

Combinatorial Optimization on Quantum Computers

Ruslan Shaydulin, Argonne National Laboratory

PART 0: SOME BIG-PICTURE CONSIDERATIONS

Complexity of solving problems

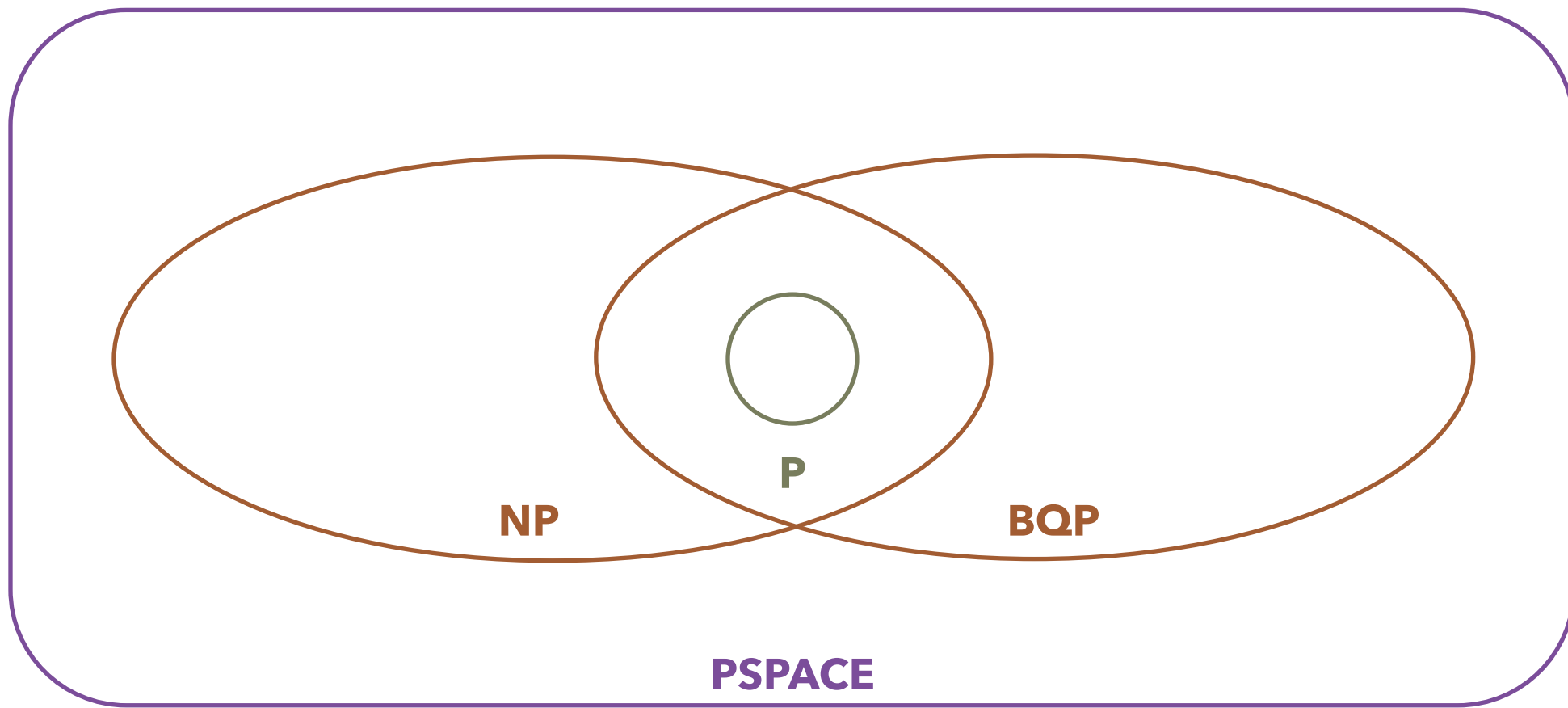
- When reasoning about complexity of solving problems (including optimization problems!), we are usually interested in how the time / memory requirements grow with problem size (asymptotic complexity)

Complexity of solving problems

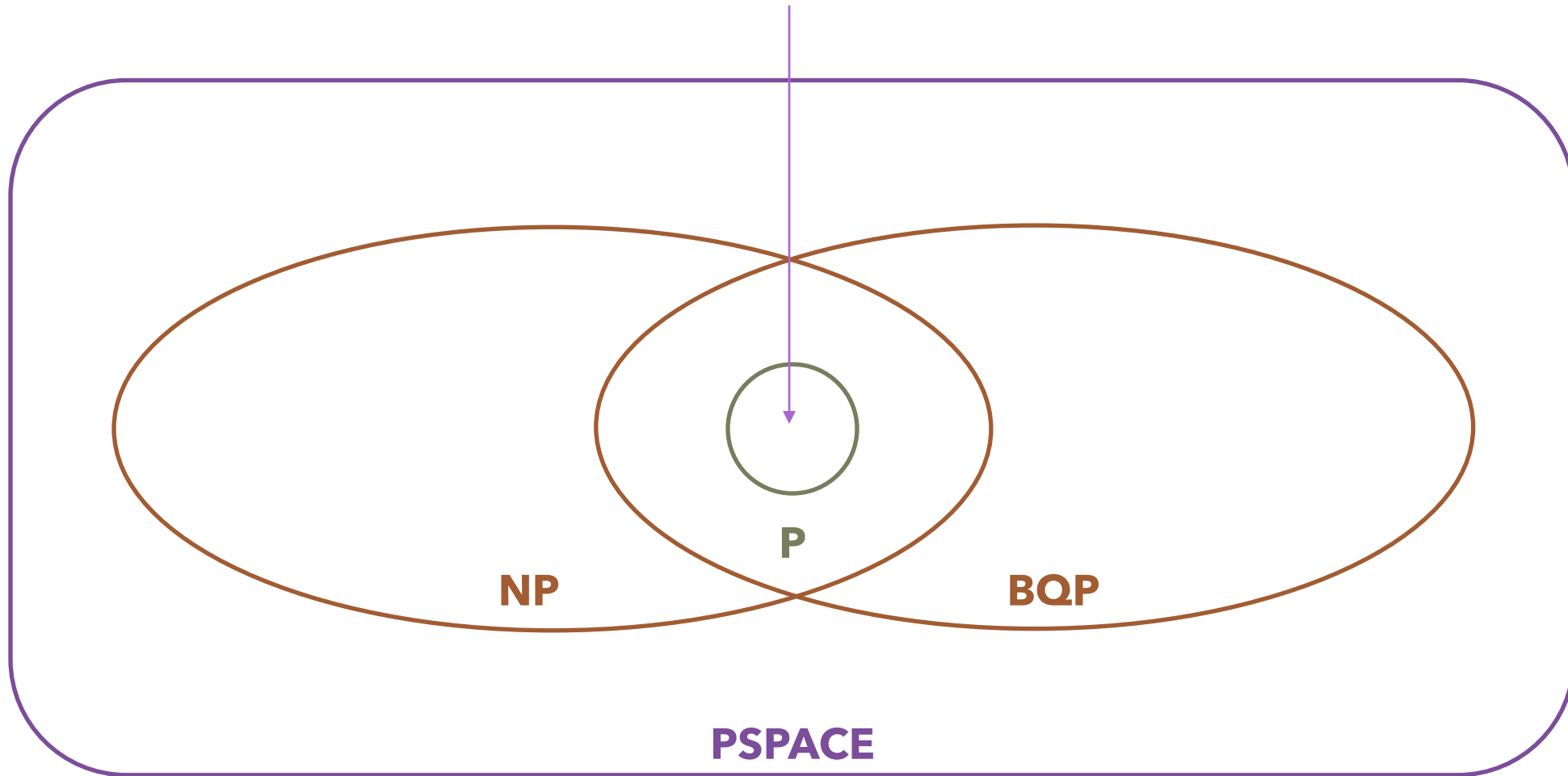
- When reasoning about complexity of solving problems (including optimization problems!), **we are usually interested in how the time / memory requirements grow with problem size** (asymptotic complexity)
- Famous classes of problems:
 - P – solvable in polynomial time
 - NP – can verify *proof* of the solution in polynomial time
 - PSPACE – solvable in polynomial space (and unlimited time)

Complexity of solving problems

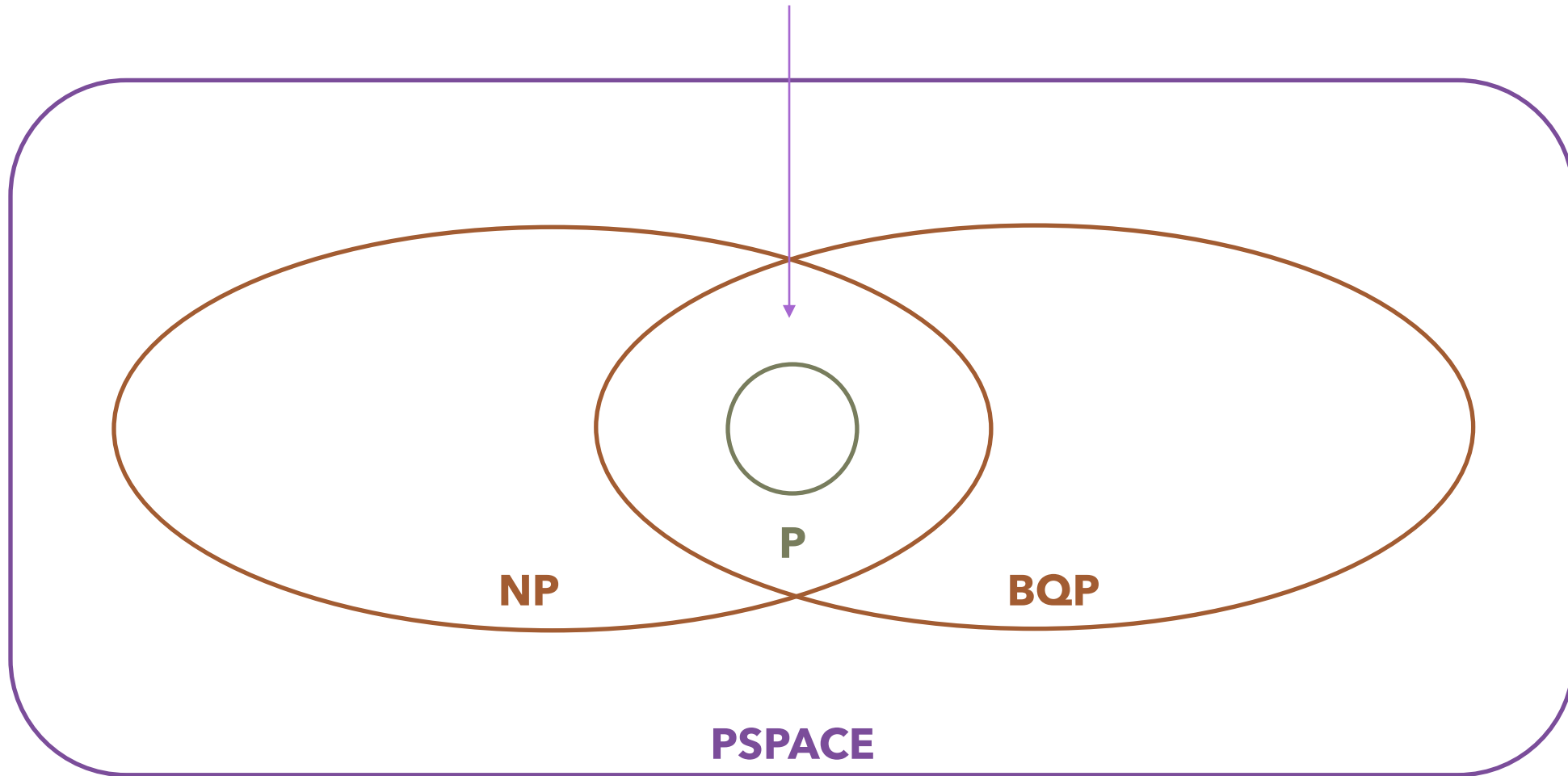
- When reasoning about complexity of solving problems (including optimization problems!), **we are usually interested in how the time / memory requirements grow with problem size** (asymptotic complexity)
- Famous classes of problems:
 - P – solvable in polynomial time
 - NP – can verify *proof* of the solution in polynomial time
 - PSPACE – solvable in polynomial time (and unlimited space)
- Quantum complexity class that we are most interested in:
 - BQP (bounded-error quantum polynomial time) – solvable on a quantum computer in polynomial time, with an error probability of at most $1/3$ for all instances



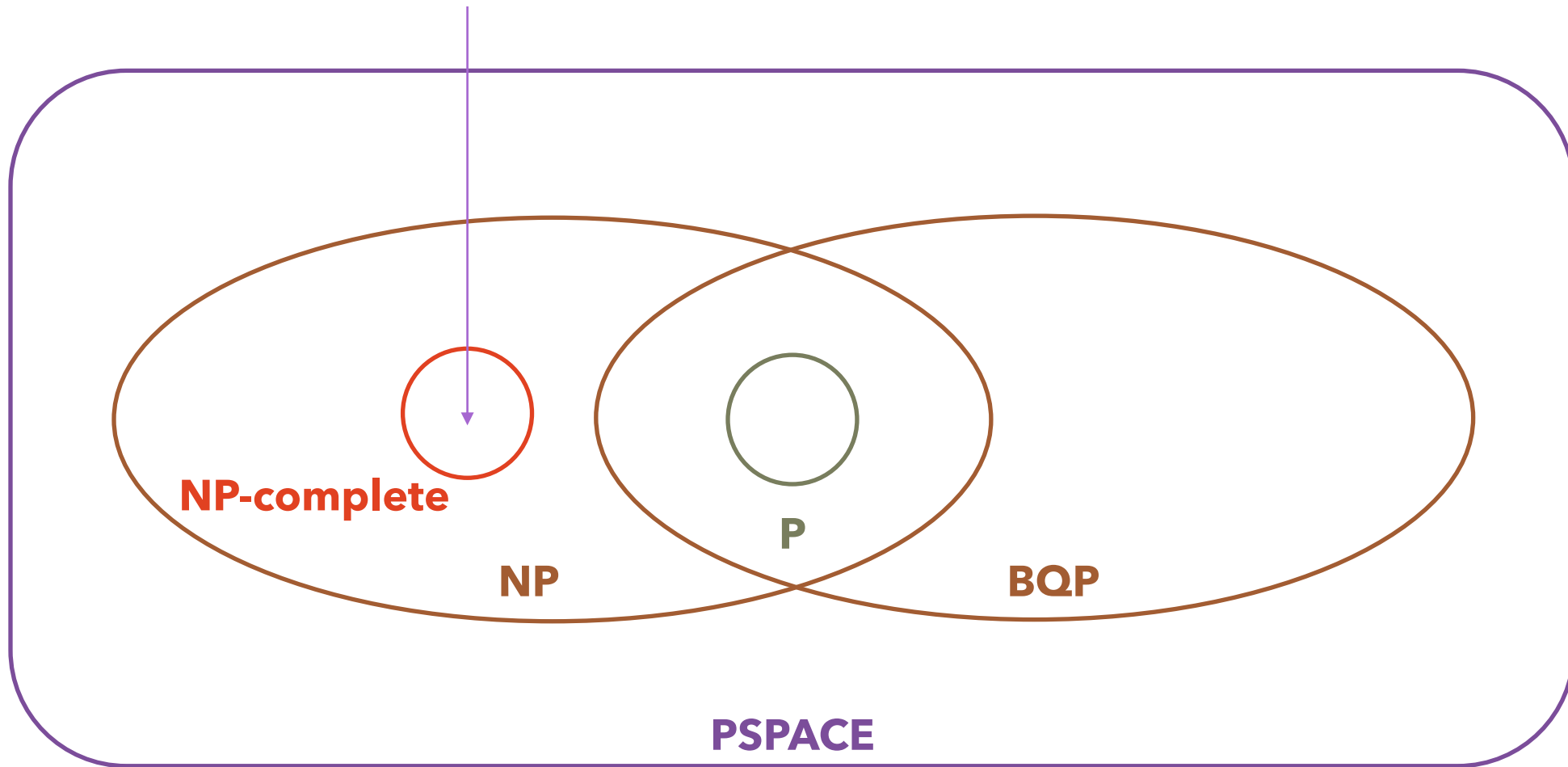
Ex: Find shortest path between two nodes in a graph



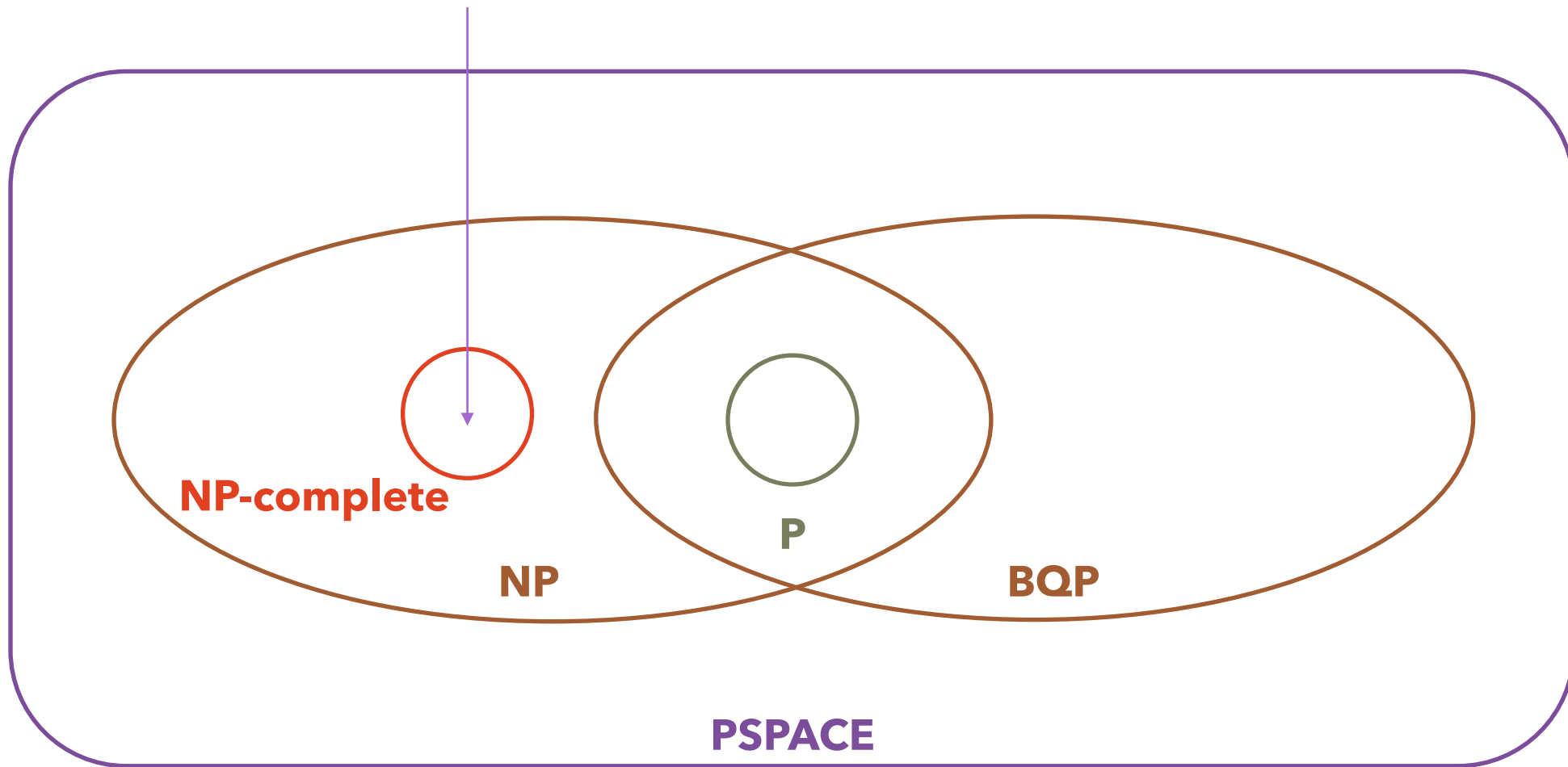
Ex: integer factorization (?)



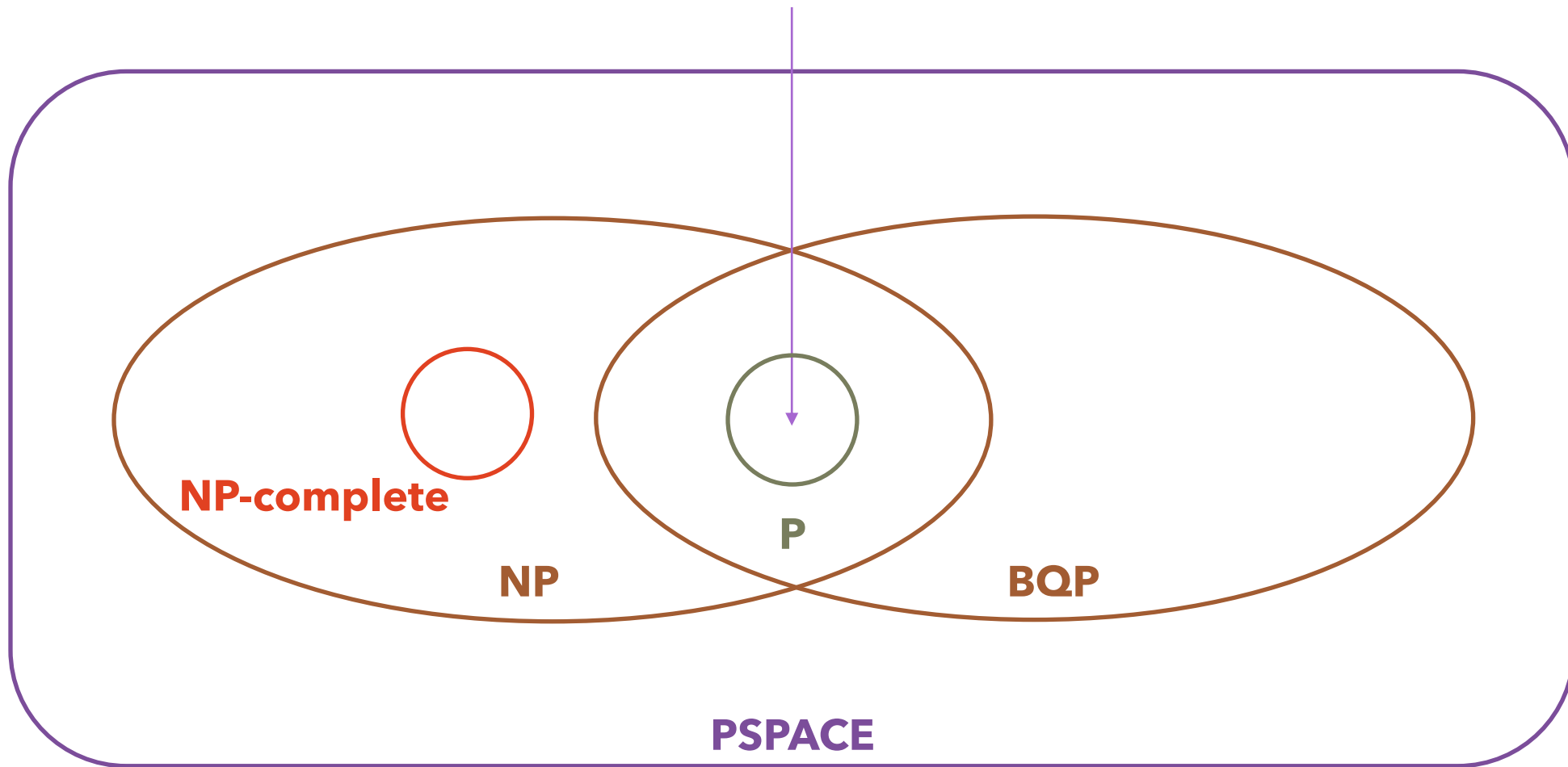
Ex: maximum cut problem



Ex: maximum cut problem **in worst case**



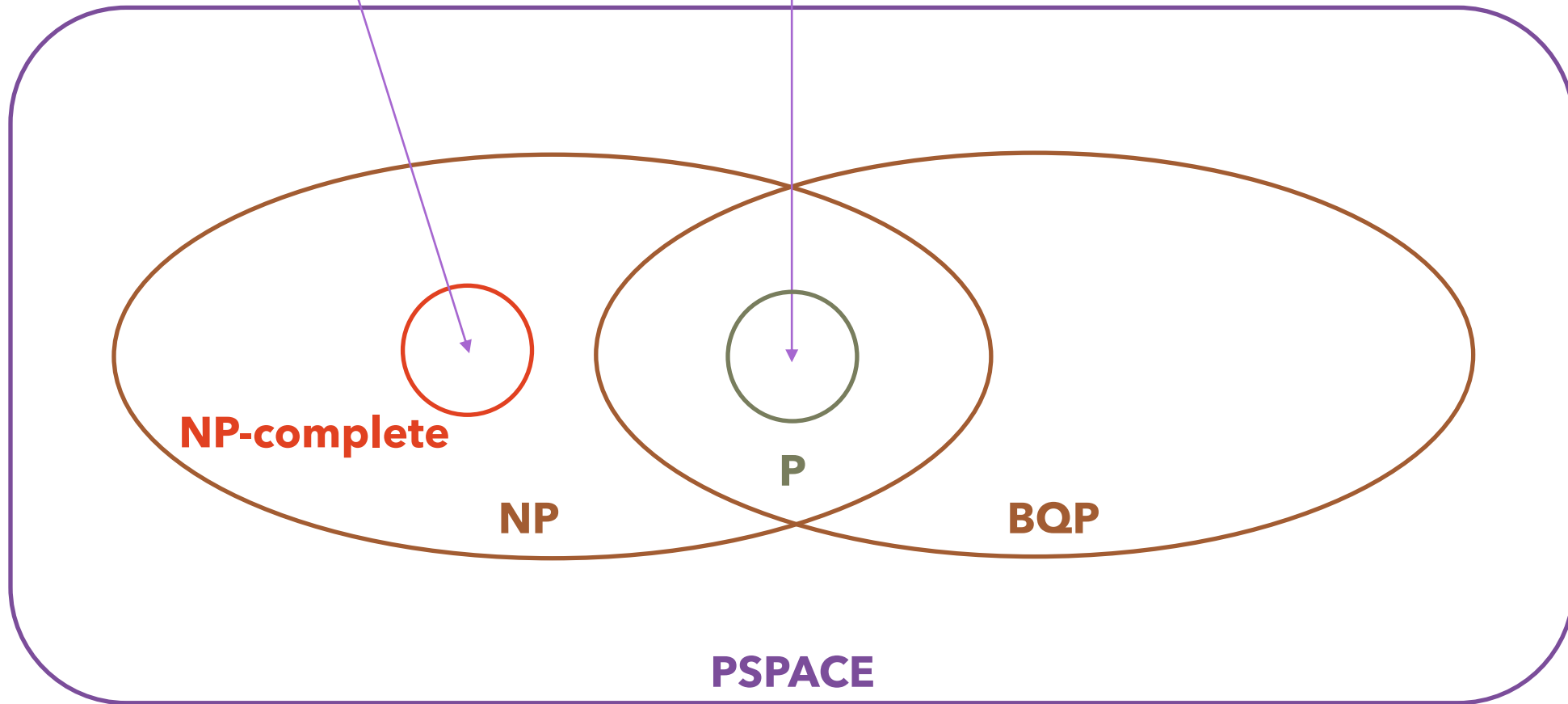
Ex: maximum cut problem **on cycle graph**



Worst-case instances are difficult to construct!
Proving asymptotic performance is even more difficult!

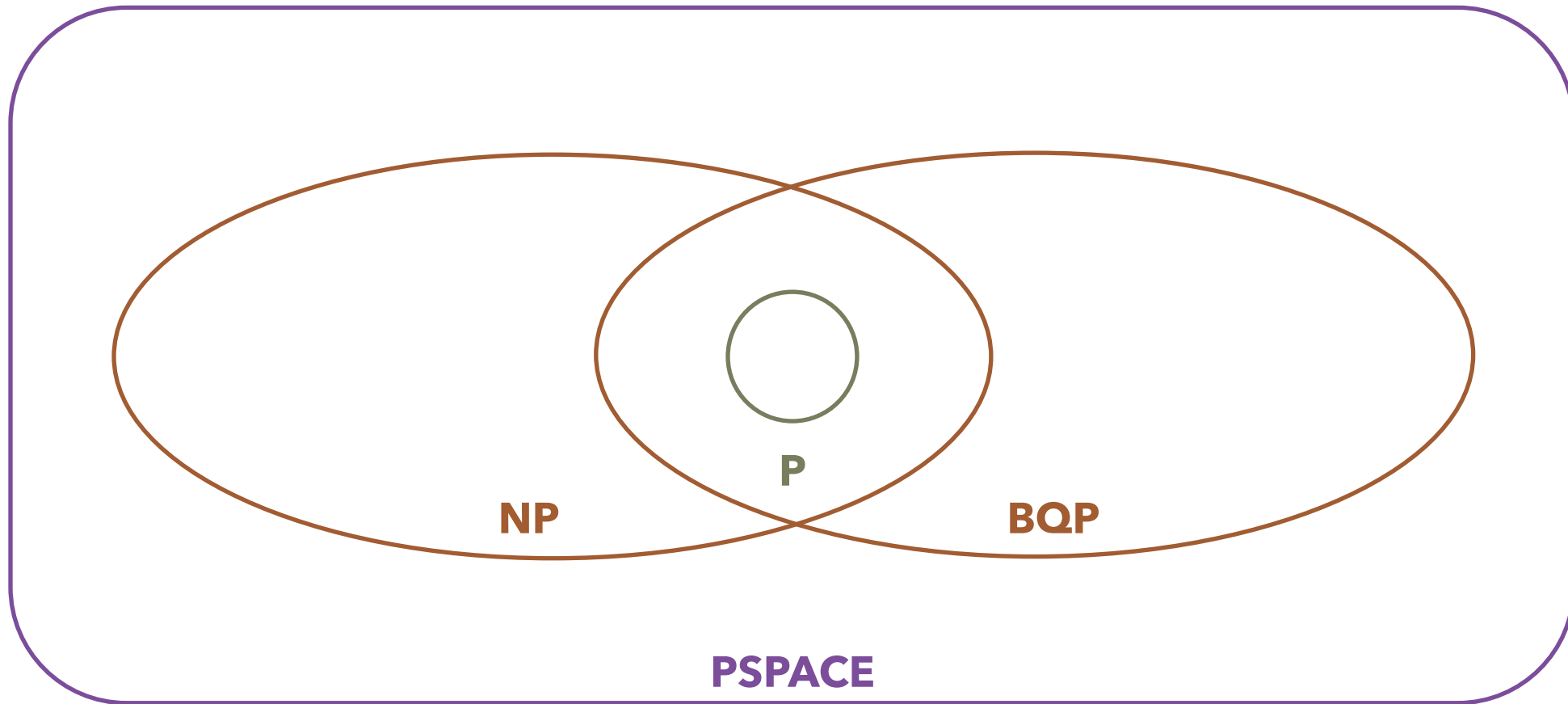
Ex: maximum cut problem **in worst case**

Ex: maximum cut problem **on cycle graph**



Disclaimer

- We are **not** going to solve NP-complete problems in polynomial time (today)



Disclaimer

- We are **not** going to solve NP-complete problems in polynomial time (today)
- Instead, we are going to adopt a different perspective...

Algorithms with proven performance

Classical

- Matrix multiplication
- Dijkstra's algorithm for shortest path

Quantum

- Shor's algorithm for integer factoring
- Grover's algorithm for unstructured search

Heuristic methods

Classical

- Gradient descent (for non-convex problems)
- Simulated annealing
- Genetic algorithm

Quantum

- Quantum annealing
- QAOA
- More to be discovered...

Algorithms with proven performance

Classical

- Matrix multiplication
- Dijkstra's algorithm for shortest path

Quantum

- Shor's algorithm for integer factoring
- Grover's algorithm for unstructured search

Heuristic methods

Classical

- Gradient descent (convex problems)
- Simulated annealing
- Genetic algorithm

Quantum

- Quantum annealing
- QAOA
- More to be discovered...

- Many of the most powerful classical algorithms are heuristics – no reason to think quantum will be different
- NISQ hardware provides a unique opportunity to develop novel *quantum* heuristic methods

PART 1:
MAPPING COMBINATORIAL OPTIMIZATION
PROBLEMS ONTO QUANTUM COMPUTERS

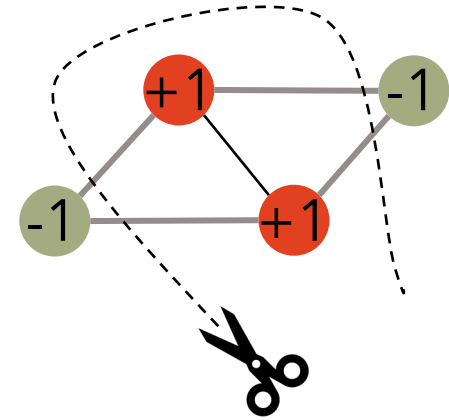
Outline

1. Maximum cut as a paradigmatic example
2. General rules for constructing Hamiltonians representing Boolean functions

Maximum Cut Problem (MAXCUT)

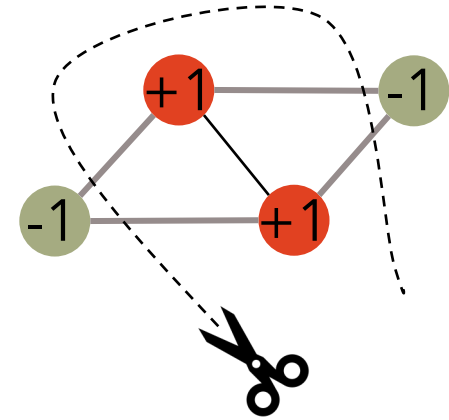
- The goal of maximum cut is to split the set of vertices V of a graph into two disjoint parts such that the number of edges spanning two parts is maximized.
- For example, if color denotes part, in the graph on the right 4 edges are cut:
- Maximum cut can be formulated as an optimization problem:

$$\max_{\mathbf{s}} \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j) \quad s_i \in \{-1, +1\}$$



Maximum Cut Problem (MAXCUT)

- The goal of maximum cut is to split the set of vertices V of a graph into two disjoint parts such that the number of edges spanning two parts is maximized.
- For example, if color denotes part, in the graph on the right 4 edges are cut:



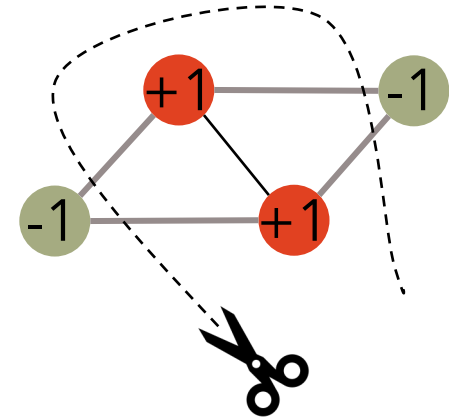
- Maximum cut can be formulated as an optimization problem:

$$\max_{\mathbf{s}} \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j) \quad s_i \in \{-1, +1\}$$

Same sign – no edge is cut (no contribution to the objective): $\frac{1}{2}(1 - s_i s_j) = 0$

Maximum Cut Problem (MAXCUT)

- The goal of maximum cut is to split the set of vertices V of a graph into two disjoint parts such that the number of edges spanning two parts is maximized.
- For example, if color denotes part, in the graph on the right 4 edges are cut:

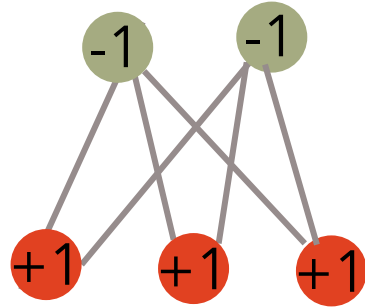


- Maximum cut can be formulated as an optimization problem:

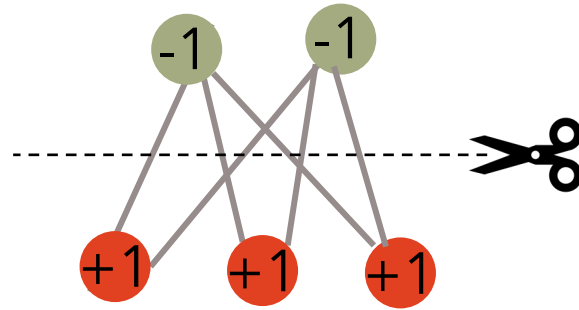
$$\max_{\mathbf{s}} \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j) \quad s_i \in \{-1, +1\}$$

Different sign – an edge is cut (contribution to the objective = 1): $\frac{1}{2}(1 - s_i s_j) = 1$

Q: What is the cut in the graph below?



Q: What is the cut in the graph below?



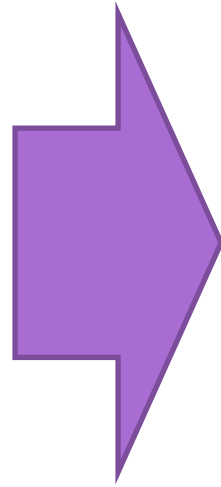
- A: six edges are cut

Solving Combinatorial Optimization Problems on a Quantum Computer

Classical

Maximizing objective

$$\max_{\mathbf{s}} C(\mathbf{s}) = \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j)$$



Quantum

Characterizing a Hamiltonian
(Hermitian operator) C

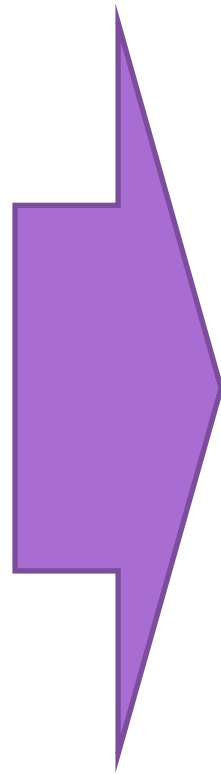
To solve an optimization problem on a quantum computer, we need to convert it into a problem of characterizing a quantum Hamiltonian

Solving Combinatorial Optimization Problems on a Quantum Computer

Classical

Objective $\max_{\mathbf{s}} C(\mathbf{s}) = \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j)$

Solution $\mathbf{s} \in \{-1, +1\}^n$



Quantum

Hamiltonian (Hermitian operator) C

Highest energy eigenstate $|s\rangle$
(largest eigenvalue eigenvector)

Solving Combinatorial Optimization Problems on a Quantum Computer

Classical

Objective $\max_{\mathbf{s}} C(\mathbf{s}) = \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j)$

Solution $\mathbf{s} \in \{-1, +1\}^n$

Quantum

Hamiltonian (Hermitian operator) C

Highest energy eigenstate $|s\rangle$
(largest eigenvalue eigenvector)

Since the Hamiltonian is classical, this eigenstate is a computational basis state*, we can measure it and **get the solution with certainty**

What does this Hamiltonian look like?

- Hamiltonian is diagonal, with values on the diagonal corresponding to the values of the objective function

$$\max_{x \in \{0,1\}^n} f(x)$$



$$C = \begin{pmatrix} f(0 \dots 00) & 0 & 0 & 0 & 0 \\ 0 & f(0 \dots 01) & 0 & 0 & 0 \\ \vdots & & \ddots & & \\ 0 & 0 & 0 & f(1 \dots 10) & 0 \\ 0 & 0 & 0 & 0 & f(1 \dots 11) \end{pmatrix}$$

What does this Hamiltonian look like?

- Hamiltonian is diagonal, with values on the diagonal corresponding to the values of the objective function

$$C |0 \dots 00\rangle = \begin{pmatrix} f(0 \dots 00) & 0 & 0 & 0 & 0 \\ 0 & f(0 \dots 01) & 0 & 0 & 0 \\ \vdots & & \ddots & & \\ 0 & 0 & 0 & f(1 \dots 10) & 0 \\ 0 & 0 & 0 & 0 & f(1 \dots 11) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} = f(0 \dots 00) |0 \dots 00\rangle$$

What does this Hamiltonian look like?

- Hamiltonian is diagonal, with values on the diagonal corresponding to the values of the objective function
- In general:

$$C|x\rangle = \begin{pmatrix} f(0\dots 00) & 0 & 0 & 0 & 0 \\ 0 & f(0\dots 01) & 0 & 0 & 0 \\ \vdots & & \ddots & & \\ 0 & 0 & 0 & f(1\dots 10) & 0 \\ 0 & 0 & 0 & 0 & f(1\dots 11) \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} = f(x)|x\rangle \quad \forall x \in \{0,1\}^n$$

What does this Hamiltonian look like?

- Hamiltonian is diagonal, with values on the diagonal corresponding to the values of the objective function
- In general:

$$C|x\rangle = \begin{pmatrix} f(0\dots 00) & 0 & 0 & 0 & 0 \\ 0 & f(0\dots 01) & 0 & 0 & 0 \\ \vdots & & \ddots & & \\ 0 & 0 & 0 & f(1\dots 10) & 0 \\ 0 & 0 & 0 & 0 & f(1\dots 11) \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} = f(x)|x\rangle \quad \forall x \in \{0,1\}^n$$

- This Hamiltonian is too large to construct explicitly!

Solving Combinatorial Optimization Problems on a Quantum Computer

Classical

Quantum

Objective $\max_{\mathbf{s}} C(\mathbf{s}) = \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j)$

Hamiltonian (Hermitian operator) C

Evaluation of

How do we construct this Hamiltonian?

state \mathbf{s}

Solution

$$\mathbf{s} \in \{-1, +1\}^n$$

Highest energy eigenstate $|s\rangle$

Notation Reminder

$$|x\rangle = \mathbf{x} = \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

column vector

$$\langle y| = |y\rangle^\dagger = [y_1^* \quad y_2^* \quad \cdots \quad y_n^*]$$

conjugate transpose

Notation Reminder

$$|x\rangle = \mathbf{x} = \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{column vector} \quad \langle y| = |y\rangle^\dagger = [y_1^* \quad y_2^* \quad \cdots \quad y_n^*] \quad \text{conjugate transpose}$$

$$\langle y|x\rangle = y_1^* x_1 + \cdots + y_n^* x_n = [y_1^* \quad y_2^* \quad \cdots \quad y_n^*] \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{inner product}$$

Notation Reminder

$$|x\rangle = \mathbf{x} = \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{column vector} \quad \langle y| = |y\rangle^\dagger = [y_1^* \quad y_2^* \quad \cdots \quad y_n^*] \quad \text{conjugate transpose}$$

$$\langle y|x\rangle = y_1^* x_1 + \cdots + y_n^* x_n = [y_1^* \quad y_2^* \quad \cdots \quad y_n^*] \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{inner product}$$

$$|x\rangle \langle y| = \begin{bmatrix} x_1 y_1^* & \cdots & x_1 y_n^* \\ \vdots & \ddots & \vdots \\ x_n y_1^* & \cdots & x_n y_n^* \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} [y_1^* \quad y_2^* \quad \cdots \quad y_n^*] \quad \text{outer product}$$

Constructing MAXCUT Hamiltonian

- MAXCUT objective:

$$\max_{\mathbf{s}} \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j) \quad s_i \in \{-1, +1\}$$

Constructing MAXCUT Hamiltonian

- MAXCUT objective:

$$\max_{\mathbf{s}} \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j) \quad s_i \in \{-1, +1\}$$

- MAXCUT Hamiltonian is constructed by mapping binary variables s_i onto the eigenvalues of Z

$$C = \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j)$$

- Want to show:

$$C |x\rangle = C(\mathbf{x}) |x\rangle$$

Pauli Z operator

- Consider Pauli Z operator

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

Pauli Z operator

- Consider Pauli Z operator

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

- Note that it has eigenvalues -1, +1 with eigenvectors being computational basis states

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

$$Z |1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} = (-1) |1\rangle$$

Pauli Z operator

- Note that it has eigenvalues -1, +1 with eigenvectors being computational basis states

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

$$Z |1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} = (-1) |1\rangle$$

- Therefore,

$$Z|x\rangle = (-1)^x |x\rangle \quad x \in \{0, 1\}$$

Pauli Z operator

- Acting on the i -th qubit:

$$\begin{aligned} Z_i |x_0 \dots x_n\rangle &= I \otimes \dots \otimes Z_i \otimes \dots \otimes I |x_0 \dots x_n\rangle \\ &= (-1)^{x_i} |x_0 \dots x_n\rangle \quad x_i \in \{0, 1\}, \quad i = 1, \dots, n \end{aligned}$$

Pauli Z operator

- Acting on the i-th qubit:

$$\begin{aligned} Z_i |x_0 \dots x_n\rangle &= I \otimes \dots \otimes Z_i \otimes \dots \otimes I |x_0 \dots x_n\rangle \\ &= (-1)^{x_i} |x_0 \dots x_n\rangle \quad x_i \in \{0, 1\}, \quad i = 1, \dots, n \end{aligned}$$

- Acting on the i-th and j-th qubit:

$$\begin{aligned} Z_i Z_j |x_0 \dots x_n\rangle &= I \otimes \dots \otimes Z_i \otimes Z_j \otimes \dots \otimes I |x_0 \dots x_n\rangle \\ &= (-1)^{x_i} (-1)^{x_j} |x_0 \dots x_n\rangle \quad x_i \in \{0, 1\}, \quad i = 1, \dots, n \end{aligned}$$

- (note that here we reorder qubits such that i-th and j-th qubit are adjacent)

Constructing MAXCUT Hamiltonian

- MAXCUT objective:

$$\max_{\mathbf{s}} \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j) \quad s_i \in \{-1, +1\}$$

- MAXCUT Hamiltonian is constructed by mapping binary variables s_i onto the eigenvalues of Z

$$C = \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j)$$

Verifying MAXCUT Hamiltonian

- MAXCUT objective:

$$\max_{\mathbf{s}} \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j) \quad s_i \in \{-1, +1\}$$

- Let's reformulate it in the following way:

$$\begin{aligned} x_i &= \frac{1}{2}(1 - s_i) \\ s_i = 1 &\rightarrow x_i = 0, & (-1)^{x_i} &= 1 = s_i \\ s_i = -1 &\rightarrow x_i = 1, & (-1)^{x_i} &= -1 = s_i \end{aligned}$$

Verifying MAXCUT Hamiltonian

- MAXCUT objective:

$$\max_{\mathbf{s}} \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j) \quad s_i \in \{-1, +1\}$$

- Let's reformulate it in the following way:

$$\begin{aligned} x_i &= \frac{1}{2}(1 - s_i) \\ s_i = 1 &\rightarrow x_i = 0, & (-1)^{x_i} &= 1 = s_i \\ s_i = -1 &\rightarrow x_i = 1, & (-1)^{x_i} &= -1 = s_i \end{aligned}$$

- New objective:

$$\max_{\mathbf{x}} \frac{1}{2} \sum_{ij \in E} (1 - (-1)^{x_i} (-1)^{x_j}) \quad x_i \in \{0, 1\}$$

Verifying MAXCUT Hamiltonian

- MAXCUT objective:

$$C(\mathbf{x}) = \frac{1}{2} \sum_{ij \in E} (1 - (-1)^{x_i} (-1)^{x_j}) \quad x_i \in \{0, 1\}$$

- MAXCUT Hamiltonian:

$$C = \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j)$$

- Want to show:

$$C |x\rangle = C(\mathbf{x}) |x\rangle$$

Verifying MAXCUT Hamiltonian

$$\begin{aligned} C|x\rangle &= \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) |x_0 \dots x_n\rangle \\ &= \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) |x_0 \dots x_n\rangle \\ &= \frac{1}{2} \sum_{ij \in E} (|x_0 \dots x_n\rangle - Z_i Z_j |x_0 \dots x_n\rangle) \\ &= \frac{1}{2} \sum_{ij \in E} (1 - (-1)^{x_i} (-1)^{x_j}) |x_0 \dots x_n\rangle = C(\mathbf{x}) |x\rangle \\ x_i &\in \{0, 1\}, \quad i = 1, \dots, n \end{aligned}$$

Constructing MAXCUT Hamiltonian

- MAXCUT objective:

$$\max_{\mathbf{s}} \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j) \quad s_i \in \{-1, +1\}$$

- MAXCUT Hamiltonian is constructed by mapping binary variables s_i onto the eigenvalues of Z

$$C = \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j)$$

- Note that the same procedure would work for any (unconstrained) binary objective!

Q: what is the size of Hamiltonian C?

$$C = \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j)$$

- Recall that: $Z_i Z_j = I \otimes \dots \otimes Z_i \otimes Z_j \otimes \dots \otimes I$
- Therefore each term in the sum is a $2^n \times 2^n$ matrix if written explicitly!
- Luckily, we do not have to write them out explicitly

Constructing a Hamiltonian for a general problem

- Recall that we want to construct a Hamiltonian C such that:

$$C |x\rangle = f(x) |x\rangle \quad \forall x \in \{0, 1\}^n$$

- How can we do this efficiently for an arbitrary function f ?
- In other words, how do we map Boolean and real functions to diagonal Hamiltonians acting on qubits?

This method is due to Hadfield, Stuart. "On the representation of Boolean and real functions as Hamiltonians for quantum computing." arXiv preprint arXiv:1804.09130 (2018).

Fourier analysis of Boolean functions

- Consider Boolean “maximum” function on two bits, $f = \max_2$

$$\max_2(+1, +1) = +1$$

$$\max_2(-1, +1) = +1$$

$$\max_2(+1, -1) = +1$$

$$\max_2(-1, -1) = -1$$

- Recall from a couple of slides ago that we can go between $x \in \{0, 1\}$ and $x \in \{-1, 1\}$ by performing a change of variables:

$$x \rightarrow (-1)^x$$

Fourier analysis of Boolean functions

- Consider Boolean “maximum” function on two bits, $f = \max_2$

$$\max_2(+1, +1) = +1$$

$$\max_2(-1, +1) = +1$$

$$\max_2(+1, -1) = +1$$

$$\max_2(-1, -1) = -1$$

- It can be expressed as a multilinear polynomial

$$\max_2(x_1, x_2) = \frac{1}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 - \frac{1}{2}x_1x_2$$

with up to 2^n terms corresponding to subsets of indices $S \subseteq [n]$

Fourier analysis of Boolean functions

- To construct this multilinear polynomial, note that:

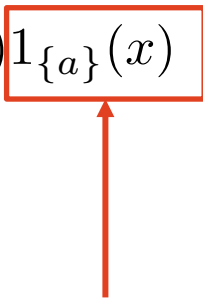
$$f(x) = \sum_{a \in \{-1,1\}^n} f(a) \boxed{1_{\{a\}}(x)}$$

Indicator polynomial



Fourier analysis of Boolean functions

- To construct this multilinear polynomial, note that:

$$f(x) = \sum_{a \in \{-1,1\}^n} f(a) \boxed{1_{\{a\}}(x)}$$


Indicator polynomial

- Indicator polynomial can be constructed as:

$$1_{\{a\}}(x) = \left(\frac{1 + a_1 x_1}{2} \right) \left(\frac{1 + a_2 x_2}{2} \right) \cdots \left(\frac{1 + a_n x_n}{2} \right)$$

Fourier analysis of Boolean functions

- To construct this multilinear polynomial, note that:

$$f(x) = \sum_{a \in \{-1,1\}^n} f(a) 1_{\{a\}}(x)$$

- For \max_2 example:

$$\begin{aligned} \max_2(x) &= (+1) \left(\frac{1+x_1}{2} \right) \left(\frac{1+x_2}{2} \right) \\ &\quad + (+1) \left(\frac{1-x_1}{2} \right) \left(\frac{1+x_2}{2} \right) \\ &\quad + (+1) \left(\frac{1+x_1}{2} \right) \left(\frac{1-x_2}{2} \right) \\ &\quad + (-1) \left(\frac{1-x_1}{2} \right) \left(\frac{1-x_2}{2} \right) = \frac{1}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 - \frac{1}{2}x_1x_2 \end{aligned}$$

Fourier analysis of Boolean functions

- Consider Boolean “maximum” function on two bits, $f = \max_2$

$$\max_2(+1, +1) = +1$$

$$\max_2(-1, +1) = +1$$

$$\max_2(+1, -1) = +1$$

$$\max_2(-1, -1) = -1$$

- It can be expressed as a multilinear polynomial

$$\max_2(x_1, x_2) = \frac{1}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 - \frac{1}{2}x_1x_2$$

with up to 2^n terms corresponding to subsets of indices $S \subseteq [n]$

- We write the monomial corresponding to S as

$$x^S = \prod_{i \in S} x_i \quad \left(\text{with } x^\emptyset = 1 \text{ by convention} \right)$$

Fourier analysis of Boolean functions

- Consider Boolean “maximum” function on two bits, $f = \max_2$

$$\max_2(+1, +1) = +1$$

$$\max_2(-1, +1) = +1$$

$$\max_2(+1, -1) = +1$$

$$\max_2(-1, -1) = -1$$

- It can be expressed as a multilinear polynomial, called the **Fourier expansion**:

$$\max_2(x_1, x_2) = \frac{1}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 - \frac{1}{2}x_1x_2 = \sum_{S \subseteq [2]} \widehat{\max_2}(S)x^S$$

with **Fourier coefficients**:

$$\widehat{\max_2}(\emptyset) = \frac{1}{2}, \quad \widehat{\max_2}(\{1\}) = \frac{1}{2}, \quad \widehat{\max_2}(\{2\}) = \frac{1}{2}, \quad \widehat{\max_2}(\{1, 2\}) = -\frac{1}{2};$$

Fourier analysis of Boolean functions

- More generally,

Theorem Every function $f : \{-1, 1\}^n \rightarrow \mathbb{R}$ can be uniquely expressed as a multilinear polynomial,

$$f(x) = \sum_{S \subseteq [n]} \hat{f}(S) x^S$$

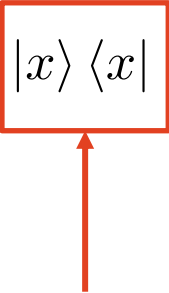
This expression is called the Fourier expansion of f , and the real number $\hat{f}(S)$ is called the Fourier coefficient of f on S . Collectively, the coefficients are called the Fourier spectrum of f .

Constructing a Hamiltonian for a general problem

- Recall that we want to construct a Hamiltonian C such that:

$$C |x\rangle = f(x) |x\rangle \quad \forall x \in \{0, 1\}^n$$

- Note that this is equivalent to:

$$C = \sum_{x \in \{0, 1\}^n} f(x) |x\rangle \langle x|$$


Projector operator

Constructing a Hamiltonian for a general problem

- Recall that we want to construct a Hamiltonian C such that:

$$C |x\rangle = f(x) |x\rangle \quad \forall x \in \{0, 1\}^n$$

- Note that this is equivalent to:

$$C = \sum_{x \in \{0, 1\}^n} f(x) |x\rangle \langle x| = \sum_{x \in \{0, 1\}^n} f(x) Pr_x$$

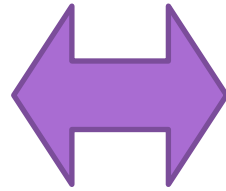
Projector operator

Constructing a Hamiltonian for a general problem

Classical

$$f(x) = \sum_{a \in \{-1,1\}^n} f(a) \boxed{1_{\{a\}}(x)}$$

Indicator polynomial



Quantum

$$C = \sum_{x \in \{0,1\}^n} f(x) \boxed{|x\rangle \langle x|} = \sum_{x \in \{0,1\}^n} f(x) Pr_x$$

Projector operator

Constructing a Hamiltonian for a general problem

- Now, note that products of Pauli Z operators implement parity functions:

$$\prod_{j \in S} Z_j |x\rangle = x^S |x\rangle$$

where

$$x^S = \prod_{i \in S} x_i \quad x_i \in \{-1, +1\}$$

Constructing a Hamiltonian for a general problem

Classical

$$f(x) = \sum_{a \in \{-1,1\}^n} f(a) 1_{\{a\}}(x)$$



Fourier
expansion

$$f(x) = \sum_{S \subseteq [n]} \hat{f}(S) x^S$$

Quantum

$$C = \sum_{x \in \{0,1\}^n} f(x) |x\rangle \langle x| = \sum_{x \in \{0,1\}^n} f(x) Pr_x$$



$$C = \sum_{S \subseteq [n]} \hat{f}(S) \prod_{j \in S} Z_j$$



Unique n-qubit Hamiltonian representing f

Constructing a Hamiltonian for a general problem

Classical

$$f(x) = \sum_{a \in \{-1,1\}^n} f(a) 1_{\{a\}}(x)$$



Fourier
expansion

$$f(x) = \sum_{S \subseteq [n]} \hat{f}(S) x^S$$

Quantum

$$C = \sum_{x \in \{0,1\}^n} f(x) |x\rangle \langle x| = \sum_{x \in \{0,1\}^n} f(x) Pr_x$$



$$C = \sum_{S \subseteq [n]} \hat{f}(S) \prod_{j \in S} Z_j$$



Unique n-qubit Hamiltonian representing f

Constructing a Hamiltonian for a general problem: a recipe

1. Perform Fourier expansion of the desired function:

$$f(x) = \sum_{S \subseteq [n]} \hat{f}(S) x^S$$

2. The unique n-qubit Hamiltonian representing f is given by:

$$C = \sum_{S \subseteq [n]} \hat{f}(S) \prod_{j \in S} Z_j$$

- Note: this is a general recipe!
- ... but in general it is #P-hard to compute the Fourier expansion

Constructing a Hamiltonian for a general problem: a *simpler* recipe

- Start with the table of building blocks:

$f(x)$	H_f	$f(x)$	H_f
x	$\frac{1}{2}I - \frac{1}{2}Z$	\bar{x}	$\frac{1}{2}I + \frac{1}{2}Z$
$x_1 \oplus x_2$	$\frac{1}{2}I - \frac{1}{2}Z_1 Z_2$	$\bigoplus_{j=1}^k x_j$	$\frac{1}{2}I - \frac{1}{2}Z_1 Z_2 \dots Z_k$
$x_1 \wedge x_2$	$\frac{1}{4}I - \frac{1}{4}(Z_1 + Z_2 - Z_1 Z_2)$	$\bigwedge_{j=1}^k x_j$	$\frac{1}{2^k} \prod_j (I - Z_j)$
$x_1 \vee x_2$	$\frac{3}{4}I - \frac{1}{4}(Z_1 + Z_2 + Z_1 Z_2)$	$\bigvee_{j=1}^k x_j$	$I - \frac{1}{2^k} \prod_j (I + Z_j)$
$\overline{x_1 x_2}$	$\frac{3}{4}I + \frac{1}{4}(Z_1 + Z_2 - Z_1 Z_2)$	$x_1 \Rightarrow x_2$	$\frac{3}{4}I + \frac{1}{4}(Z_1 - Z_2 + Z_1 Z_2)$

- Combine them as follows:

$$\begin{aligned}
 H_{\neg f} &= H_{\bar{f}} = I - H_f & H_{f \Rightarrow g} &= I - H_f + H_f H_g \\
 H_{f \wedge g} &= H_{fg} = H_f H_g & H_{f \vee g} &= H_f + H_g - H_f H_g \\
 H_{f \oplus g} &= H_f + H_g - 2H_f H_g & H_{af+bg} &= aH_f + bH_g \quad a, b \in \mathbb{R}.
 \end{aligned}$$

PART 2:
QUANTUM APPROXIMATE OPTIMIZATION
ALGORITHM (QAOA)

Quantum Approximate Optimization Algorithm (QAOA)

- QAOA prepares a parameterized “trial” (ansatz) state of the form:

$$\begin{aligned} |\psi(\boldsymbol{\theta})\rangle &= |\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle \\ &= e^{-i\beta_p B} e^{-i\gamma_p C} \dots e^{-i\beta_1 B} e^{-i\gamma_1 C} H^{\otimes n} |0\rangle. \end{aligned}$$

- Here C is the problem Hamiltonian, e.g. for MAXCUT: $C = \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j)$
- B is the mixer Hamiltonian: $B = \sum_i X_i$

Quantum Approximate Optimization Algorithm (QAOA)

- QAOA prepares a parameterized “trial” (ansatz) state of the form:

$$\begin{aligned} |\psi(\boldsymbol{\theta})\rangle &= |\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle \\ &= e^{-i\beta_p B} e^{-i\gamma_p C} \dots e^{-i\beta_1 B} e^{-i\gamma_1 C} H^{\otimes n} |0\rangle . \end{aligned}$$

- Then a classical optimizer is used to vary the parameters $\boldsymbol{\beta}, \boldsymbol{\gamma}$ to maximize:

$$f(\boldsymbol{\beta}, \boldsymbol{\gamma}) = \langle \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) | C | \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) \rangle .$$

Quantum Approximate Optimization Algorithm (QAOA)

- QAOA prepares a parameterized “trial” (ansatz) state of the form:

$$\begin{aligned} |\psi(\boldsymbol{\theta})\rangle &= |\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle \\ &= e^{-i\beta_p B} e^{-i\gamma_p C} \dots e^{-i\beta_1 B} e^{-i\gamma_1 C} H^{\otimes n} |0\rangle . \end{aligned}$$

- Note that for $p \rightarrow \infty$ QAOA can *at least* exactly approximate adiabatic quantum evolution and can therefore find the exact optimal solution
- For small p , picture is more mixed, but there is some indication of the potential for quantum advantage

Quantum Approximate Optimization Algorithm (QAOA)

- QAOA prepares a parameterized “trial” (ansatz) state of the form:

$$\begin{aligned} |\psi(\boldsymbol{\theta})\rangle &= |\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle \\ &= e^{-i\beta_p B} e^{-i\gamma_p C} \dots e^{-i\beta_1 B} e^{-i\gamma_1 C} H^{\otimes n} |0\rangle . \end{aligned}$$

How do we implement this circuit in gates?

Implementing QAOA

- Let's assume the following gate set

$$\begin{aligned} X &\equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; & Y &\equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}; & Z &\equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}; & H &\equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \\ S &\equiv \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}; & R_z(\theta) &\equiv e^{-i\theta Z/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} \end{aligned}$$

Implementing QAOA

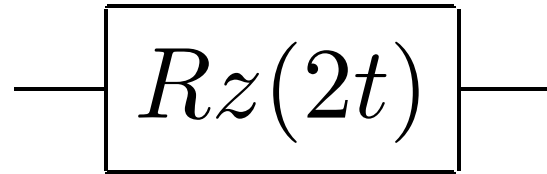
- Let's start with simple operator $e^{-iZt} = R_z(2t)$

$$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad R_z(\theta) \equiv e^{-i\theta Z/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$$

Implementing QAOA

- Let's start with simple operator $e^{-iZt} = R_z(2t)$

- Circuit:



$$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad R_z(\theta) \equiv e^{-i\theta Z/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$$

Implementing QAOA

- Slightly more complicated operator: $e^{-iZZt} = e^{-iZ \otimes Zt}$
- Remember that Z has eigenvectors $|0\rangle, |1\rangle$ with eigenvalues 1,-1 and $e^A |v\rangle = e^\lambda |v\rangle$ if $A |v\rangle = \lambda |v\rangle$

- Then:

$$e^{-iZ \otimes Zt} |00\rangle = e^{-i(1 \times 1)t} |00\rangle = e^{-it} |00\rangle$$

$$e^{-iZ \otimes Zt} |01\rangle = e^{-i(1 \times -1)t} |01\rangle = e^{it} |01\rangle$$

$$e^{-iZ \otimes Zt} |10\rangle = e^{-i(-1 \times 1)t} |10\rangle = e^{it} |10\rangle$$

$$e^{-iZ \otimes Zt} |11\rangle = e^{-i(-1 \times -1)t} |11\rangle = e^{-it} |11\rangle$$

- Adds a phase factor with the sign depending on parity! In general:

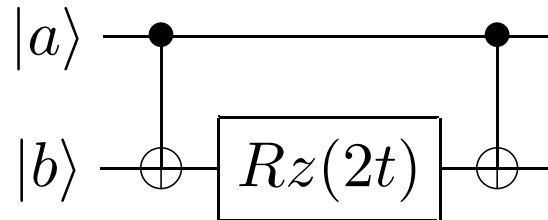
$$e^{-iZ \otimes Zt} |ab\rangle = e^{-i(-1)^{a \oplus b}t} |ab\rangle$$

Implementing QAOA

- Slightly more complicated operator: $e^{-iZZt} = e^{-iZ \otimes Z t}$
- Adds a phase factor with the sign depending on parity! In general:

$$e^{-iZ \otimes Z t} |ab\rangle = e^{-i(-1)^{a \oplus b} t} |ab\rangle$$

- Circuit:

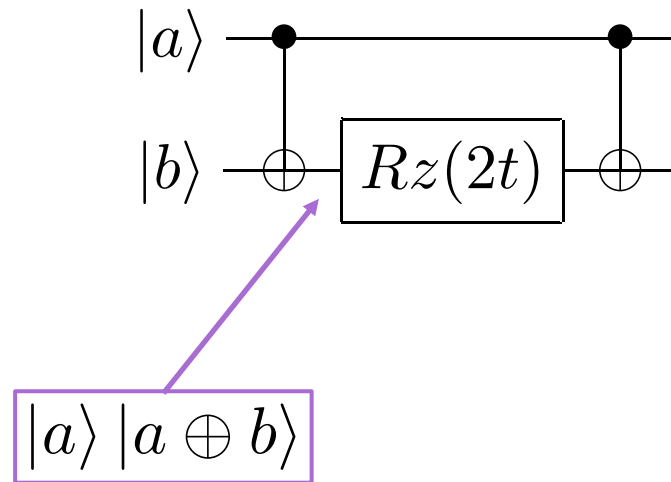


Implementing QAOA

- Slightly more complicated operator: $e^{-iZZt} = e^{-iZ \otimes Z t}$
- Adds a phase factor with the sign depending on parity! In general:

$$e^{-iZ \otimes Z t} |ab\rangle = e^{-i(-1)^{a \oplus b} t} |ab\rangle$$

- Circuit:

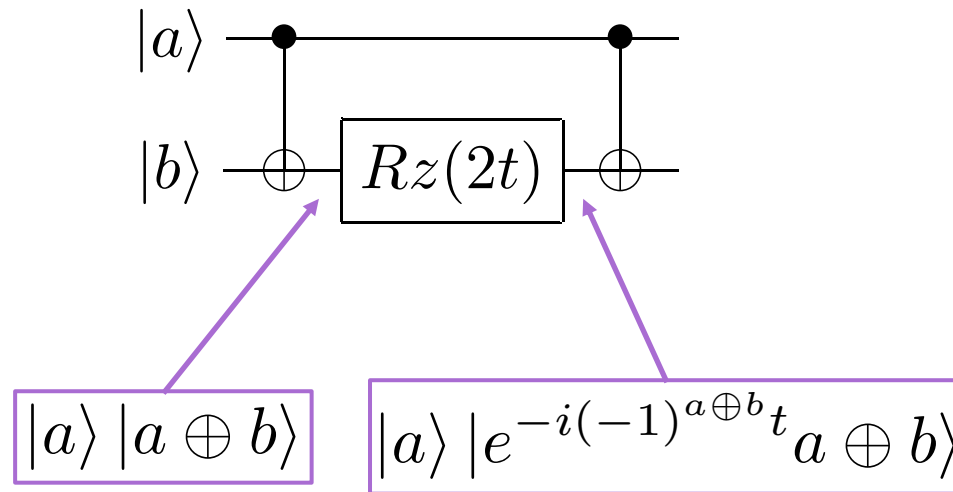


Implementing QAOA

- Slightly more complicated operator: $e^{-iZZt} = e^{-iZ \otimes Zt}$
- Adds a phase factor with the sign depending on parity! In general:

$$e^{-iZ \otimes Zt} |ab\rangle = e^{-i(-1)^{a \oplus b} t} |ab\rangle$$

- Circuit:

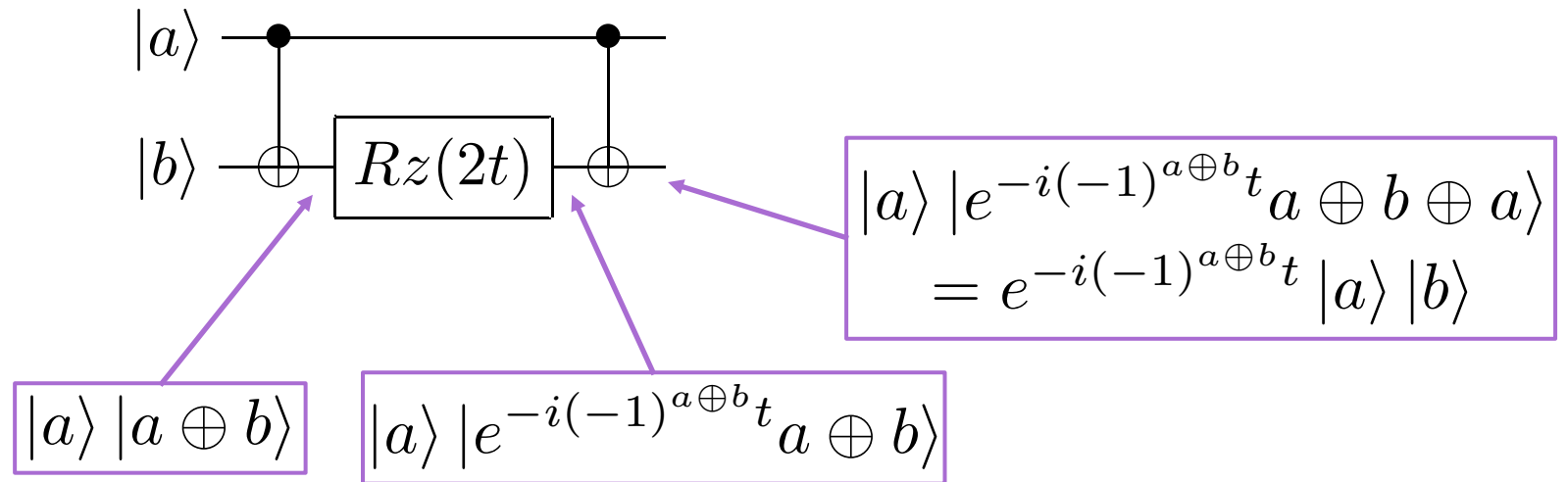


Implementing QAOA

- Slightly more complicated operator: $e^{-iZZt} = e^{-iZ \otimes Z t}$
- Adds a phase factor with the sign depending on parity! In general:

$$e^{-iZ \otimes Z t} |ab\rangle = e^{-i(-1)^{a \oplus b} t} |ab\rangle$$

- Circuit:

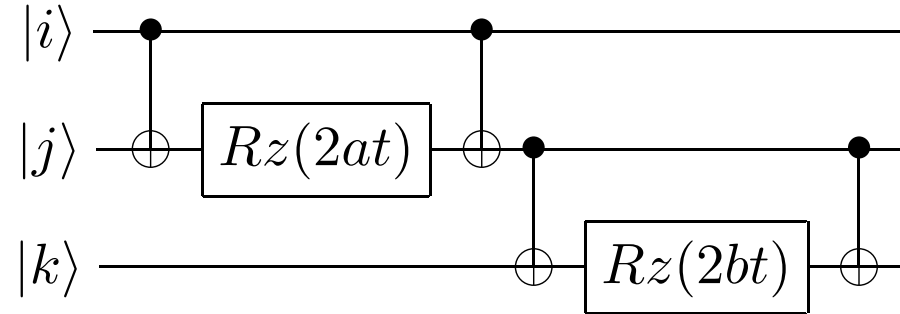


Implementing QAOA

- If terms of our Hamiltonian commute, we can just concatenate corresponding circuits:

$$e^{-iaZ_iZ_jt-ibZ_jZ_kt} = e^{-iaZ_iZ_jt}e^{-ibZ_jZ_kt}$$

- Circuit:

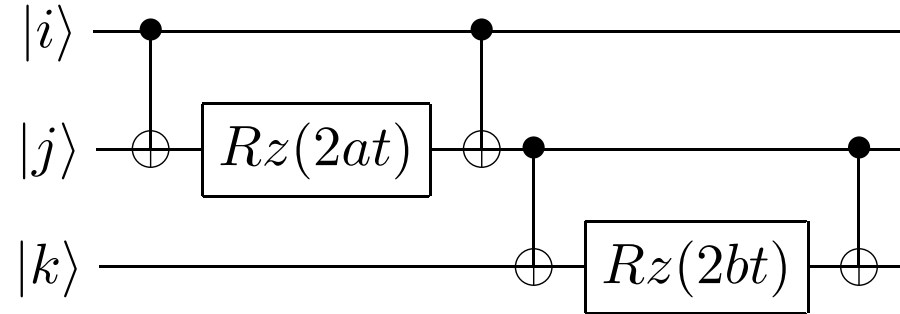


Implementing QAOA

- If terms of our Hamiltonian commute, we can just concatenate corresponding circuits:

$$e^{-iaZ_iZ_jt-ibZ_jZ_kt} = e^{-iaZ_iZ_jt}e^{-ibZ_jZ_kt}$$

- Circuit:



- Now we can simulate the operator corresponding to the problem Hamiltonian:

$$e^{-i\gamma C} = e^{-i\gamma \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j)}$$

Implementing QAOA

- QAOA prepares a parameterized “trial” (ansatz) state of the form:

$$\begin{aligned} |\psi(\boldsymbol{\theta})\rangle &= |\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle \\ &= e^{-i\beta_p B} e^{-i\gamma_p C} \dots e^{-i\beta_1 B} e^{-i\gamma_1 C} H^{\otimes n} |0\rangle . \end{aligned}$$

- Now we can simulate the operator corresponding to the problem Hamiltonian:

$$e^{-i\gamma C} = e^{-i\gamma \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j)}$$

Implementing QAOA

- Mixer operator: $e^{-i\beta B} = e^{-i\beta \sum_j X_j}$
- One term: e^{-iXt}

- Note that

$$e^{-iZt} = \sum_{j=0}^{\infty} \frac{(-iZt)^j}{j!} = I - iZt - \frac{Z^2t^2}{2!} + \dots$$

$$\begin{aligned} He^{-iZt}H &= H \left(I - iZt - \frac{Z^2t^2}{2!} + \dots \right) H = I - iHZHt - \frac{HZHHZHt^2}{2!} + \dots \\ &= I - iXt - \frac{X^2t^2}{2!} + \dots = e^{-iXt} \end{aligned}$$

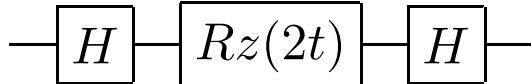
Implementing QAOA

- Mixer operator: $e^{-i\beta B} = e^{-i\beta \sum_j X_j}$
- One term: e^{-iXt}

- Note that

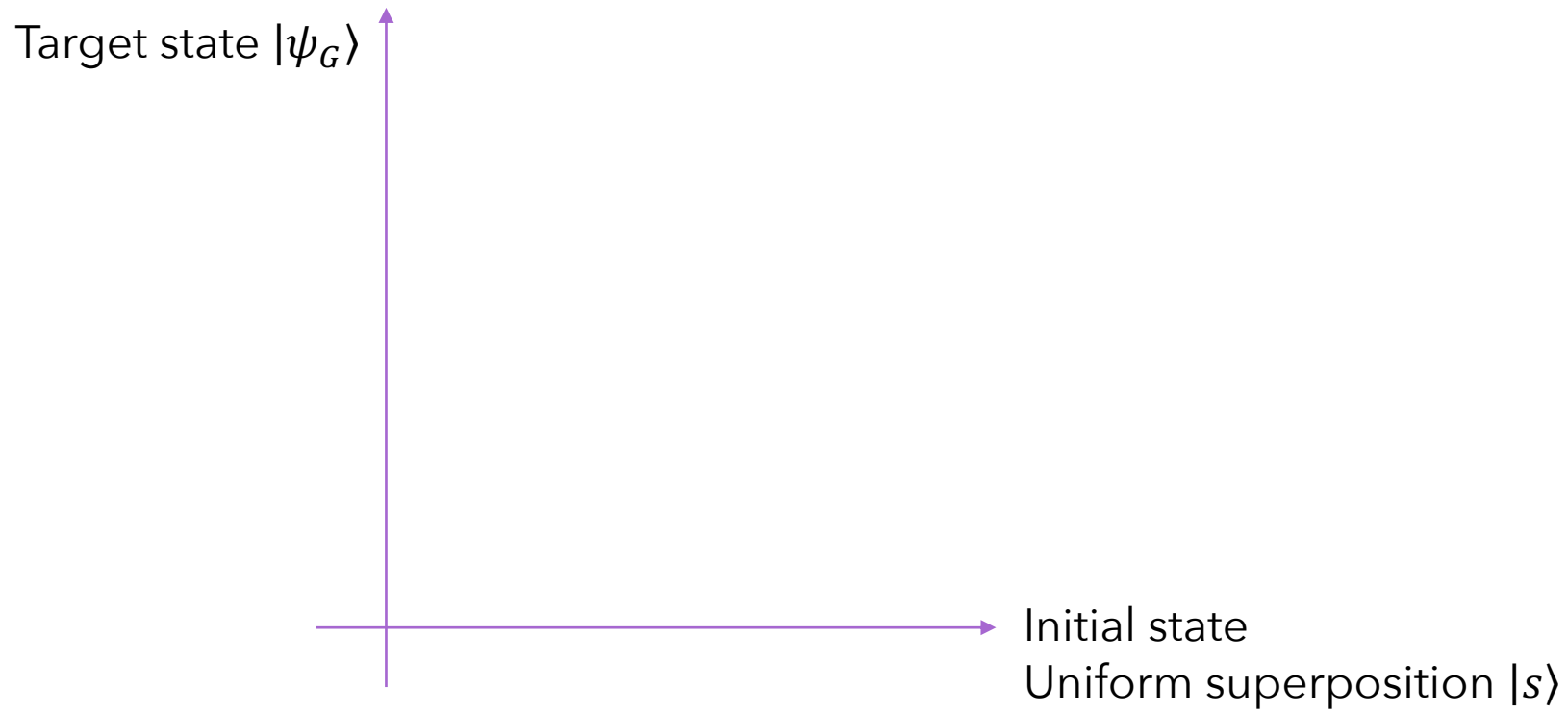
$$e^{-iZt} = \sum_{j=0}^{\infty} \frac{(-iZt)^j}{j!} = I - iZt - \frac{Z^2t^2}{2!} + \dots$$

$$\begin{aligned} He^{-iZt}H &= H \left(I - iZt - \frac{Z^2t^2}{2!} + \dots \right) H = I - iHZHt - \frac{HZHHZHt^2}{2!} + \dots \\ &= I - iXt - \frac{X^2t^2}{2!} + \dots = e^{-iXt} \end{aligned}$$

- Circuit: 

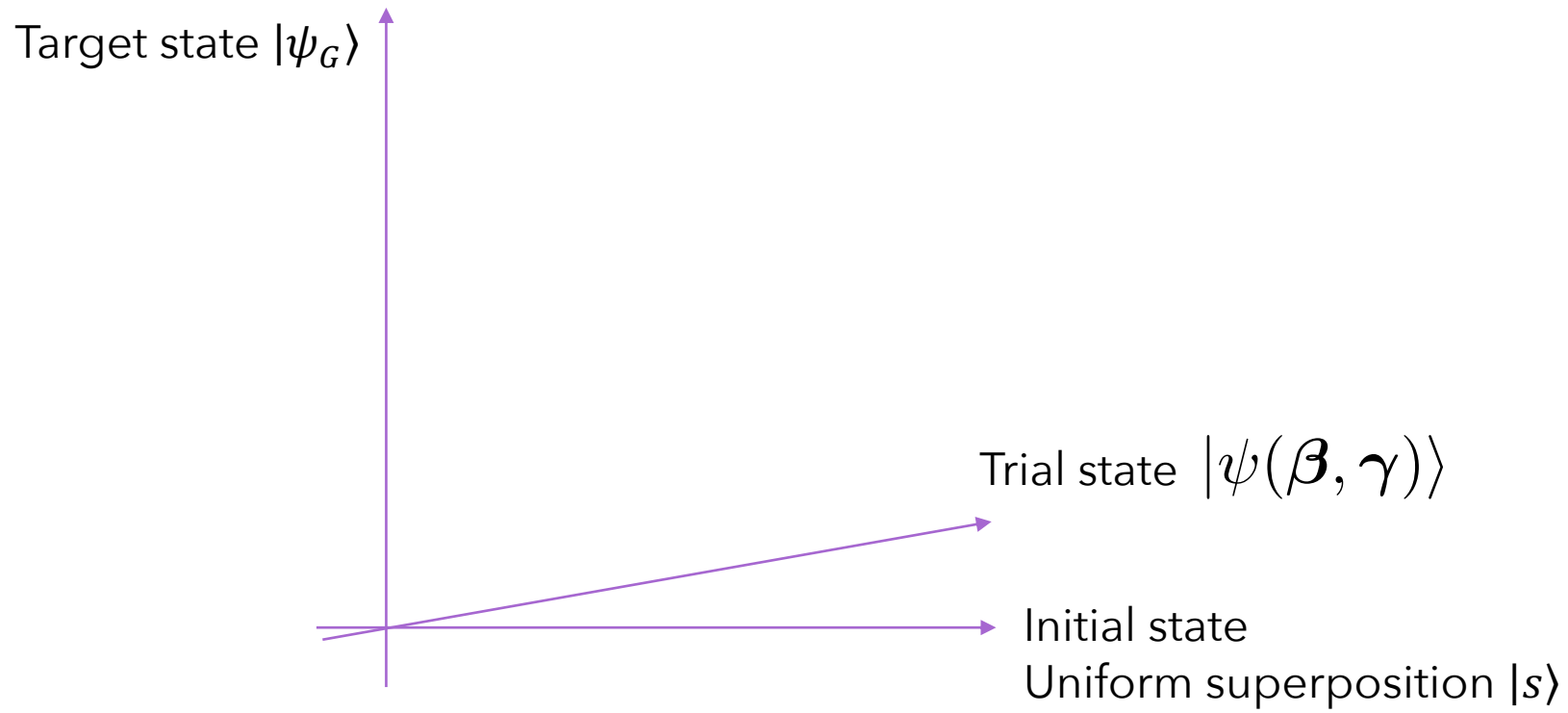
Quantum Approximate Optimization Algorithm (QAOA): Geometric Interpretation

- QAOA prepares a parameterized “trial” (ansatz) state $|\psi(\beta, \gamma)\rangle$



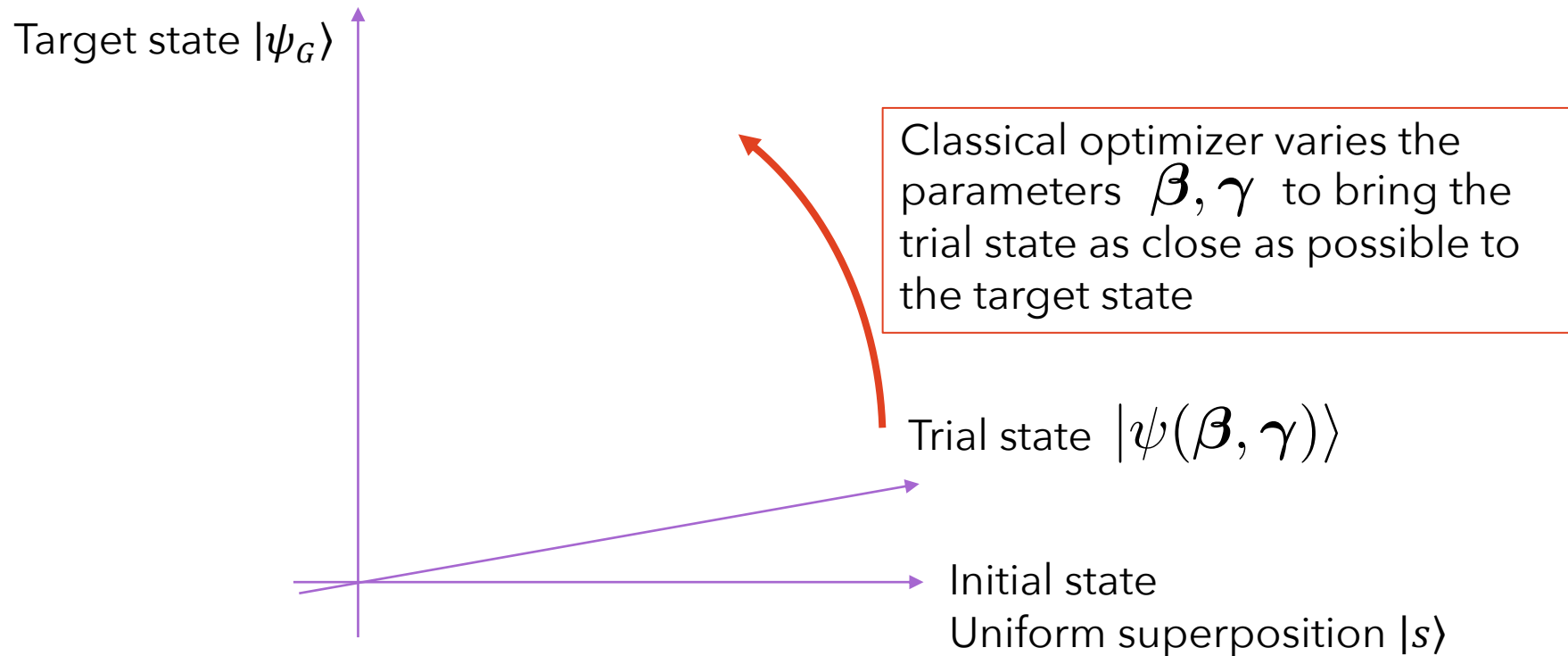
Quantum Approximate Optimization Algorithm (QAOA): Geometric Interpretation

- QAOA prepares a parameterized “trial” (ansatz) state $|\psi(\beta, \gamma)\rangle$



Quantum Approximate Optimization Algorithm (QAOA): Geometric Interpretation

- QAOA prepares a parameterized “trial” (ansatz) state $|\psi(\beta, \gamma)\rangle$



Quantum Approximate Optimization Algorithm (QAOA)

- QAOA prepares a parameterized “trial” (ansatz) state of the form:

$$\begin{aligned} |\psi(\boldsymbol{\theta})\rangle &= |\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle \\ &= e^{-i\beta_p \hat{H}_M} e^{-i\gamma_p \hat{H}_C} \dots e^{-i\beta_1 \hat{H}_M} e^{-i\gamma_1 \hat{H}_C} |+\rangle^{\otimes n}. \end{aligned}$$

- Crucially, the quality of QAOA solution heavily depends on the quality of the parameters found by the classical optimizer

Adiabatic Quantum Computation (AQC)

- A way to find the ground / highest energy state of the problem Hamiltonian, inspired by adiabatic approximation theorems
- Adiabatic approximation states, roughly, that a system prepared in an eigenstate (e.g. a ground state) of some time-dependent Hamiltonian $H(t)$ will remain in the corresponding eigenstate provided that $H(t)$ is varied “slowly enough”.
- **Idea:** prepare a system in an eigenstate of a simple Hamiltonian and adiabatically interpolate it into the corresponding eigenstate of problem Hamiltonian
- Interpolation is given by a time-dependent Hamiltonian:

$$H(t) = (1 - s(t))H_D + s(t)H_P$$

- $s(t)$ is a smooth function s.t. $s(t = 0) = 0$ and $s(t = T) = 1$

Adiabatic Quantum Computation for MAXCUT

- Note minus because by convention AQC goes from ground state to ground state

$$\begin{aligned}H(t) &= (1 - s(t))H_D + s(t)H_P \\H_P &= -C = -\frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) \\H_D &= -\sum_j X_j \\s &= |+\rangle^{\otimes n} = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} |x\rangle\end{aligned}$$

- s is the ground state of driver Hamiltonian H_D and is the initial state

Simulation of AQC

- How can we simulate this time-dependent Hamiltonian on a quantum computer?

$$\begin{aligned}H(t) &= (1 - s(t))H_D + s(t)H_P \\H_P &= -C = -\frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) \\H_D &= -\sum_j X_j \\s &= |+\rangle^{\otimes n} = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} |x\rangle\end{aligned}$$

- To answer this, let's go to the basics

Simulation of AQC

- Consider a general quantum evolution, defined by a unitary U :

$$|\psi(t)\rangle = U(t) |\psi(0)\rangle$$

- Evolution is related to system Hamiltonian by:

$$i \frac{dU(t)}{dt} = H(t)U(t)$$

- For a time-dependent Hamiltonian such that $[H(t_1), H(t_2)] = 0$, $\forall t_1, t_2$
the solution of this equation is:

$$U(t) = e^{-i \int_0^t H(t) dt}$$

Simulation of AQC

- Consider a general quantum evolution, defined by a unitary U :

$$|\psi(t)\rangle = U(t) |\psi(0)\rangle$$

- Evolution is related to system Hamiltonian:

$$i \frac{dU(t)}{dt} = H(t)U(t)$$

- For a time-independent Hamiltonian solution this simplifies to:

$$U(t) = e^{-iHt}$$

Approximating Non-Commuting Hamiltonians

- (**Trotter formula**) For Hermitian A and B and any real t:

$$\lim_{n \rightarrow \infty} (e^{iAt/n} e^{iBt/n})^n = e^{i(A+B)t}$$

Approximating Non-Commuting Hamiltonians

- (**Trotter formula**) For Hermitian A and B and any real t :

$$e^{i(A+B)t} = \lim_{n \rightarrow \infty} (e^{iAt/n} e^{iBt/n})^n$$

- Corollary (**Lie-Trotter-Suzuki** decompositions):

$$e^{i(A+B)t} = e^{iAt} e^{iBt} + O(t^2)$$

$$e^{i(A+B)t} = e^{iBt/2} e^{iAt} e^{iBt/2} + O(t^3)$$

Discretizing AQC

- Let's now consider our time-dependent Hamiltonian $H(t)$ and evolution operator:

$$U(t) = e^{-i \int_0^t H(t) dt}$$

Discretizing AQC

- Let's now consider our time-dependent Hamiltonian $H(t)$ and evolution operator:

$$U(t) = e^{-i \int_0^t H(t) dt}$$

- We can break it up as follows:

$$U(T, 0) = U(T, T - \Delta t)U(T - \Delta t, T - 2\Delta t) \cdots U(\Delta t, 0)$$

Discretizing AQC

- Let's now consider our time-dependent Hamiltonian $H(t)$ and evolution operator:

$$U(t) = e^{-i \int_0^t H(t) dt}$$

- We can break it up as follows:

$$U(T, 0) = U(T, T - \Delta t)U(T - \Delta t, T - 2\Delta t) \cdots U(\Delta t, 0)$$

- Now, choose time step Δt to be small enough so that $H(t)$ is approximately constant over the interval:

$$U(j\Delta t, (j-1)\Delta t) = e^{-i \int_{(j-1)\Delta t}^{j\Delta t} H(t) dt} \approx e^{-i H(j\Delta t) \Delta t}$$

Discretizing AQC

- Let's now consider our time-dependent Hamiltonian $H(t)$ and evolution operator:

$$U(t) = e^{-i \int_0^t H(t) dt}$$

- We can break it up as follows:

$$U(T, 0) = U(T, T - \Delta t)U(T - \Delta t, T - 2\Delta t) \cdots U(\Delta t, 0)$$

- Now, choose time step Δt to be small enough so that $H(t)$ is approximately constant over the interval:

$$U(j\Delta t, (j-1)\Delta t) \approx e^{-iH(j\Delta t)\Delta t}$$

- Combining both, we get:
$$U(T, 0) \approx \prod_{j=1}^p e^{-iH(j\Delta t)\Delta t}$$

Discretizing AQC

$$U(T, 0) \approx \prod_{j=1}^p e^{-iH(j\Delta t)\Delta t}$$

- Now we can apply Lie-Trotter-Suzuki decomposition:

$$e^{i(A+B)t} = e^{iAt}e^{iBt} + O(t^2)$$

Discretizing AQC

$$\begin{aligned} U(T, 0) &\approx \prod_{j=1}^p e^{-iH(j\Delta t)\Delta t} = \prod_{j=1}^p e^{i((1-s(j\Delta t))H_D + s(j\Delta t)H_P)\Delta t} \\ &= \left[\text{applying Lie-Trotter-Suzuki decomposition: } e^{i(A+B)t} = e^{iAt}e^{iBt} + O(t^2) \right] \\ &\approx \prod_{j=1}^p e^{i(1-s(j\Delta t))\Delta t H_D} e^{i s(j\Delta t)\Delta t H_P} \end{aligned}$$

Understanding QAOA

$$\begin{aligned} |\gamma, \beta\rangle &\equiv U_{\text{QAOA}_p}(\gamma, \beta) |s\rangle \\ U_{\text{QAOA}_p}(\gamma, \beta) &\equiv \prod_{j=1}^p e^{-i\beta_j B} e^{-i\gamma_j C} \\ C &= \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) \\ B &= \sum_j X_j \\ s &= |+\rangle^{\otimes n} \end{aligned}$$

QAOA

$$\begin{aligned} |\psi\rangle &= U_{\text{AQC}} |s\rangle \\ U_{\text{AQC}} &= \prod_{j=1}^p e^{i(1-s(j\Delta t))\Delta t H_D} e^{i s(j\Delta t)\Delta t H_P} \\ H_P &= -C = -\frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) \\ H_D &= -B = -\sum_j X_j \\ s &= |+\rangle^{\otimes n} \end{aligned}$$

Simulated AQC

Understanding QAOA

$$\begin{aligned} |\gamma, \beta\rangle &\equiv U_{\text{QAOA}_p}(\gamma, \beta) |s\rangle \\ U_{\text{QAOA}_p}(\gamma, \beta) &\equiv \prod_{j=1}^p e^{-i\beta_j B} e^{-i\gamma_j C} \\ C &= \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) \\ B &= \sum_j X_j \\ s &= |+\rangle^{\otimes n} \end{aligned}$$

QAOA

$$\begin{aligned} |\psi\rangle &= U_{\text{AQC}} |s\rangle \\ U_{\text{AQC}} &= \prod_{j=1}^p e^{i(1-s(j\Delta t))\Delta t H_D} e^{i s(j\Delta t)\Delta t H_P} \\ H_P &= -C = -\frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) \\ H_D &= -B = -\sum_j X_j \\ s &= |+\rangle^{\otimes n} \end{aligned}$$

Simulated AQC

- QAOA is equivalent to simulated adiabatic computation if we set

$$\gamma_j = -s(j\Delta t)\Delta t, \quad \beta_j = -1[1 - s(j\Delta t)]\Delta t$$

Understanding QAOA

$$\begin{aligned} |\gamma, \beta\rangle &\equiv U_{\text{QAOA}_p}(\gamma, \beta) |s\rangle \\ U_{\text{QAOA}_p}(\gamma, \beta) &\equiv \prod_{j=1}^p e^{-i\beta_j B} e^{-i\gamma_j C} \\ C &= \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) \\ B &= \sum_j X_j \\ s &= |+\rangle^{\otimes n} \end{aligned}$$

QAOA

$$\begin{aligned} |\psi\rangle &= U_{\text{AQC}} |s\rangle \\ U_{\text{AQC}} &= \prod_{j=1}^p e^{i(1-s(j\Delta t))\Delta t H_D} e^{i s(j\Delta t)\Delta t H_P} \\ H_P &= -C = -\frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) \\ H_D &= -B = -\sum_j X_j \\ s &= |+\rangle^{\otimes n} \end{aligned}$$

Simulated AQC

- However, the non-adiabatic mechanism of QAOA is quite different!

See e.g. Jiang, Zhang, Eleanor G. Rieffel, and Zhihui Wang. "Near-optimal quantum circuit for Grover's unstructured search using a transverse field." *Physical Review A* 95.6 (2017): 062317.

Evaluating shallow-depth QAOA energy classically

- QAOA prepares a parameterized “trial” (ansatz) state of the form:

$$\begin{aligned} |\psi(\boldsymbol{\theta})\rangle &= |\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle \\ &= e^{-i\beta_p B} e^{-i\gamma_p C} \dots e^{-i\beta_1 B} e^{-i\gamma_1 C} H^{\otimes n} |0\rangle . \end{aligned}$$

- Then a classical optimizer is used to vary the parameters $\boldsymbol{\beta}, \boldsymbol{\gamma}$ to maximize:

$$f(\boldsymbol{\beta}, \boldsymbol{\gamma}) = \langle \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) | C | \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) \rangle .$$

- If we can evaluate $f(\boldsymbol{\beta}, \boldsymbol{\gamma})$ efficiently classically, we can train QAOA without access to a quantum computer!

Evaluating shallow-depth QAOA energy classically

- Let $U(B, \beta) = e^{-i\beta B}, U(C, \gamma) = e^{-i\gamma C}$

Evaluating shallow-depth QAOA energy classically

- Let $U(B, \beta) = e^{-i\beta B}$, $U(C, \gamma) = e^{-i\gamma C}$
- Then QAOA energy for $p=1$:

$$\begin{aligned} f(\beta_1, \gamma_1) &= \langle s | U^\dagger(C, \gamma_1) U^\dagger(B, \beta_1) \frac{1}{2} \sum_{(j,k) \in E} (I - Z_j Z_k) U(B, \beta_1) U(C, \gamma_1) | s \rangle \\ &= \frac{|E|}{2} - \frac{1}{2} \sum_{(j,k) \in E} \langle s | U^\dagger(C, \gamma_1) U^\dagger(B, \beta_1) Z_j Z_k U(B, \beta_1) U(C, \gamma_1) | s \rangle \\ &= \frac{|E|}{2} - \frac{1}{2} \sum_{(j,k) \in E} f_{(j,k)}(\beta_1, \gamma_1) \end{aligned}$$

Evaluating shallow-depth QAOA energy classically

- Let $U(B, \beta) = e^{-i\beta B}$, $U(C, \gamma) = e^{-i\gamma C}$
- Then QAOA energy for $p=1$:

$$\begin{aligned} f(\beta_1, \gamma_1) &= \langle s | U^\dagger(C, \gamma_1) U^\dagger(B, \beta_1) \frac{1}{2} \sum_{(j,k) \in E} (I - Z_j Z_k) U(B, \beta_1) U(C, \gamma_1) | s \rangle \\ &= \frac{|E|}{2} - \frac{1}{2} \sum_{(j,k) \in E} \langle s | U^\dagger(C, \gamma_1) U^\dagger(B, \beta_1) Z_j Z_k U(B, \beta_1) U(C, \gamma_1) | s \rangle \\ &= \frac{|E|}{2} - \frac{1}{2} \sum_{(j,k) \in E} f_{(j,k)}(\beta_1, \gamma_1) \end{aligned}$$

- Let's look more closely at the contribution of one edge:

$$f_{(j,k)}(\beta_1, \gamma_1) = \langle s | U^\dagger(C, \gamma_1) U^\dagger(B, \beta_1) Z_j Z_k U(B, \beta_1) U(C, \gamma_1) | s \rangle$$

Evaluating shallow-depth QAOA energy classically

- Let's look more closely at the contribution of one edge:

$$f_{(j,k)}(\beta_1, \gamma_1) = \langle s | U^\dagger(C, \gamma_1) U^\dagger(B, \beta_1) Z_j Z_k U(B, \beta_1) U(C, \gamma_1) | s \rangle$$

- Let's go through it step-by-step, from the middle out:

$$\begin{aligned} U^\dagger(B, \beta_1) Z_j Z_k U(B, \beta_1) &= (e^{i\beta_1 \sum_m X_m}) Z_j Z_k (e^{-i\beta_1 \sum_m X_m}) \\ &= (\prod_m e^{i\beta_1 I \otimes \dots \otimes X_m \otimes \dots I}) (I \otimes \dots \otimes Z_j Z_k \otimes \dots \otimes I) (\prod_m e^{-i\beta_1 I \otimes \dots \otimes X_m \otimes \dots I}) \end{aligned}$$

Evaluating shallow-depth QAOA energy classically

- Let's look more closely at the contribution of one edge:

$$f_{(j,k)}(\beta_1, \gamma_1) = \langle s | U^\dagger(C, \gamma_1) U^\dagger(B, \beta_1) Z_j Z_k U(B, \beta_1) U(C, \gamma_1) | s \rangle$$

- Let's go through it step-by-step, from the middle out:

$$\begin{aligned} U^\dagger(B, \beta_1) Z_j Z_k U(B, \beta_1) &= (e^{i\beta_1 \sum_m X_m}) Z_j Z_k (e^{-i\beta_1 \sum_m X_m}) \\ &= (\prod_m e^{i\beta_1 I \otimes \dots \otimes X_m \otimes \dots I}) (I \otimes \dots \otimes Z_j Z_k \otimes \dots \otimes I) (\prod_m e^{-i\beta_1 I \otimes \dots \otimes X_m \otimes \dots I}) \end{aligned}$$

- Note that whenever $m \notin \{j, k\}$ we can commute the term $e^{i\beta_1 X_m} = e^{i\beta_1 I \otimes \dots \otimes X_m \otimes \dots I}$ through $Z_j Z_k$:

$$\begin{aligned} e^{i\beta_1 I \otimes \dots \otimes X_m \otimes \dots I} (I \otimes \dots \otimes Z_j Z_k \otimes \dots \otimes I) (e^{-i\beta_1 I \otimes \dots \otimes X_m \otimes \dots I}) &= Z_j Z_k e^{i\beta_1 I \otimes \dots \otimes X_m \otimes \dots I} (e^{-i\beta_1 I \otimes \dots \otimes X_m \otimes \dots I}) \\ &= Z_j Z_k \end{aligned}$$

Evaluating shallow-depth QAOA energy classically

- From this it is clear that out of all mixer terms only the terms acting on the qubits j, k contribute to $f_{(j,k)}$:

$$U^\dagger(B, \beta_1) Z_j Z_k U(B, \beta_1) = (e^{i\beta_1(X_j + X_k)}) Z_j Z_k (e^{-i\beta_1(X_j + X_k)})$$

Evaluating shallow-depth QAOA energy classically

- From this it is clear that out of all mixer terms only the terms acting on the qubits j, k contribute to $f_{(j,k)}$:

$$U^\dagger(B, \beta_1) Z_j Z_k U(B, \beta_1) = (e^{i\beta_1(X_j + X_k)}) Z_j Z_k (e^{-i\beta_1(X_j + X_k)})$$

- Now, let's add the phase separation operator. It's easy to see that similarly all terms that do not have either j or k in them will commute through and cancel out:

$$\begin{aligned} U^\dagger(C, \gamma_1) U^\dagger(B, \beta_1) Z_j Z_k U(B, \beta_1) U(C, \gamma_1) &= \\ &= U^\dagger(C, \gamma_1) (e^{i\beta_1(X_j + X_k)}) Z_j Z_k (e^{-i\beta_1(X_j + X_k)}) U(C, \gamma_1) \\ &= (e^{i\gamma_1 \sum_{(u,v) \in \text{nbhd}((j,k))} Z_u Z_v}) (e^{i\beta_1(X_j + X_k)}) Z_j Z_k (e^{-i\beta_1(X_j + X_k)}) (e^{-i\gamma_1 \sum_{(u,v) \in \text{nbhd}((j,k))} Z_u Z_v}) \end{aligned}$$

Evaluating shallow-depth QAOA energy classically

- Full term:

$$\begin{aligned} f_{(j,k)}(\beta_1, \gamma_1) &= \langle RCC | (e^{i\gamma_1 \sum_{(u,v) \in \text{nbhd}((j,k))} Z_u Z_v}) (e^{i\beta_1 (X_j + X_k)}) Z_j Z_k (e^{-i\beta_1 (X_j + X_k)}) (e^{-i\gamma_1 \sum_{(u,v) \in \text{nbhd}((j,k))} Z_u Z_v}) | RCC \rangle \\ &= \langle RCC_{tree}(\beta_1, \gamma_1) | Z_j Z_k | RCC_{tree}(\beta_1, \gamma_1) \rangle \end{aligned}$$

- Here we denote a uniform superposition over states on the qubits in reverse causal cone of $Z_j Z_k$ by $|RCC\rangle = \bigotimes_{m \in \text{nbhd}((j,k))} |+\rangle$
- Now, this can be easily extended to larger neighborhoods in BFS fashion by adding more QAOA operators, since the k -th operator will only "bring into play" terms $Z_u Z_v$ corresponding to the edges from k -neighborhood of (j,k) (the other terms will commute through and cancel out like shown above). In other words, only the terms in the *light cone* of $Z_j Z_k$ affect the energy.

Summary

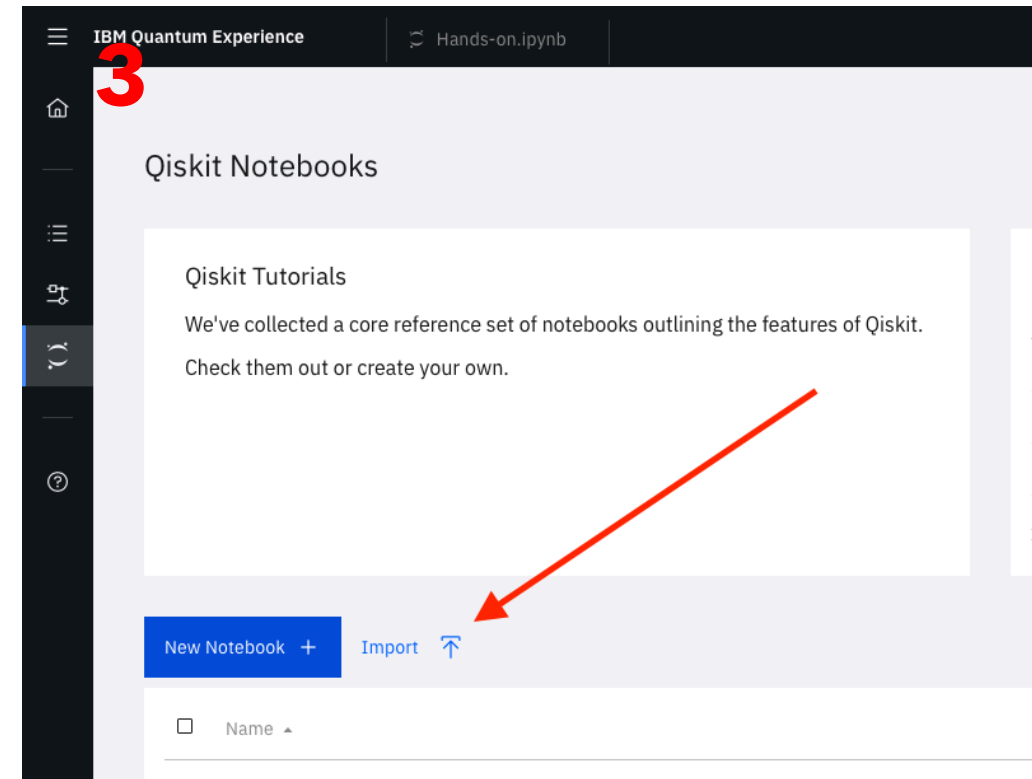
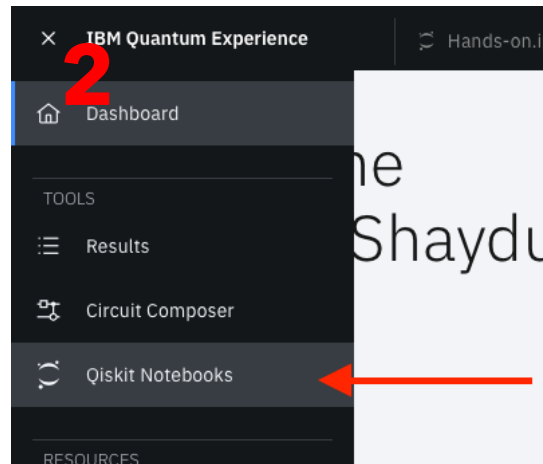
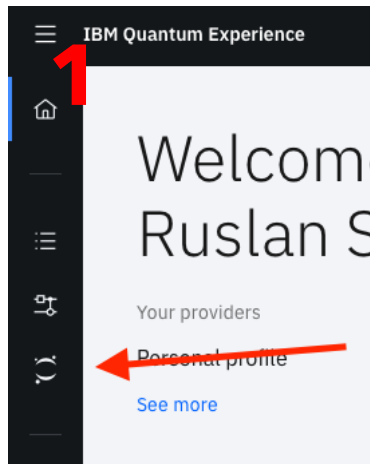
- QAOA will not solve NP-complete problems in polynomial time
- QAOA is a promising *heuristic* for the NISQ era
- For any Boolean function on n bits we can construct a unique n -qubit Hamiltonian representing it
- QAOA is deeply connected to Adiabatic Quantum Optimization Algorithm
- QAOA energy can often be evaluated purely classically

Summary

- QAOA will not solve NP-complete problems in polynomial time
- QAOA is a promising *heuristic* for the NISQ era
- For any Boolean function on n bits we can construct a unique n -qubit Hamiltonian representing it
- QAOA is deeply connected to Adiabatic Quantum Optimization Algorithm
- QAOA energy can often be evaluated purely classically
- **Coming up next:** hands-on, decomposition and advanced simulation

PART 3: HANDS-ON

- Get the latest version of the notebook from https://github.com/rsln-s/IEEE_QW_2020 or <http://bit.ly/qw2020tut>
- Go to quantum-computing.ibm.com and login (you might need to create an IBM ID if you haven't already!)
- Go to quantum-computing.ibm.com/jupyter
If you don't like typing, there's a button on home page that takes you there:



- Import the notebook Hands-on.ipynb:
- Note that you can also run the notebook locally on your machine, but you'll have to set up your own environment