Quantum Computing

Chapter 09: Quantum Optimization

Prachya Boonkwan

Version: January 14, 2026

Sirindhorn International Institute of Technology
Thammasat University, Thailand

Who? Me?

- Nickname: Arm (P'/N' Arm, etc.)
- Born: Aug 1981
- Work
  - Researcher at NECTEC 2005-2024
  - Lecturer at SIIT, Thammasat University 2025-now
- Education
  - B.Eng & M.Eng in Computer Engineering, Kasetsart University, Thailand
  - Obtained Ministry of Science and Technology Scholarship of Thailand in early 2008
  - Did a PhD in Informatics (AI & Computational Linguistics) at University of Edinburgh, UK from 2008 to 2013

Table of Contents

Simulated Annealing

Local and Global Optima

- An optimum refers to a point whose gradient is zero, and an optimum can be either minimum or maximum
  - A minimum has a positive curvature

$$\frac{\partial^2}{\partial \Theta^2} L \quad > \quad 0$$

  - A maximum has a negative curvature

$$\frac{\partial^2}{\partial \Theta^2} L \quad < \quad 0$$

- Local optimum does not guarantee the lowest prediction loss
- Global optimum is more preferable in parameter optimization

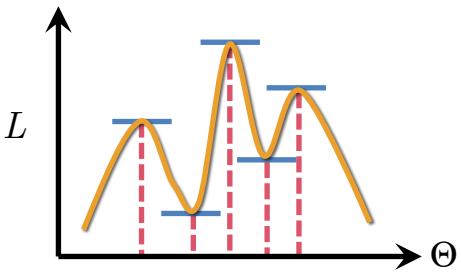


Figure 1: Local and global optima

Simulated Annealing in Classical Computing

- We approximate the global optimum by annealing the prediction loss

$$L_{\text{ann}}(\Theta, T) \quad = \quad \left[L(\Theta)\right]^{1/T}$$

- Annealing refers to heating the metal or glass and allowing it to cool slowly, resulting in flatter prediction loss and wider coverage of area exploration

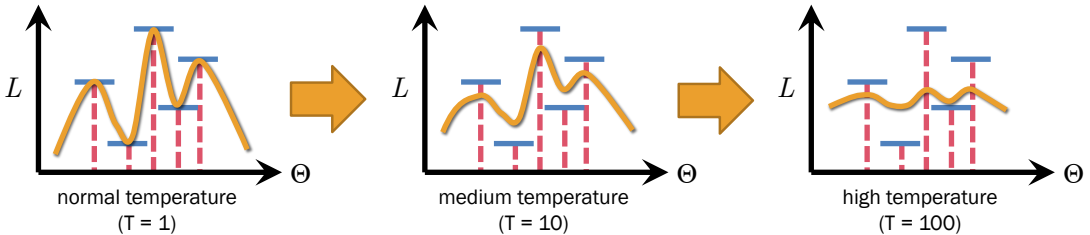


Figure 2: Effects of annealing the prediction loss. As the temperature increases, the hills get smaller, making it easier to hop over them for any gradient-based algorithm.

Simulated Annealing in Classical Computing

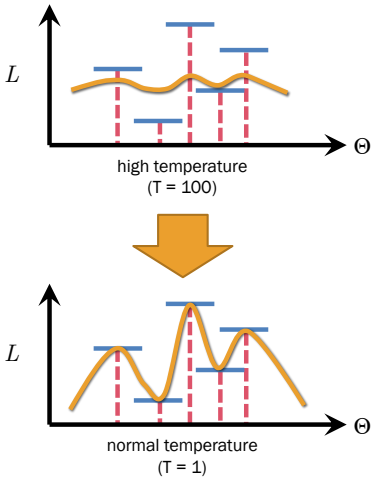


Figure 3: Simulated annealing

- Simulated annealing is a gradient-based optimization algorithm with the temperature and learning rate gradually decreased during the optimization process
  1. Begin with high temperature $T = T_{\max}$
  2. Randomize a starting point $\Theta^{(0)}$
  3. Compute a new parameter

$$\Theta^{(k+1)} = \Theta^{(k)} - \eta \frac{\partial}{\partial \Theta} L_{\text{ann}}(\Theta, T)$$

where the gradient of the annealed prediction loss is

$$\frac{\partial}{\partial \Theta} L_{\text{ann}}(\Theta, T) = \frac{\partial}{\partial \Theta} \left[L(\Theta) \right]^{1/T}$$

  4. Gradually decrease T and $\eta \Rightarrow$ scheduling
  5. Stop when the temperature becomes 1
  6. Otherwise, go to step 3

Applying Simulated Annealing to PennyLane: Training the XOR Model

```
loss_fn = N.MSELoss()
opt = O.Adam(xor_model.parameters(), lr=0.1)
scheduler = O.lr_scheduler.ExponentialLR(opt, gamma=0.95)        # Simulated annealing
no_iters = 30
batch_size = 20
loss_threshold = 1e-5

for i in range(no_iters):
    np.random.shuffle(training_data)
    total_loss = 0.0
    for j in range(no_training // batch_size):
        batch = training_data[j * batch_size : (j + 1) * batch_size]
        inmat = batch[:, 0:2]
        outmat = xor_model(inmat).flatten()
        goldmat = batch[:, 2]
        loss = loss_fn(outmat, goldmat)
        total_loss += loss.detach().data.item()
        opt.zero_grad()
        loss.backward()
        opt.step()
    scheduler.step()                                             # Reduce the temperature
    print(f'Total loss {i}: {loss.detach().data.item()}')
    if total_loss < loss_threshold: break
```

Exercise 9.1: Simulated Annealing

Answer the following questions.

1. Use a calculator to anneal the following values with the specified temperatures.

Values	$T = 100$	$T = 50$	$T = 10$	$T = 1$
0.001				
0.01				
0.1				
0.4				
0.9				
1.0				

Table 1: Simulated annealing of value L is $L^{1/T}$, where T is a temperature.

2. From the table above, explain why annealing helps solve the issue of barren plateaux manifested in the prediction loss.

Variational Quantum Eigensolver

Systems Energy and the Ground State

- In quantum chemistry, the stability of molecular structures is described in terms of energy level
- Stable structure has the lowest energy level

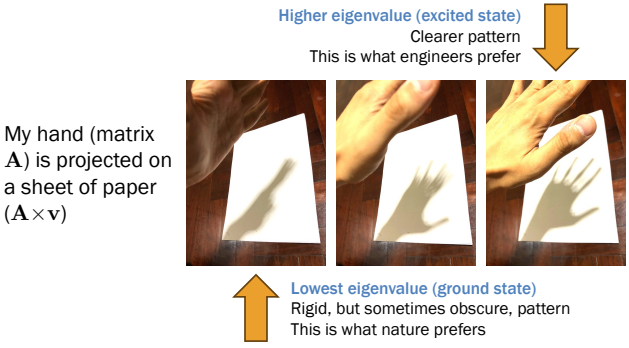


Figure 4: Excited state and ground state

- With this analogy, the optimal parameters will lower the systems energy down to the ground state
- For any state $|\psi\rangle = U(\Theta)$, the Rayleigh-Ritz principle holds:

$$E_0 \leq \frac{\langle \psi | \hat{\mathbf{H}} | \psi \rangle}{\langle \psi | \psi \rangle}$$

where $\hat{\mathbf{H}}$ is the Hamiltonian of $|\psi\rangle$, and E_0 is the ground energy

- We want to find the optimal parameters Θ such that $\mathbf{U}(\Theta)$ reaches the ground state $|\psi_0\rangle$

Ground State of a Hamiltonian

- Example: Let's find the ground state of a Hamiltonian

$$\hat{H} = \begin{bmatrix} 3 & 2 \\ 1 & 2 \end{bmatrix}$$

- First, we eigen-decompose it by equating

$$\begin{aligned} \det \begin{bmatrix} 3-\lambda & 2 \\ 1 & 2-\lambda \end{bmatrix} &= 0 \\ (3-\lambda)(2-\lambda)-2 &= 0 \\ \lambda^2-5\lambda+4 &= 0 \\ (\lambda-1)(\lambda-4) &= 0 \end{aligned}$$

- We obtain the following eigenvalues

$$\lambda_1 = 1 \qquad \lambda_2 = 4$$

- The ground energy is the eigenvalue with the lowest amplitude:

$$E_0 = \arg\min_k |\lambda_k| = 1$$

- We solve for the eigenvector of $E_0 = 1$

$$\left[\begin{array}{cc|c} 3-E_0 & 2 & 0 \\ 1 & 2-E_0 & 0 \end{array}\right] \sim \left[\begin{array}{cc|c} 1 & 1 & 0 \\ 0 & 0 & 0 \end{array}\right]$$

- Therefore, the ground state is $\frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

Adiabatic Theorem

- If a quantum system starts in a ground state of its Hamiltonian $\hat{\mathbf{H}}$, and its $\hat{\mathbf{H}}$ evolves within a certain limit $T_{\max}$, the system will remain in the ground state throughout the evolution
  - *adiabatic* [ˌeɪˌdaɪəˈbædɪk] (adj.) relating to a condition in which heat does not enter or escape the system concerned
- For each t-th iteration of time evolution $\left|\Psi^{(t)}\right\rangle$, we eigen-decompose it to $(\lambda_0, \mathbf{v}_0)$, $(\lambda_1, \mathbf{v}_1)$, ..., $(\lambda_N, \mathbf{v}_N)$, where $\lambda_0 < ... < \lambda_N$
- The energy gap for $\left|\Psi^{(t)}\right\rangle$ is defined as:

$$\Delta E(t) \quad = \quad \lambda_1 - \lambda_0$$

- Maximum limit of iterations $T_{\max}$ is

$$T_{\max} \quad = \quad O(1/(\Delta E_{\min})^2)$$

where $\Delta E_{\min}$ is the minimum energy gap

$$\Delta E_{\min} \quad = \quad \min_{1 \le t \le T} \Delta E(t)$$

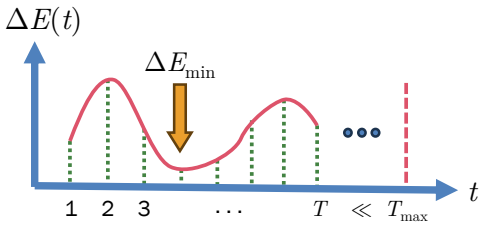


Figure 5: Adiabatic theorem

Adiabatic Quantum Computing (AQC)

- Adiabatic quantum computing is a quantum algorithm for simulation based on the adiabatic theorem, whereby time evolution is constrained by the energy gap
- By the Power Method, repetitive state transitions will converge to an eigenvector:

$$\mathrm{U}(\mathrm{U}^N\mathbf{x}) \approx \lambda(\mathrm{U}^N\mathbf{x}) = \mathbf{v}$$

where N is very large, $\mathbf{v}$ is an eigenvector, and λ is its corresponding eigenvalue

- If we want to stay at the ground state (least dominant eigenvector), every state transition must be subtle; otherwise, the system will jump to an excited state

- Assumption: Every time we make a state transition, we should NOT exceed the energy gap, w.r.t. the Adiabatic Theorem

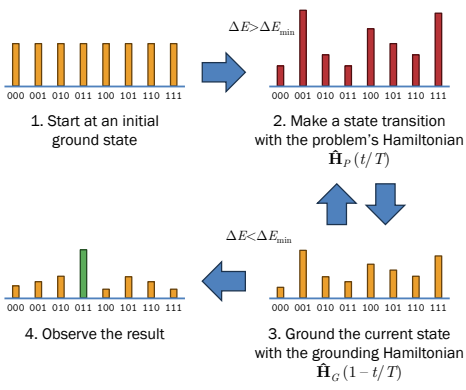


Figure 6: Adiabatic quantum computing

Adiabatic Quantum Computing (cont'd)

- Objective: We want to solve a combinatory optimization problem, where the solution $\mathbf{x}^*$ yields the ground energy
- Step 1 (preparation): We start from a ground state, which should be easy to prepare; for example,

$$|\psi_0\rangle = |+\rangle^{\otimes N}$$

- Step 2 (problem encoding): We encode our hard problem as a quantum algorithm with Hamiltonian $\hat{\mathrm{H}}_P$
- Step 3 (state grounding): We encode our state grounding method as a quantum algorithm with Hamiltonian $\hat{\mathrm{H}}_G$
- Step 4 (adiabatic transition): We iteratively make a state transition by trotterization:

$$\begin{aligned} \hat{\mathrm{H}}^{(t)} &= \left(1 - \frac{t}{T}\right)\hat{\mathrm{H}}_G + \frac{t}{T}\hat{\mathrm{H}}_P \\ |\psi_{t+1}\rangle &= \mathrm{cis}\left(-\hat{\mathrm{H}}^{(t)}\right)|\psi_t\rangle \end{aligned}$$

- Step 5 (measurement): We measure the solution from the final state $|\psi_T\rangle$

$$\mathbf{x}^* = \arg\max_{\mathbf{x}\in\{0,1\}^N} P(\mathbf{x}|\psi_T)$$

- Why it works? Each state transition is always within the energy gap

Adiabatic Quantum Computing in PennyLane/1

```
import pennylane as qml
from pennylane import numpy as np

wires = ['x1', 'x2']
dev = qml.device('default.qubit', wires=wires)

# Grounding Hamiltonian: H_G = - X(x1) - X(x2)
hmlt_grounding = qml.Hamiltonian([-1, -1], [qml.X(wires='x1'), qml.X(wires='x2')])
# Problem Hamiltonian: H_P = - Z(x1) Z(x2)
hmlt_problem = qml.Hamiltonian([-1], [qml.Z(wires='x1') @ qml.Z(wires='x2')])

# Total number of steps
no_steps = 100

# Maximum time
t_max = 10

# One step of adiabatic transition
def adiabatic_transition(t, t_max, hmlt_grounding, hmlt_problem):
    return (1 - t / t_max) * hmlt_grounding + (t / t_max) * hmlt_problem
```

Adiabatic Quantum Computing in PennyLane/2

```
@qml.qnode(dev, shots=1000)
def circuit():
    # Ground state = |+>
    qml.Hadamard(wires='x1')
    qml.Hadamard(wires='x2')
    # Adiabatic evolution
    delta_t = t_max / no_steps
    for i in range(no_steps):
        # Compute the Hamiltonian for state transition t
        t = i * delta_t
        hmlt_t = adiabatic_transition(t, t_max, hmlt_grounding, hmlt_problem)
        # Perform state transition with the precomputed Hamiltonian
        qml.ApproxTimeEvolution(hmlt_t, time=delta_t, n=1)
    # Result measurement
    return qml.probs(wires=wires)

probs = circuit()
print(f'State probability:\n{probs}')
```

Variational Quantum Eigensolver (VQE)

- Variational quantum eigensolver is a variational quantum algorithm that minimizes the ground energy, so that we obtain the optimal parameters Θ at the ground state
- The main benefit of VQE is that we can approximate the globally optimal parameters via the ground energy in each iteration

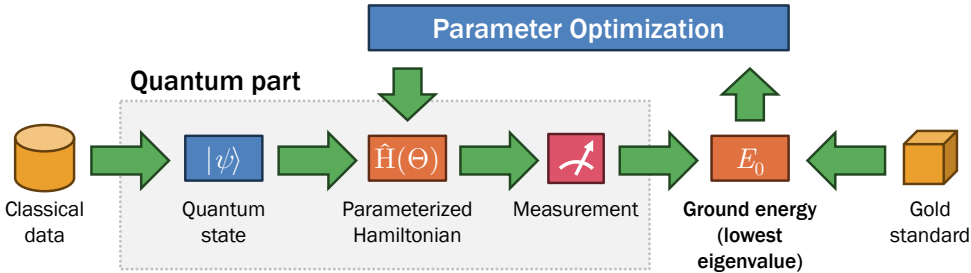


Figure 7: Variational quantum eigensolver (VQE)

Exercise 9.2: Energy State and Adiabatic Quantum Computing

Compute the ground energy and ground state of the following Hamiltonians.

1. $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
2. $\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
3. $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
4. $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Answer the following questions.

5. In adiabatic quantum computing, why do we prefer slow evolution? What happens if the time evolution progresses too fast?
6. Modify the AQC code aforementioned, such that

$$\begin{aligned} |\psi_0\rangle &= |+\rangle^{\otimes 2} \\ \hat{\mathrm{H}}_P &= -Z_1 - Z_2 + Z_1 Z_2 \\ \hat{\mathrm{H}}_G &= -X_1 - X_2 \end{aligned}$$

7. VQE seeks to find the optimal parameters that minimize the system's ground energy. In the context of hybrid ML, what would be comparable to the ground energy?

QUBO Problems

Quadratic Unconstrained Binary Optimization (QUBO) Problem

- QUBO is a combinatorial optimization problem with an objective function

$$\begin{aligned} G(\mathbf{x}|\mathbf{W}) &= \mathbf{x}^\top \mathbf{W}\mathbf{x} \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} W_{ij}x_i x_j \end{aligned}$$

  and $\mathbf{x} \in \{0,1\}^N$ is an input bitstring

- QUBO can be rewritten in the matrix form:

$$G(\mathbf{x}|\mathbf{W}', \mathbf{w}) = \mathbf{x}^\top \mathbf{W}'\mathbf{x} + \mathbf{w}^\top \mathbf{x}$$

  where $\mathbf{W}' \in \mathbb{R}^{N\times N}$ are non-diagonal parameters, $\mathbf{w} \in \mathbb{R}^N$ are diagonal parameters, and $\mathbf{W} = \mathbf{W}' + \mathrm{diag}(\mathbf{w})$

- In some cases, we define $\mathbf{W}$ as a right-upper matrix, obtaining

$$G(\mathbf{x}|\mathbf{W}) = \sum_{i=1}^{N}\sum_{j=i}^{N} W_{ij}x_i x_j$$

- Example: Suppose we let

$$\mathbf{W} = \begin{bmatrix} 1 & -3 & -4 \\ 0 & 2 & -2 \\ 0 & 0 & 5 \end{bmatrix}$$

  Our QUBO problem is

$$\begin{aligned} \mathbf{x}^* = \arg\min_{\mathbf{x}\in\{0,1\}^3} \big[& x_1^2 - 3x_1x_2 - 4x_1x_3 \\ & + 2x_2^2 - 2x_2x_3 + 5x_3^2 \big] \end{aligned}$$

QUBO and Graph Structure

- Since the objective function of QUBO is

$$G(\mathbf{x}|\mathbf{W}) \quad = \quad \sum_{i=1}^{N}\sum_{j=1}^{N} W_{ij}x_i x_j$$

we can treat $\mathbf{W}$ as an adjacency matrix, where each W_{ij} is a weight of edge (i,j) in a graph structure

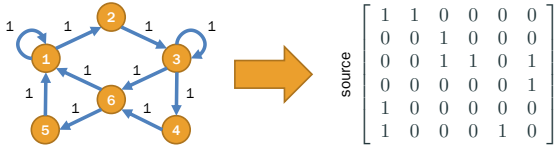


Figure 8: Adjacency matrix of a graph. Every edge has equal weight of 1.

- Equivalently, we can construct a quantum algorithm for $\mathbf{W}$ by treating edge (i,j) as an interaction between two qubits i and j
- Ising coupling gates cope with such interaction on different basis vectors

$$\begin{aligned}
\mathrm{XX}(\theta) &= \mathrm{cis}\left(-\frac{\theta}{2}\mathrm{X}\otimes\mathrm{X}\right)\\
\mathrm{YY}(\theta) &= \mathrm{cis}\left(-\frac{\theta}{2}\mathrm{Y}\otimes\mathrm{Y}\right)\\
\mathrm{ZZ}(\theta) &= \mathrm{cis}\left(-\frac{\theta}{2}\mathrm{Z}\otimes\mathrm{Z}\right)\\
\mathrm{XY}(\theta) &= \mathrm{cis}\left[-\frac{\theta}{4}\left(\mathrm{X}\otimes\mathrm{X}+\mathrm{Y}\otimes\mathrm{Y}\right)\right]
\end{aligned}$$

- For any self-loop (i,i), $\mathrm{ZZ}(\theta) = \mathrm{RZ}(\theta)$

Converting QUBO Problem to Ansatz

- We design an ansatz w.r.t. our objective function G by substituting each edge with an operator of Ising coupling gates
- In practice, we choose $\mathbf{ZZ}$ because Z-axis projection yields a value in $[-1, 1]$ for computing the objective function

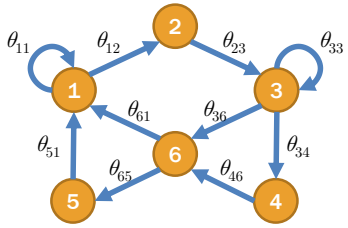


Figure 9: An example QUBO problem

- Our ansatz becomes

$$\mathbf{U}(\Theta) = \prod_{(i,i)\in E} \mathbf{RZ}_i^{(N)}(\theta_{ii}) \times \prod_{(i,j)\in E} \mathbf{ZZ}_{i,j}^{(N)}(\theta_{ij})$$

- Projective measurement of our ansatz is

$$\left\langle \mathbf{Z}^{(\mathbf{U}(\Theta)|x\rangle)} \right\rangle = \sum_{(i,i)\in E} \frac{\theta_{ii}}{2} Z_i^{(N)} + \sum_{(i,j)\in E} \frac{\theta_{ij}}{2} Z_i^{(N)} Z_j^{(N)}$$

where N is the total number of nodes, E is the set of edges, $\Theta = (\theta_{11}, \ldots, \theta_{NN})$, and Z_k is an observation of qubit k on the Z-axis (sometimes called a decision variable)

Computing the Objective Function of QUBO

- If we have an input vector $\mathbf{x}$, we have the output state

$$\mathbf{U}(\Theta)|x\rangle = \left[\text{cis}\left(-\sum_{\text{any }(i,j)} \hat{\mathbf{H}}_{ij}\right)\right]|x\rangle$$

- To compute the objective function for our parameters Θ, we have to enumerate all possible inputs $\mathbf{x}$ and measure their expectations on the Z-axis

$$\begin{aligned} Q(\Theta) &= \sum_{\mathbf{x}\in\{0,1\}^L} \left\langle \mathbf{Z}^{\mathbf{U}(\Theta)|x\rangle} \right\rangle \\ &= \sum_{\mathbf{x}\in\{0,1\}^L} \langle x|\,\mathbf{U}^*(\Theta)\,\mathbf{Z}\,\mathbf{U}(\Theta)\,|x\rangle \end{aligned}$$

- Now we can optimize the parameters Θ by any optimization algorithm

$$\Theta^* = \arg\min_{\Theta} Q(\Theta)$$

- The optimal argument $\mathbf{x}^*$ of G can be computed by

$$\begin{aligned} \mathbf{x}^* &= \sum_{\mathbf{x}\in\{0,1\}^L} p(\mathbf{x})\,|x\rangle \\ &= \mathbf{U}(\Theta^*)\left[\mathbf{H}^{\otimes L}|0\rangle^{\otimes L}\right] \end{aligned}$$

where $p(\mathbf{x})$ is the probability of input vector $\mathbf{x}$

Properties of QUBO Problems

- In the general form, QUBO problems are NP-hard $O(2^N)$ and cannot be solved within polynomial time
- There are special cases where QUBO can be solved in polynomial time
  - Case 1: If all $W_{ij} > 0$, the minimum is trivially $\mathbf{x}^* = \mathbf{0}$
  - Case 2: If all $W_{ij} < 0$, the minimum is trivially $\mathbf{x}^* = \mathbf{1}$
  - Case 3: If $\mathbf{W}$ is a diagonal matrix, then the problem can be solved in polynomial time
  - Case 4: If all non-diagonal elements of $\mathbf{W}$ are non-positive (i.e. zero or negative), then the problem can also be solved in polynomial time
- In classical computing, we solve a QUBO problem by converting it to a zero-one linear programming (i.e. mixture of linear regressions with constraints) and solving it with Linear Integer Programming
- Time complexity for solving a zero-one linear optimization is exponential in Lenstra's (1983) algorithm:

$$O\left(2^{O(n^3)} \cdot (m \cdot \log V)^{O(1)}\right)$$

where n is the number of variables, m is the number of equations, and V is the maximum absolute value of all coefficients in the equation system

Exercise 9.3: QUBO Problems

Reformulate the following objective functions as QUBO problems: $G(\mathbf{x}|\mathbf{W}) = \mathbf{x}^\top \mathbf{W}\mathbf{x}$, where $\mathbf{x} = [x_1, \ldots, x_N]^\top$.

1. $G(\mathbf{x}) = 3x_1x_2$
2. $G(\mathbf{x}) = 4x_1^2$
3. $G(\mathbf{x}) = -5x_2^2$
4. $G(\mathbf{x}) = x_1^2 - 2x_2^2$
5. $G(\mathbf{x}) = 9x_1^2 + 8x_1x_2 + 7x_2^2$
6. $G(\mathbf{x}) = 3x_1^2 + 4x_2^2 + 5x_3^2$
7. $G(\mathbf{x}) = 3x_1^2 - x_1x_2 + 4x_2^2 + 2x_2x_3 + x_3^2 - x_1x_3$
8. $G(\mathbf{x}) = 3x_1x_2 - 7x_2x_3 + 9x_3x_4 - 5x_1x_4$
9. $G(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2$
10. $G(\mathbf{x}) = x_1^2 + 2x_1x_3 + 3x_2^2 + 4x_2x_4 + 5x_1x_2x_3x_4$

Answer the following questions.

11. Convert the objective functions in questions 1-10 into equivalent ansatzes.
12. Convert the ansatzes in the previous question into corresponding projective measurement.
13. Explain why solving a QUBO problem in classical computing is of $O(2^N)$ time.
14. For a QUBO problem $G(\mathbf{x}|\mathbf{W}) = \mathbf{x}^\top \mathbf{W}\mathbf{x}$, show that if all $W_{ij} > 0$, we then have that $\arg\min_{\mathbf{x}} G(\mathbf{x}|\mathbf{W}) = \mathbf{0}$.
15. For a QUBO problem $G(\mathbf{x}|\mathbf{W}) = \mathbf{x}^\top \mathbf{W}\mathbf{x}$, show that if all $W_{ij} < 0$, we then have that $\arg\min_{\mathbf{x}} G(\mathbf{x}|\mathbf{W}) = \mathbf{1}$.

Quantum Approximate Optimization Algorithm

Transverse-Field Ising Model

- Ising model is a mathematical model of ferromagnetism in statistical mechanics
- Transverse field is a lattice of nearest neighbor interaction determined by the alignment and anti-alignment of atomic spin projections on the Z axis
- Neighboring spins that agree have a lower energy than those that disagree
- External heat and magnetic field disburb this tendency, creating a probability distribution of different structural phases

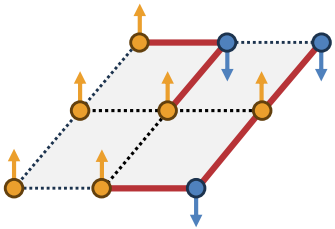


Figure 10: 2-dimensional Ising model. Dotted lines denote alignment of neighboring magnetic spins, while thick red lines denote their anti-alignment.

System Energy and Configuration

- System energy is described by the Hamiltonian function

$$H(\mathbf{s}|\mathbf{J},\mathbf{h}) = -\sum_{\text{any } i,j} J_{ij} s_i s_j - \mu \sum_j h_j s_j$$

where

  - configuration $\mathbf{s}$ is a vector of magnetic spins of all sites,
  - each member $s_j \in \{-1, +1\}$ is the spin at site j,
  - J_{ij} is the interaction between i and j,
  - μ is the magnetic moment, and
  - h_j is an external magnetic force at site j

- $(\mathbf{J},\mathbf{h})$ are the model's parameters

- The configuration probability is given by the Boltzmann distribution

$$P_\beta(\mathbf{s}|\mathbf{J},\mathbf{h}) = \frac{1}{Z_\beta} \exp\left[-\beta H(\mathbf{s}|\mathbf{J},\mathbf{h})\right]$$

where β is an inverse temperature and $Z_\beta = \sum_{\text{all } \mathbf{s}} P_\beta(\mathbf{s}|\mathbf{J},\mathbf{h})$ is a normalizing constant for all possible configurations

- We will borrow the idea of transverse field and spin interaction to describe the parameter interaction in a variational quantum algorithm

Quantum Annealing

- Quantum annealing is an optimization algorithm that seeks for the global minimum of QUBO through evolution
- We imitate space exploration of the simulated annealing by a superposition over the parameter space ⇒ the transverse field (extending across sth.)

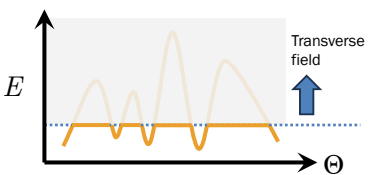


Figure 11: Quantum annealing

- Quantum annealing can also be seen as a kind of adiabatic quantum algorithms
- Procedure
  1. We start from a superposition of all possible states with equal weights.
  2. The system gradually evolves through time using a quantum operator that represents our QUBO problem.
  3. The amplitudes (weights) of the candidate states are manipulated by the prediction loss. We are exploring the entire parameter space in parallel.
  4. As the transverse field gradually subsides, the peaks and troughs start to appear.
  5. We choose the minimum amplitude as our solution.

Quantum Approximate Optimization Algorithm (QAOA)

- QAOA is a hybrid annealing algorithm that solves QUBO problems with trotterization
- Time evolution of QAOA consists of two steps: (1) parameter space revelation and (2) prediction loss minimization:

$$\hat{\mathbf{H}}(r) \quad = \quad (1-r)\hat{\mathbf{H}}_M + r\hat{\mathbf{H}}_L$$

where $r \in [0,1]$ is the ratio of the two steps, $\hat{\mathbf{H}}_M$ is the mixer Hamiltonian, and $\hat{\mathbf{H}}_L$ is the loss Hamiltonian (a.k.a. cost Hamiltonian)

- To imitate annealing, we interleave the mixer Hamiltonian and the Hamiltonian of the prediction loss for r times

- We trotterize QAOA into r steps

$$\mathbf{U}_{\text{QAOA}}(\mathbf{p},\mathbf{m},\Theta) \quad \approx \quad \prod_{k=1}^{r} \mathbf{U}_L(p_k,\Theta)\mathbf{U}_M(m_k)$$

where $\mathbf{U}_L$ is the ansatz for prediction loss

$$\mathbf{U}_L(p_k,\Theta) \quad = \quad \text{cis}[-p_k\hat{\mathbf{H}}_L(\Theta)]$$

and $\mathbf{U}_M$ is the mixer ansatz

$$\mathbf{U}_M(m_k) \quad = \quad \text{cis}[-m_k\hat{\mathbf{H}}_M]$$

- X-mixer Hamiltonian is the most popular

$$\hat{\mathbf{H}}_M \quad = \quad \sum_{j=1}^{N} \mathbf{X}_j^{(N)}$$

Quantum Approximate Optimization Algorithm (cont'd)

- We repeatedly compute the gradient of the prediction loss and update the parameters $(\mathbf{p}, \mathbf{m}, \Theta)$, until they converge to $(\mathbf{p}^*, \mathbf{m}^*, \Theta^*)$ and yield the globally minimum loss
- Output: the optimal solution $\mathbf{x}^* = \mathbf{U}_{\text{QAOA}}(\mathbf{p}^*, \mathbf{m}^*, \Theta^*)$

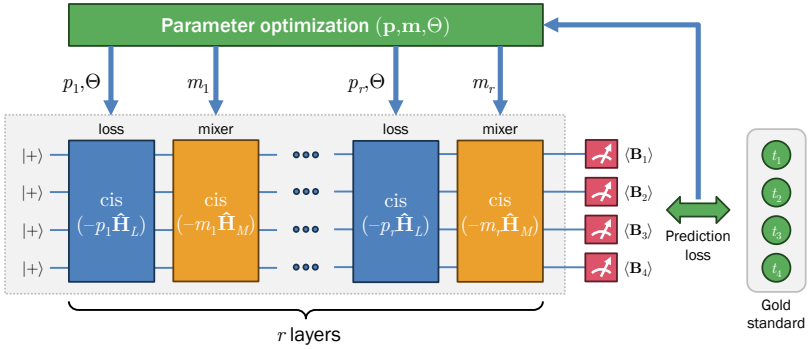


Figure 12: Quantum approximate optimization algorithm. Time complexity is significantly reduced to $O(rNMS)$, where M is the number of iterations and S is the number of labels.

Design Patterns of Loss Hamiltonians

Rewards:

- If our QUBO problem is conditioned on $x_i = 0$, we use the decision variable Z_i; otherwise (i.e. the condition is on $x_i = 1$), we use $-Z_i$ instead
- If our problem is conditioned on an interaction of input bits, we use the decision variable N_i, where

$$N_i = \frac{1}{2}(1 - Z_i)$$

which allows us to compute

$$N_i N_j = \frac{1}{4}(1 - Z_i - Z_j + Z_i Z_j)$$

Penalty:

- If there are constraints in our problem, we employ a penalty term

$$r = \rho \times (\text{constraint} - \text{target value})^2$$

where ρ is the penalty coefficient

- Because the penalty term must be large enough to demote invalid solutions to our problem, the rule of thumb for ρ is

$$\rho > \max(\text{objective terms})$$

- Example: The penalty coefficient for the objective terms $Z_1 + Z_2 + Z_3 Z_4$ is $\rho = 1 + 1 + 1 \times 1 = 3$

Design Patterns of Mixer Hamiltonian and Initial State

Mixer Hamiltonian

- The mixer must be able to transition from the initial state to any other state in the search space
- The mixer Hamiltonian $\hat{\mathbf{H}}_M$ must NOT commute with the loss Hamiltonian $\hat{\mathbf{H}}_L$;

$$[\hat{\mathbf{H}}_M, \hat{\mathbf{H}}_L] \neq 0$$

  If they commute, the QAOA will get stuck at some states

- If our problem is unconstrained, the most popular choice is the X-mixer Hamiltonian due to its capability of space exploration

Initial state

- If our problem is unconstrained, we can use the uniform superposition of all possible states

$$|\psi_0\rangle = \mathbf{H}^{\otimes N} |0\rangle^{\otimes N}$$

- If our problem is constrained, we should start from a feasible solution that satisfies the constraints
- The initial state should be easy to set up using simple quantum gates, such as $\mathbf{X}$ and $\mathbf{CNOT}$

Max-Cut Problem

- Max-cut problem is a graph-coloring problem, where all nodes are partitioned into two sets S and T so that the number of edges between S and T is maximum
- Imitating QUBO, each partition can be represented by a bitstring, e.g. 00101 means $S = \{1, 2, 4\}$ and $T = \{3, 5\}$
- Initial state: $|\psi_0\rangle = \mathrm{H}^{\otimes N} |0\rangle^{\otimes N}$

- Our loss Hamiltonian is

$$\begin{aligned} \hat{\mathrm{H}}_L &= \text{rewards of different colors} \\ &\quad - \text{penalty of similar colors} \\ &= \sum_{\text{any}(i,j)} \left(Z_i Z_j - I_i I_j\right) \end{aligned}$$

- We use the X-mixer: $\hat{\mathrm{H}}_M = \sum_{j=1}^{N} \mathrm{X}_j^{(N)}$

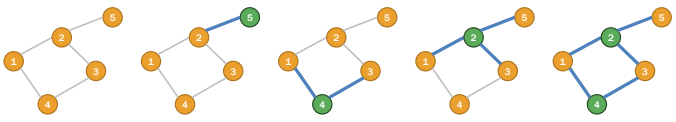


Figure 13: Max-cut problem. Thick blue lines represent cuts (edges linking nodes of different colors).

QAOA in PennyLane: Max-Cut Problem/1

```
import pennylane as qml
from pennylane import numpy as np
from networkx import Graph

##### SETTING UP A GRAPH #####

# We create a graph of 5 nodes and 10 edges
no_nodes = 5
no_edges = 10

# We create qubits of the same number as the nodes
wires = list(range(0, no_nodes))
dev = qml.device('default.qubit', wires=wires)

# We sample 10 edges of random source and target nodes
srcidxs = np.random.randint(no_nodes, size=no_edges).data
tgtidxs = np.random.randint(no_nodes, size=no_edges).data
graph = Graph([ (srcidxs[i], tgtidxs[i])
                for i in range(no_edges) ])

# We will use three QAOA layers
circuit_depth = 3
```

QAOA in PennyLane: Max-Cut Problem/2

```
##### MAXCUT PROBLEM #####

hmlt_loss, hmlt_mixer = qml.qaoa.maxcut(graph)    # Construct the loss and mixer Hamiltonians
                                                  # from our graph

def qaoa_layer(p, m):                             # This is one QAOA layer
    qml.qaoa.cost_layer(p, hmlt_loss)             # First, apply the loss Hamiltonian
    qml.qaoa.mixer_layer(m, hmlt_mixer)           # Then, apply the mixer Hamiltonian

def circuit(params):                              # QAOA circuit
    for i in wires:
        qml.Hadamard(i)
    qml.layer(qaoa_layer, circuit_depth, params[0], params[1])

@qml.qnode(dev, shots=100)
def loss_function(params):                        # Prediction loss yields expectation
    circuit(params)
    return qml.expval(hmlt_loss)

@qml.qnode(dev, shots=100)
def predict(params):                              # Prediction yields state counts
    circuit(params)
    return qml.counts(wires=wires)
```

QAOA in PennyLane: Max-Cut Problem/3

```
##### QUANTUM APPROXIMATE OPTIMIZATION ALGORITHM #####

# Model parameters: always 2 x circuit depth dimensions
# requires_grad=True means optimizable parameters
params = np.array(np.random.rand(2, circuit_depth), requires_grad=True)

# Stochastic gradient descent algorithm
opt = qml.GradientDescentOptimizer(stepsize=0.01)
no_epochs = 50

# Estimate the parameters for 50 epochs
for i in range(no_epochs):
    params, loss = opt.step_and_cost(loss_function, params)
    print(f'Loss {i} = {loss}')

# We have obtained the optimal parameters
print(f'Optimal parameters:\n{params}')

##### PREDICTION #####

counts = predict(params)
print(f'Probability distribution:\n{counts}')
```

Max-Clique Problem

- Max-clique problem is a graph-coloring problem, where all nodes are partitioned into S and T so that all nodes in T form the largest clique
- Clique is a graph whose all possible pairs of its nodes are linked by at least one edge
- Initial state: $|\psi_0\rangle = |0\rangle^{\otimes N}$

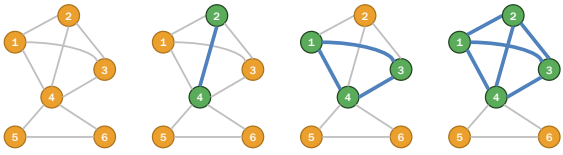


Figure 14: Max-clique problem

- Our loss Hamiltonian is

$$\begin{aligned}\hat{\mathrm{H}}_L &= \text{rewards of being in a clique} \\ &= \sum_{v \in V} Z_v\end{aligned}$$

- We use the bit-flipping mixer:

$$\hat{\mathrm{H}}_M = \sum_{v \in V} \left[\frac{1}{2^{d(v)}} X_v \prod_{w \in N(v)} \left(I + (-1)^b Z_w \right) \right]$$

where V is the set of all nodes, $d(v)$ is the degree of v, and $b \in \{0,1\}$ imposes bit-flipping only when $v = |b\rangle$

- `qml.max_clique(graph)`

Maximum-Weighted Cycle Problem

- Maximum-weighted cycle problem is a graph-coloring problem, where each node x_{ij} denotes an edge (i,j) with weight c_{ij}
- We convert a node graph $G=(V,E)$ into an edge graph $G'=(V',E')$ by
  - For each edge $(i,j)\in E$, add a node (i,j) into V'
  - For each edge $(i,k)\in E$, if there also exist $(i,j),(j,k)\in E$, add $((i,j),(j,k))$ into E'
- We partition these nodes into S and T so that all nodes in T form a cycle with the maximum score

$$P = \prod_{(i,j)\in E} \left[(c_{ij}-1)x_{ij}+1\right]$$

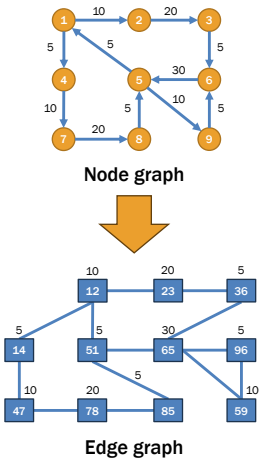


Figure 15: From node graph to edge graph

Maximum-Weighted Cycle Problem (cont'd)

- Initial state: $|\psi_0\rangle = \mathrm{H}^{\otimes N} |0\rangle^{\otimes N}$
- Our loss Hamiltonian is

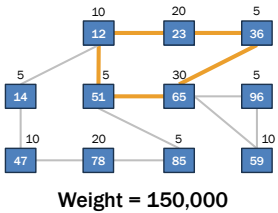
$$\begin{aligned} \hat{\mathrm{H}}_L &= \text{rewards of convenience} \\ &= \sum_{(i,j)\in E} Z_{ij} \log c_{ij} \end{aligned}$$

- We use the cycle mixer:

$$\hat{\mathrm{H}}_M = \sum_{(i,j),(i,k),(j,k)\in E} X_{ij}X_{ik}X_{jk} + Y_{ij}Y_{ik}X_{jk} + Y_{ij}X_{ik}Y_{jk} - X_{ij}Y_{ik}X_{jk}$$

where i, j, k are bit- and sign-flipped

- `qml.max_weight_cycle(graph)`



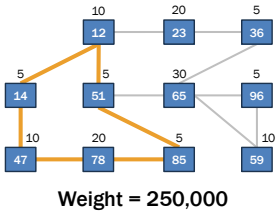


Figure 16: Max-weighted cycle problem

Maximum Independent Set Problem

- Maximum independent set problem is a graph-coloring problem, where all nodes are partitioned into S and T so that all nodes in T do not any neighbors in T
- Definition: For every node $v \in T$, its neighbors $N(v) \cap T = \emptyset$
- Initial state: $|\psi_0\rangle = |0\rangle^{\otimes N}$

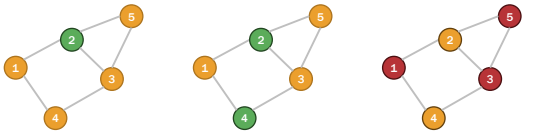


Figure 17: Maximum independent set problem

- Our loss Hamiltonian is

$$\begin{aligned} \hat{\mathrm{H}}_L &= \text{rewards of independence} \\ &= \sum_{v \in V} Z_v \end{aligned}$$

- We use the bit-flipping mixer:

$$\hat{\mathrm{H}}_M = \sum_{v \in V} \left[\frac{1}{2^{d(v)}} X_v \prod_{w \in N(v)} (I + (-1)^b Z_w) \right]$$

where V is the set of all nodes, $d(v)$ is the degree of v, and $b \in \{0,1\}$ imposes bit-flipping only when $v = |b\rangle$

- `qml.max_independent_set(graph)`

Minimum Vertex Cover Problem

- Minimum vertex cover problem is a graph-coloring problem, where all nodes are partitioned into S and T so that all nodes in T are the minimum vertex cover
- Vertex cover is a set of nodes that include at least one endpoint of every edge
- Initial state: $|\psi_0\rangle = |1\rangle^{\otimes N}$

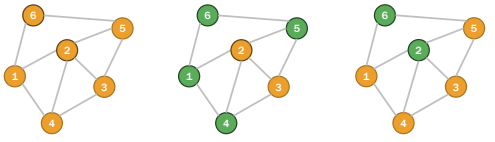


Figure 18: Minimum vertex cover problem

- Our loss Hamiltonian is

$$\begin{aligned}\hat{\mathrm{H}}_L &= \text{penalty of independence} \\ &= -\sum_{v\in V} Z_v\end{aligned}$$

- We use the bit-flipping mixer:

$$\hat{\mathrm{H}}_M = \sum_{v\in V}\left[\frac{1}{2^{d(v)}} X_v \prod_{w\in N(v)} (I+(-1)^b Z_w)\right]$$

where V is the set of all nodes, $d(v)$ is the degree of v, and $b \in \{0,1\}$ imposes bit-flipping only when $v = |b\rangle$

- `qml.min_vertex_cover(graph)`

Traveling Salesman Problem

- Traveling salesman problem (TSP) is a graph-coloring problem, where each node x_{it} denotes visiting the city i at timestep t
- We partition these nodes into S and T so that all nodes in T form a path that cover all nodes in V
- TSP is not included in PennyLane 0.43, so we have to prepare our own Hamiltonians

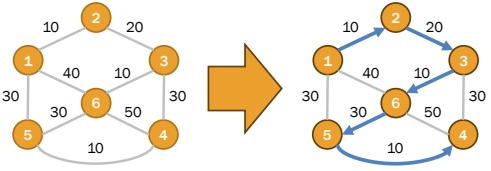


Figure 19: Traveling salesman problem

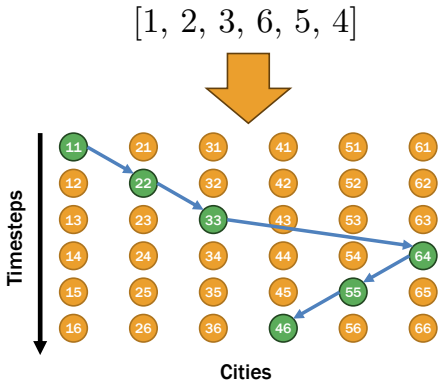


Figure 20: From sequence to timestep graph. The space complexity is increased from $O(n)$ to $O(n^2)$.

Traveling Salesman Problem (cont'd)

- For a node graph $G = (V, E)$, we convert a sequence $Q = [v_1, \ldots, v_N]$ into a timestep graph $G' = (V', E')$ by
  - For each node $v \in V$, add a node (v, t) for each timestep $1 \leq t \leq N$ into V'
  - For each element $1 \leq t < N$, add an edge $((v_t, t), (v_{t+1}, t+1))$ into E'
- We define a node-timestep Hamiltonian

$$N_{it} = \frac{1}{2}(I - Z_{it})$$

- Initial state: $|1\rangle$ for each city i and timestep i
- We use the X-mixer: $\hat{\mathbf{H}}_M = \sum_{j=1}^{N} \mathbf{X}_j^{(N)}$

- Our loss Hamiltonian is

$$\begin{aligned}\hat{\mathbf{H}}_L &= \text{rewards of convenience} \\ &\quad - \text{one-visit-per-city penalty} \\ &\quad - \text{one-city-per-timestep penalty} \\ &= \sum_{t=1}^{N-1} \sum_{(i,j)\in E} d_{ij} N_{i,t} N_{j,t+1} \\ &\quad + \sum_{i \in V} \left(\sum_{t=1}^{N} N_{it} - 1 \right)^2 \\ &\quad + \sum_{t=1}^{N} \left(\sum_{i \in V} N_{it} - 1 \right)^2\end{aligned}$$

where d_{ij} is the distance between i and j

Traveling Salesman Problem in PennyLane/1

```
import pennylane as qml
from pennylane import numpy as np

##### SETTING UP A DISTANCE MATRIX #####

no_cities = 4
# distance_matrix = np.random.randint(20, size=(no_cities, no_cities)) + 1
distance_matrix = np.array([ [10, 20, 30, 40],
                             [30, 10, 20, 10],
                             [60, 10, 70, 30],
                             [40, 30, 20, 10] ])

##### SETTING UP THE QUANTUM COMPUTER #####

wires = range(no_cities ** 2)
dev = qml.device('lightning.qubit', wires=wires)

def idx_qubit(cityidx, timestep):
    return cityidx * no_cities + timestep
```

Traveling Salesman Problem in PennyLane/2

```
##### TRAVELING SALESMAN PROBLEM #####

def compute_rewards(distance_matrix, no_cities):
    co_pairs = []
    for t in range(no_cities):
        t_next = (t + 1) % no_cities
        for i in range(no_cities):
            for j in range(no_cities):
                if i == j: continue
                idx_i = idx_qubit(i, t)
                idx_j = idx_qubit(j, t_next)
                # N[i,t] * N[j,t+1] = 1/4 (1 - Z[i] - Z[j] + Z[i] * Z[j])
                # We ignore the constant part
                co_pairs.extend([
                    (-0.25 * distance_matrix[i,j], qml.Z(idx_i)),
                    (-0.25 * distance_matrix[i,j], qml.Z(idx_j)),
                    (0.25 * distance_matrix[i,j], qml.Z(idx_i) @ qml.Z(idx_j))
                ])
    penalty = 3 * no_cities**3
    return co_pairs, penalty
```

Traveling Salesman Problem in PennyLane/3

```
def compute_penalty(distance_matrix, no_cities, penalty):
    co_pairs = []
    for i in range(no_cities):
        for t1 in range(no_cities):
            for t2 in range(t1 + 1, no_cities):
                idx_1 = idx_qubit(i, t1)
                idx_2 = idx_qubit(i, t2)
                # We peanlize two 1's in a row/column by
                # adding P / 2 * Z[i] * Z[j]
                co_pairs.append( (0.5 * penalty, qml.Z(idx_1) @ qml.Z(idx_2)) )
    return co_pairs

def compute_loss_hmlt(distance_matrix, no_cities):
    co_pairs_rewards, penalty = compute_rewards(distance_matrix, no_cities)
    co_pairs_penalty = compute_penalty(distance_matrix, no_cities, penalty)
    co_pairs = co_pairs_rewards + co_pairs_penalty
    coeffs = [pair[0] for pair in co_pairs]
    obsvs = [pair[1] for pair in co_pairs]
    return qml.Hamiltonian(coeffs, obsvs)

hmlt_loss = compute_loss_hmlt(distance_matrix, no_cities)
hmlt_mixer = qml.qaoa.x_mixer(wires)
```

Traveling Salesman Problem in PennyLane/4

```
##### QUANTUM CIRCUIT #####

def qaoa_layer(p, m):
    qml.qaoa.cost_layer(p, hmlt_loss)
    qml.qaoa.mixer_layer(m, hmlt_mixer)

def circuit(params):
    # Initial state: 1 for each city i and timestep i
    for i in range(no_cities):
        qml.X(idx_qubit(i, i))
    qml.layer(qaoa_layer, circuit_depth, params[0], params[1])

@qml.qnode(dev, shots=100)
def loss_function(params):
    circuit(params)
    return qml.expval(hmlt_loss)

@qml.qnode(dev, shots=100)
def predict(params):
    circuit(params)
    return qml.counts(wires=wires)
```

Traveling Salesman Problem in PennyLane/5

```
##### QUANTUM APPROXIMATE OPTIMIZATION ALGORITHM #####

circuit_depth = 2

params = np.array(np.random.rand(2, circuit_depth), requires_grad=True)
opt = qml.AdamOptimizer(stepsize=0.001)
no_steps = 10

for i in range(no_steps):
    params, cost = opt.step_and_cost(loss_function, params)
    print(f'Loss {i} = {cost}')

print(f'Optimal parameters:\n{params}')

counts = predict(params)
print(f'Probability distribution:\n{counts}')

##### WARNING: IT TAKES AGES TO TRAIN THE MODEL WITHOUT GPU/QPU #####
```

Exercise 9.4: Quantum Approximate Optimization Algorithm

1. Discuss similarities and differences between the adiabatic quantum algorithms and the quantum annealing.
2. Explain why the parameter space of QUBO is compared with a transverse field. In particular, identify the parameter interaction and self-loops of QUBO problems in the system energy equation.
3. Regarding QAOA, we interleave the mixer Hamiltonian and the loss Hamiltonian via trotterization. Make a comparison of this process with the adiabatic quantum computing. [*Hint*: Notice the difference in the weight terms t/T and $1-t/T$.]
4. Compare the time complexity of solving QUBO problems in classical computing against that of QAOA.
5. Discuss the similarity between the max-cut problem and the 2D Ising Model.
6. Discuss the differences between the X-mixer and the bit-flipping mixer. Explain how each of them allows the exploration of the parameter space.
7. Study the following QUBO problems: max-cut, max-clique, max-weighted cycle, max-independent set, and min-vertex cover. Explain how the design patterns of loss Hamiltonians are applied to them.

Conclusion

Conclusion

- Simulated annealing is a classical optimization technique for finding the global optimum
- Variational quantum eigensolver (VQE) is a parameter optimization method for complex VQAs based on the adiabatic quantum computing
- QUBO (quadratic unconstrained binary optimization) is a set of problems, where we seek to find the right binary combination of the input such that the objective function becomes minimized
- With VQE, we can solve QUBO problems via optimization on graph structures by decomposing them into a loss Hamiltonian and a corresponding mixer Hamiltonian
- QAOA helps solve QUBO problems on graph structures by interleaving the loss and mixer Hamiltonians for several layers to enhance the parameter space exploration

Questions?

Target Learning Outcomes/1

- Simulated Annealing
  1. Explain the concept of local and global optima in the context of gradient-based optimization and the mathematical role of curvature.
  2. Describe the mechanism of simulated annealing, specifically how increasing temperature flattens the prediction loss function to facilitate area exploration.
  3. Calculate annealed values for given temperatures and use these results to explain how the technique addresses the problem of barren plateaux in prediction loss.
- Variational Quantum Eigensolver (VQE)
  1. Differentiate between excited states and the ground state in quantum systems using the Rayleigh-Ritz principle.
  2. Define the role of the Hamiltonian as an energy operator and explain how VQE seeks the lowest eigenvalue to find stable molecular or parameter structures.
  3. Compare VQE as a parameter optimization method against adiabatic quantum computing.

Target Learning Outcomes/2

- QUBO Problems
  1. Define quadratic unconstrained binary optimization (QUBO) and its objective to find binary combinations that minimize a specific function.
  2. Analyze common QUBO problems such as Max-Cut, Max-Clique, and Min-Vertex Cover.
  3. Design patterns for loss Hamiltonians to translate graph-based optimization problems into a quantum-accessible format.
- Quantum Approximate Optimization Algorithm (QAOA)
  1. Explain the QAOA Framework, specifically the interleaving of loss Hamiltonians and mixer Hamiltonians across multiple layers.
  2. Discuss the differences between the X-mixer and bit-flipping mixer in terms of exploring the parameter space.
  3. Contrast the time complexity and methodology of solving QUBO problems via QAOA versus classical computing approaches.