Combinatorial Optimization on Quantum Computers

Ruslan Shaydulin, Argonne National Laboratory

Argonne NATIONAL LABORATORY

PART 0:
SOME BIG-PICTURE CONSIDERATIONS

Complexity of solving problems

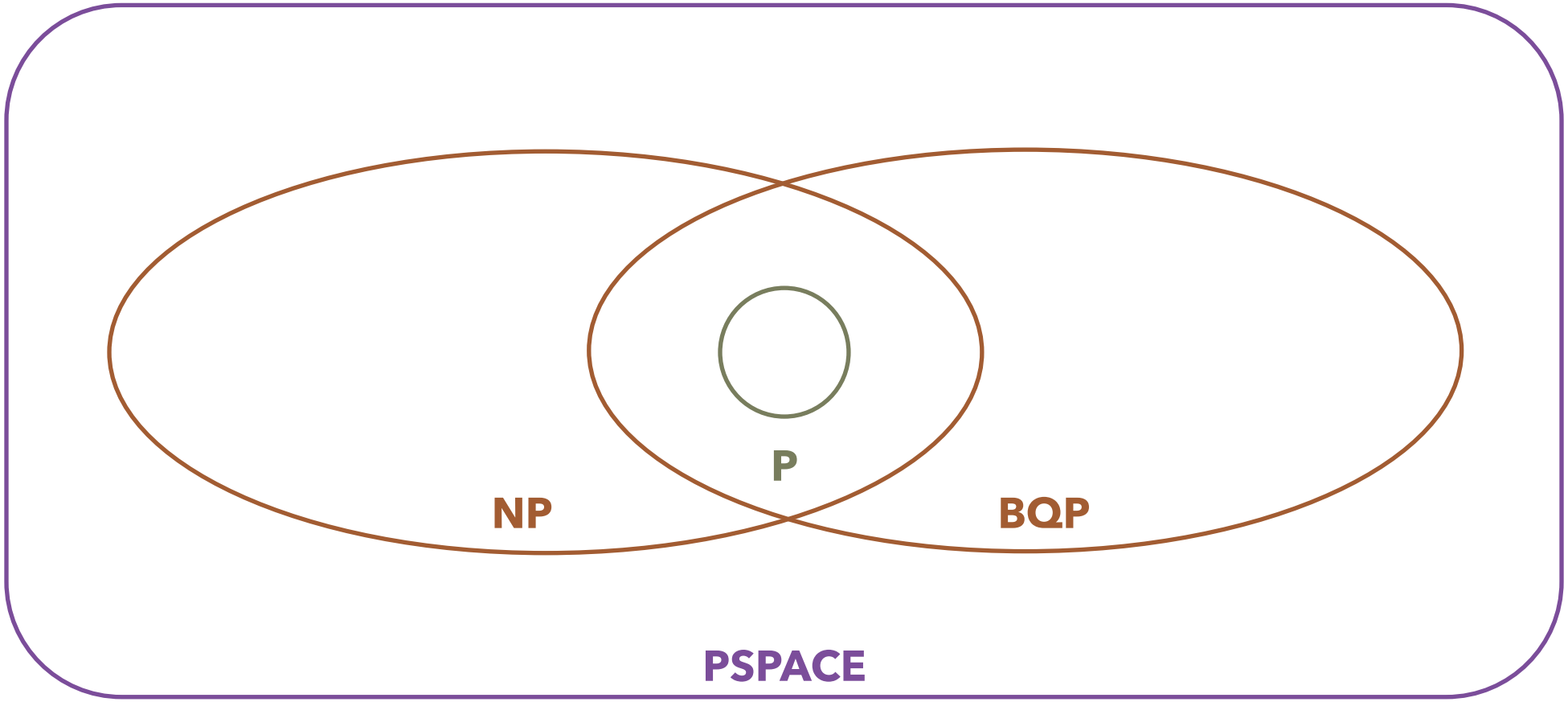
- When reasoning about complexity of solving problems (including optimization problems!), **we are usually interested in how the time / memory requirements grow with problem size** (asymptotic complexity)

Complexity of solving problems

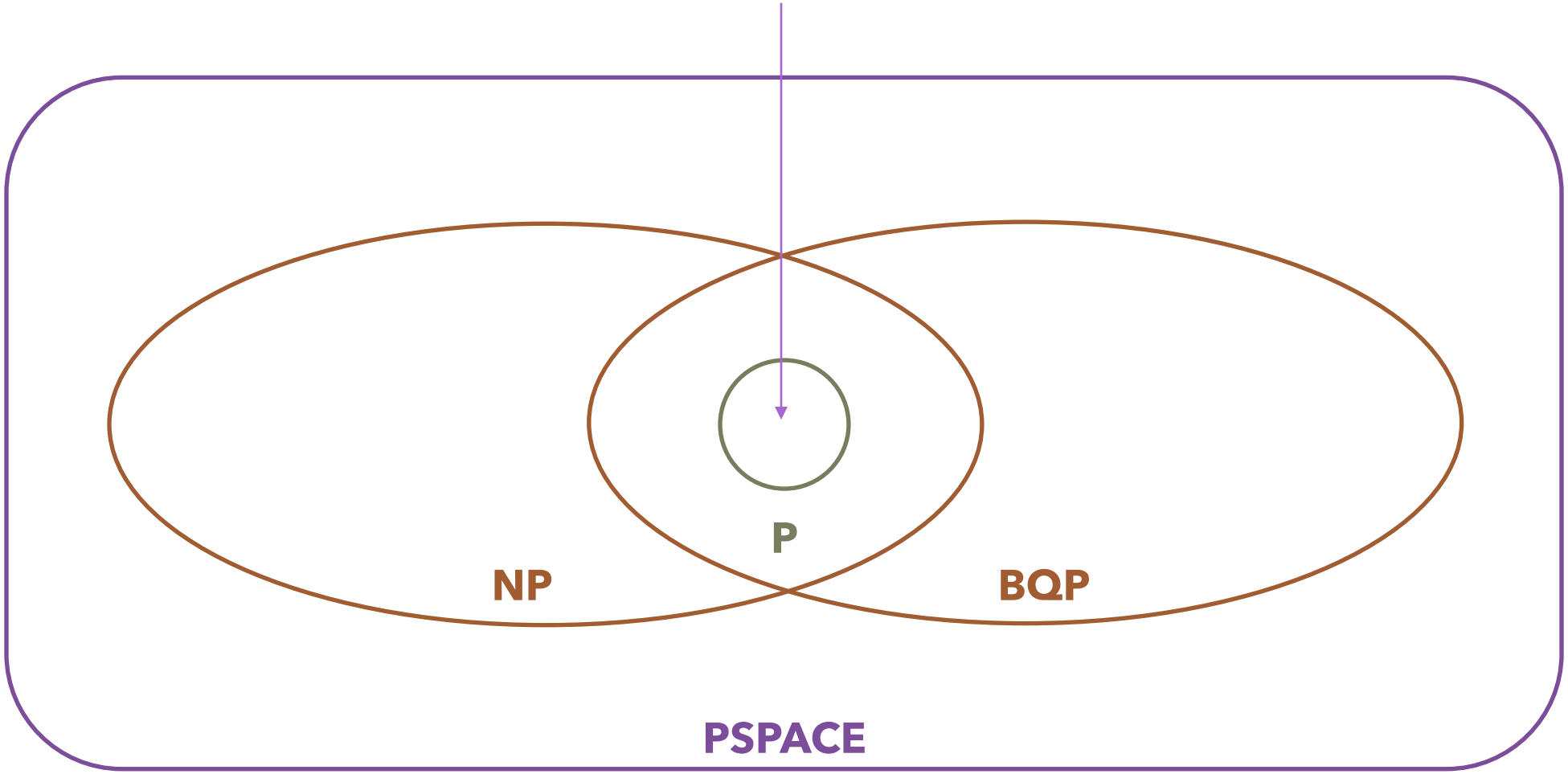
- When reasoning about complexity of solving problems (including optimization problems!), **we are usually interested in how the time / memory requirements grow with problem size** (asymptotic complexity)
- Famous classes of problems:
  - P – solvable in polynomial time
  - NP – can verify *proof* of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

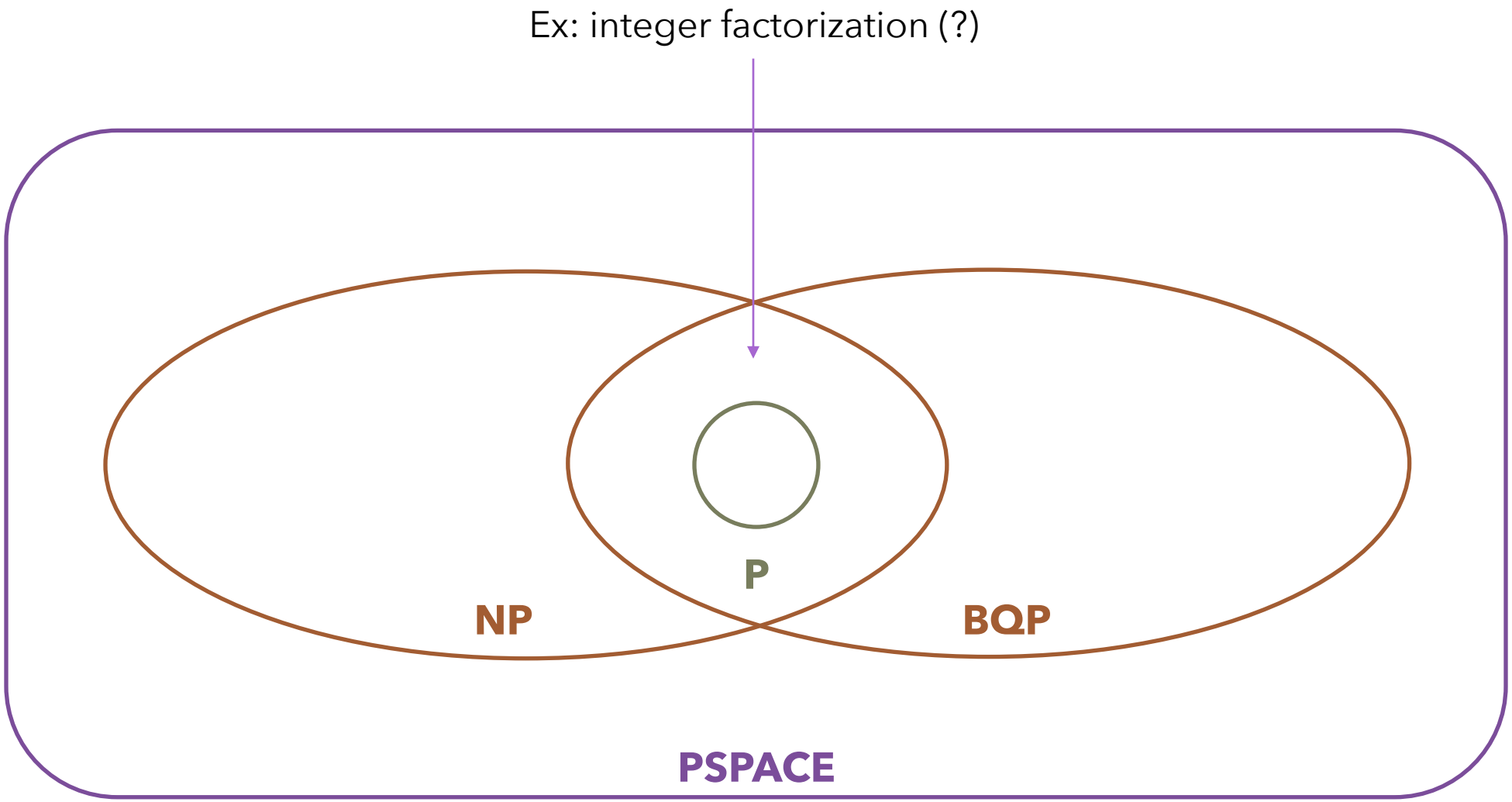
Complexity of solving problems

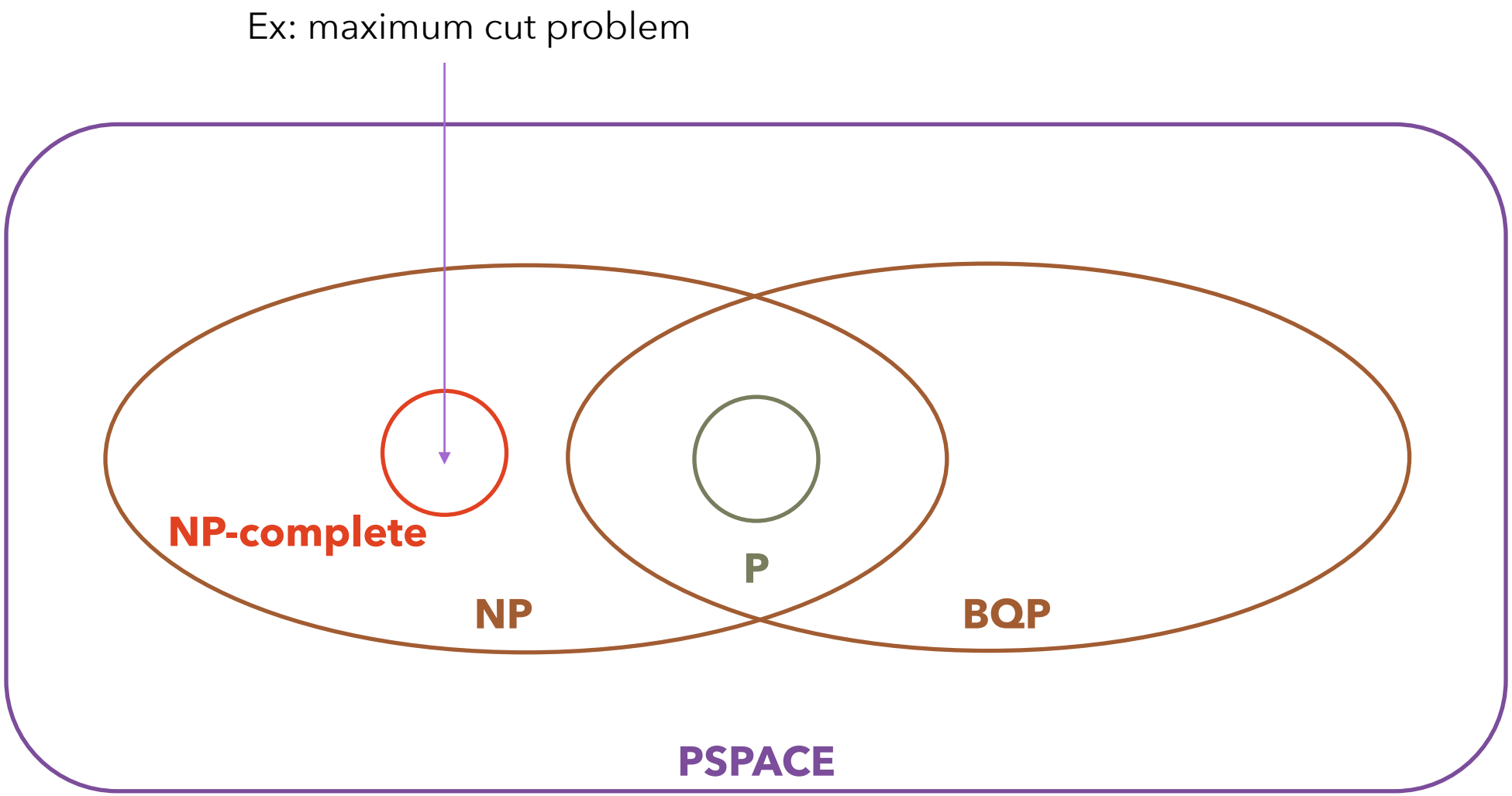
- When reasoning about complexity of solving problems (including optimization problems!), **we are usually interested in how the time / memory requirements grow with problem size** (asymptotic complexity)
- Famous classes of problems:
  - P – solvable in polynomial time
  - NP – can verify *proof* of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


P
NP
BQP
PSPACE

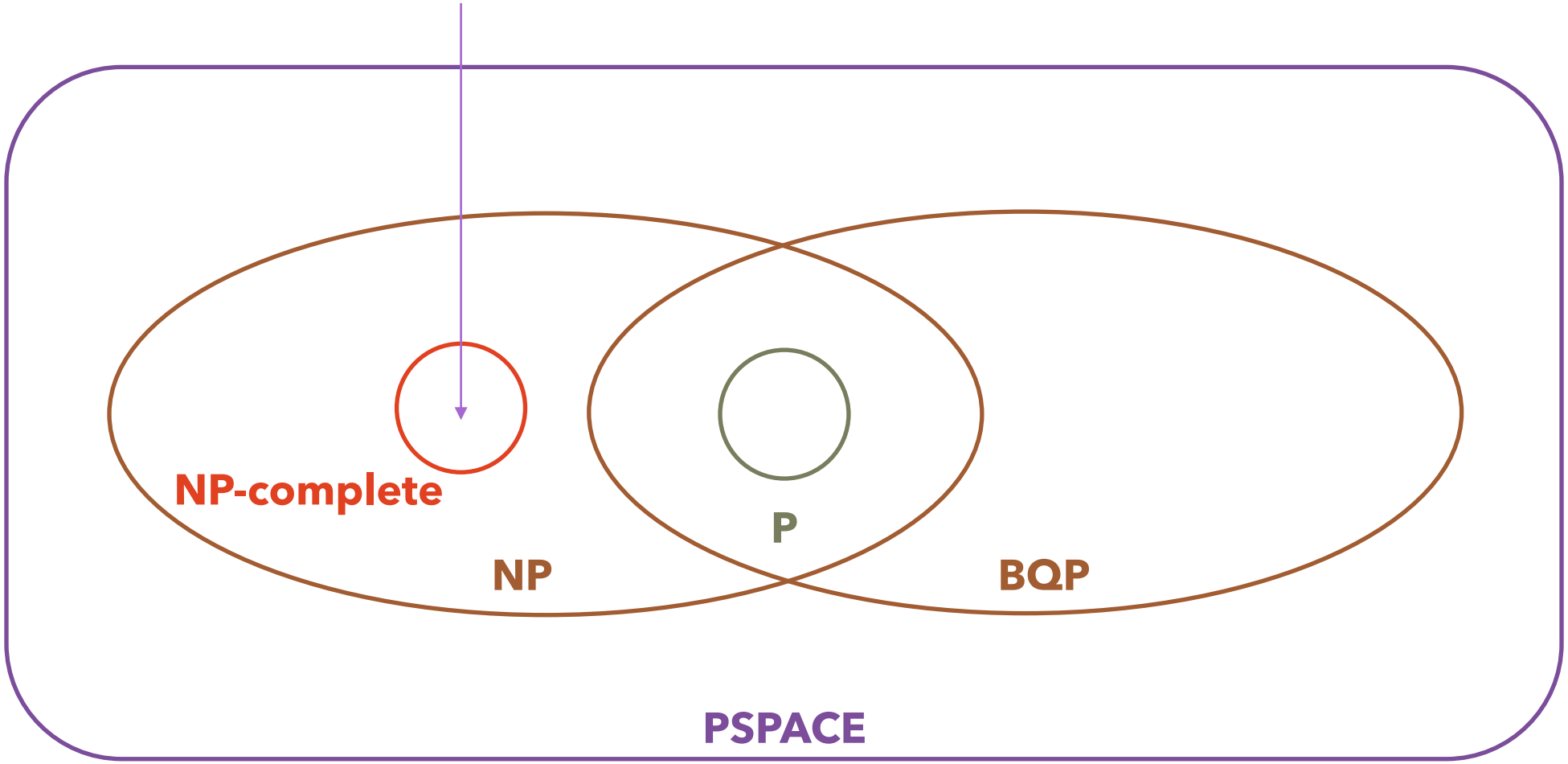
Ex: Find shortest path between two nodes in a graph



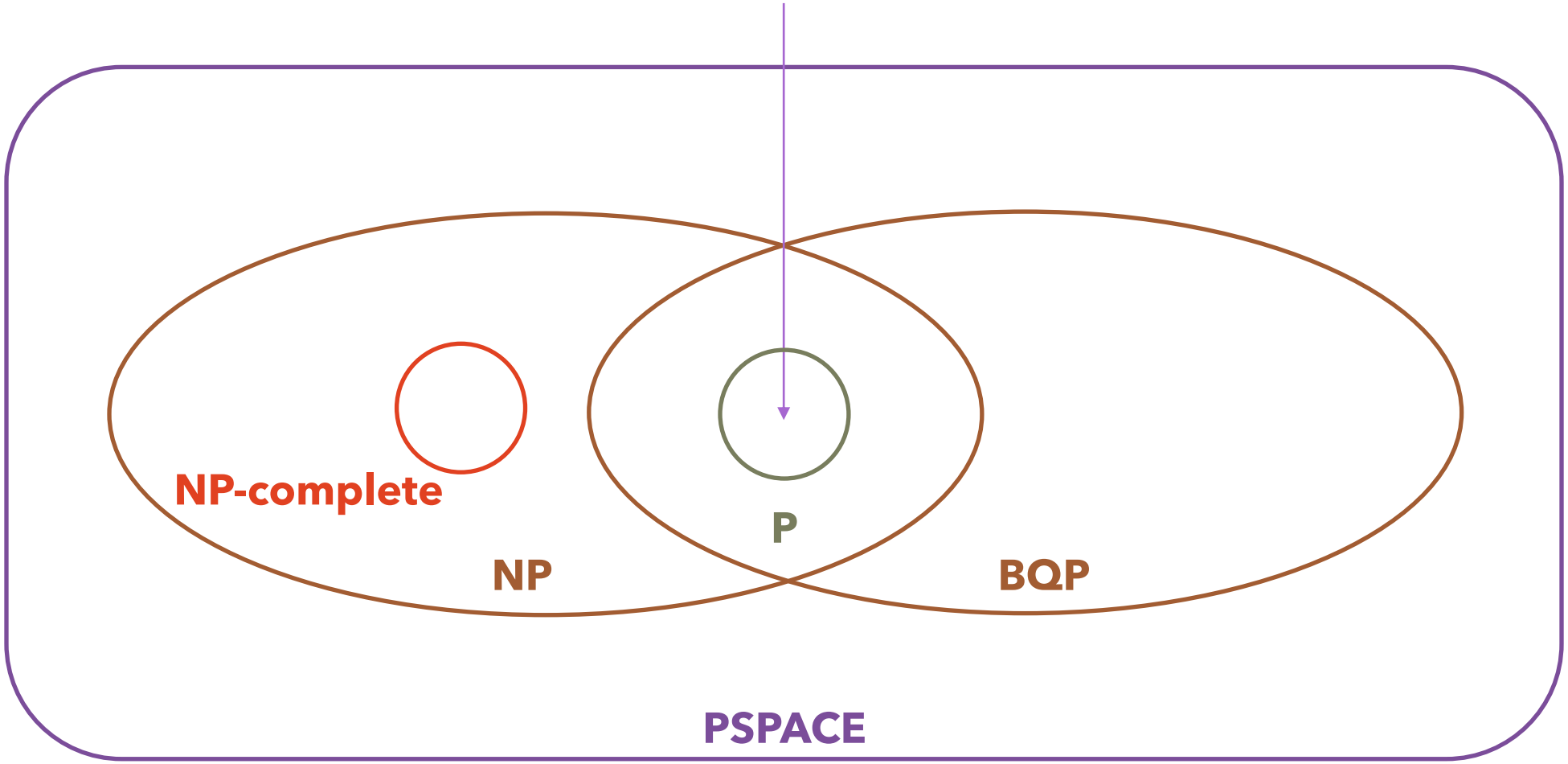

Ex: integer factorization (?)
P
NP
BQP
PSPACE


Ex: maximum cut problem
NP-complete
P
NP
BQP
PSPACE

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



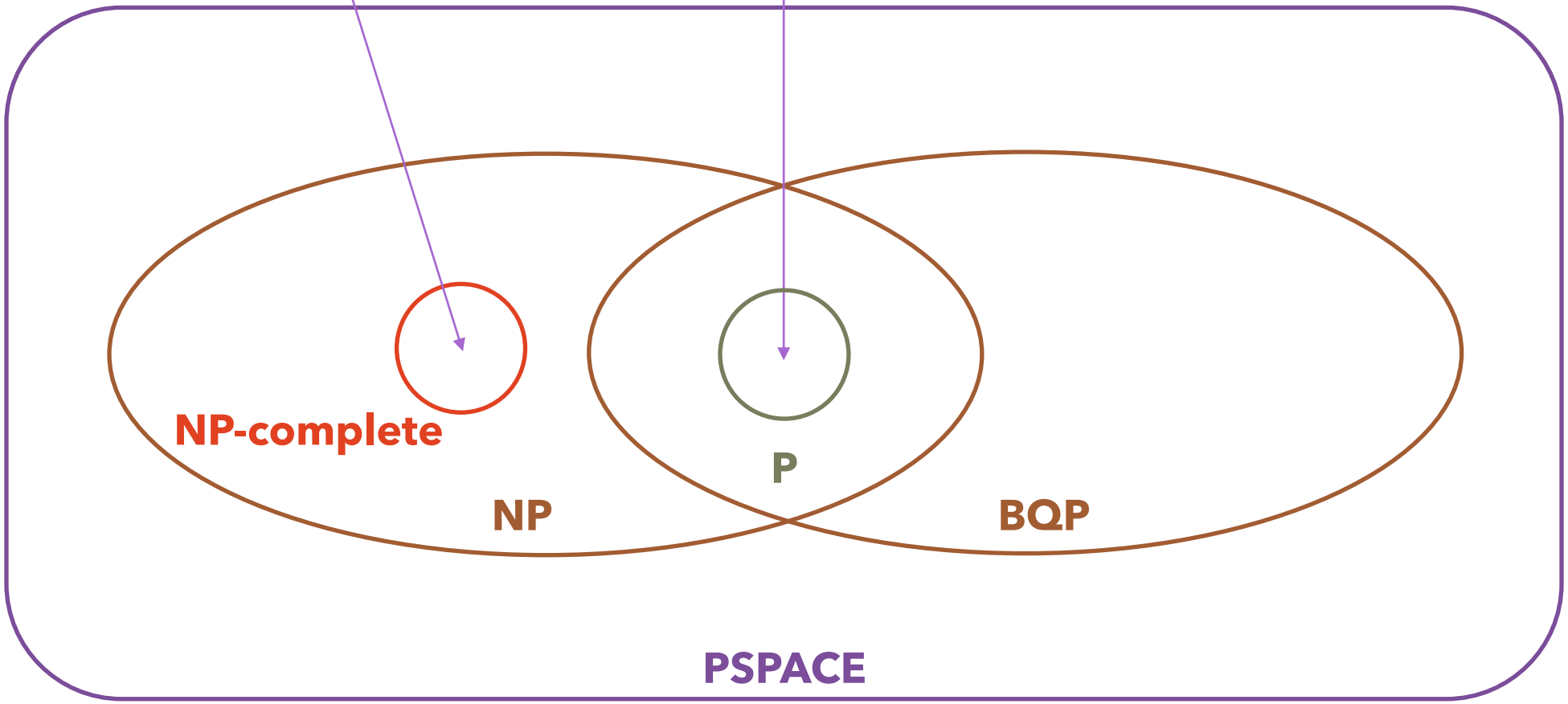
Ex: maximum cut problem **on cycle graph**



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

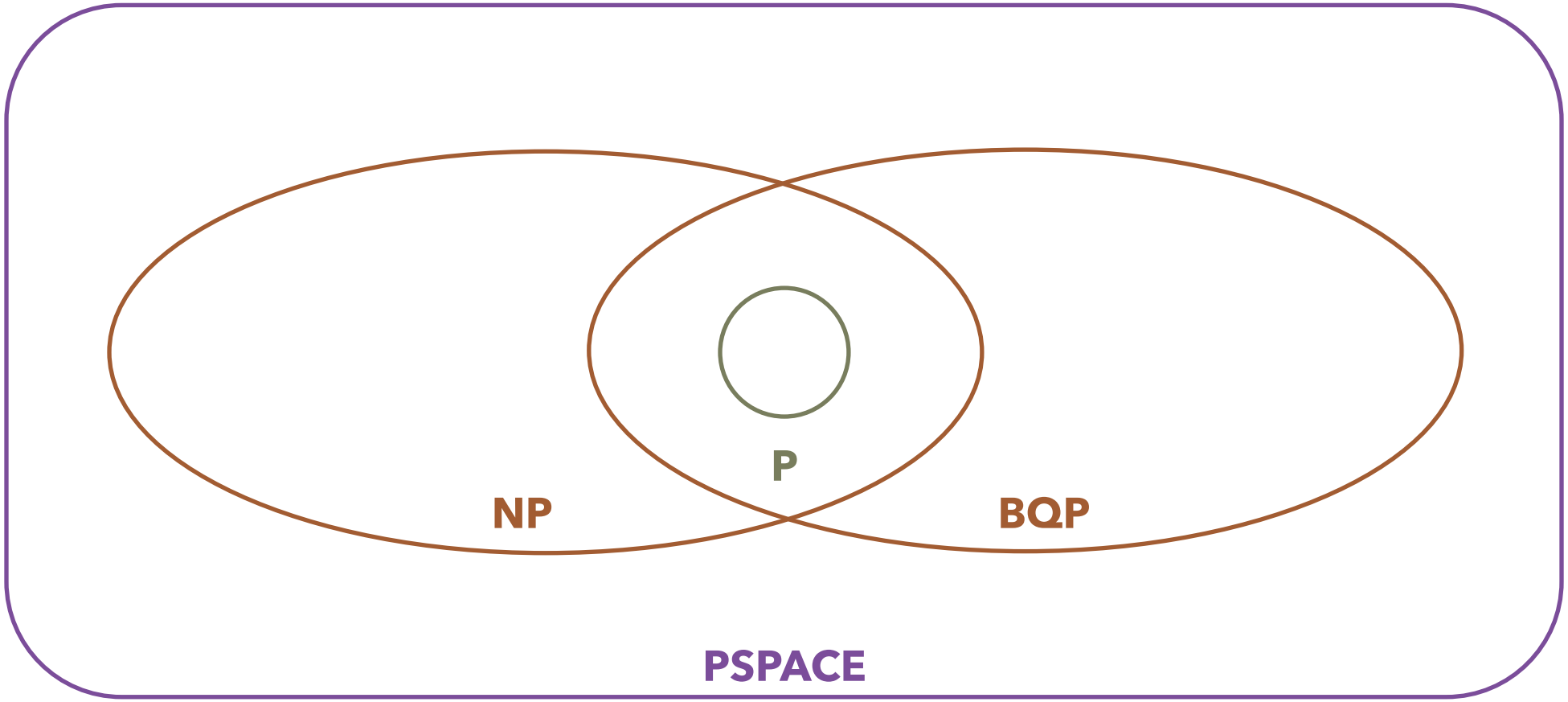
Ex: maximum cut problem **in worst case**

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 annealin
- Genetic algorithm

Quantum

- Quantum annealin
- 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

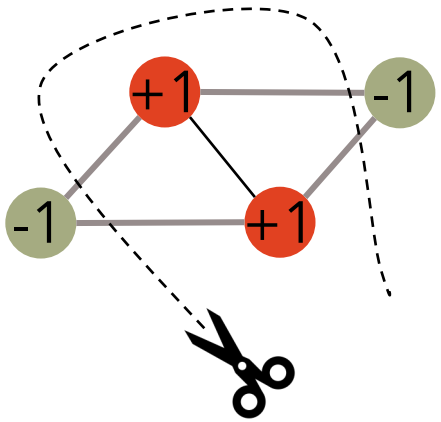
Outline

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:

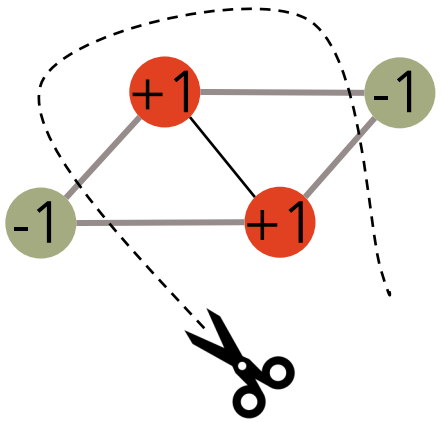


- Maximum cut can be formulated as an optimization problem:

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

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:



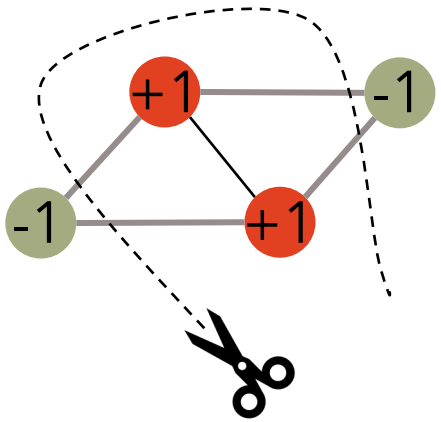
- Maximum cut can be formulated as an optimization problem:

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

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:

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

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

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

One way of solving an optimization problem on a quantum computer is by converting it into a problem of characterizing a Hamiltonian

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
Solution	$\mathbf{s} \in \{-1, +1\}^n$	Highest energy eigenstate (largest eigenvalue eigenvector) $\lvert s \rangle$

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
Solution	$\mathbf{s} \in \{-1, +1\}^n$	Highest energy eigenstate (largest eigenvalue eigenvector) $\lvert s \rangle$

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

$$\max_{x \in \{0,1\}^n} f(x) \quad \Rightarrow \quad C = \begin{pmatrix} f(0\ldots00) & 0 & 0 & 0 & 0 \\ 0 & f(0\ldots01) & 0 & 0 & 0 \\ \vdots & & \ddots & & \\ 0 & 0 & 0 & f(1\ldots10) & 0 \\ 0 & 0 & 0 & 0 & f(1\ldots11) \end{pmatrix}$$

What does this Hamiltonian look like?

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

$$C\,|0\ldots00\rangle = \begin{pmatrix} f(0\ldots00) & 0 & 0 & 0 & 0 \\ 0 & f(0\ldots01) & 0 & 0 & 0 \\ \vdots & & \ddots & & \\ 0 & 0 & 0 & f(1\ldots10) & 0 \\ 0 & 0 & 0 & 0 & f(1\ldots11) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} = f(0\ldots00)\,|0\ldots00\rangle$$

What does this Hamiltonian look like?

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

$$C\,|x\rangle = \begin{pmatrix} f(0\ldots00) & 0 & 0 & 0 & 0 \\ 0 & f(0\ldots01) & 0 & 0 & 0 \\ \vdots & & \ddots & & \\ 0 & 0 & 0 & f(1\ldots10) & 0 \\ 0 & 0 & 0 & 0 & f(1\ldots11) \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} = f(x)\,|x\rangle \qquad \forall x \in \{0,1\}^n$$

What does this Hamiltonian look like?

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

$$C\,|x\rangle = \begin{pmatrix} f(0\ldots00) & 0 & 0 & 0 & 0 \\ 0 & f(0\ldots01) & 0 & 0 & 0 \\ \vdots & & \ddots & & \\ 0 & 0 & 0 & f(1\ldots10) & 0 \\ 0 & 0 & 0 & 0 & f(1\ldots11) \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} = f(x)\,|x\rangle \qquad \forall x \in \{0,1\}^n$$

- 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 c		state s
Solution	$\mathbf{s} \in \{-1, +1\}^n$	Highest energy eigenstate $\lvert s \rangle$

How do we construct this Hamiltonian?

Notation Reminder

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

$$\langle y| = |y\rangle^\dagger = \begin{bmatrix} y_1^* & y_2^* & \cdots & y_n^* \end{bmatrix}$$ conjugate transpose

Notation Reminder

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

$$\langle y| = |y\rangle^\dagger = \begin{bmatrix} y_1^* & y_2^* & \cdots & y_n^* \end{bmatrix} \qquad \text{conjugate transpose}$$

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

Notation Reminder

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

$$\langle y| = |y\rangle^\dagger = \begin{bmatrix} y_1^* & y_2^* & \cdots & y_n^* \end{bmatrix} \qquad \text{conjugate transpose}$$

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

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

Constructing MAXCUT Hamiltonian

- MAXCUT objective:

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

Constructing MAXCUT Hamiltonian

- MAXCUT objective:

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

- 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:

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

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

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

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

- Therefore,

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

Pauli Z operator

- Acting on the i-th qubit:

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

Pauli Z operator

- Acting on the i-th qubit:

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

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

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

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

Constructing MAXCUT Hamiltonian

- MAXCUT objective:

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

- 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:

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

- Let's reformulate it in the following way:

$$x_i = \frac{1}{2}(1 - s_i)$$

$$s_i = 1 \rightarrow x_i = 0, \qquad (-1)^{x_i} = 1 = s_i$$

$$s_i = -1 \rightarrow x_i = 1, \qquad (-1)^{x_i} = -1 = s_i$$

Verifying MAXCUT Hamiltonian

- MAXCUT objective:

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

- Let's reformulate it in the following way:

$$x_i = \frac{1}{2}(1 - s_i)$$

$$s_i = 1 \rightarrow x_i = 0, \qquad (-1)^{x_i} = 1 = s_i$$

$$s_i = -1 \rightarrow x_i = 1, \qquad (-1)^{x_i} = -1 = s_i$$

- New objective:

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

Verifying MAXCUT Hamiltonian

- MAXCUT objective:

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

- MAXCUT Hamiltonian:

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

- Want to show:

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

Verifying MAXCUT Hamiltonian

$$
\begin{aligned}
C\,|x\rangle &= \frac{1}{2}\sum_{ij\in E}(I - Z_iZ_j)\,|x_0\ldots x_n\rangle \\
&= \frac{1}{2}\sum_{ij\in E}(I - Z_iZ_j)\,|x_0\ldots x_n\rangle \\
&= \frac{1}{2}\sum_{ij\in E}(|x_0\ldots x_n\rangle - Z_iZ_j\,|x_0\ldots x_n\rangle) \\
&= \frac{1}{2}\sum_{ij\in E}(1 - (-1)^{x_i}(-1)^{x_j}))\,|x_0\ldots x_n\rangle = C(\mathbf{x})\,|x\rangle
\end{aligned}
$$

$$x_i \in \{0,1\}, \quad i = 1,\ldots n$$

Constructing MAXCUT Hamiltonian

- MAXCUT objective:

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

- 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: $Z_i Z_j = I \otimes \ldots \otimes Z_i \otimes Z_j \otimes \ldots \otimes I$

- Therefore each term in the sum is a 2^nx2^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:

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

- 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, $f = max_2$

$$\begin{aligned}\max_2(+1,+1) &= +1\\ \max_2(-1,+1) &= +1\\ \max_2(+1,-1) &= +1\\ \max_2(-1,-1) &= -1\end{aligned}$$

- 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:

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

O'Donnell, Ryan. Analysis of boolean functions. Cambridge University Press, 2014.

Fourier analysis of Boolean functions

- Consider Boolean "maximum" function on two bits, $f = max_2$

$$\begin{aligned}\max_2(+1,+1) &= +1\\ \max_2(-1,+1) &= +1\\ \max_2(+1,-1) &= +1\\ \max_2(-1,-1) &= -1\end{aligned}$$

- 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 $S \subseteq [n]$

O'Donnell, Ryan. Analysis of boolean functions. Cambridge University Press, 2014.

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:

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

Fourier analysis of Boolean functions

- Consider Boolean "maximum" function on two bits, $f = max_2$

$$\begin{aligned}\max_2(+1,+1) &= +1\\ \max_2(-1,+1) &= +1\\ \max_2(+1,-1) &= +1\\ \max_2(-1,-1) &= -1\end{aligned}$$

- 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

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

O'Donnell, Ryan. Analysis of Boolean functions. Cambridge University Press, 2014.

Fourier analysis of Boolean functions

- Consider Boolean "maximum" function on two bits, $f = max_2$

$$\begin{aligned}\max_2(+1,+1) &= +1\\ \max_2(-1,+1) &= +1\\ \max_2(+1,-1) &= +1\\ \max_2(-1,-1) &= -1\end{aligned}$$

- 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:**

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

O'Donnell, Ryan. Analysis of boolean functions. Cambridge University Press, 2014.

Fourier analysis of Boolean functions

- More generally,

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

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

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

O'Donnell, Ryan. Analysis of boolean functions. Cambridge University Press, 2014.

Constructing a Hamiltonian for a general problem

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

$$C|x\rangle = f(x)|x\rangle = \sum_{S \subseteq [n]} \widehat{f}(S) x^S |x\rangle \quad \forall x \in \{0,1\}^n$$

Constructing a Hamiltonian for a general problem

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

$$C|x\rangle = f(x)|x\rangle = \sum_{S \subseteq [n]} \widehat{f}(S) x^S |x\rangle \quad \forall x \in \{0,1\}^n$$

- 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

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

$$C|x\rangle = f(x)|x\rangle = \sum_{S \subseteq [n]} \widehat{f}(S) x^S |x\rangle \quad \forall x \in \{0,1\}^n$$

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

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

- The unique n-qubit Hamiltonian representing *f*

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

Constructing a Hamiltonian for a general problem: a recipe

1. Perform Fourier expansion of the desired function:

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

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

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

- 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	$\frac{1}{2}I - \frac{1}{2}Z$	$\bar{x}$	$\frac{1}{2}I + \frac{1}{2}Z$
$x_1 \oplus x_2$	$\frac{1}{2}I - \frac{1}{2}Z_1Z_2$	$\bigoplus_{j=1}^{k} x_j$	$\frac{1}{2}I - \frac{1}{2}Z_1Z_2\ldots Z_k$
$x_1 \wedge x_2$	$\frac{1}{4}I - \frac{1}{4}(Z_1 + Z_2 - Z_1Z_2)$	$\Lambda_{j=1}^{k} x_j$	$\frac{1}{2^k}\prod_j (I - Z_j)$
$x_1 \vee x_2$	$\frac{3}{4}I - \frac{1}{4}(Z_1 + Z_2 + Z_1Z_2)$	$V_{j=1}^{k} x_j$	$I - \frac{1}{2^k}\prod_j (I + Z_j)$
$\overline{x_1x_2}$	$\frac{3}{4}I + \frac{1}{4}(Z_1 + Z_2 - Z_1Z_2)$	$x_1 \Rightarrow x_2$	$\frac{3}{4}I + \frac{1}{4}(Z_1 - Z_2 + Z_1Z_2)$

- Combine them as follows:

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

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

PART 2:
QUANTUM APPROXIMATE OPTIMIZATION ALGORITHM (QAOA)

Quantum Approximate Optimization Algorithm (QAOA)

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

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

- 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:

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

- Then a classical optimizer is used to vary the parameters $\boldsymbol{\beta}, \boldsymbol{\gamma}$ to maximize the expected objective value of the quantum evolution measurement outcomes:

$$
\langle C \rangle := \langle \vec{\beta}, \vec{\gamma} | \, C \, | \vec{\beta}, \vec{\gamma} \rangle = \sum_{x \in \{0,1\}^n} \mathsf{Pr}(x) f(x).
$$

Quantum Approximate Optimization Algorithm (QAOA)

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

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

- Note that for p → ∞ 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:

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

How do we implement this circuit in gates?

Implementing QAOA

- Let's assume the following gate set

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \quad Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}; \quad Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}; \quad H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix};$$

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

Implementing QAOA

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

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

Implementing QAOA

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

- Circuit:

$$-\boxed{Rz(2t)}-$$

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

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:

$$
\begin{aligned}
e^{-iZ\otimes Zt}|00\rangle &= e^{-i(1\times 1)t}|00\rangle = e^{-it}|00\rangle \\
e^{-iZ\otimes Zt}|01\rangle &= e^{-i(1\times -1)t}|01\rangle = e^{it}|01\rangle \\
e^{-iZ\otimes Zt}|10\rangle &= e^{-i(-1\times 1)t}|10\rangle = e^{it}|10\rangle \\
e^{-iZ\otimes Zt}|11\rangle &= e^{-i(-1\times -1)t}|11\rangle = e^{-it}|11\rangle
\end{aligned}
$$

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

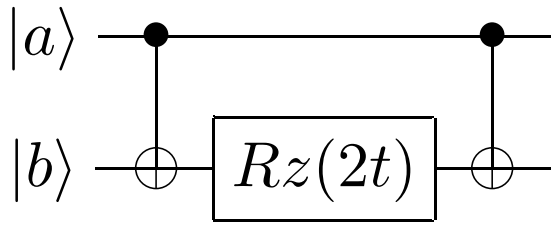
$$
e^{-iZ\otimes Zt}|ab\rangle = e^{-i(-1)^{a\oplus b}t}|ab\rangle
$$

Implementing QAOA

- Slightly more complicated operator: $e^{-iZZt} = e^{-iZ\otimes Zt}$
- 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:

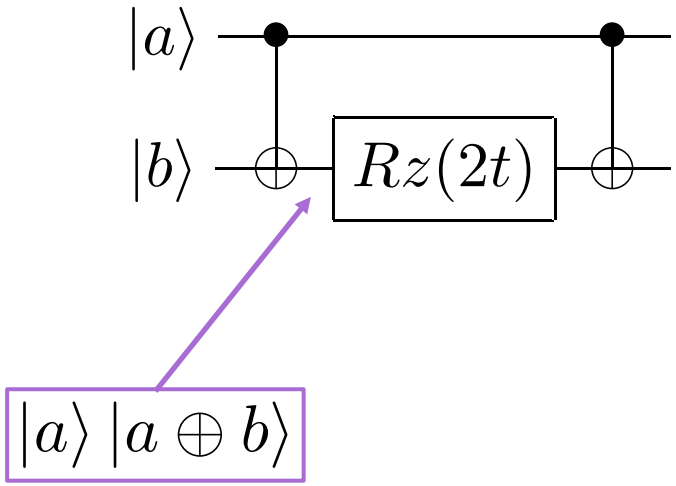


Implementing QAOA

- Slightly more complicated operator: $e^{-iZZt} = e^{-iZ\otimes Zt}$
- 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:

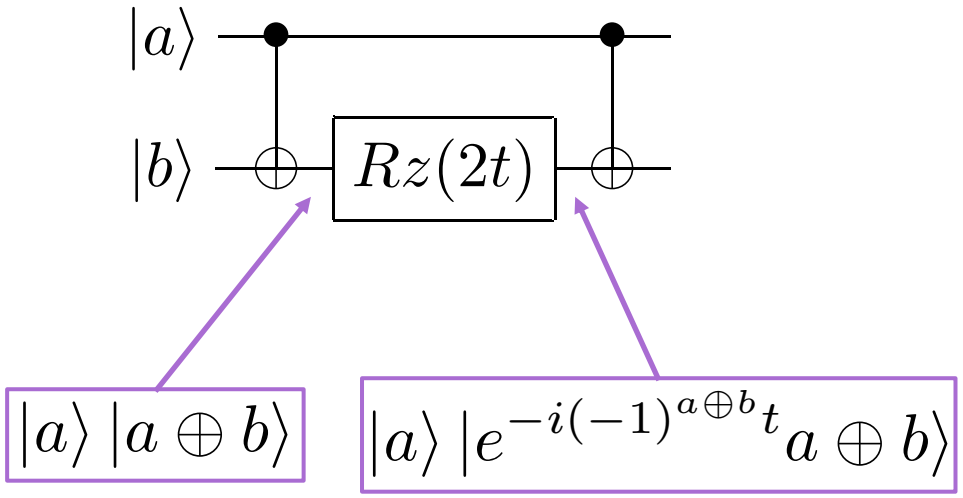


Implementing QAOA

- Slightly more complicated operator: $e^{-iZZt} = e^{-iZ\otimes Zt}$
- 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:

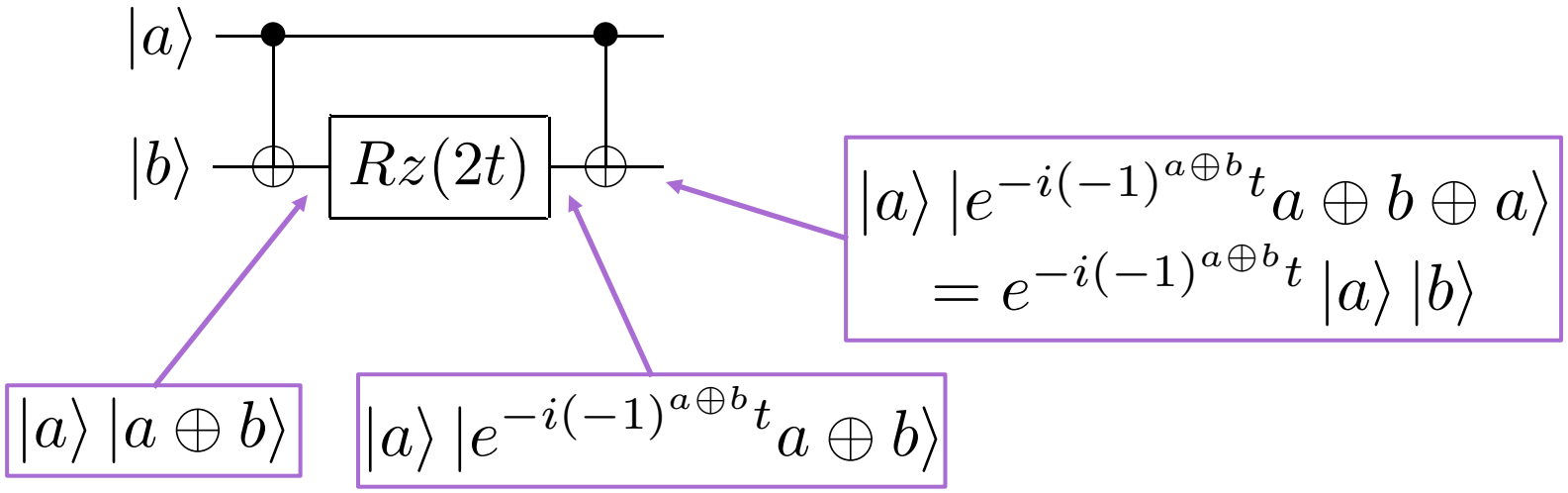


Implementing QAOA

- Slightly more complicated operator: $e^{-iZZt} = e^{-iZ\otimes Zt}$
- 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:

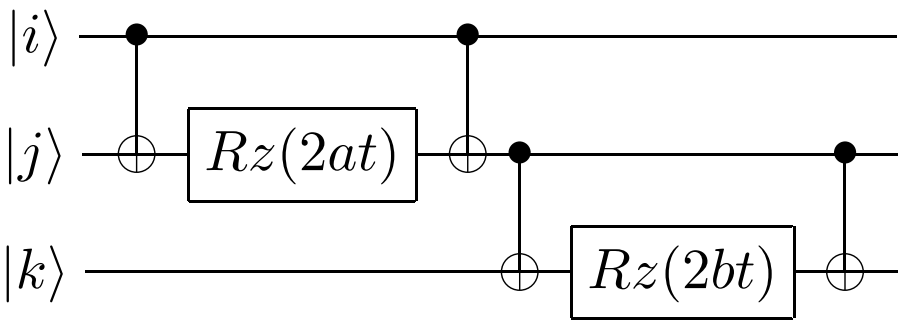


Implementing QAOA

- 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:

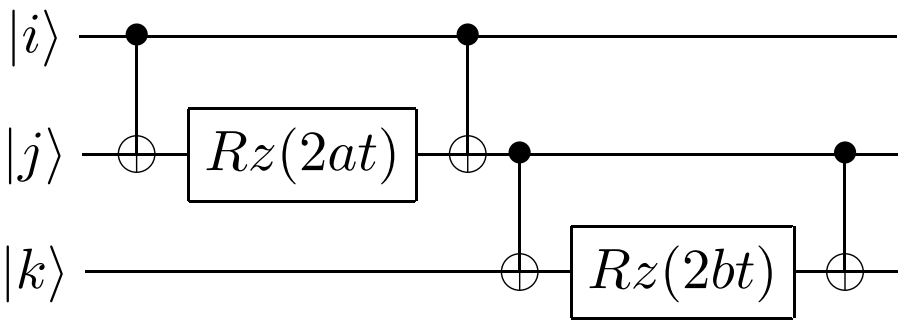


Implementing QAOA

- 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:

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

Implementing QAOA

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

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

- 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

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

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

Implementing QAOA

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

- Note that

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

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

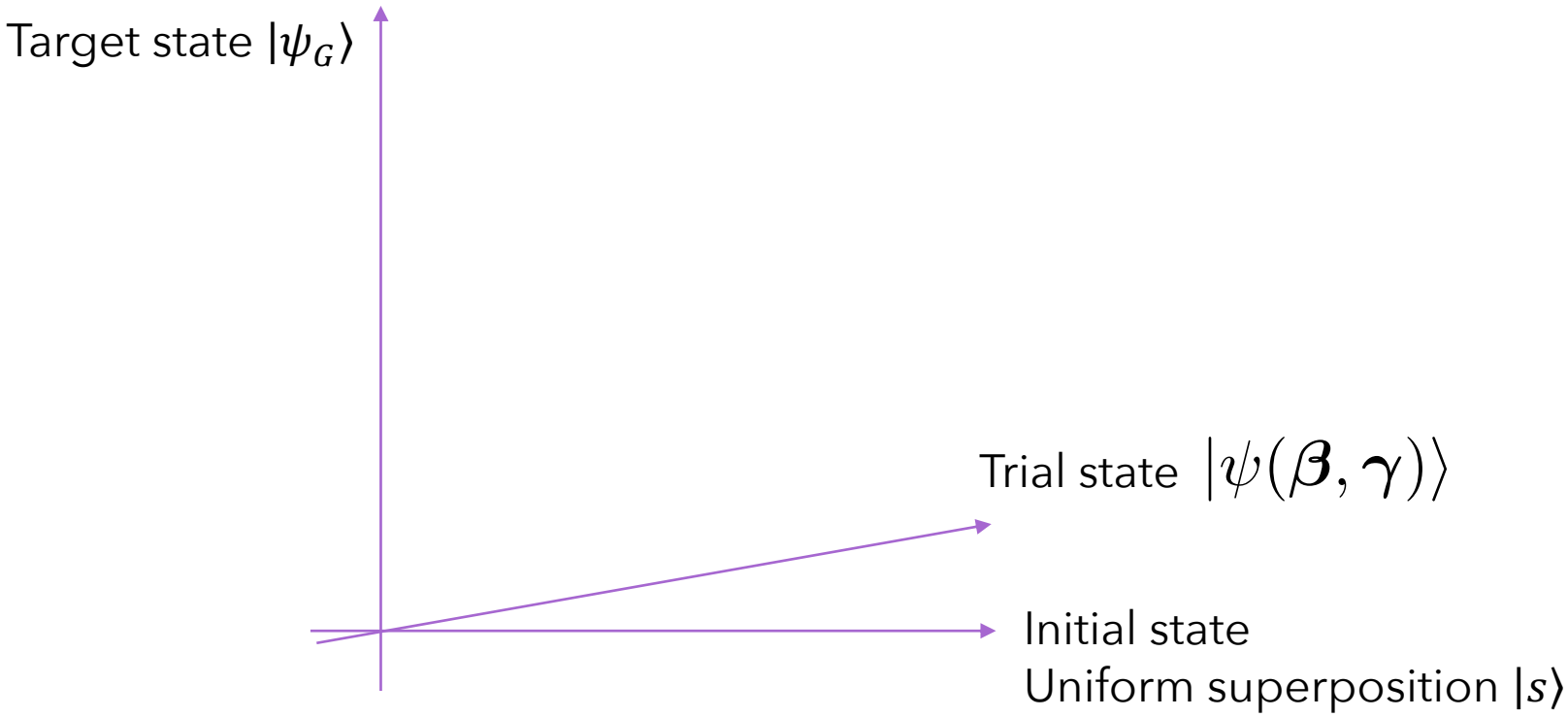
- Circuit: $-\boxed{H}-\boxed{Rz(2t)}-\boxed{H}-$

Quantum Approximate Optimization Algorithm (QAOA): Geometric Interpretation

- QAOA prepares a parameterized "trial" (ansatz) state $|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle$

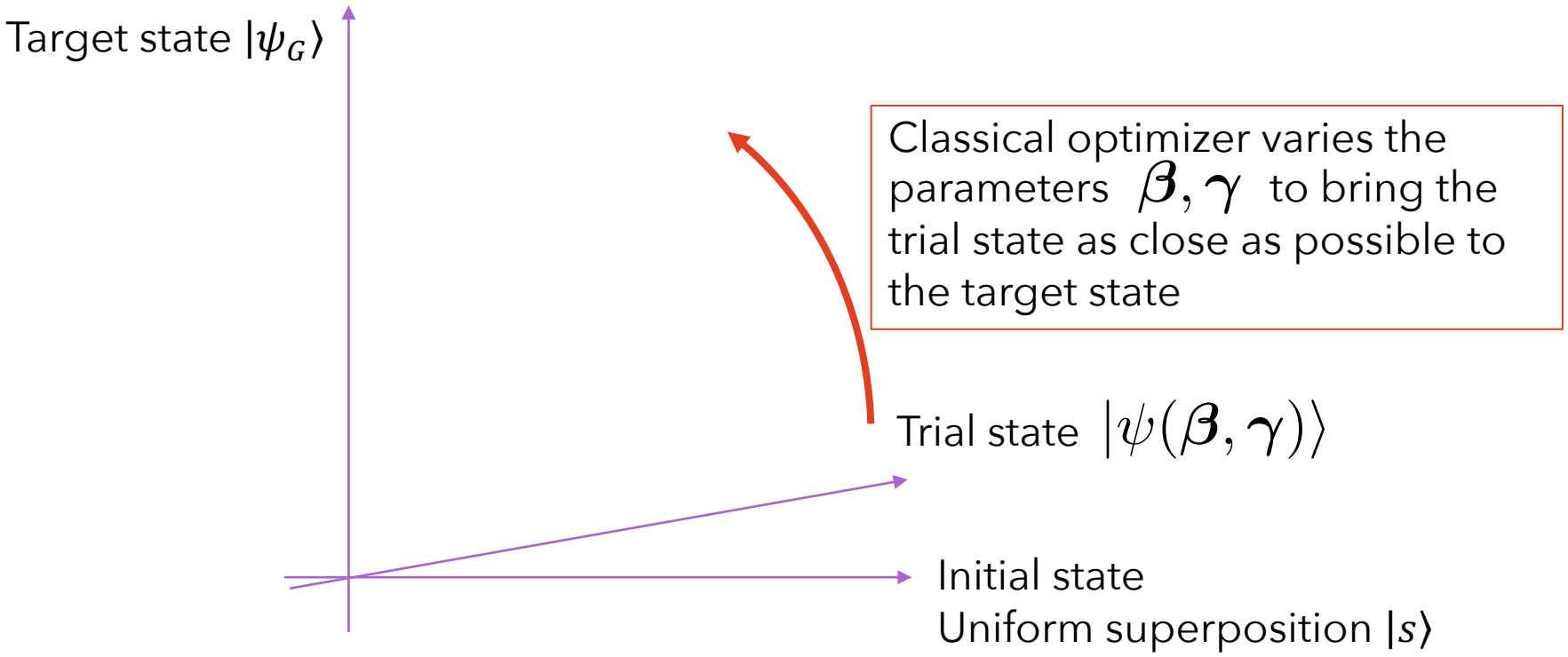
Quantum Approximate Optimization Algorithm (QAOA): Geometric Interpretation

- QAOA prepares a parameterized "trial" (ansatz) state $|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle$



Quantum Approximate Optimization Algorithm (QAOA): Geometric Interpretation

- QAOA prepares a parameterized "trial" (ansatz) state $|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle$



Quantum Approximate Optimization Algorithm (QAOA)

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

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

- 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

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

- 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?

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

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

Simulation of AQC

- Consider a general quantum evolution described by the Schrödinger equation:

$$i\frac{d}{dt}|\psi(t)\rangle = H|\psi(t)\rangle$$

Simulation of AQC

- Consider a general quantum evolution described by the Schrödinger equation:

$$i\frac{d}{dt}|\psi(t)\rangle = H|\psi(t)\rangle$$

- The solution to this equation is

$$|\psi(t)\rangle = U(t)|\psi(0)\rangle = e^{-i\int_0^t H(t)dt}|\psi(0)\rangle$$

Simulation of AQC

- Consider a general quantum evolution described by the Schrödinger equation:

$$i\frac{d}{dt}|\psi(t)\rangle = H|\psi(t)\rangle$$

- The solution to this equation is

$$|\psi(t)\rangle = U(t)|\psi(0)\rangle = e^{-i\int_0^t H(t)dt}|\psi(0)\rangle$$

- We can break it up as follows:

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

Simulation of AQC

- Consider a general quantum evolution described by the Schrödinger equation:

$$i\frac{d}{dt}|\psi(t)\rangle = H|\psi(t)\rangle$$

- The solution to this equation is

$$|\psi(t)\rangle = U(t)|\psi(0)\rangle = e^{-i\int_0^t H(t)dt}|\psi(0)\rangle$$

- 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:

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

Simulation of AQC

- Consider a general quantum evolution described by the Schrödinger equation:

$$i\frac{d}{dt}|\psi(t)\rangle = H|\psi(t)\rangle$$

- The solution to this equation is

$$|\psi(t)\rangle = U(t)|\psi(0)\rangle = e^{-i\int_0^t H(t)dt}|\psi(0)\rangle$$

- We can break it up as follows:

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

- Combining both, we get:

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

Discretizing AQC

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

Understanding QAOA

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

QAOA

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

Simulated AQC

Understanding QAOA

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

QAOA

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

Simulated AQC

- QAOA is equivalent to simulated adiabatic computation if we set

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

Understanding QAOA

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

QAOA

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

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:

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

- 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

- Recall that we can write

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

Evaluating shallow-depth QAOA energy classically

- Recall that we can write

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

- From linearity:

$$\begin{aligned} E_p(\vec{\beta}, \vec{\gamma}) &= \langle \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) | \, C \, | \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) \rangle \\ &= \langle \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) | \sum_{S \subset [n]} \hat{f}(S) \prod_{j \in S} \mathrm{Z}_j \, | \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) \rangle \\ &= \sum_{S \subset [n]} \hat{f}(S) \, \langle \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) | \prod_{j \in S} \mathrm{Z}_j \, | \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) \rangle \\ &= \sum_{S \subset [n]} \hat{f}(S) E_p(\vec{\beta}, \vec{\gamma}, S). \end{aligned}$$

Evaluating shallow-depth QAOA energy classically

$$\langle \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) | \, C \, | \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) \rangle = \sum_{S \subset [n]} \hat{f}(S) E_p(\vec{\beta}, \vec{\gamma}, S).$$

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

- Consider one term in the sum:

$$E_p(\vec{\beta}, \vec{\gamma}, S) = \langle \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) | \prod_{j \in S} \mathrm{Z}_j \, | \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) \rangle =$$

$$= \langle s | \, U_P^\dagger(\gamma_1) U_B^\dagger(\beta_1) \cdots U_P^\dagger(\gamma_p) \underbrace{U_B^\dagger(\beta_p) \prod_{j \in S} \mathrm{Z}_j U_B(\beta_p)}_{O(S)} U_P(\gamma_p) \cdots U_B(\beta_1) U_P(\gamma_1) \, | s \rangle \, .$$

Evaluating shallow-depth QAOA energy classically

$$\langle\psi(\boldsymbol{\beta},\boldsymbol{\gamma})|\, C\, |\psi(\boldsymbol{\beta},\boldsymbol{\gamma})\rangle = \sum_{S\subset[n]} \hat{f}(S) E_p(\vec{\beta},\vec{\gamma},S).$$

- Consider one term in the sum:

$$E_p(\vec{\beta},\vec{\gamma},S) = \langle\psi(\boldsymbol{\beta},\boldsymbol{\gamma})| \prod_{j\in S} \mathrm{Z}_j\, |\psi(\boldsymbol{\beta},\boldsymbol{\gamma})\rangle =$$

$$= \langle s|\, U_P^\dagger(\gamma_1) U_B^\dagger(\beta_1)\cdots U_P^\dagger(\gamma_p)\, \underbrace{U_B^\dagger(\beta_p) \prod_{j\in S} \mathrm{Z}_j U_B(\beta_p)}_{O(S)}\, U_P(\gamma_p)\cdots U_B(\beta_1) U_P(\gamma_1)\, |s\rangle\,.$$

- Terms acting on qubits not in S commute through:

$$O(S) = \prod_{j\in S} e^{i\beta \mathrm{X}_j} \prod_{j\in S} \mathrm{Z}_j \prod_{j\in S} e^{-i\beta \mathrm{X}_j}$$

Evaluating shallow-depth QAOA energy classically

- Analogously, since

$$
\begin{aligned}
U_P(\gamma) &= e^{-i\gamma H} \\
&= e^{-i\gamma \sum_{\tilde{S}\subset[n]} \hat{f}(\tilde{S}) \prod_{j\in\tilde{S}} \mathrm{Z}_j} \\
&= \prod_{\tilde{S}\subset[n]} e^{-i\gamma \hat{f}(\tilde{S}) \prod_{j\in\tilde{S}} \mathrm{Z}_j}.
\end{aligned}
$$

- All terms that do not "touch" S commute through, and support after 1 step of QAOA is on the set of qubits

$$S_{p=1} = S \cup \{k \in \tilde{S} : \tilde{S} \cap S \neq \emptyset\}$$

which does not grow with overall system size

Evaluating shallow-depth QAOA energy classically

- All terms that do not "touch" *S* commute through, and support after 1 step of QAOA is on the set of qubits

$$S_{p=1} = S \cup \{k \in \tilde{S} : \tilde{S} \cap S \neq \emptyset\}$$

  which does not grow with overall system size

- Support of the operator (and this complexity of evaluating energy) after *p* steps corresponds to the *reverse causal cone* of $\prod_{j \in S} \mathrm{Z}_j$

Summary

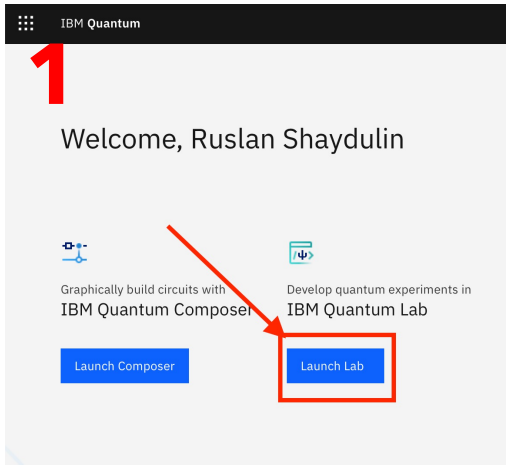
- QAOA will not solve NP-complete problems in polynomial time
- QAOA is a promising *heuristic* for the NISQ era
- For any Boolean function on n bits we can construct a unique n-qubit Hamiltonian representing it
- QAOA is deeply connected to Adiabatic Quantum Optimization Algorithm
- QAOA energy can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

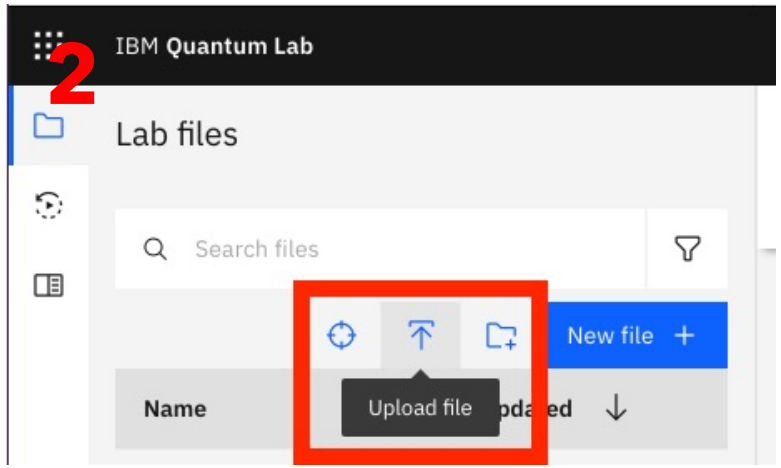
Summary

- 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/QAOA_tutorial
- Go to quantum-computing.ibm.com and login (you might need to create an IBM ID if you haven't already!)
- Launch Jupyter lab





- Upload 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