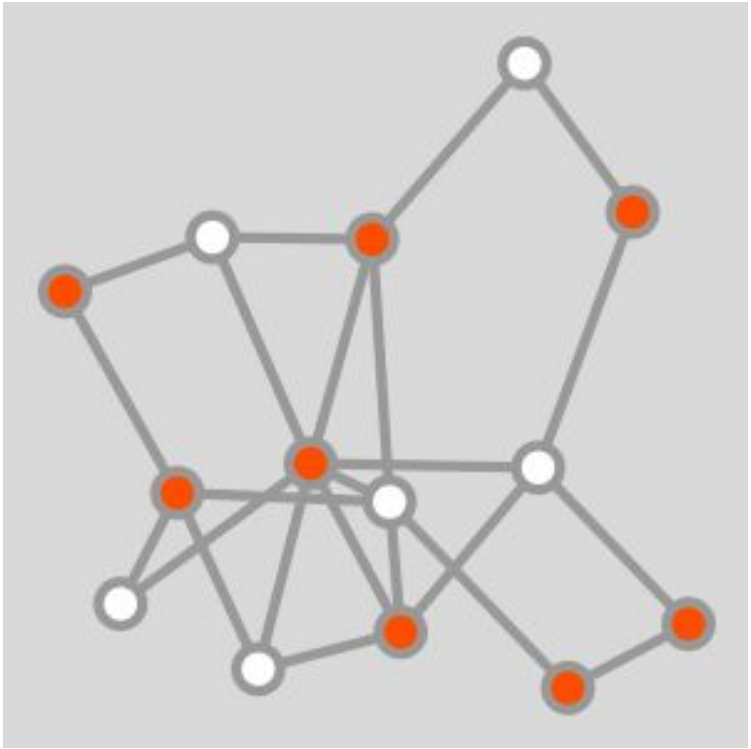


Image:https://www.wolfram.com/language/12/core-graphs-and-networks/solve-max-cut-problem.html?product=mathematica

Pauli Matrices

$$\sigma_1 = \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\sigma_2 = \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

$$\sigma_3 = \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

- Called “gates”
- These are operations that can be applied to a qubit in different combinations

In Qiskit:

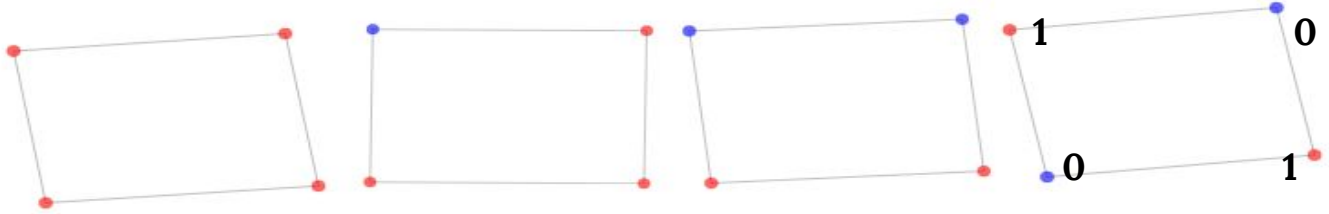
Image:https://www.wolfram.com/language/12/core-graphs-and-networks/solve-max-cut-problem.html?product=mathematica

Classical Solution

```
In [36]: def objective_value(x, w):
             X = np.outer(x, (1 - x))
             w_01 = np.where(w != 0, 1, 0)
             return np.sum(w_01 * X)

         def brute_force():
             # use the brute-force way to generate the oracle
             def bitfield(n, L):
                 result = np.binary_repr(n, L)
                 return [int(digit) for digit in result]  # [2:] to chop off the "0b" part

             L = num_nodes
             max = 2**L # max number of combinations to put L nodes into two groups
             maximum_v = np.inf
             for i in range(max):
                 cur = bitfield(i, L) #binary representation of i

                 how_many_nonzero = np.count_nonzero(cur)
                 if how_many_nonzero * 2 != L:  # checks if # of 0 and 1s is balanced, continues if not
                     continue

                 cur_v = objective_value(np.array(cur), w)
                 if cur_v < maximum_v: # replaces maximum with new maximum to find true max number of edges
                     maximum_v = cur_v
             return maximum_v

         sol = brute_force()
         print(f'Objective value computed by the brute-force method is {sol}')

Objective value computed by the brute-force method is 3
```

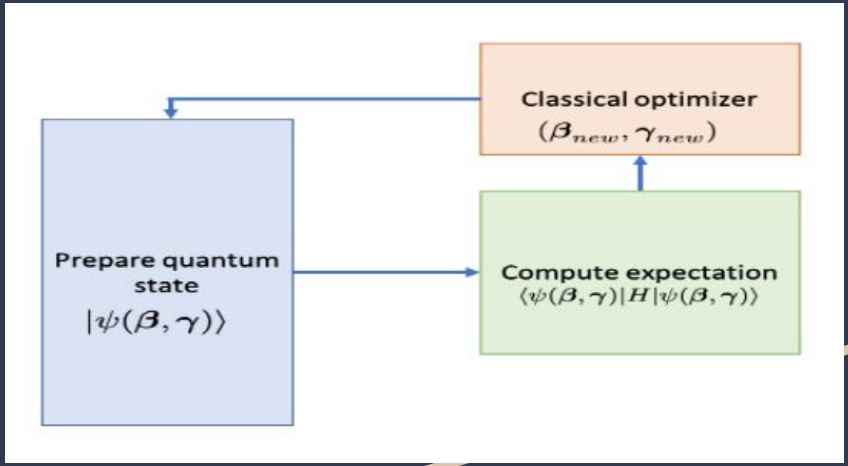
- Brute force method stores each combination of red-blue nodes as a bitstring and tests 2^L combinations
- Time to solve using brute force method increases exponentially as # of nodes increases

0 = blue
1 = red



Source: https://qiskit.org/textbook/ch-applications/qaoa.html

Quantum Solution



Solving with QAOA depends on two parameters, β and γ

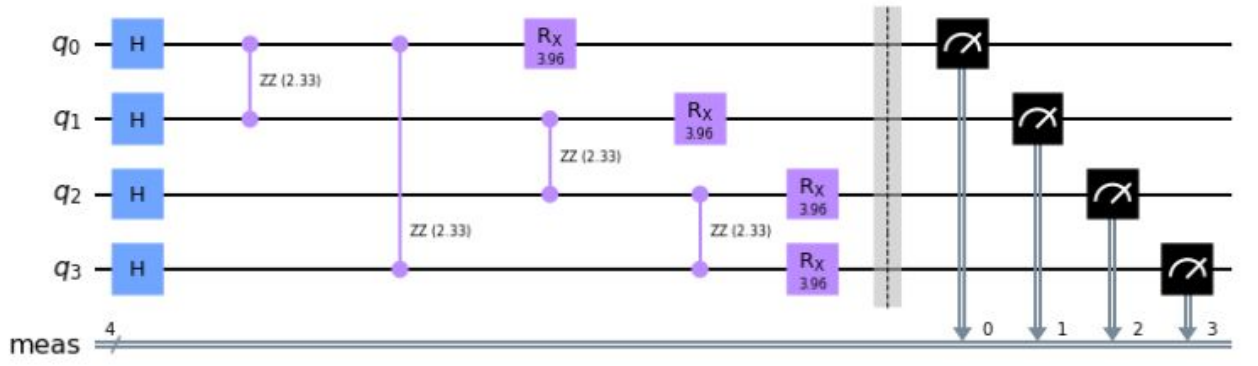
1. Prepare equal superposition state
2. Apply QAOA circuit of alternating cost and mixer layers
3. Measurement in computational basis → bitstring samples
4. Evaluate cut values
5. Optimize β and γ using classical optimizer, repeat with new parameters

Image: https://qiskit.org/textbook/ch-applications/qaoa.html#The-QAOA-circuit

The QAOA Circuit: An Overview

```
In [22]: qc_res.draw('mpl')
```

Out[22]:



Prepare initial state

Cost & mixer layers

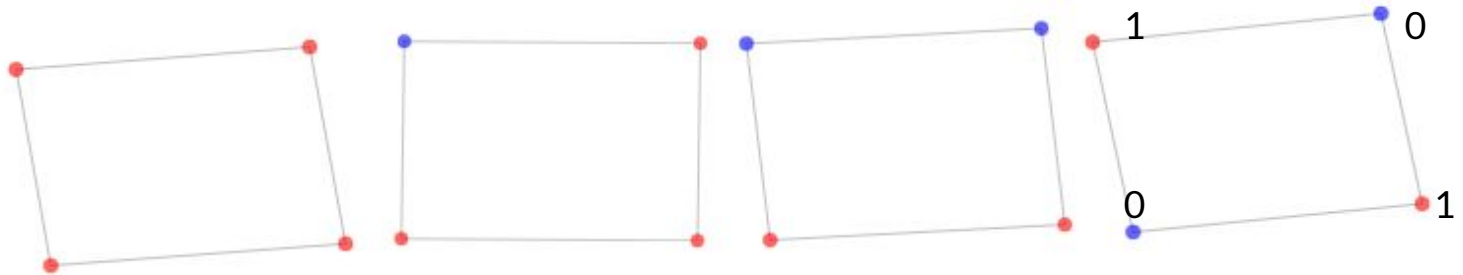
Measurement

Quantum Solution

Construct the Hamiltonian for the problem

- Assigns edge weight of 1 when i =1 , j = 0

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



Source: https://qiskit.org/textbook/ch-applications/qaoa.html

The Cost Layer (γ)

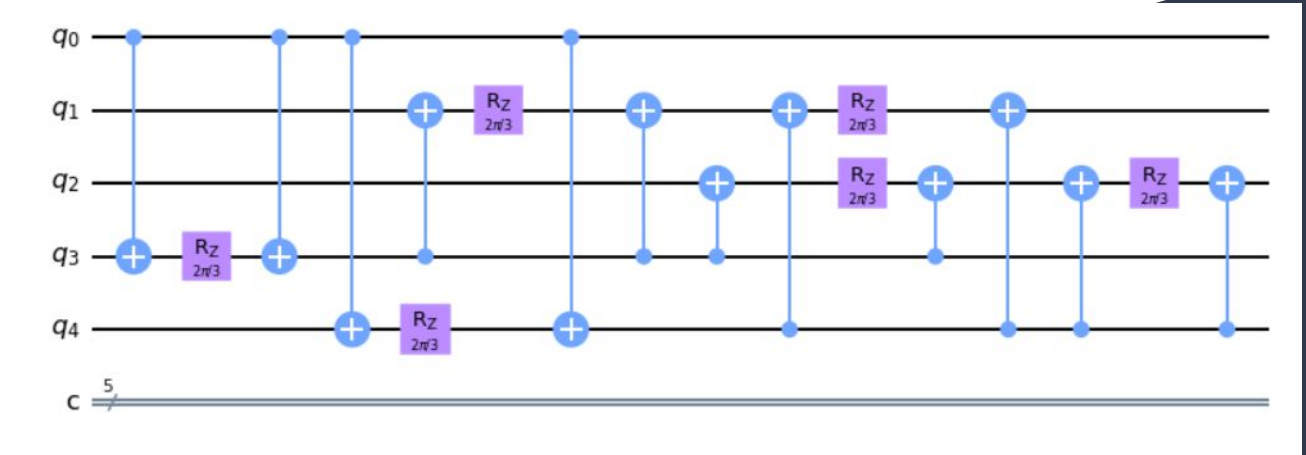
The cost layer is constructed with the exponentiation of the cost Hamiltonian, which decomposes to a combination of CNOT gates and R_{ZZ} rotation gates

Cost Hamiltonian:

$$H_C = \sum_{i,j=1}^{n} \frac{1}{4} Q_{ij} Z_i Z_j - \sum_{i=1}^{n} \frac{1}{2} \left(c_i + \sum_{j=1}^{n} Q_{ij} \right) Z_i$$

The Cost Layer (γ)

The cost layer is constructed with the exponentiation of the cost Hamiltonian, which decomposes to a combination of CNOT gates and R_{ZZ} rotation gates



The Mixer Layer (β)

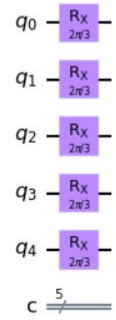
- The mixer layer “mixes” the quantum state so that the cost layer can be applied again to test more quantum states, parameterized by β.
- Constructed by the exponentiation of the mixer Hamiltonian
- X rotation gate by 2β

$$H_M = \sum_{i=1}^{n} X_i$$

```
In [8]: def append_x_term(qc, q1, beta):
            qc.rx(2*beta, q1)

        def get_mixer_operator_circuit(G, beta):
            N = G.number_of_nodes()
            qc = QuantumCircuit(N,N)
            for n in G.nodes():
                append_x_term(qc, n, beta)
            return qc
```

```
In [10]: qc = get_mixer_operator_circuit(G, np.pi/3)
         qc.draw('mpl')
```

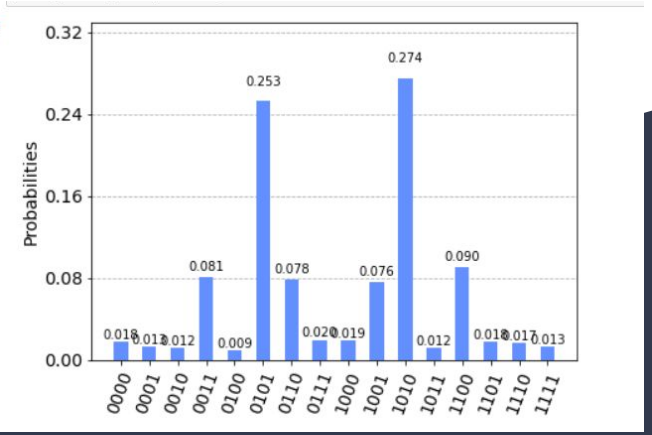
Out[10]:

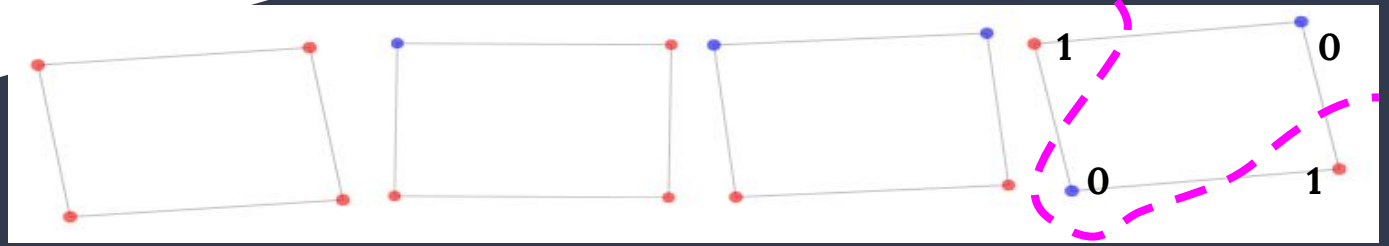


Results

In [21]:
```
from qiskit.visualization import plot_histogram

backend = Aer.get_backend('aer_simulator')
backend.shots = 512

qc_res = create_qaoa_circ(G, res.x)

counts = backend.run(qc_res, seed_simulator=10).result().get_counts()

plot_histogram(counts)
```

Out[21]:

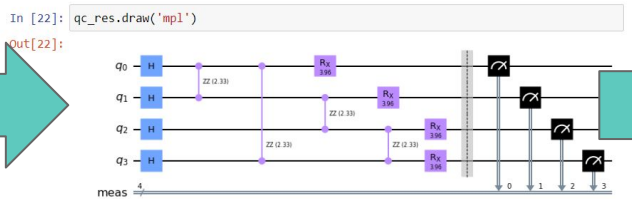




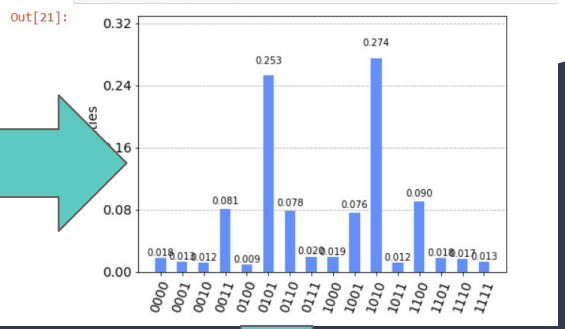
Recap

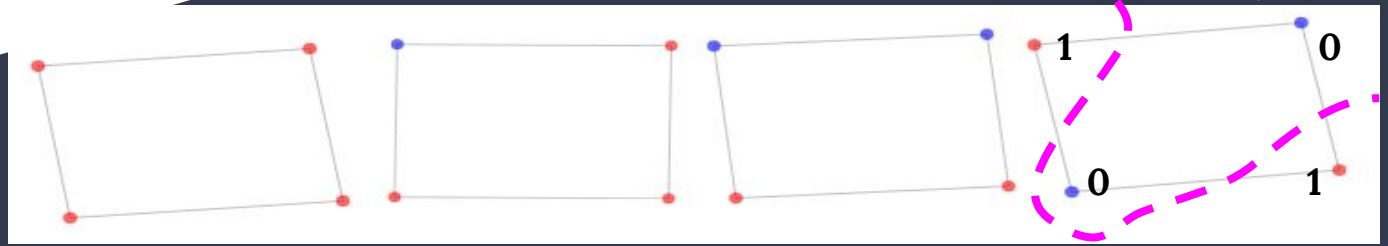
Graph

QAOA circuit



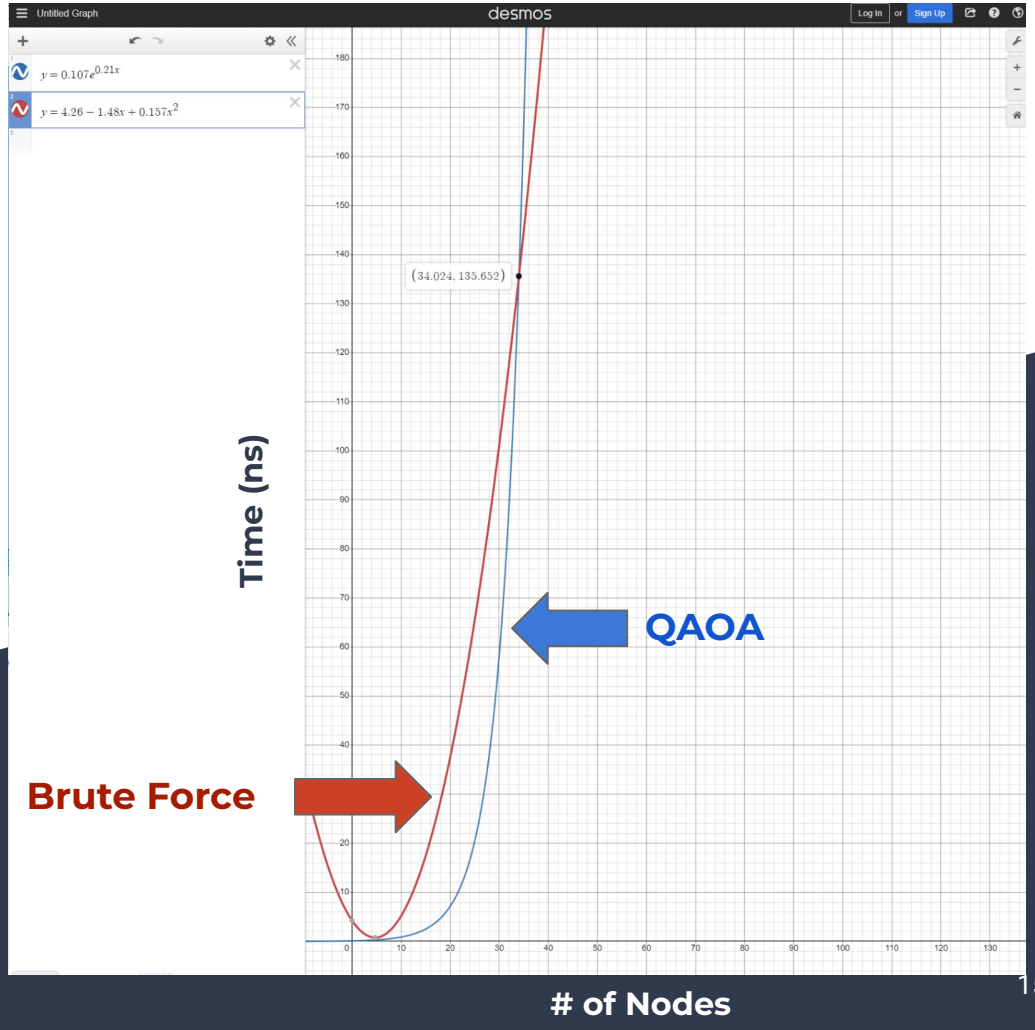
Results





Results

- Classical computing time modeled by exponential equation, QAOA time modeled by polynomial equation
- QAOA operates in polynomial time and is faster than classical approach after threshold of 34+ node graph
-



Sources

Qiskit Summer School 2021 Lab 2
https://learn.qiskit.org/summer-school/2021/lab2-variational-algorithms

Solving combinatorial optimization problems using QAOA
https://qiskit.org/textbook/ch-applications/qaoa.html

Quantum Approximation Optimization Algorithm
https://qiskit.org/documentation/locale/de_DE/tutorials/algorithms/05_qaoa.html

Surface Codes

Logical Qubits for Error Correction

Weixin Lu

Austin G. Fowler, Matteo Mariantoni, John M. Martinis, Andrew N. Cleland, Surface codes: Towards practical large-scale quantum computation

- **Error Correction**
- **Error Detection**
- **Surface Code**
- **Logical Operators**

Qubit Gates / Pauli Operators

Qubit: 2 eigenstates |g>, |e>

Gates are Hermitian: $AA^H = I$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$Y = \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix}$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\hat{X}^2 = -\hat{Y}^2 = \hat{Z}^2 = \hat{I}$$

$$\hat{X}\hat{Z} = -\hat{Z}\hat{X}$$

$$[\hat{X}, \hat{Y}] \equiv \hat{X}\hat{Y} - \hat{Y}\hat{X} = -2\hat{Z}$$

Qubit Errors

- X bit-flip
- Z phase-flip
- Y = ZX combined
- measurement
- ...

Gates are Hermitian: $AA^H = I$

Error Detection

Detect errors by repeatedly measuring each qubit

Measurement operators must commute!!

$$[X, Z] \neq 0$$

$$[X_a X_b, Z_a Z_b] = 0$$

$\hat{Z}_a\hat{Z}_b$	$\hat{X}_a\hat{X}_b$	$\|\psi\rangle$
$+1$	$+1$	$(\|gg\rangle + \|ee\rangle)/\sqrt{2}$
$+1$	-1	$(\|gg\rangle - \|ee\rangle)/\sqrt{2}$
-1	$+1$	$(\|ge\rangle + \|eg\rangle)/\sqrt{2}$
-1	-1	$(\|ge\rangle - \|eg\rangle)/\sqrt{2}$

Error Detection

Errors cannot be uniquely identified

Initial State: $(|gg\rangle + |ee\rangle)/\sqrt{2}$

X_a or X_b error: $(|ge\rangle + |eg\rangle)/\sqrt{2}$

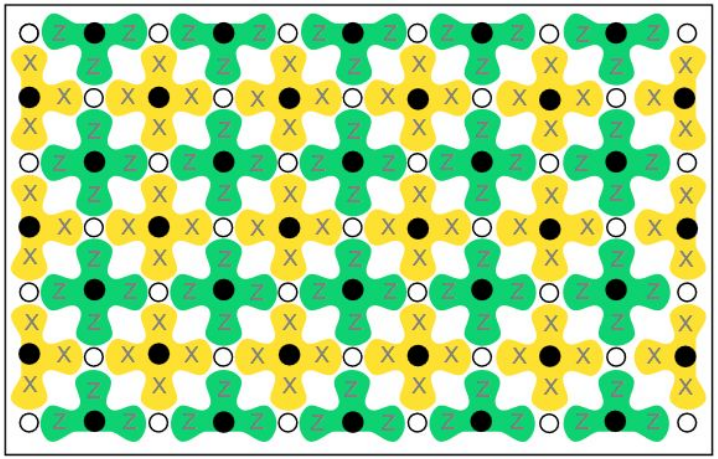
$\hat{Z}_a\hat{Z}_b$	$\hat{X}_a\hat{X}_b$	$\|\psi\rangle$
$+1$	$+1$	$(\|gg\rangle + \|ee\rangle)/\sqrt{2}$
$+1$	-1	$(\|gg\rangle - \|ee\rangle)/\sqrt{2}$
-1	$+1$	$(\|ge\rangle + \|eg\rangle)/\sqrt{2}$
-1	-1	$(\|ge\rangle - \|eg\rangle)/\sqrt{2}$

Surface Codes

- Error Detection
- Logical Qubits

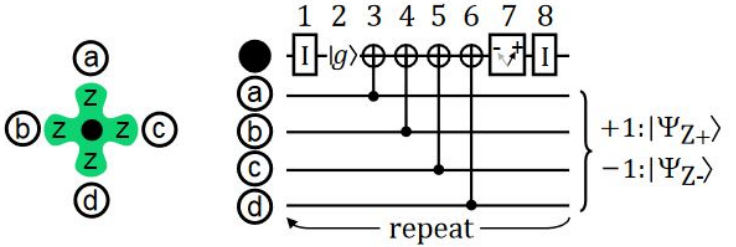
Surface Code

- Data qubits
- Measurement qubits

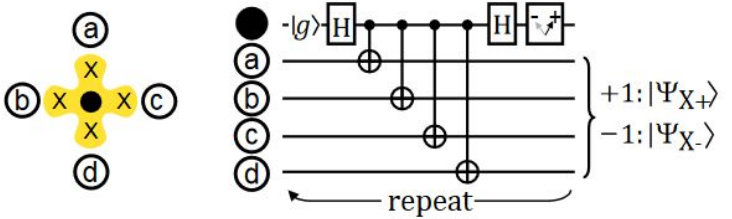


Stabilizers

$$Z_a Z_b Z_c Z_d$$



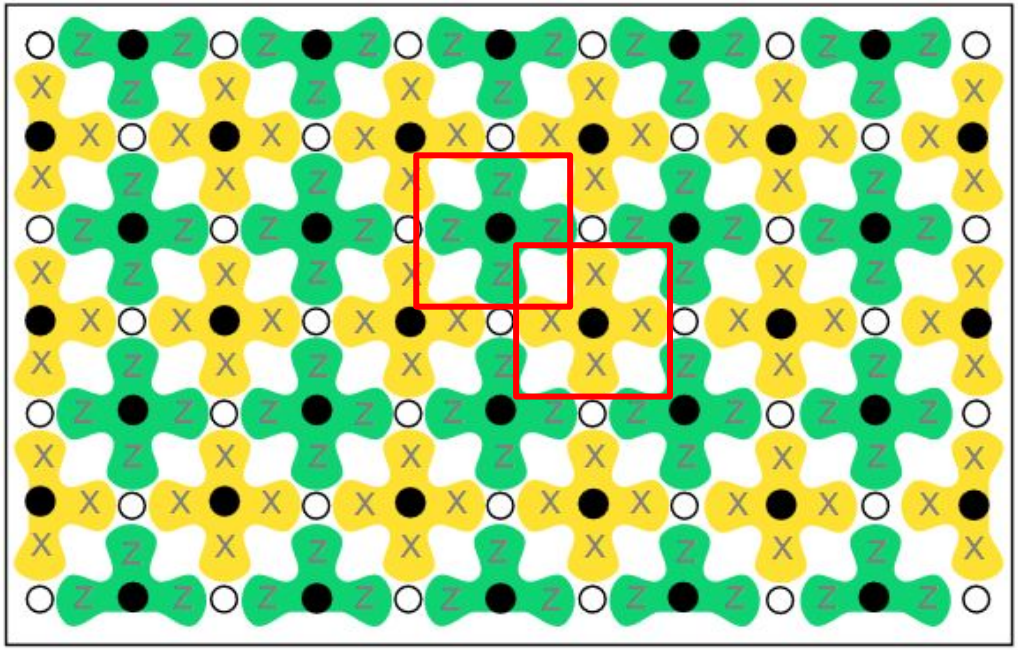
$$X_a X_b X_c X_d$$



Error Detection

Stabilizers are commute

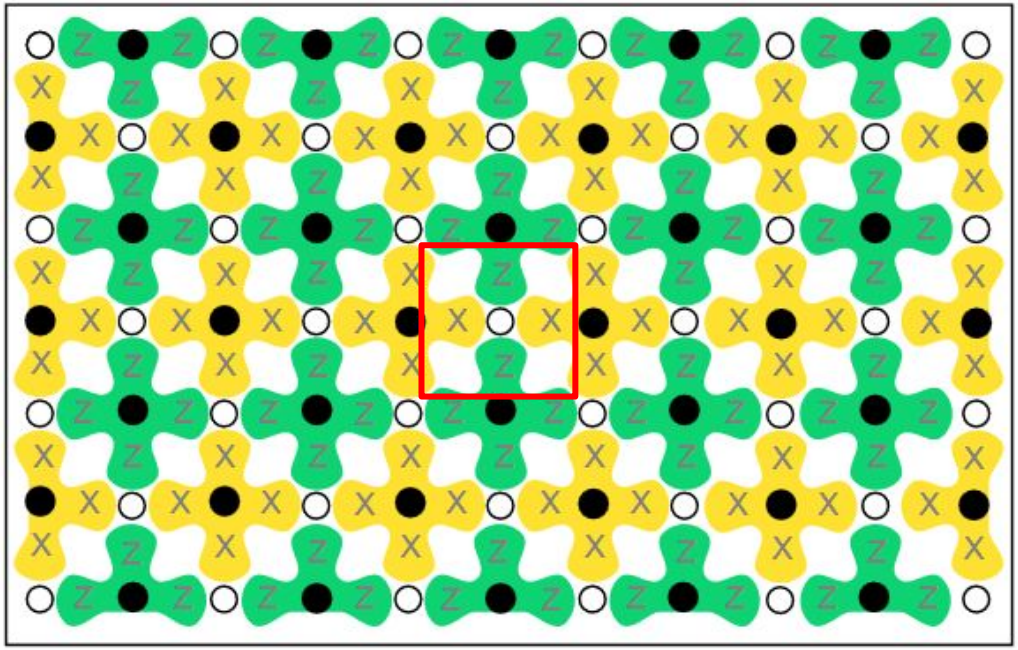
$$\begin{aligned}\left[\hat{X}_a\hat{X}_b\hat{X}_c\hat{X}_d\ ,\hat{Z}_a\hat{Z}_b\hat{Z}_e\hat{Z}_f\right] \\ &= \left(\hat{X}_a\hat{Z}_a\right)\left(\hat{X}_b\hat{Z}_b\right)\hat{X}_c\hat{X}_d\hat{Z}_e\hat{Z}_f \\ &- \left(\hat{Z}_a\hat{X}_a\right)\left(\hat{Z}_b\hat{X}_b\right)\hat{X}_c\hat{X}_d\hat{Z}_e\hat{Z}_f \\ &= 0\end{aligned}$$



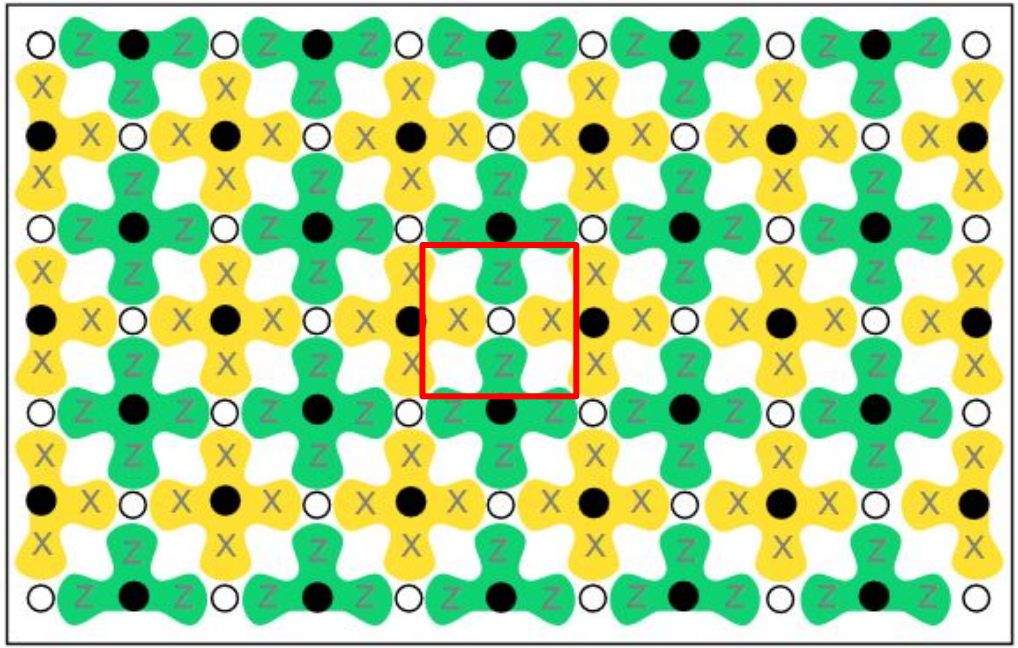
Error Detection

Z error

$$
\begin{aligned}
\hat{X}_a\hat{X}_b\hat{X}_c\hat{X}_d\left(\hat{Z}_a|\psi\rangle\right) &= -\hat{Z}_a\left(\hat{X}_a\hat{X}_b\hat{X}_c\hat{X}_d|\psi\rangle\right) \\
&= -X_{abcd}\left(\hat{Z}_a|\psi\rangle\right) \\
\hat{Z}_a\hat{Z}_b\hat{Z}_c\hat{Z}_d\left(\hat{Z}_a|\psi\rangle\right) &= \hat{Z}_a\left(\hat{Z}_a\hat{Z}_b\hat{Z}_c\hat{Z}_d|\psi\rangle\right) \\
&= Z_{abcd}\left(\hat{Z}_a|\psi\rangle\right),
\end{aligned}
$$



Error Detection



X error: flip the 2 neighboring Z-measurement qubits

Y error: flip all 4 neighboring qubits

measurement error: flip 1 measurement qubits

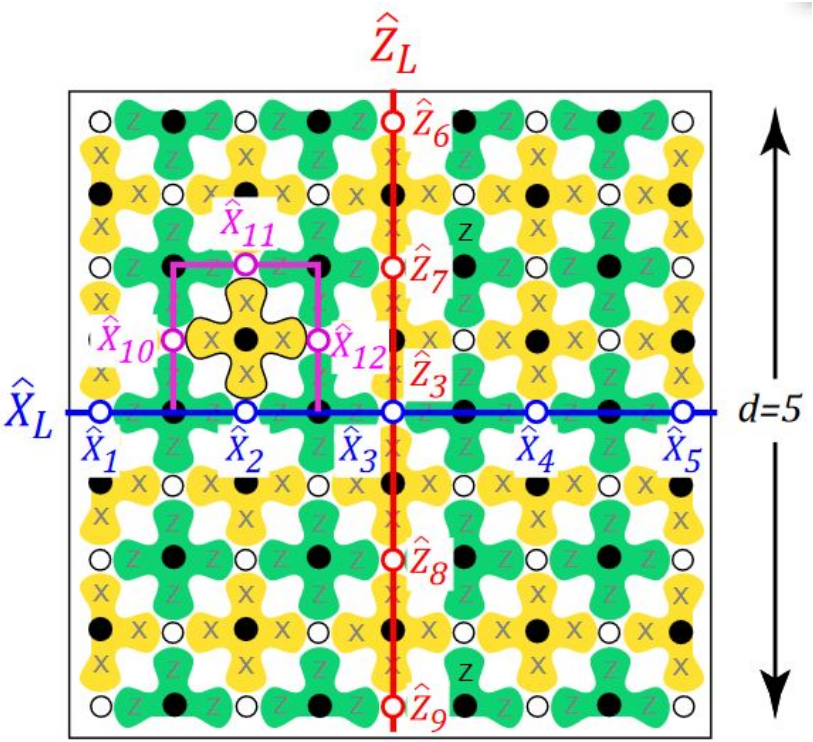
From Physical Qubits to Logical Qubits

additional degrees of freedom

– logical operators

$$\hat{X}_L = \hat{X}_1\hat{X}_2\hat{X}_3\hat{X}_4\hat{X}_5$$

$$\hat{Z}_L = \hat{Z}_6\hat{Z}_7\hat{Z}_3\hat{Z}_8\hat{Z}_9$$

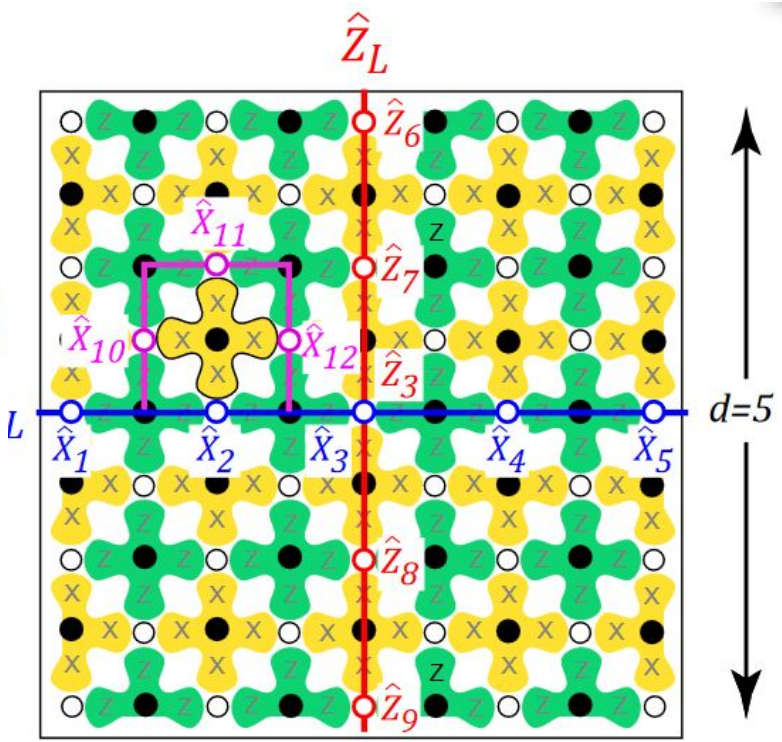


Logical Qubits

More operators?

$$\begin{aligned}\hat{X}'_L &= \hat{X}_1\hat{X}_{10}\hat{X}_{11}\hat{X}_{12}\hat{X}_3\hat{X}_4\hat{X}_5 \\ &= \left(\hat{X}_2\hat{X}_{10}\hat{X}_{11}\hat{X}_{12}\right)\left(\hat{X}_1\hat{X}_2\hat{X}_3\hat{X}_4\hat{X}_5\right) \\ &= \boxed{\left(\hat{X}_2\hat{X}_{10}\hat{X}_{11}\hat{X}_{12}\right)}\hat{X}_L,\end{aligned}$$

Stabilizer



Logical Operators

$$\begin{aligned}\hat{X}_L\hat{Z}_L &= \left(\hat{X}_1\hat{X}_2\hat{X}_3\hat{X}_4\hat{X}_5\right)\left(\hat{Z}_9\hat{Z}_{10}\hat{Z}_3\hat{Z}_{11}\hat{Z}_{12}\right)\\ &= \hat{X}_3\hat{Z}_3\left(\hat{X}_1\hat{X}_2\hat{X}_4\hat{X}_5\right)\left(\hat{Z}_9\hat{Z}_{10}\hat{Z}_{11}\hat{Z}_{12}\right)\\ &= -\hat{Z}_3\hat{X}_3\left(\hat{Z}_9\hat{Z}_{10}\hat{Z}_{11}\hat{Z}_{12}\right)\left(\hat{X}_1\hat{X}_2\hat{X}_4\hat{X}_5\right)\\ &= -\hat{Z}_L\hat{X}_L,\end{aligned}$$

$$\hat{Y}_L = \hat{Z}_L\hat{X}_L$$

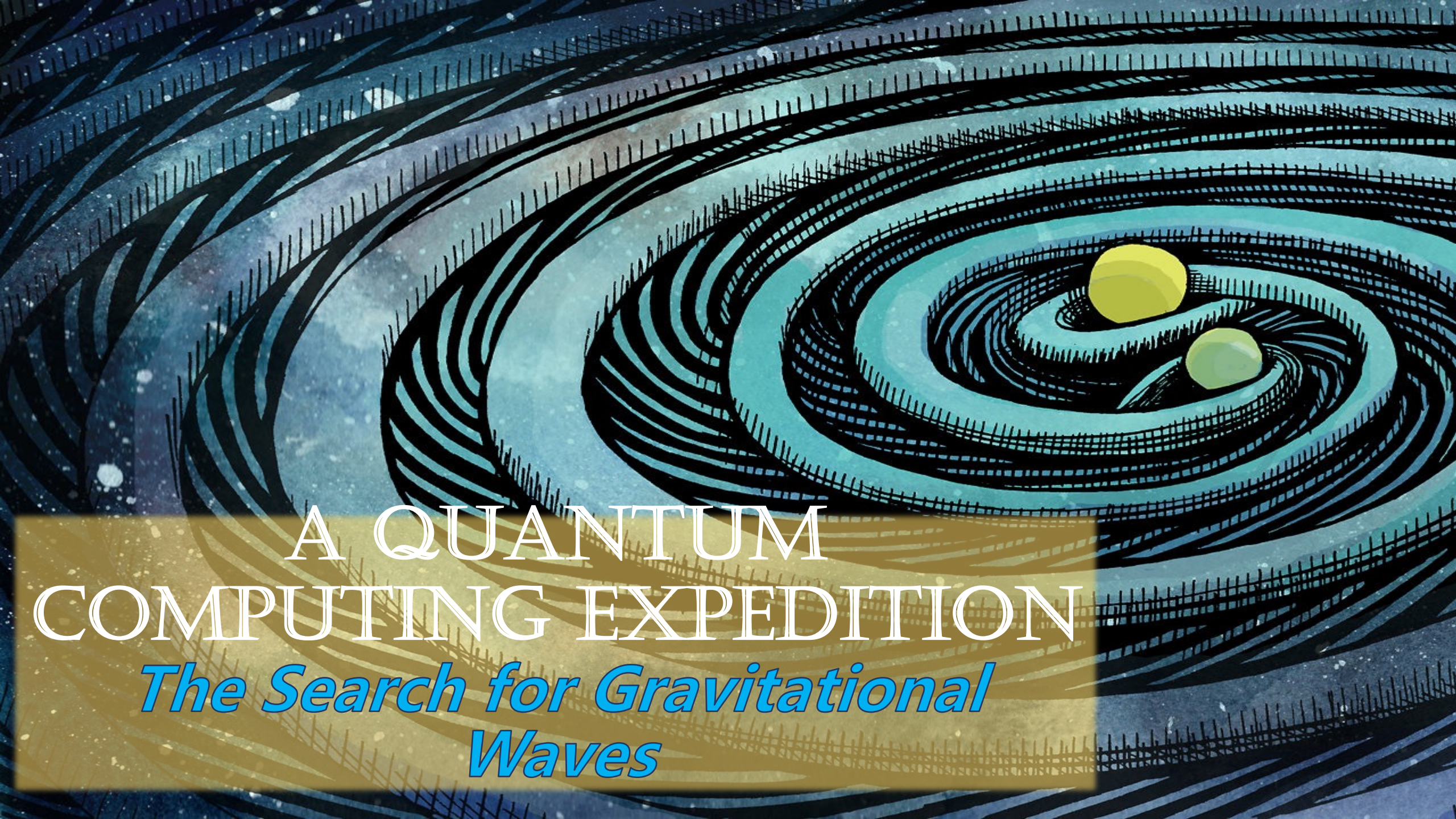
Recall:

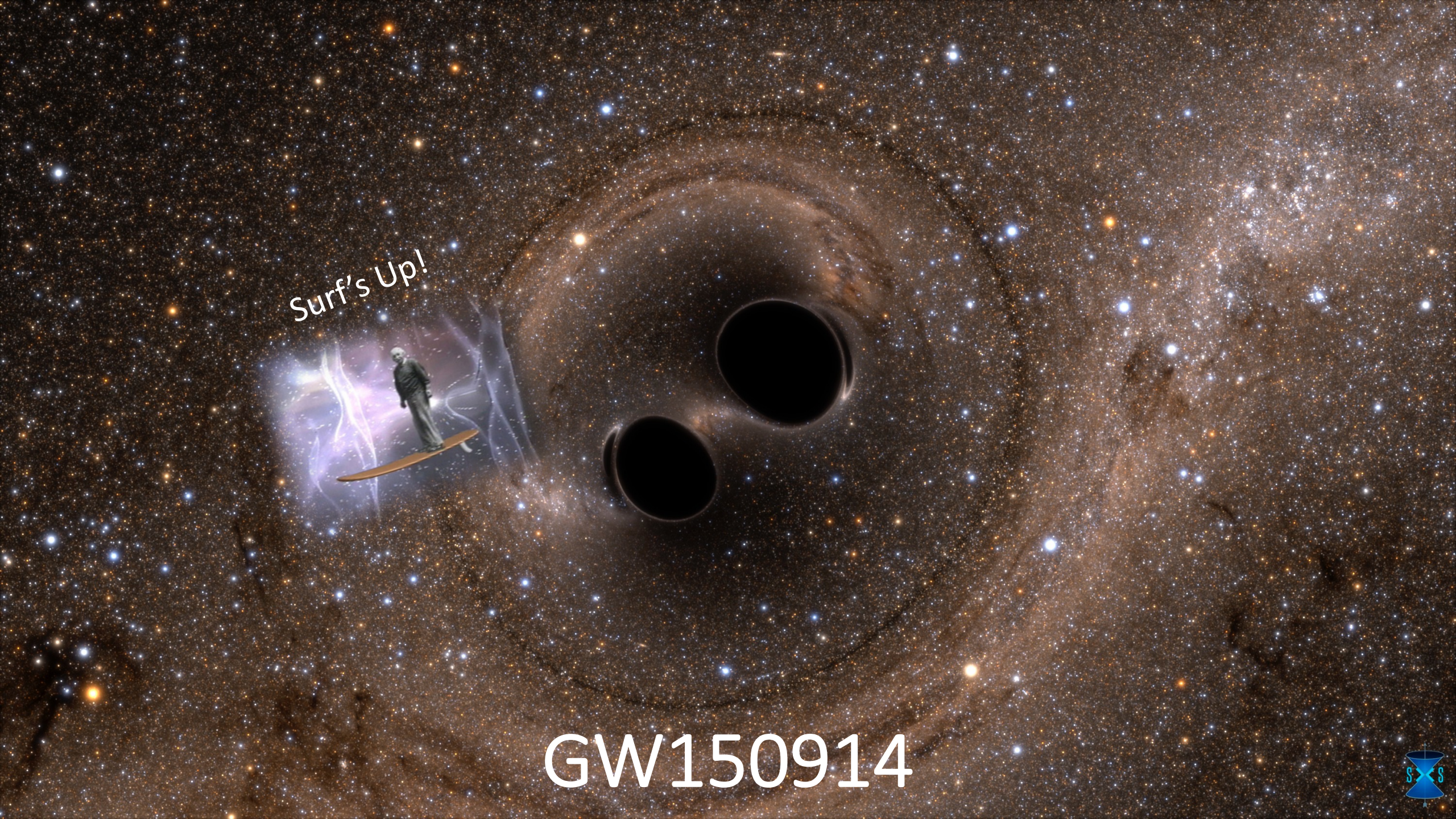
$$\hat{X}^2 = -\hat{Y}^2 = \hat{Z}^2 = \hat{I}$$

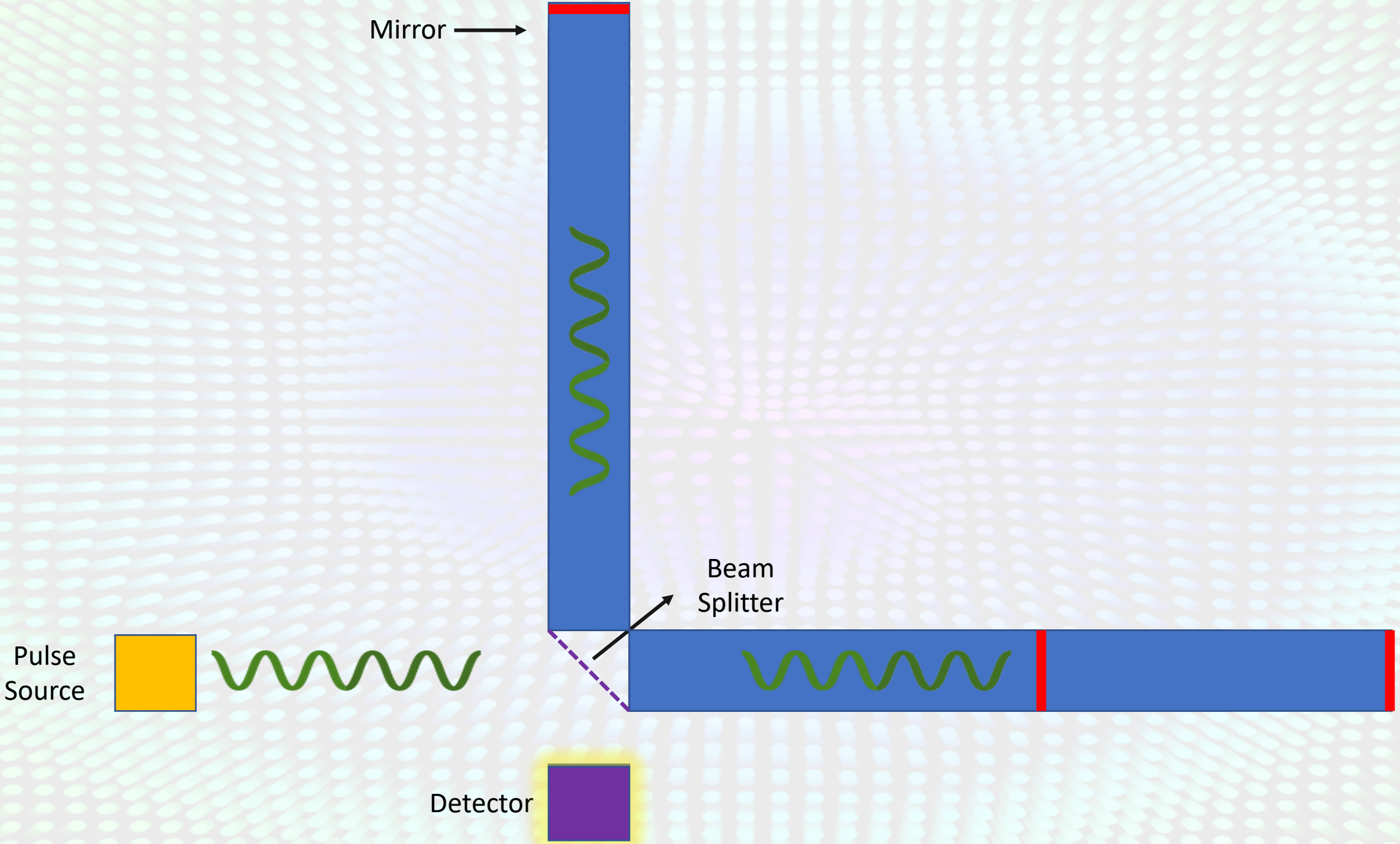
$$\hat{X}\hat{Z} = -\hat{Z}\hat{X}$$

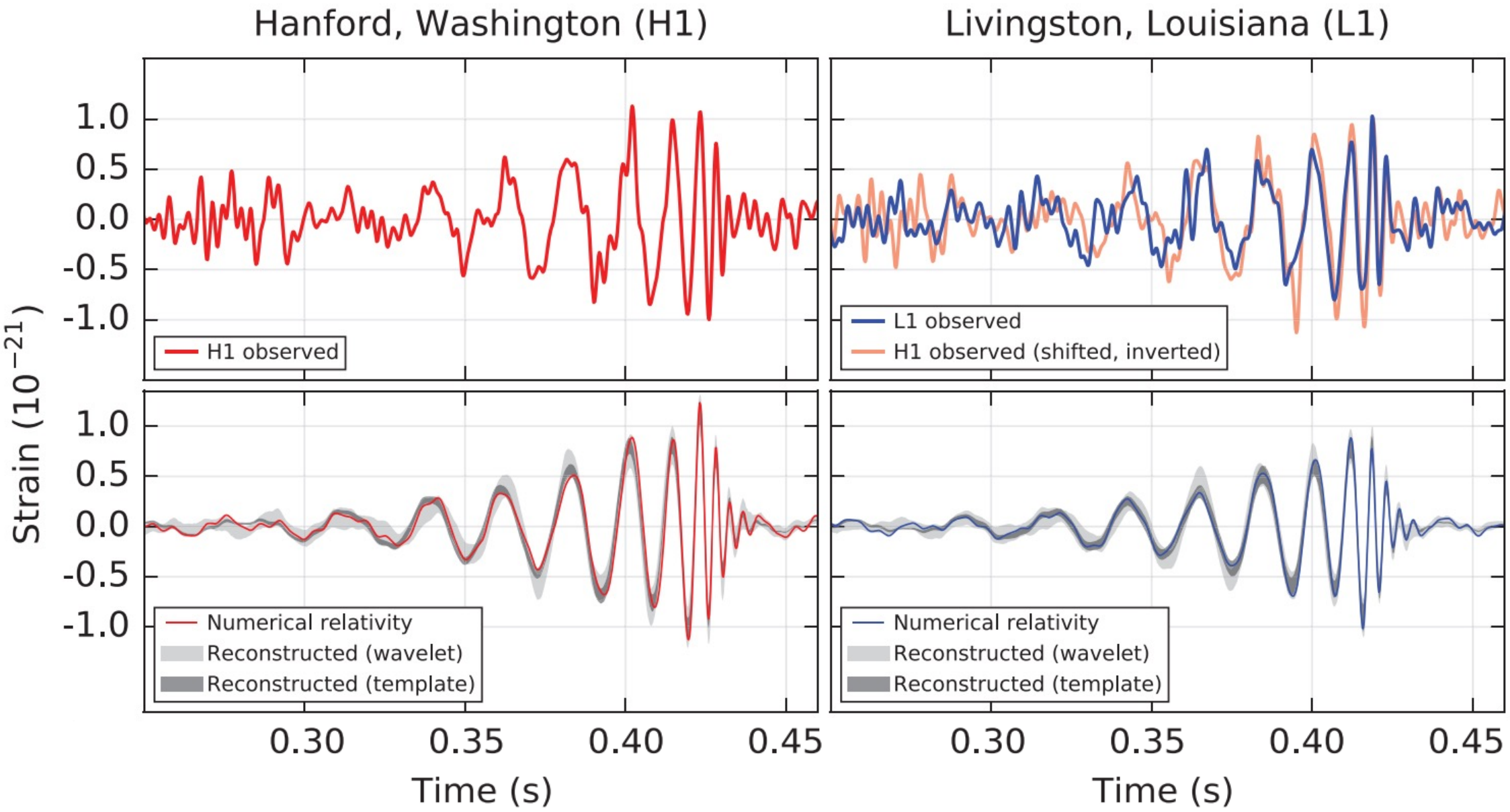
$$[\hat{X},\hat{Y}] \equiv \hat{X}\hat{Y} - \hat{Y}\hat{X} = -2\hat{Z}$$

Questions?


A QUANTUM
COMPUTING EXPEDITION
The Search for Gravitational Waves


Surf's Up!
GW150914


Mirror
Beam
Splitter
Pulse
Source
Detector



Abbott, B. P., et al. 2016. **"Observation of Gravitational Waves from a Binary Black Hole Merger."** Physical Review Letters. American Physical Society (APS).

Matched Filtering

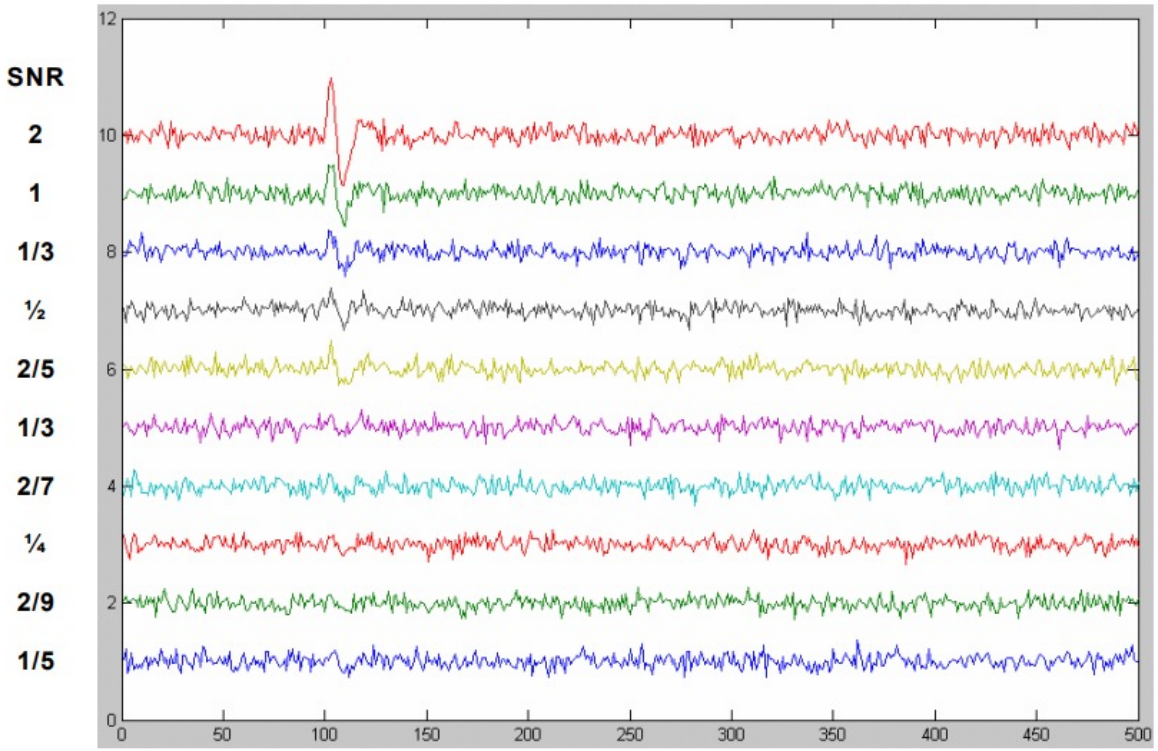


FIG. 3: Ten noisy traces and wavelets with a SNR that varies from 2.0 to 0.2.

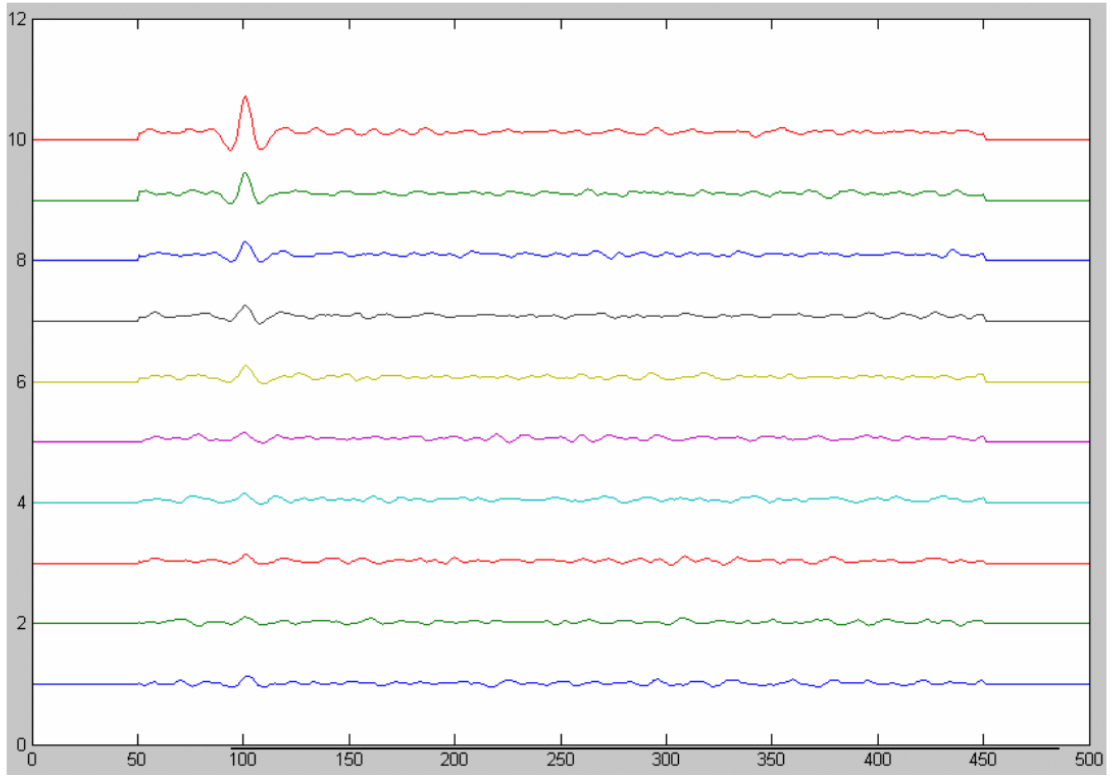


FIG. 4: Results of a matched filter from cross-correlating the wavelet in Figure 2 with the noisy signals in Figure 3,

J. C. Bancroft, Introduction to matched filters - Crewes, 2002

A quantum algorithm for gravitational wave matched filtering

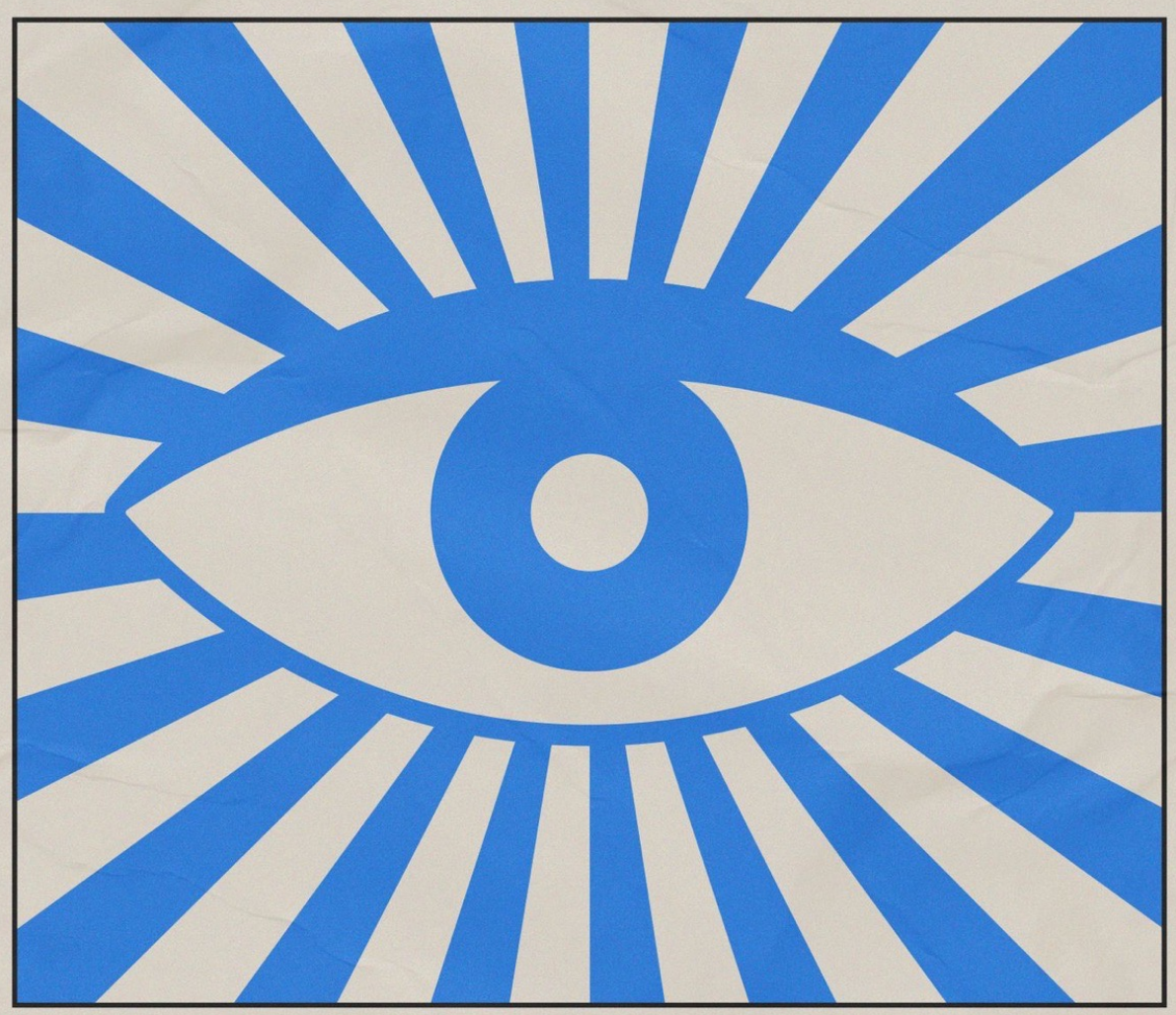
Sijia Gao [1,*] Fergus Hayes [1,†] Sarah Croke [1] Chris Messenger [1] and John Veitch [1]

[1] *SUPA, School of Physics and Astronomy, University of Glasgow, Glasgow G12 8QQ, United Kingdom*

(Dated: September 6, 2021)

Quantum computational devices, currently under development, have the potential to accelerate data analysis techniques beyond the ability of any classical algorithm. We propose the application of a quantum algorithm for the detection of unknown signals in noisy data. We apply Grover's algorithm to matched-filtering, a signal processing technique that compares data to a number of candidate signal templates. In comparison to the classical method, this provides a speed-up proportional to the square-root of the number of templates, which would make possible otherwise intractable searches. We demonstrate both a proof-of-principle quantum circuit implementation, and a simulation of the algorithm's application to the detection of the first gravitational wave signal GW150914. We discuss the time complexity and space requirements of our algorithm as well as its implications for the currently computationally-limited searches for continuous gravitational waves.

PACS numbers: 03.67.Ac; 04.30.-w; 07.05.Kf
Keywords: Quantum algorithm, matched filtering, Grover's algorithm, gravitational waves, continuous waves, data analysis



The Oracle

The Set Up

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle,$$

$|s\rangle$ = The superposition of all theoretical data templates in a template bank.
$|i\rangle$ = The i-th template in your template bank
N = The number of templates in your template bank

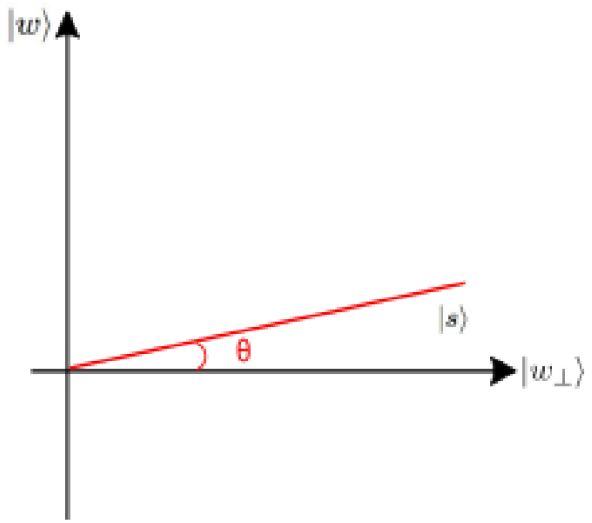
$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle, \longrightarrow |s\rangle = \sqrt{\frac{r}{N}}|w\rangle + \sqrt{\frac{N-r}{N}}|w_\perp\rangle.$$

$|w\rangle$ = The superposition of all the templates that would be **marked** as matching templates.

$|w_\perp\rangle$ = The superposition of all the other templates in our bank.

r = Number of matching templates

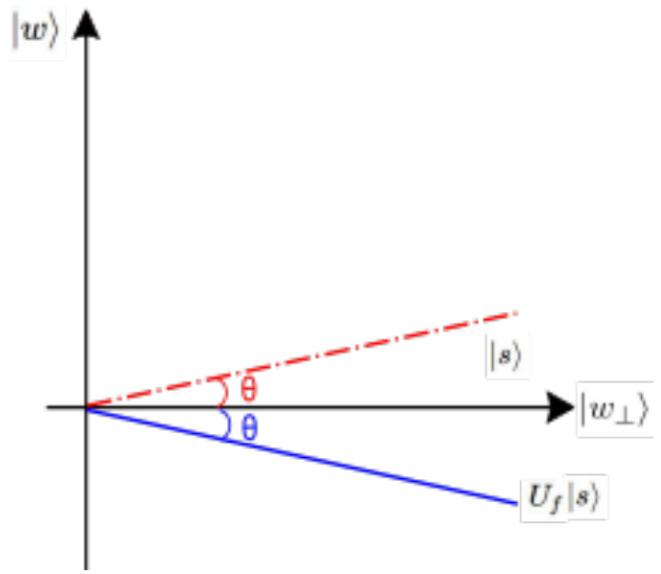
$$\theta = \arcsin\left(\langle w|s\rangle\right) = \arcsin\left(\sqrt{\frac{r}{N}}\right).$$



(a) The input state, represented by the red line.

We then apply an oracle U_f that is constructed to mark a solution with a negative sign.

$$U_f|s\rangle = -\sqrt{\frac{r}{N}}|w\rangle + \sqrt{\frac{N-r}{N}}|w_\perp\rangle,$$



(b) The state after the oracle is applied, represented by the blue line.

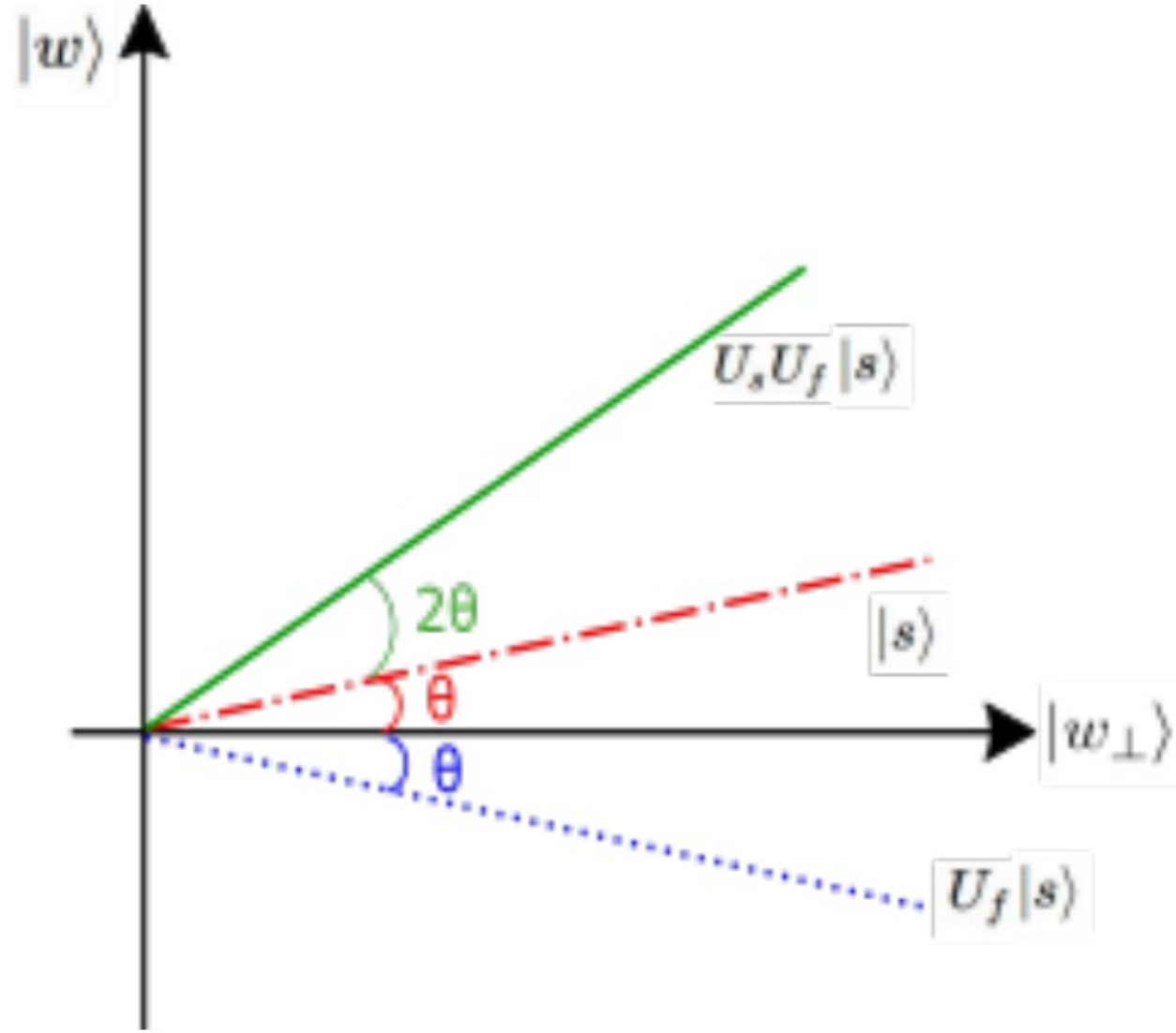
We finally apply the Diffusion Operator

$$U_s = 2|s\rangle\langle s| - \hat{\mathrm{I}},$$

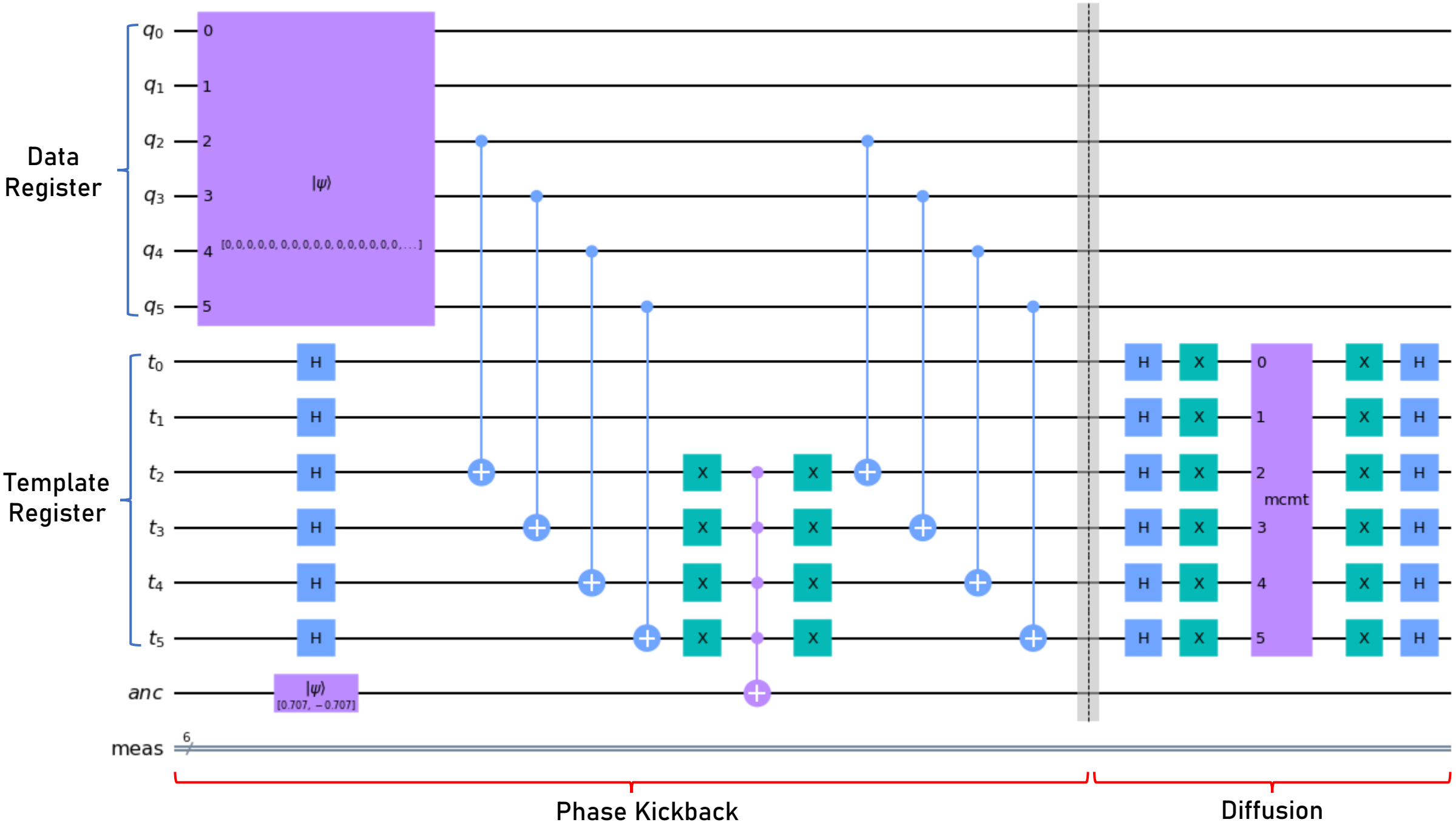
$$\mathrm{U}_s \left[-\sqrt{\tfrac{r}{N}}|w\rangle + \sqrt{\tfrac{N-r}{N}}|w_\perp\rangle \right] = (2|s\rangle\langle s| - I) \left[-\sqrt{\tfrac{r}{N}}|w\rangle + \sqrt{\tfrac{N-r}{N}}|w_\perp\rangle \right]$$

$$\left(3 - \tfrac{4r}{N}\right) \sqrt{\tfrac{r}{N}}|w\rangle + \left(1 - \tfrac{4r}{N}\right) \sqrt{\tfrac{N-r}{N}}|w_\perp\rangle$$

Matching Templates Superposition Amplified!

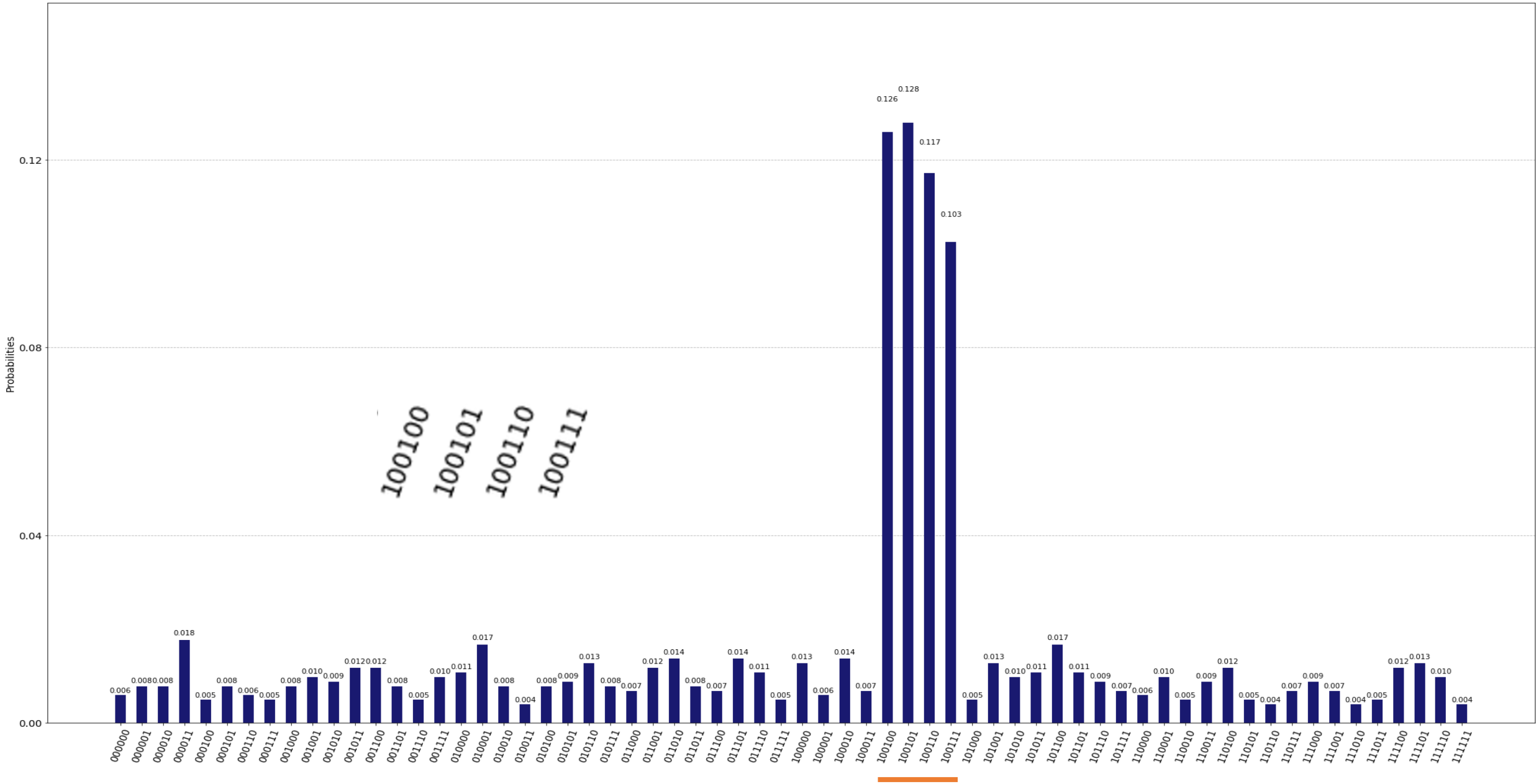


(c) The state after the diffusion operator, represented by the green line.

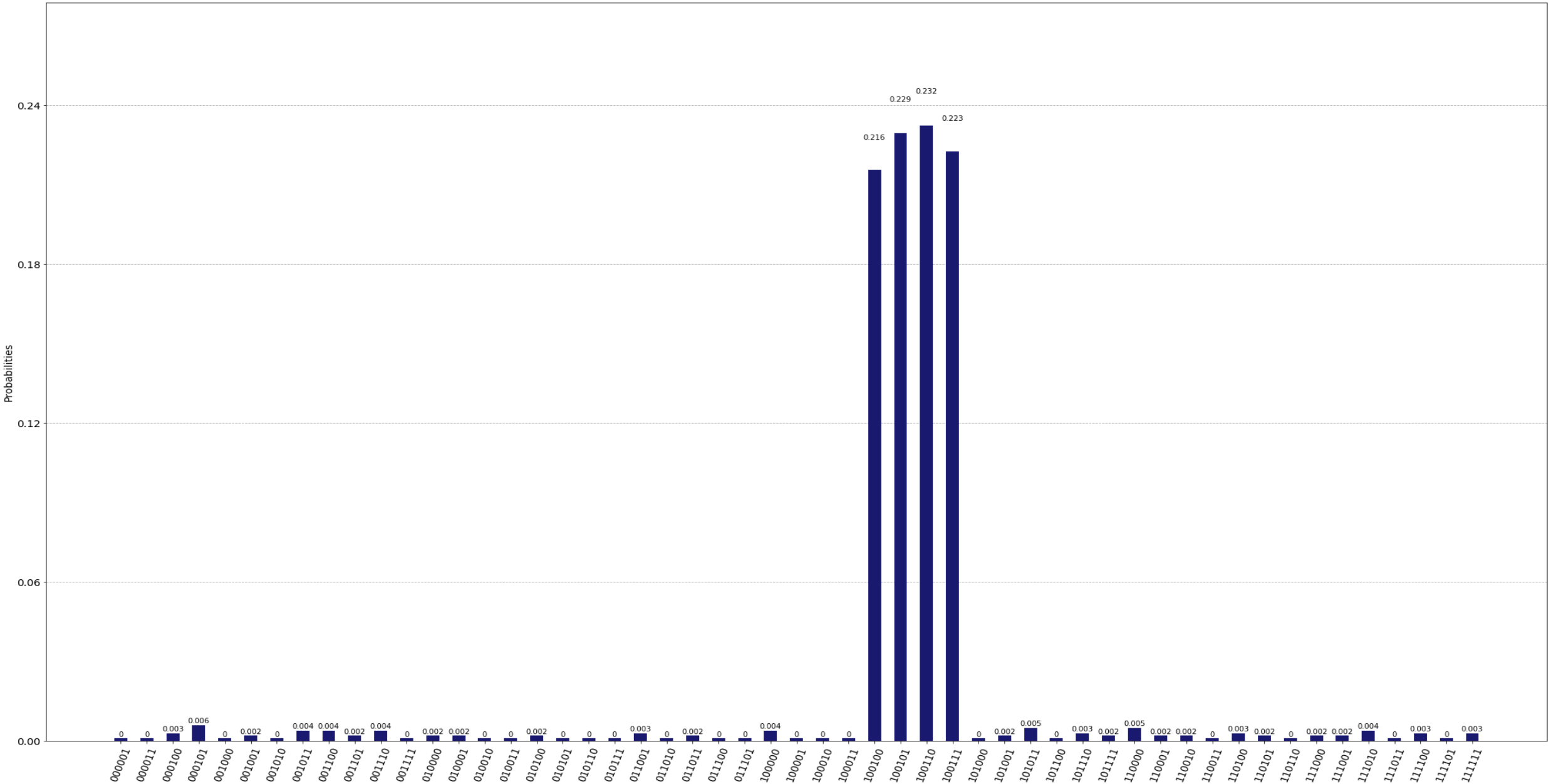

Data Register
Template Register
q_0
q_1
q_2
q_3
q_4
q_5
t_0
t_1
t_2
t_3
t_4
t_5
anc
meas
$|\psi\rangle$
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]
$|\psi\rangle$
[0.707, −0.707]
H
X
mcmt
6
Phase Kickback
Diffusion

Input Signal (Solution): 100101

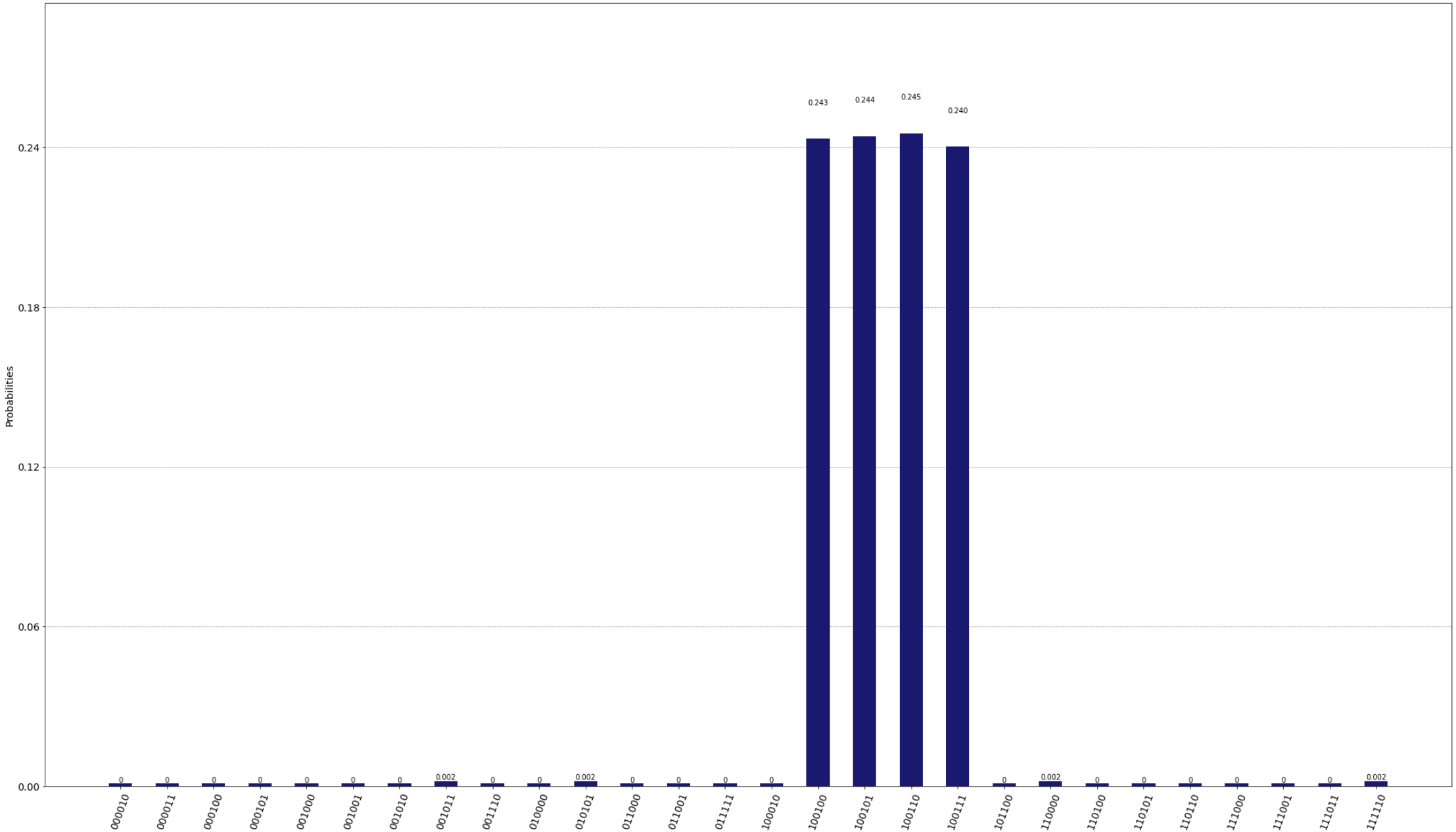
1 Grover Iteration



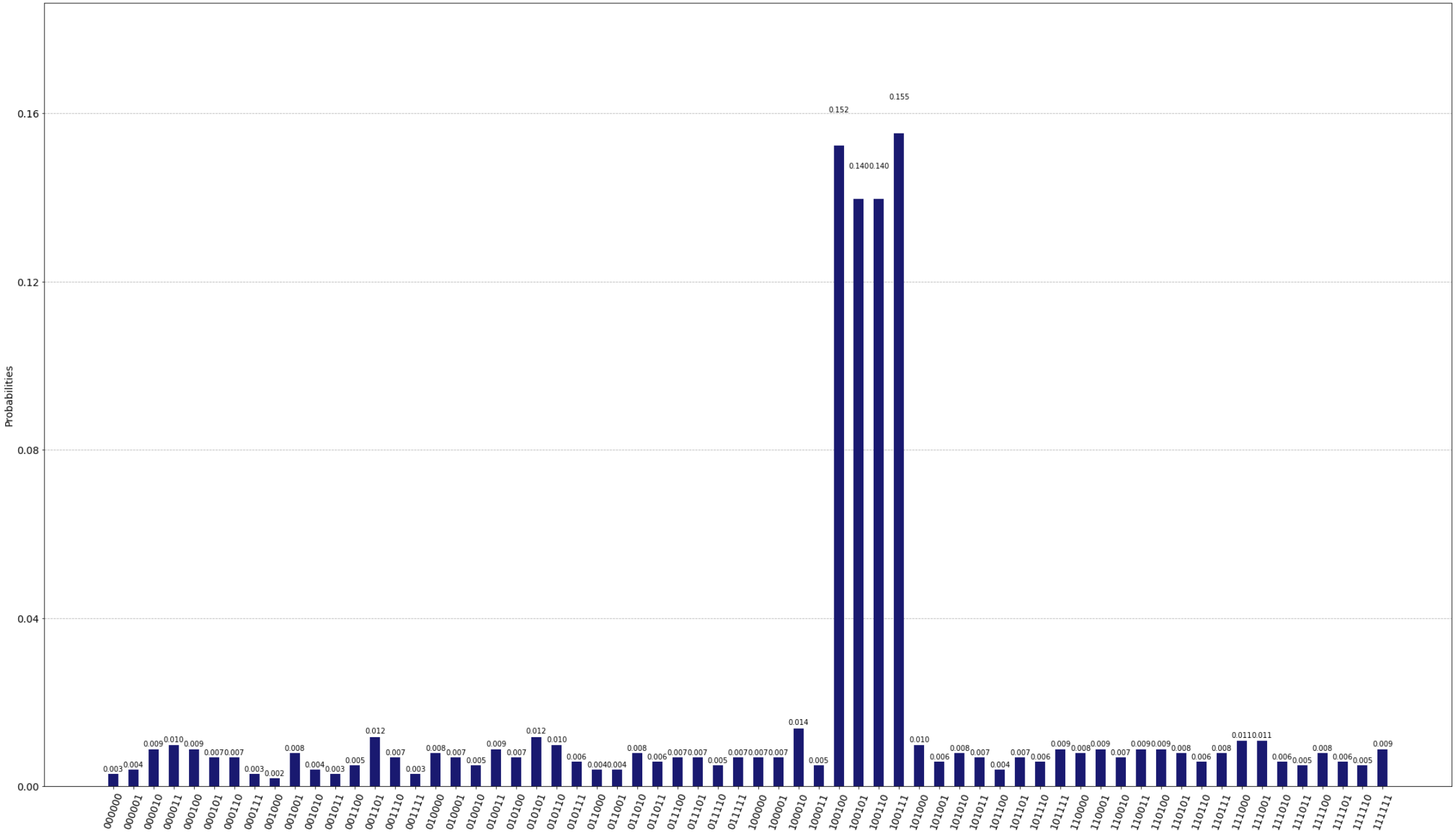
2 Grover Iterations

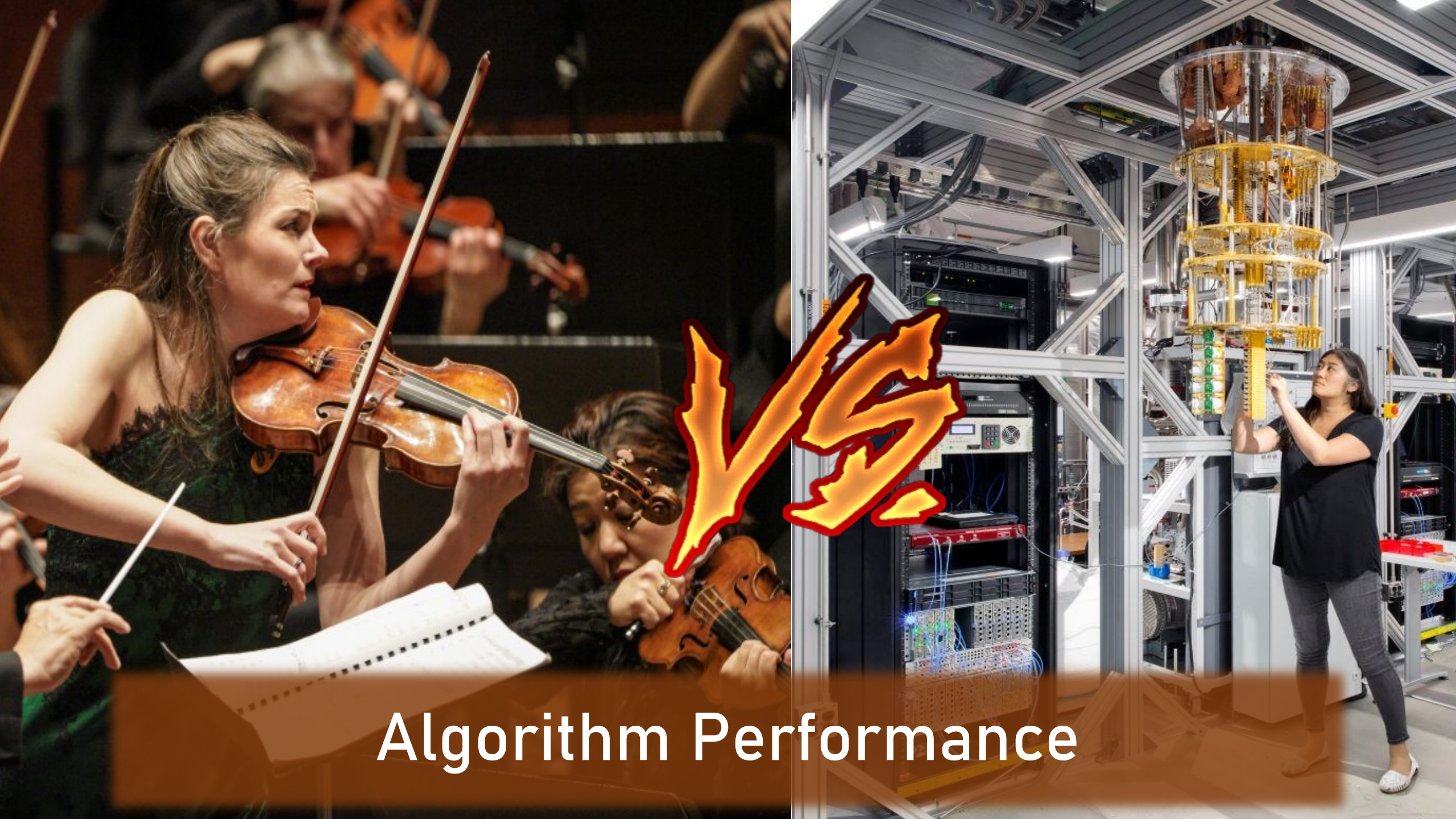


3 Grover Iterations

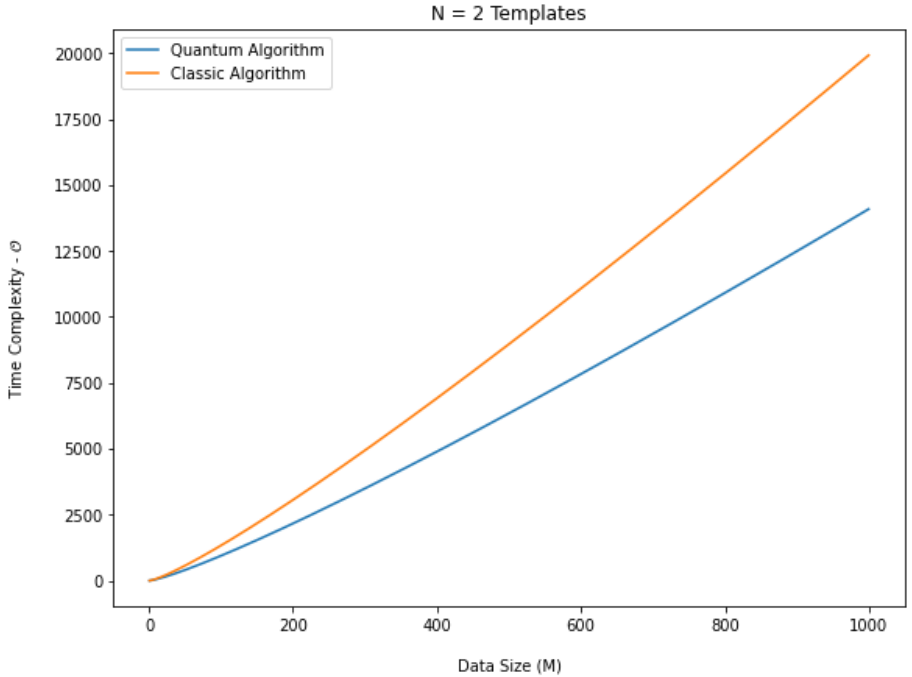


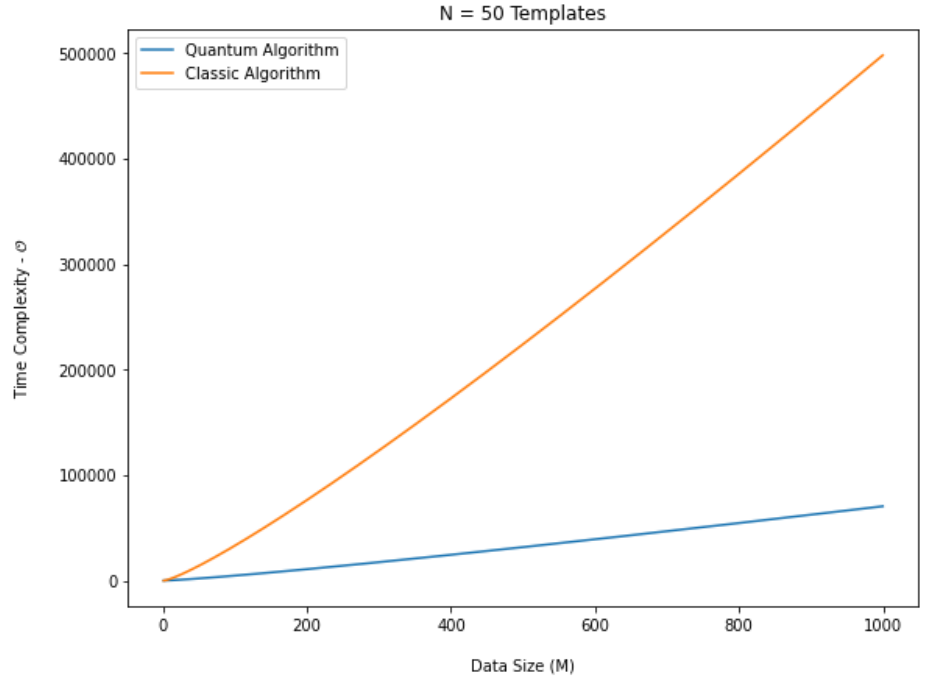
4 Grover Iterations

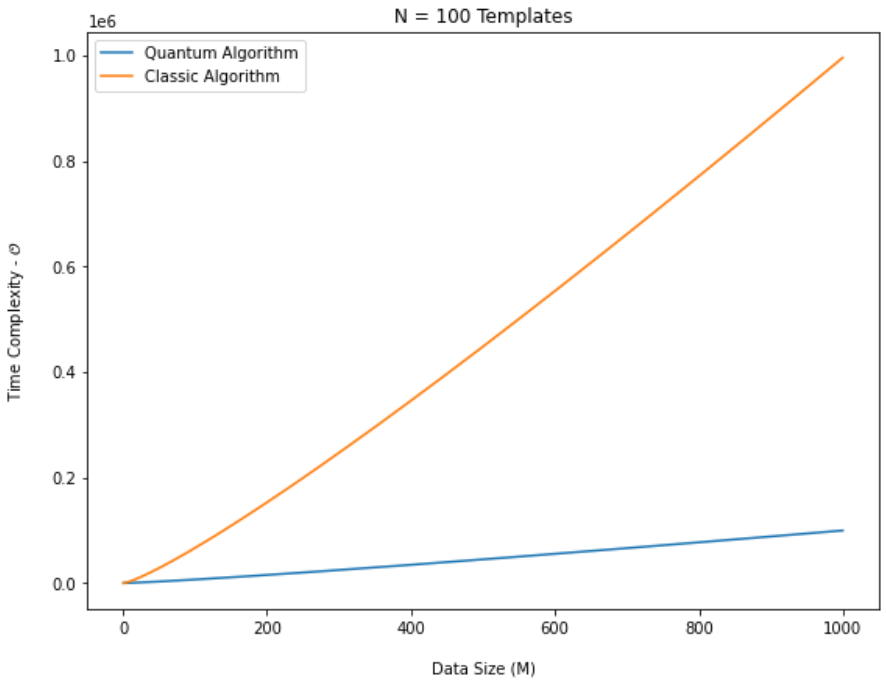


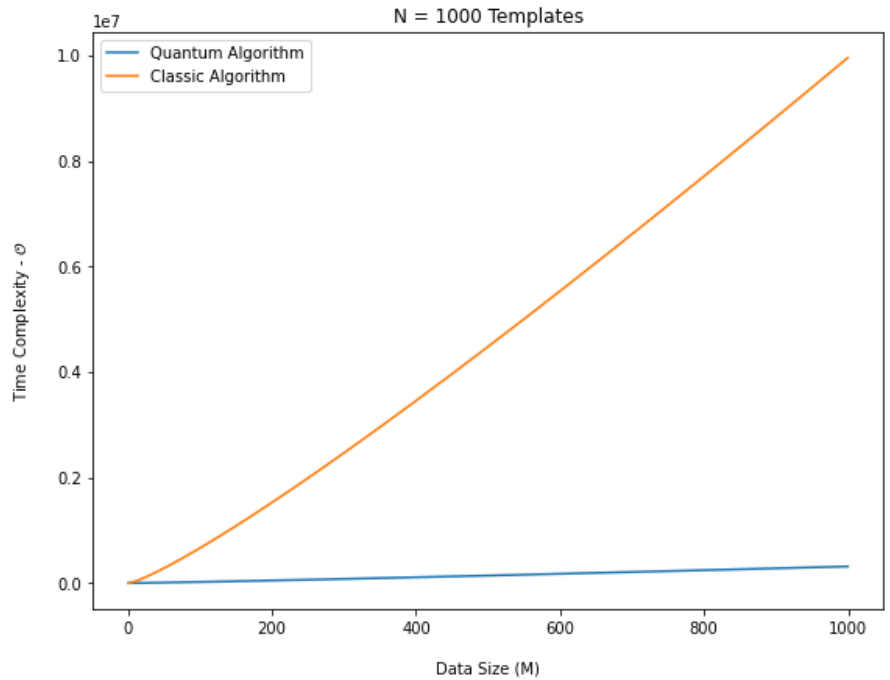


Algorithm Performance


N = 2 Templates
Quantum Algorithm
Classic Algorithm
Time Complexity - $\mathcal{O}$
Data Size (M)
0
2500
5000
7500
10000
12500
15000
17500
20000
0
200
400
600
800
1000


N = 50 Templates
Quantum Algorithm
Classic Algorithm
Time Complexity - $\mathcal{O}$
Data Size (M)
0
100000
200000
300000
400000
500000
0
200
400
600
800
1000


N = 100 Templates
1e6
Quantum Algorithm
Classic Algorithm
Time Complexity - $\mathcal{O}$
Data Size (M)
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000


N = 1000 Templates
1e7
Quantum Algorithm
Classic Algorithm
Time Complexity - $\mathcal{O}$
Data Size (M)
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000


YOUR ATTENTION
I THANK YOU FOR