CPEN 400Q / EECE 571Q Lecture 19
Solving combinatorial optimization problems with QAOA

Tuesday 22 March 2022

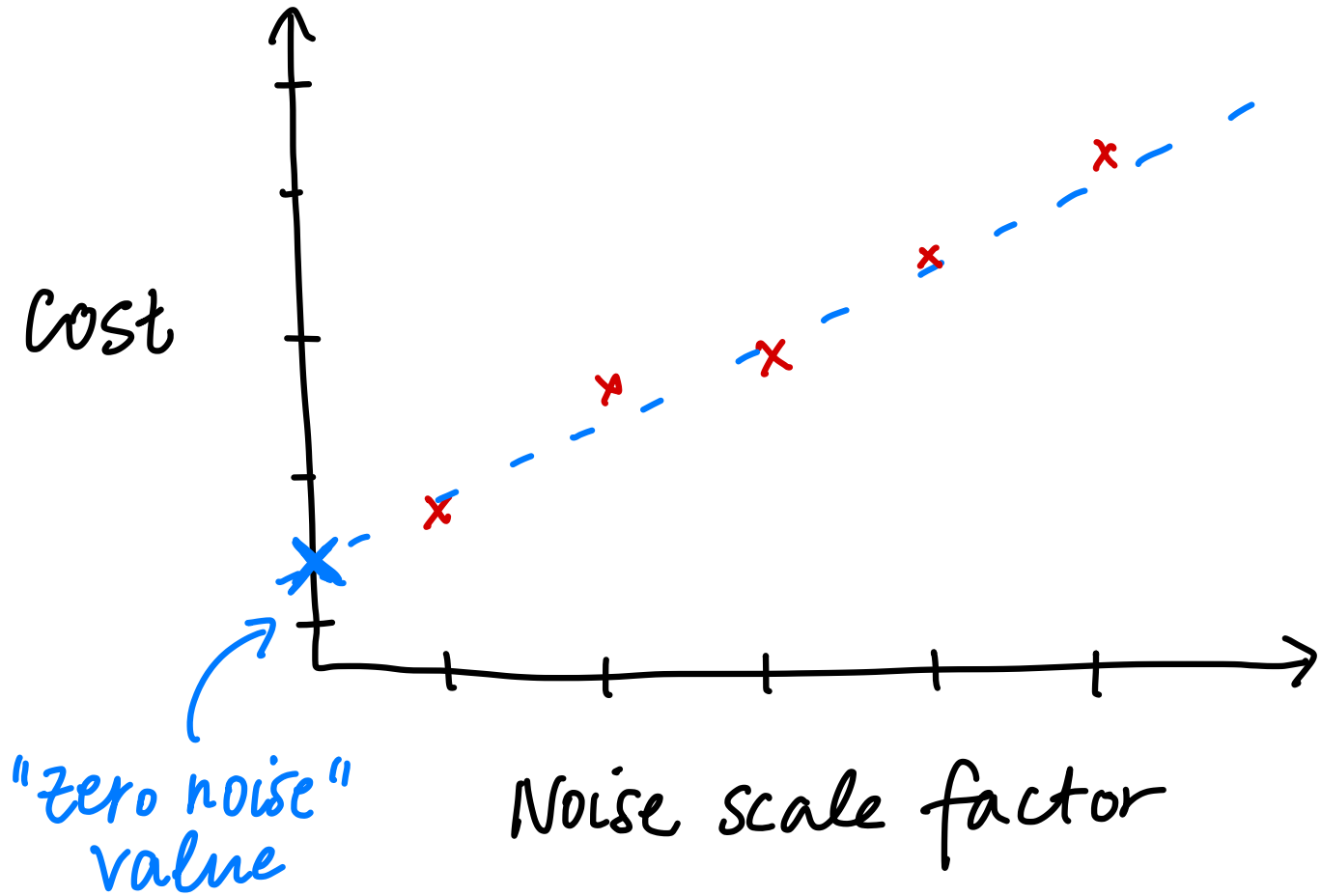
Announcements

- Assignment 4 available (due Friday 8 April at 23:59)
- Last quiz today

Please follow submission instructions for assignments (branch name, make PR, etc.), and update using `requirements.txt`.

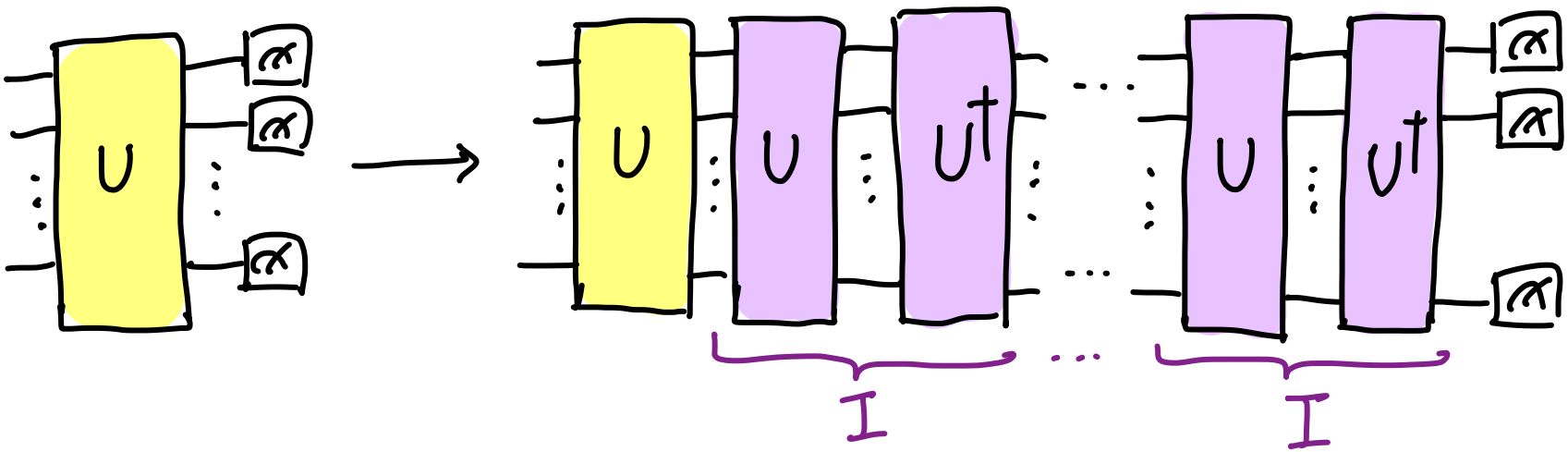
Last time

We explored an error-mitigation technique called zero-noise extrapolation (ZNE), and used linear regression to extrapolate to the noise-free expectation values.



Last time

We coded up ZNE using *unitary folding*.



We found that, applying ZNE to expectation values computed from a (simulated) noisy device gave a better reconstruction of a quantum state (using basic quantum state tomography).

Last time

We started exploring how optimization problems can be mapped to the domain of quantum computing by formulating it as an energy minimization problem:

$$\min_{\vec{x}} \ \text{cost}(\vec{x}) \quad \text{subject to constraints}(\vec{x})$$

Optimization	**Physical system**
$\vec{x}$	State of the system
$\text{cost}(\vec{x})$	Hamiltonian
Optimum $\vec{x^*}$	Ground state
$\text{cost}(\vec{x^*})$	Ground state energy

Learning outcomes

- Convert cost functions of simple graph theory problems to Hamiltonians
- Solve combinatorial optimization problems with QAOA in PennyLane

Adiabatic quantum computing (AQC)

1. Design a Hamiltonian whose ground state represents the solution to our optimization problem
2. Prepare a system in ground state of an "easy" Hamiltonian
3. Perform **adiabatic evolution** to transform the system from the ground state of the "easy" Hamiltonian to the ground state of the problem Hamiltonian

Adiabatic quantum computing (AQC)

Let H_m be a **mixer Hamiltonian** whose ground state can be easily prepared.

Let H_c be a **cost Hamiltonian** whose ground state represents the solution to a problem of interest.

Adiabatic evolution is expressed mathematically as the function

$$H(s) = A(s)H_m + B(s)H_c$$

The parameter s is representative of time; s goes from 0 to 1; $A(s)$ decreases to 0 with time and $B(s)$ increases from 0.

Quantum approximate optimization algorithm (QAOA)

QAOA is a gate-model algorithm that can obtain approximate solutions to combinatorial optimization problems.

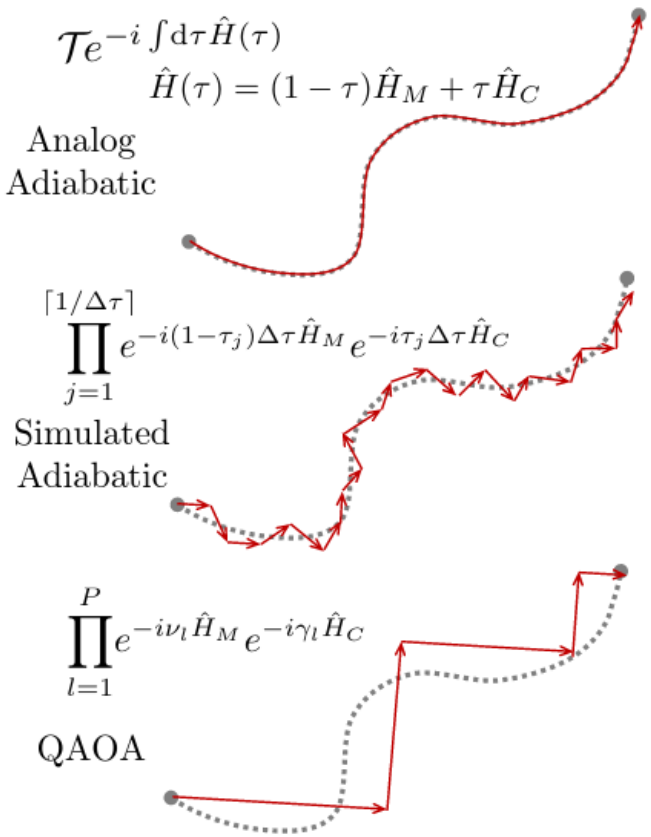
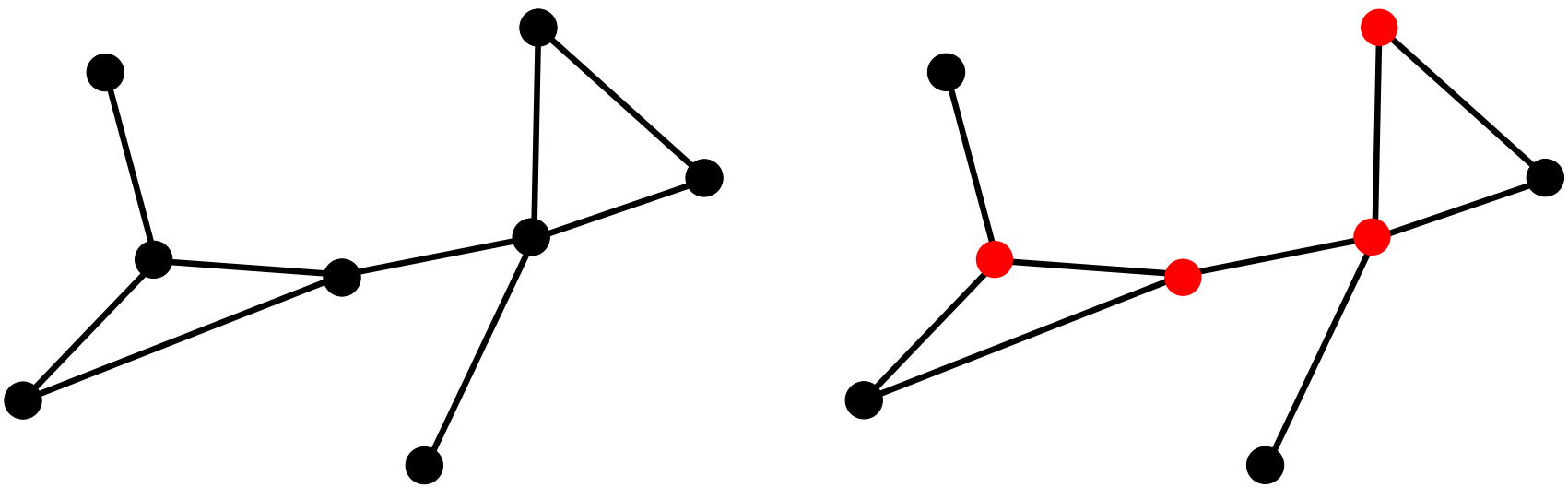


Image credit: G. Verdon, M. Broughton, J. Biamonte. *A quantum algorithm to train neural networks using low-depth circuits.* `https://arxiv.org/abs/1712.05304`

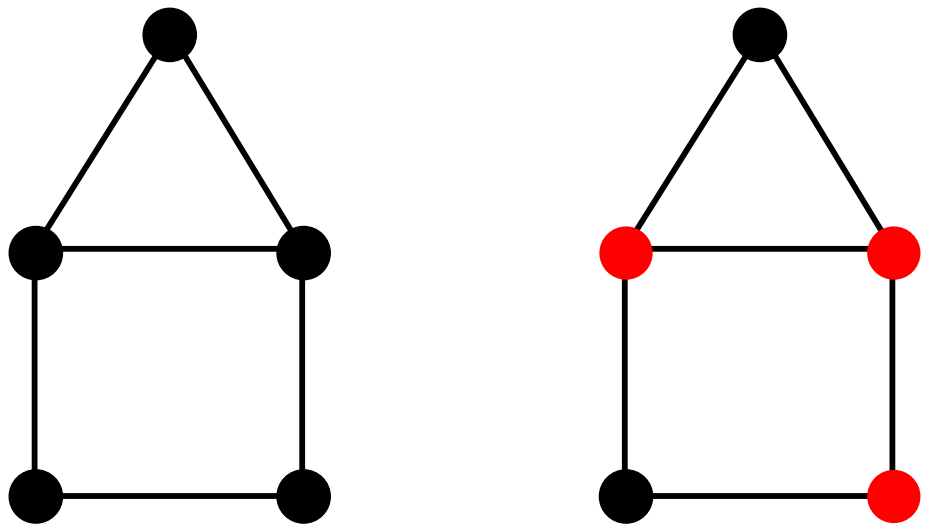
Combinatorial optimization

Example: Given a graph $G = (V, E)$, what is the *smallest number of vertices* you can colour such that every edge in the graph is attached to at least one coloured vertex?



Motivating example: vertex cover

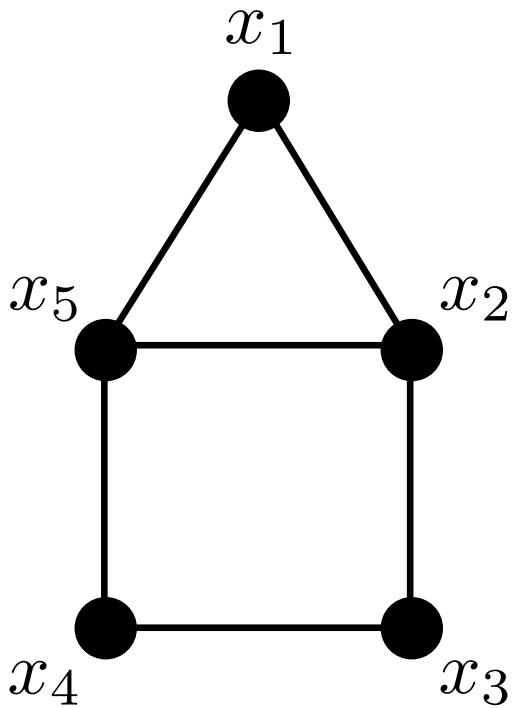
How do we turn an optimization problem for some graph into a Hamiltonian?



First, we will define a cost function, who minimum cost will correspond to the optimal set of vertices to colour. Then, we will turn it into a Hamiltonian.

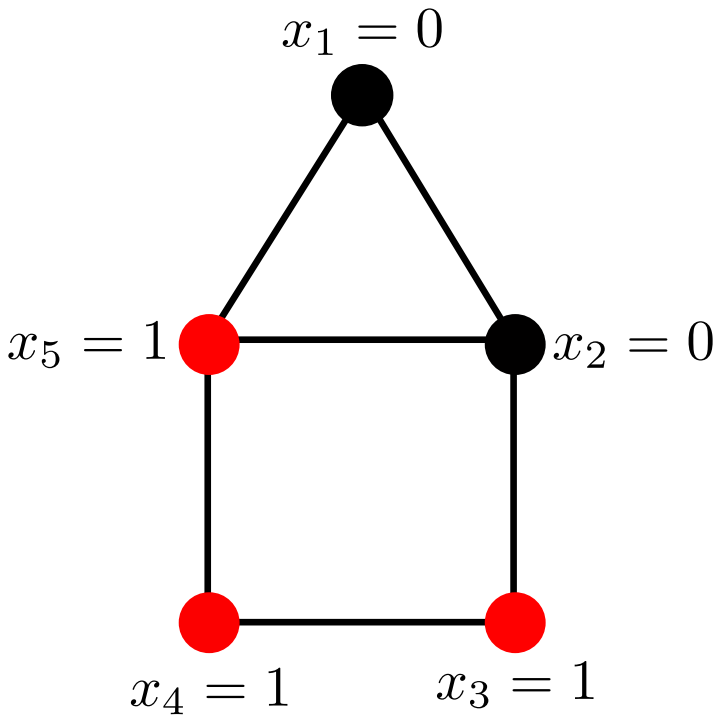
Motivating example: vertex cover

Whether a vertex is coloured is a *binary variable*.



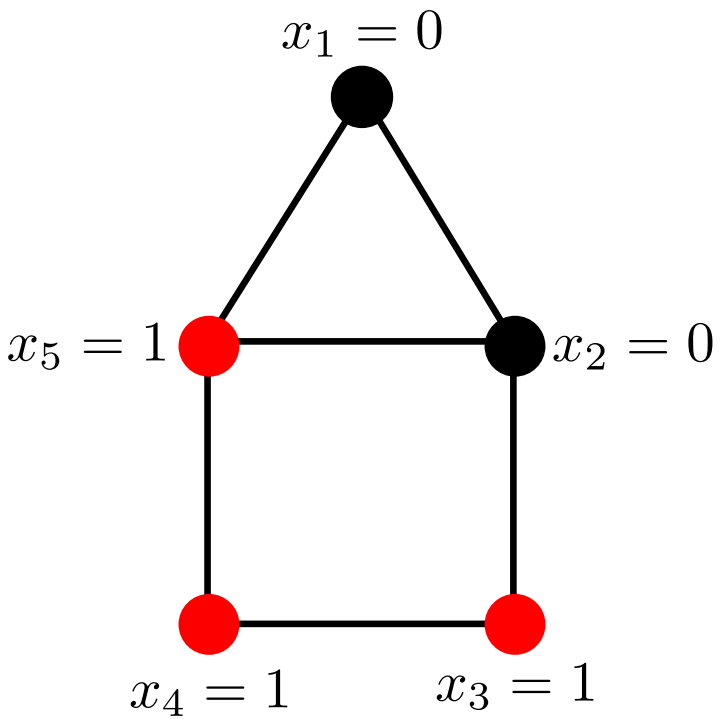
Motivating example: vertex cover

Let's assign coloured vertices to have value 1, and un-coloured 0.



Now that we have our variables, how do we come up with a minimizable cost function that represents the problem?

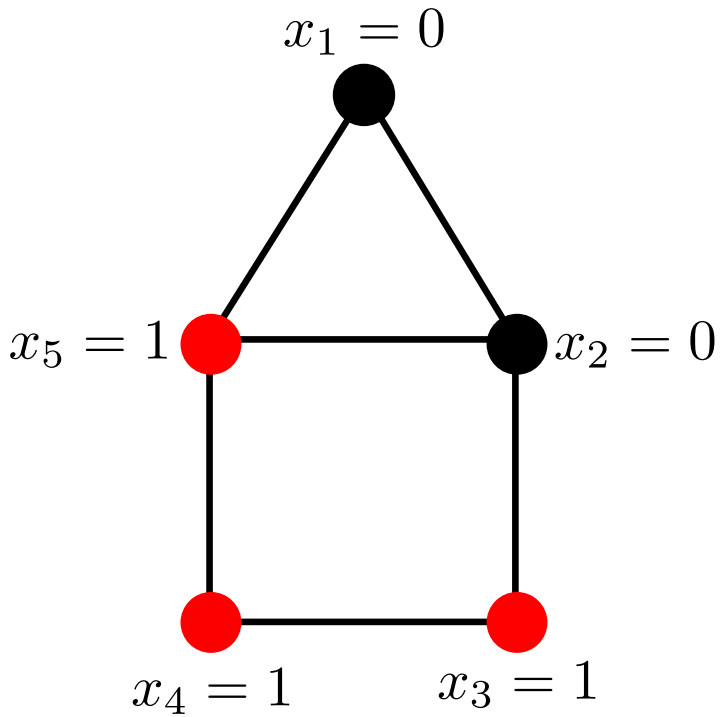
Motivating example: vertex cover



We need every edge to be next to a coloured vertex. Design a cost function that penalizes edges that are not, but favours ones that are.

Intuitively, find a function of two vertices that is 0 if the colouring is valid, and 1 if it is not.

Motivating example: vertex cover



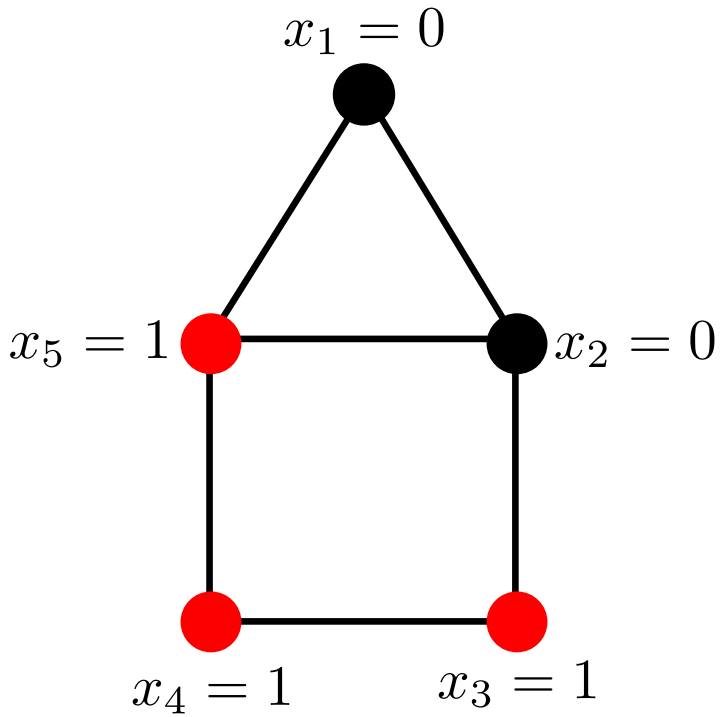
Consider for each edge ij the function

$$f(x_i, x_j) = (1 - x_i)(1 - x_j)$$

The possible values are:

$$f(x_i, x_j) = \begin{cases} 0 & \text{if } x_i = x_j = 1 \\ 0 & \text{if } x_i = 1 \text{ or } x_j = 1 \\ 1 & \text{if } x_i = x_j = 0 \end{cases}$$

Motivating example: vertex cover



Then in an optimal colouring,

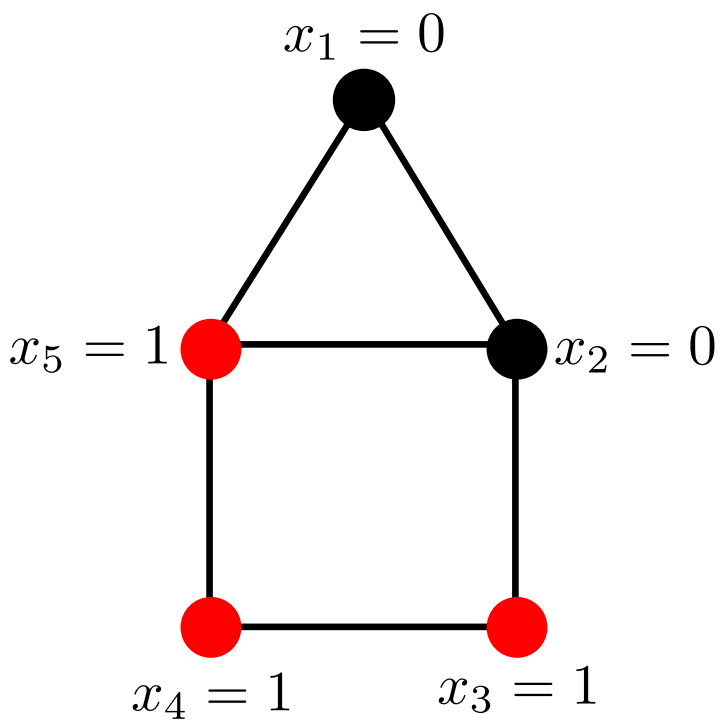
$$f(x_i, x_j) = (1 - x_i)(1 - x_j) = 0$$

for all edges $ij \in E$.

So we can write

$$\min_{\vec{x}} \sum_{ij \in E} (1 - x_i)(1 - x_j)$$

Motivating example: vertex cover



However recall that we also want to colour the fewest vertices. The cost should also depend on the number of coloured vertices.

Solution: add to our cost

$$\sum_{i \in V} x_i$$

Motivating example: vertex cover

The full cost function is then

$$\min_{\vec{x}} \left(\sum_{ij \in E} (1 - x_i)(1 - x_j) + \sum_{i \in V} x_i \right)$$

1. How do we turn this into a Hamiltonian?
2. How do we find its minimum energy / configuration on a quantum computer?

Hamiltonian translation

$$\min_{\vec{x}} \left(\sum_{ij \in E} (1 - x_i)(1 - x_j) + \sum_{i \in V} x_i \right)$$

First thing to consider is the problem domain: x_i are binary variables. We have qubits, which have basis states $|0\rangle$ and $|1\rangle$.

But since we want to turn this into a Hamiltonian and compute a cost (i.e., measure its expectation value), it's more straightforward to map 0 and 1 to *expectation values* associated to $|0\rangle$ and $|1\rangle$.

Hamiltonian translation

Usually we consider expectation values of Pauli Z.

We will make the mapping

$$x_i \to \frac{1}{2}(1 - z_i), \quad z_i \in \{-1, 1\}$$

This associates $x_i = 0$ to $z_i = 1$ (corresponds to $|0\rangle$), and $x_i = 1$ to $z_i = -1$ (corresponds to $|1\rangle$).

Hamiltonian translation

Let's expand our cost function and make this substitution.

$$\sum_{ij \in E} (1 - x_i)(1 - x_j) + \sum_{i \in V} x_i$$

$$\sum_{ij \in E} (1 - x_i - x_j + x_i x_j) + \sum_{i \in V} x_i$$

Hamiltonian translation

$$\sum_{ij \in E} (1 - x_i - x_j + x_i x_j) + \sum_{i \in V} x_i$$

Substitute:

$$\sum_{ij \in E} \left(1 - \frac{1}{2}(1 - z_i) - \frac{1}{2}(1 - z_j) + \frac{1}{4}(1 - z_i)(1 - z_j)\right) + \sum_{i \in V} \frac{1}{2}(1 - z_i)$$

Expand:

$$\sum_{ij \in E} \left(1 - \frac{1}{2} + \frac{1}{2}z_i - \frac{1}{2} + \frac{1}{2}z_j + \frac{1}{4} - \frac{1}{4}z_i - \frac{1}{4}z_j + \frac{1}{4}z_i z_j\right) + \sum_{i \in V} \frac{1}{2}(1 - z_i)$$

Collect:

$$\sum_{ij \in E} \left(\frac{1}{4} + \frac{1}{4}z_i + \frac{1}{4}z_j + \frac{1}{4}z_i z_j\right) + \sum_{i \in V} \frac{1}{2}(1 - z_i)$$

Hamiltonian translation

$$\sum_{ij \in E} \left(\frac{1}{4} + \frac{1}{4} z_i + \frac{1}{4} z_j + \frac{1}{4} z_i z_j \right) + \sum_{i \in V} \frac{1}{2} (1 - z_i)$$

Consider now that: the total number of edges and vertices are constant - they will provide only an "offset" to the cost, and the values of the variables don't matter.

$$\sum_{ij \in E} \left(\frac{1}{4} z_i + \frac{1}{4} z_j + \frac{1}{4} z_i z_j \right) - \sum_{i \in V} \frac{1}{2} z_i$$

And finally, the absolute value doesn't matter, so we can rescale:

$$\sum_{ij \in E} (z_i + z_j + z_i z_j) - 2 \sum_{i \in V} z_i$$

Hamiltonian translation

Can also weight the terms differently depending on which constraint is more important (i.e., if you care more about just getting a valid colouring, weight the first one more).

$$\gamma \sum_{ij \in E} (z_i + z_j + z_i z_j) - 2\lambda \sum_{i \in V} z_i$$

To turn this into a Hamiltonian, recall that

- Each z_i represents an expectation value of Z_i
- Computing expectation values is linear

Hamiltonian translation

$$\gamma \sum_{ij \in E} (z_i + z_j + z_i z_j) - 2\lambda \sum_{i \in V} z_i$$

$$\hat{H} = \gamma \sum_{ij \in E} (Z_i + Z_j + Z_i Z_j) - 2\lambda \sum_{i \in V} Z_i$$

QAOA

We also need a *mixer* Hamiltonian. The mixer must have a special property: it *cannot commute* with the cost Hamiltonian.

Let's try and understand why not...

Our cost Hamiltonian

$$\hat{H}_c = \gamma \sum_{ij \in E} (Z_i + Z_j + Z_i Z_j) - 2\lambda \sum_{i \in V} Z_i$$

consists of a sum of Pauli Z operators. This means it is just a diagonal matrix, and its eigenstates are the computational basis states just like those of individual Pauli Z.

$$\hat{H}_c |\mathbf{z}\rangle = E_{\mathbf{z}} |\mathbf{z}\rangle, \quad \mathbf{z} \in \{0,1\}^n$$

Any state can be expressed in terms of the computational basis:

$$|\psi\rangle = \sum_{\mathbf{z}} \alpha_{\mathbf{z}} |\mathbf{z}\rangle$$

Evolve this under the cost Hamiltonian:

$$\begin{aligned} e^{-it\hat{H}_c}|\psi\rangle &= e^{-it\hat{H}_C} \sum_{\mathbf{z}} \alpha_{\mathbf{z}} |\mathbf{z}\rangle \\ &= \sum_{\mathbf{z}} \alpha_{\mathbf{z}} e^{-it\hat{H}_C} |\mathbf{z}\rangle \\ &= \sum_{\mathbf{z}} \alpha_{\mathbf{z}} e^{-itE_{\mathbf{z}}} |\mathbf{z}\rangle \end{aligned}$$

Have we actually changed anything?

Original state:

$$|\psi\rangle = \sum_{\mathbf{z}} \alpha_{\mathbf{z}} |\mathbf{z}\rangle \quad \rightarrow \quad \Pr(\mathbf{z}) = \alpha_{\mathbf{z}} \alpha_{\mathbf{z}}^* = |\alpha_{\mathbf{z}}|^2$$

New state:

$$e^{-it\hat{H}_c}|\psi\rangle = \sum_{\mathbf{z}} \alpha_{\mathbf{z}} e^{-itE_{\mathbf{z}}} |\mathbf{z}\rangle \quad \rightarrow \quad \Pr(\mathbf{z}) = \alpha_{\mathbf{z}} e^{-itE_{\mathbf{z}}} \cdot \alpha_{\mathbf{z}}^* e^{itE_{\mathbf{z}}} = |\alpha_{\mathbf{z}}|^2$$

Simply evolving under the cost Hamiltonian doesn't change the probability distribution of the state.

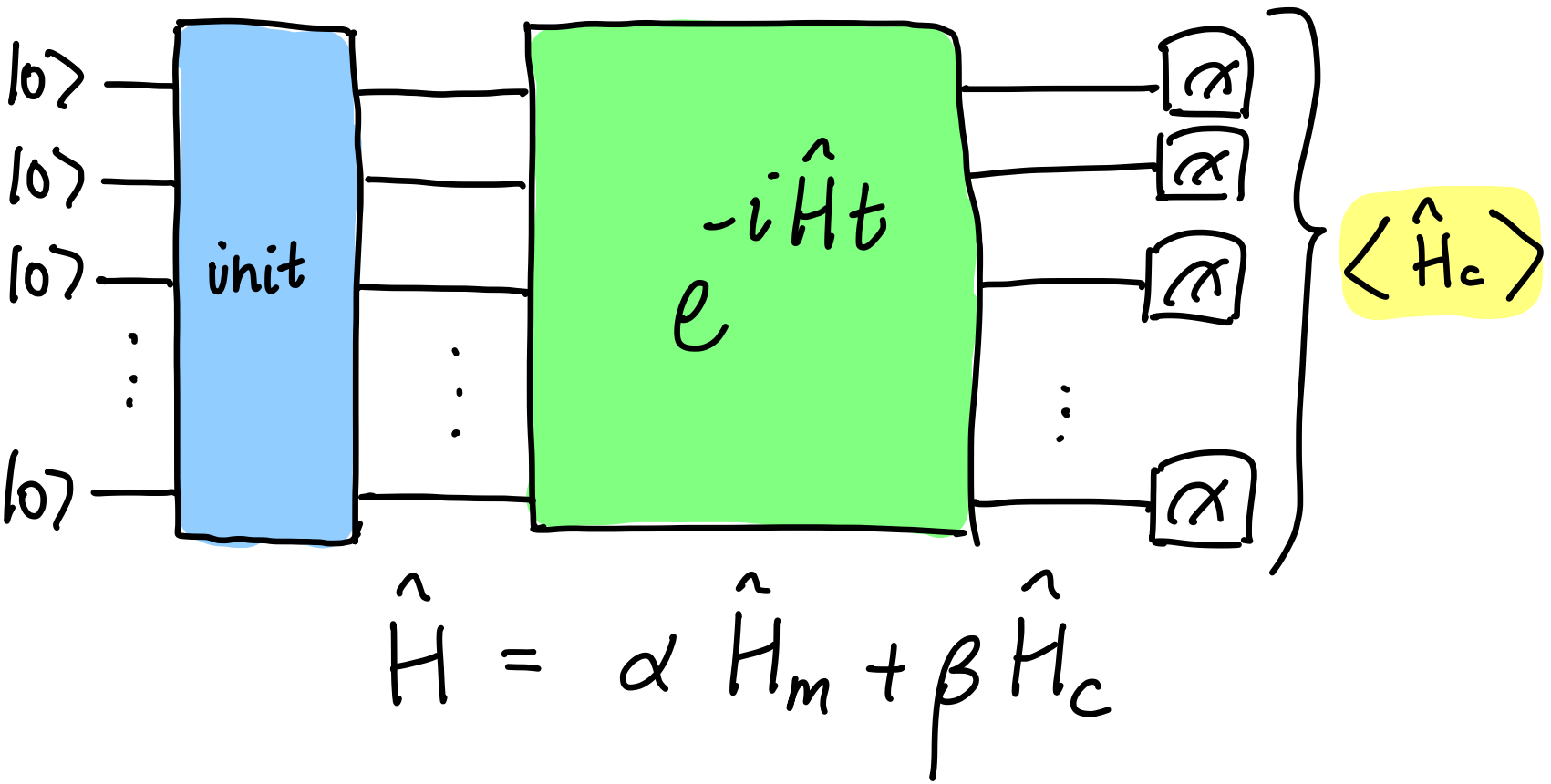
If $\hat{H}_m$ commutes with $\hat{H}_c$, then $\hat{H}_c$ and $\hat{H}_m$ have a shared set of eigenvectors so evolving under $\hat{H}_m$ doesn't affect the state either.

Need a mixer which *does not commute* with $\hat{H}_c$. Something like

$$\hat{H}_m = \sum_i X_i$$

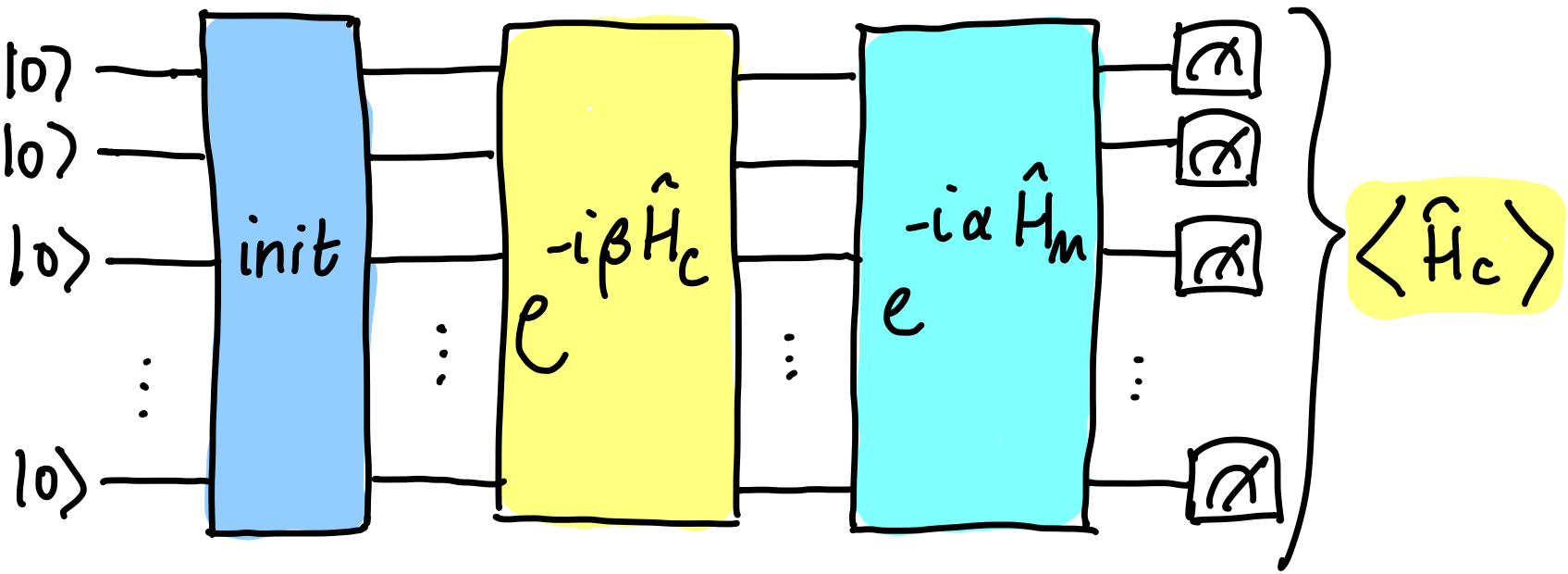
Uniform superposition is an "easy to prepare" eigenstate of $\hat{H}_m$.

Initial idea: apply the unitary that evolves the Hamiltonian?



How do we implement this circuit?

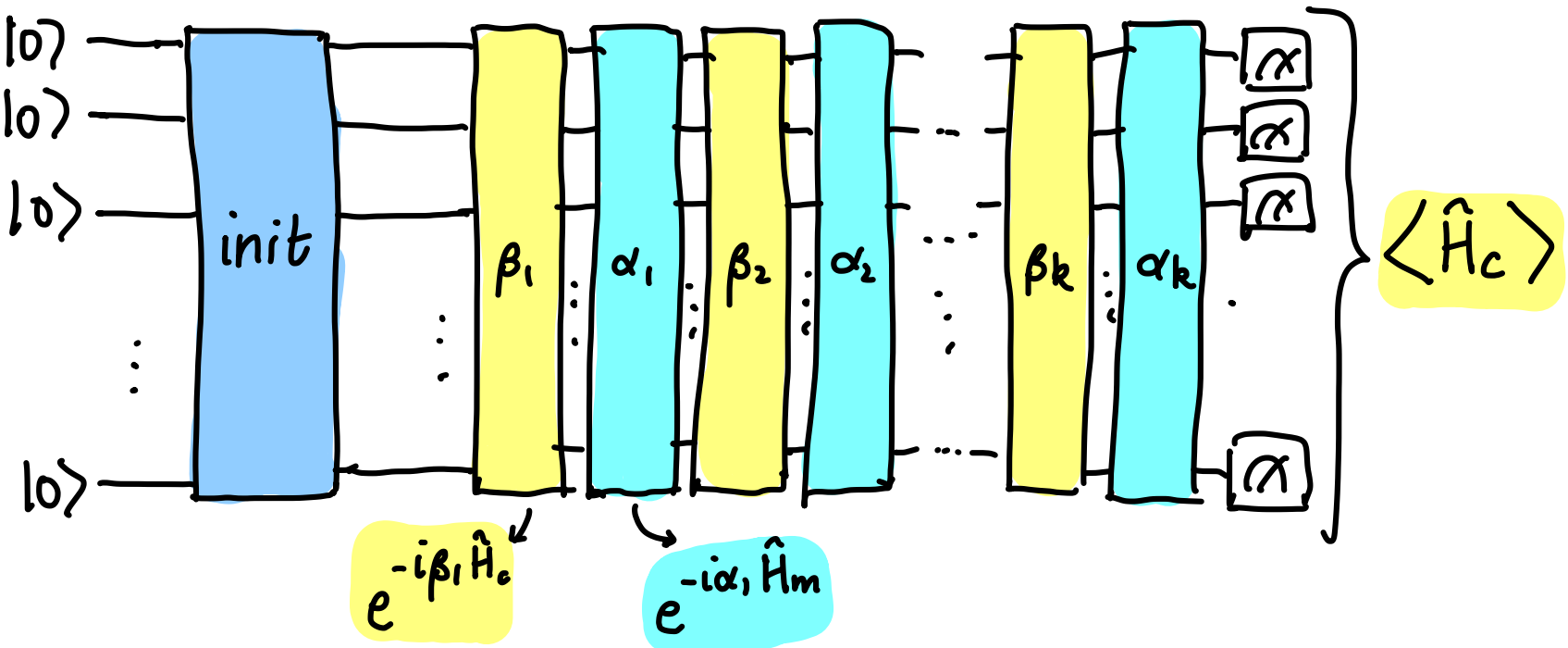
You might think that since $\hat{H}$ is a sum of terms...



But this is only true when $\hat{H}_c$ and $\hat{H}_m$ *commute* (we will talk about this more on Thursday).

QAOA

QAOA does something similar to this but instead of applying each block for a fixed "time", "time" is a trainable parameter.



Let's implement this, and find parameters that minimize the cost.

Next time

Content:

- Basics of Hamiltonian simulation

Action items:

1. Assignment 4 (can do all problems)
2. Final project

Recommended reading:

- Original QAOA paper `https://arxiv.org/abs/1411.4028`
- PennyLane Intro to QAOA tutorial
  `https://pennylane.ai/qml/demos/tutorial_qaoa_intro.html`
- Qiskit QAOA tutorial
  `https://qiskit.org/textbook/ch-applications/qaoa.html`