Combinatorial Optimization on Quantum Computers

Ruslan Shaydulin, Clemson University

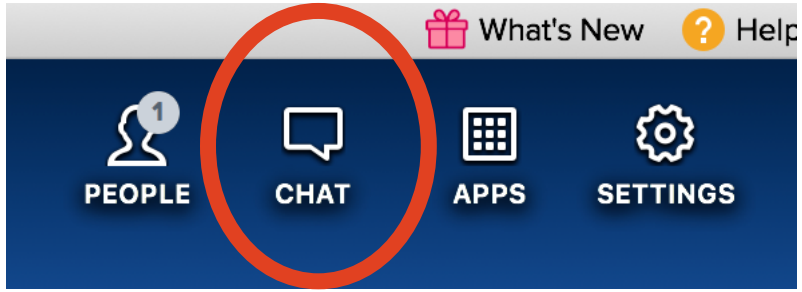
CLEMSON UNIVERSITY

Slides and notebooks

- To download ZIP archive, go to https://bit.ly/qtutorial2020
- If you prefer Git: https://github.com/rsln-s/ANL_tutorial_June_15
- If you do not have an IBM account, go to quantum-computing.ibm.com to create one

Questions for / from the audience

- Post your questions and your replies in the chat



Warmup questions: are chats fun?

1. Yes!
2. Wow, much fun! 
3. No, I cannot find the chat button 😢

PART 0:
SOME BIG-PICTURE CONSIDERATIONS

Complexity of solving problems

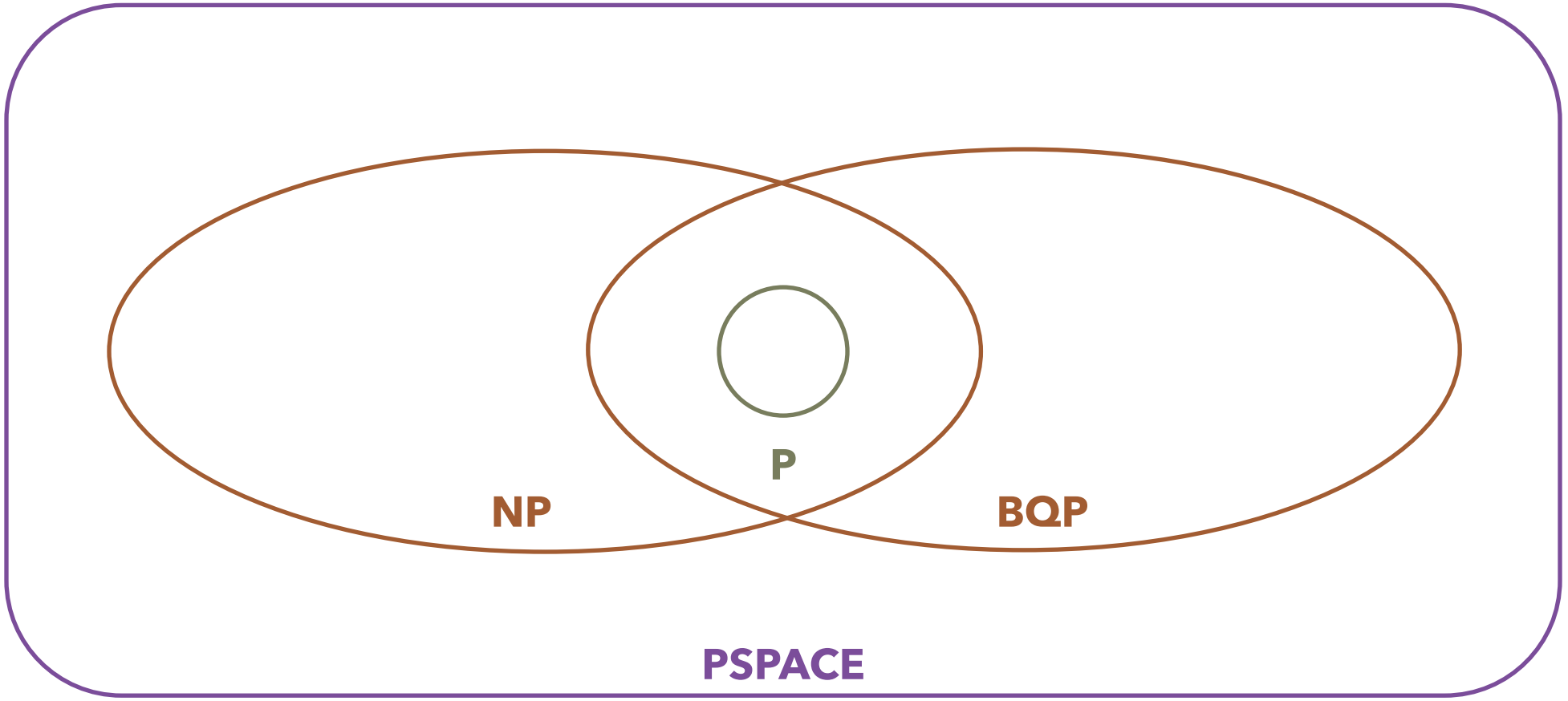
- When reasoning about complexity of solving problems (including optimization problems!), **we are usually interested in how the time / memory requirements grow with problem size** (asymptotic complexity)

Complexity of solving problems

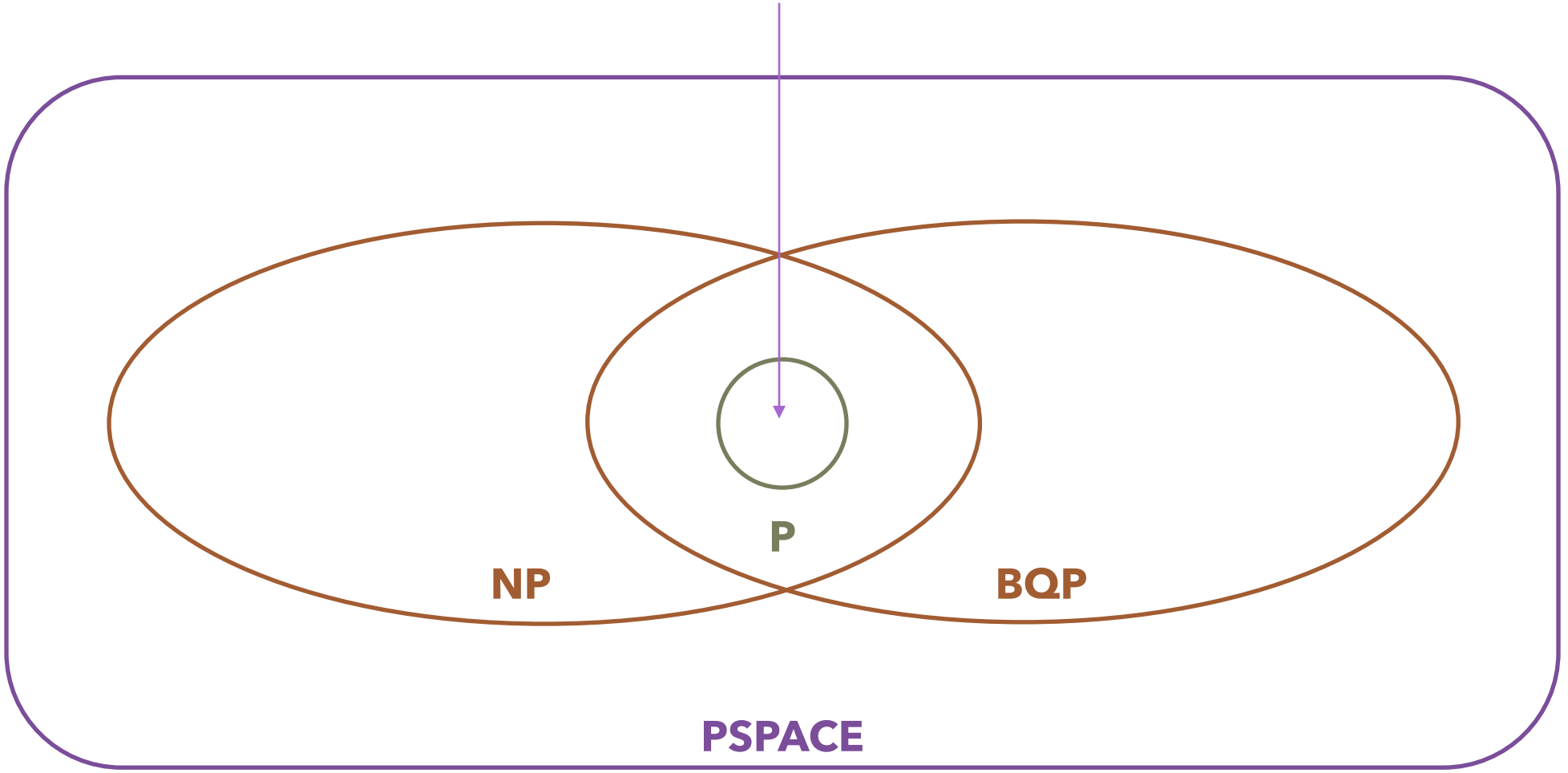
- When reasoning about complexity of solving problems (including optimization problems!), **we are usually interested in how the time / memory requirements grow with problem size** (asymptotic complexity)
- Famous classes of problems:
  - P – solvable in polynomial time
  - NP – can verify *proof* of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

Complexity of solving problems

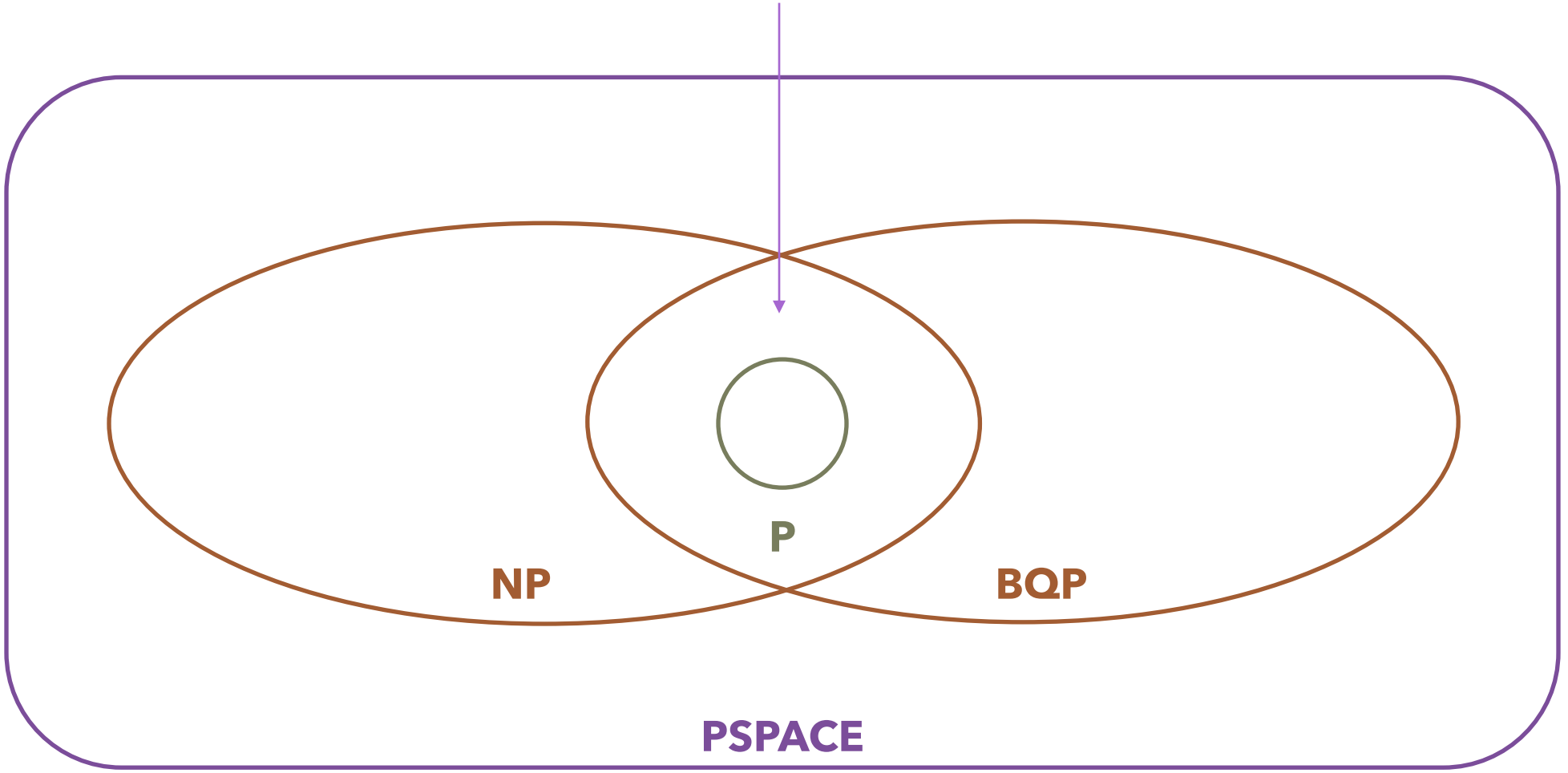
- When reasoning about complexity of solving problems (including optimization problems!), **we are usually interested in how the time / memory requirements grow with problem size** (asymptotic complexity)
- Famous classes of problems:
  - P – solvable in polynomial time
  - NP – can verify *proof* of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


NP
P
BQP
PSPACE

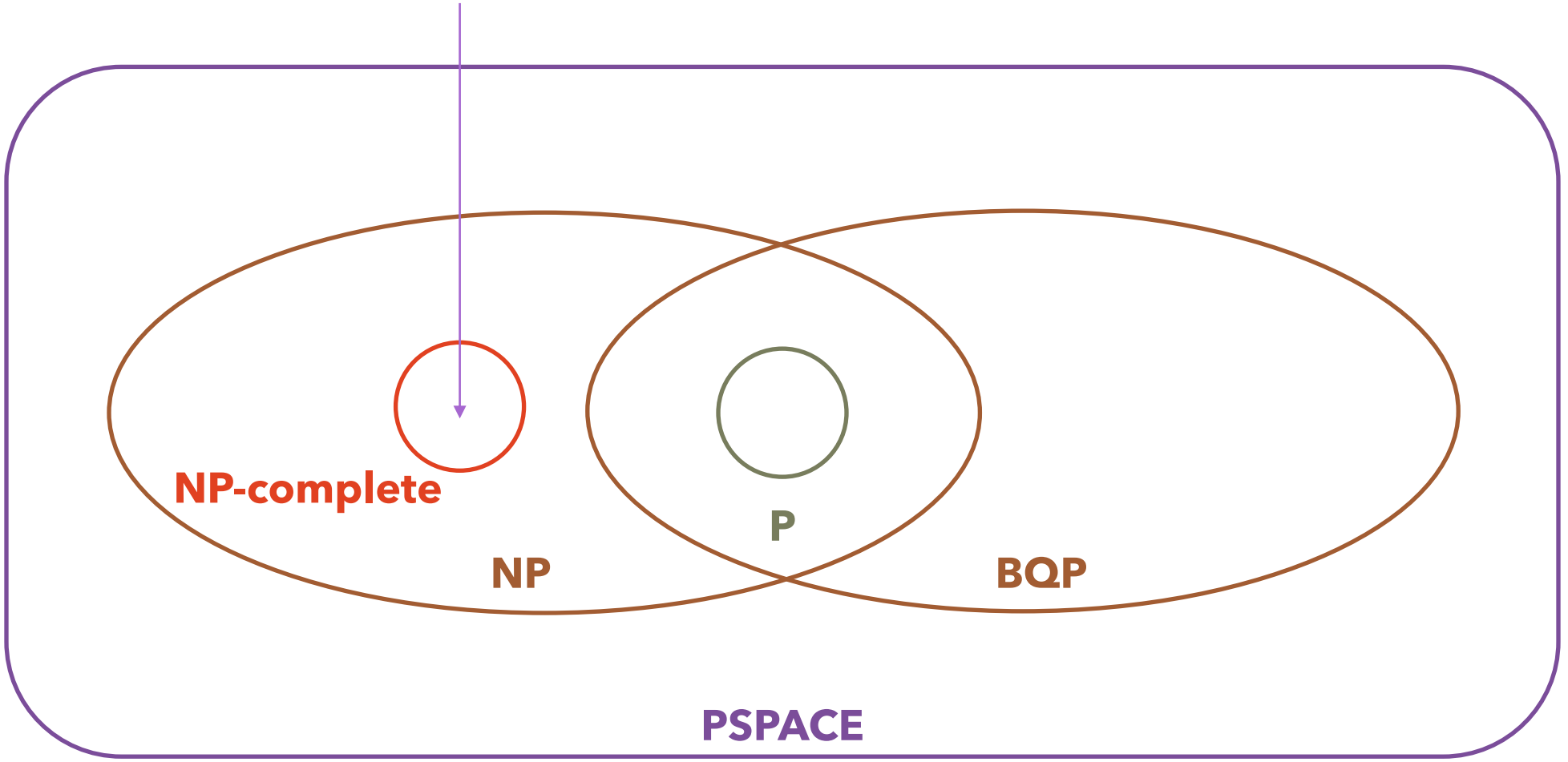
Ex: Find shortest path between two nodes in a graph



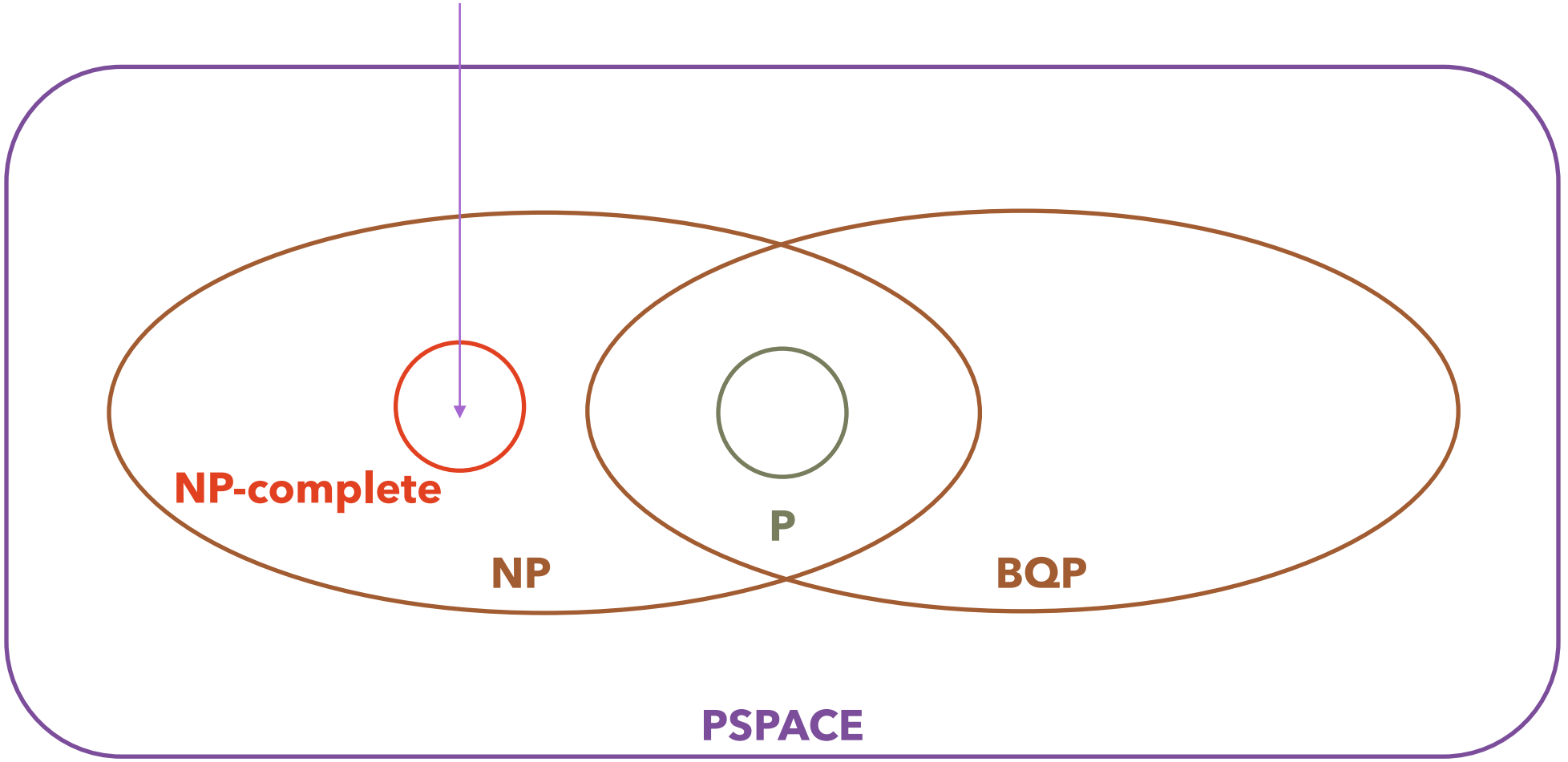
Ex: integer factorization (?)



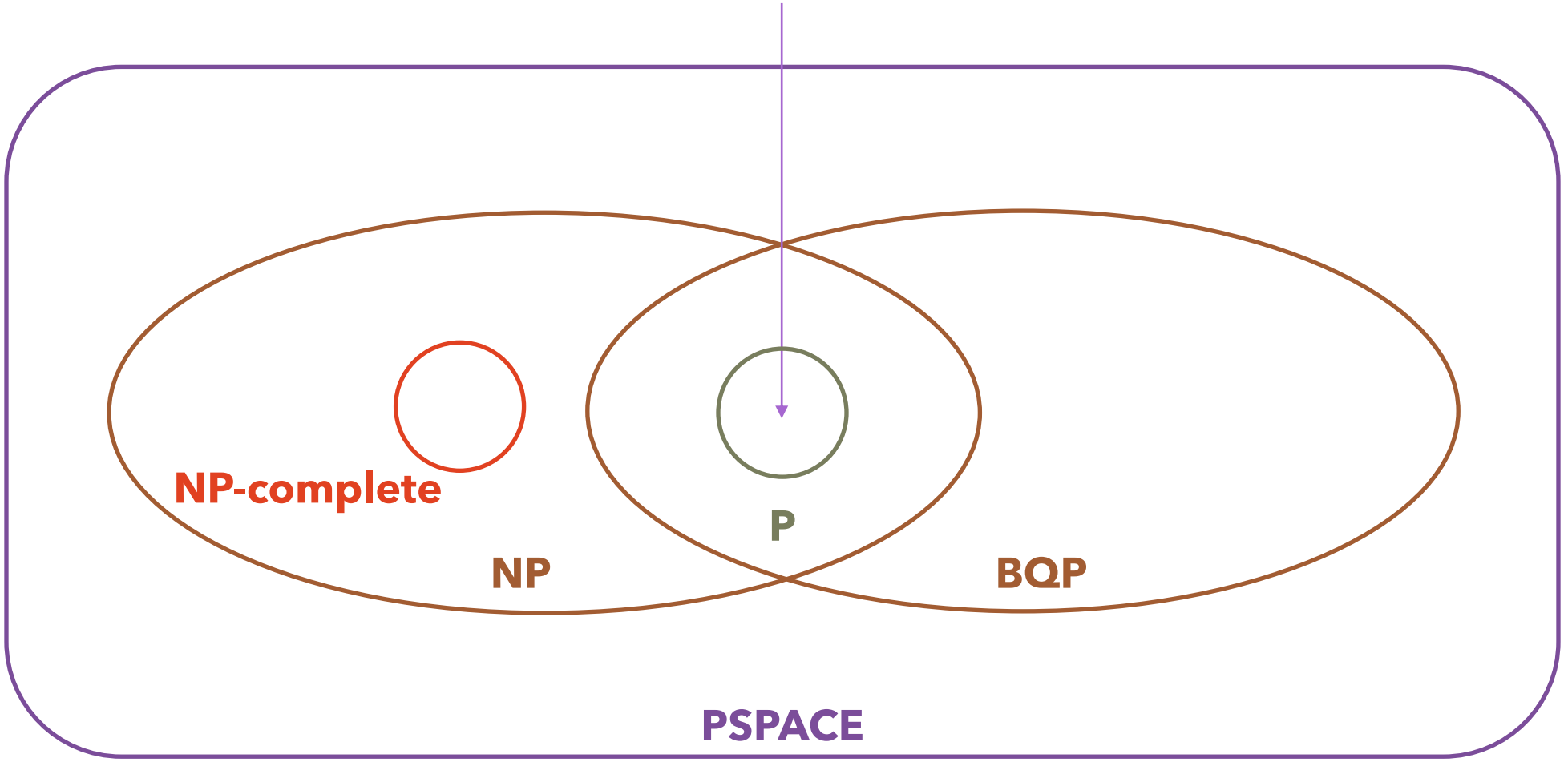
Ex: maximum cut problem



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



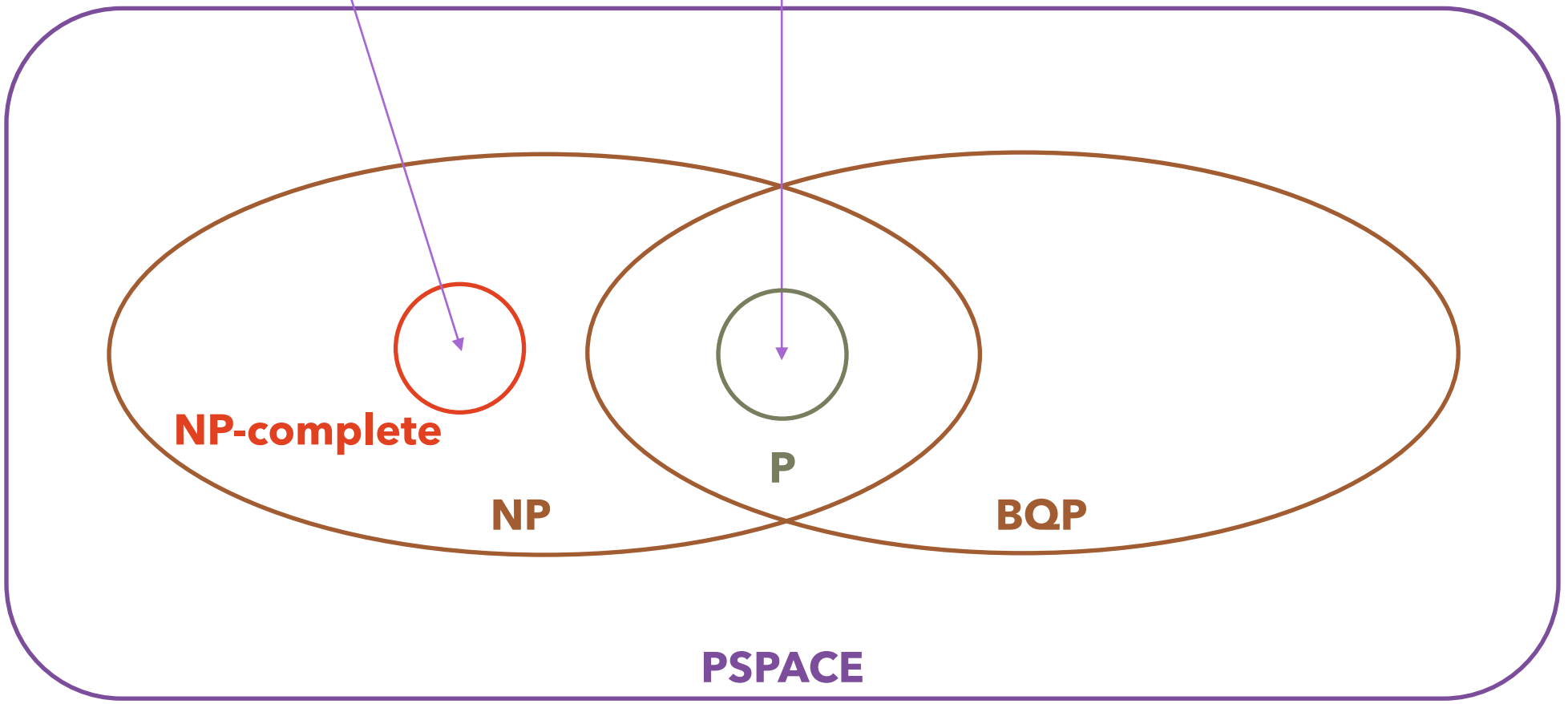
Ex: maximum cut problem **on cycle graph**



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

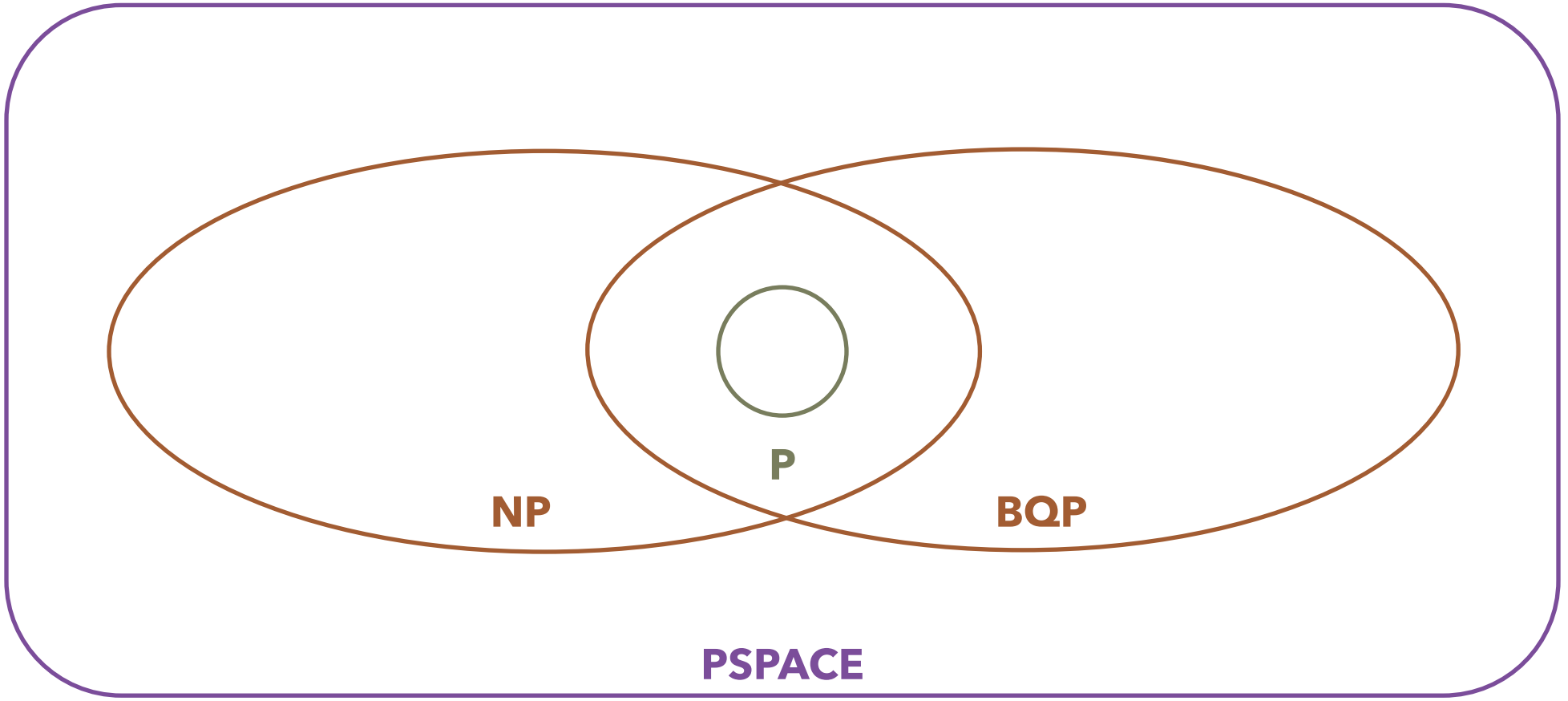
Ex: maximum cut problem **in worst case**

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 anneali
- 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

Q: can we solve NP-complete problems in polynomial time on a quantum computer?

1. Yes, because a quantum computer can try all 2^n solutions at the same time
2. Yes, because a quantum computer can solve integer factoring, which is NP-complete, in polynomial time
3. No, because quantum computers are inherently random
4. Probably not, but we do not have definitive proof yet

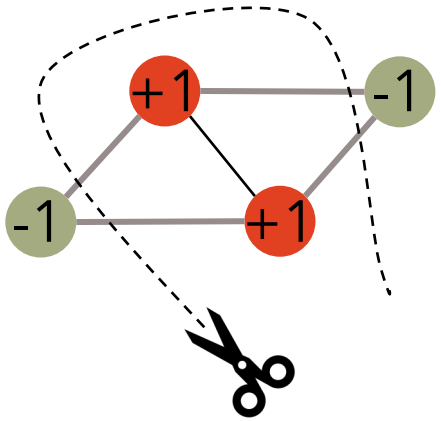
Q: can we solve NP-complete problems in polynomial time on a quantum computer?

1. Yes, because a quantum computer can try all 2^n solutions at the same time
2. Yes, because a quantum computer can solve integer factoring, which is NP-complete, in polynomial time
3. No, because quantum computers are inherently random
4. **Probably not, but we do not have definitive proof yet**

PART 1:
QUANTUM APPROXIMATE OPTIMIZATION ALGORITHM (QAOA)

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:

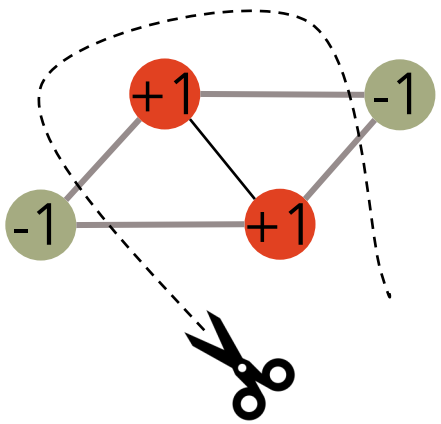


- Maximum cut can be formulated as an optimization problem:

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

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 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:



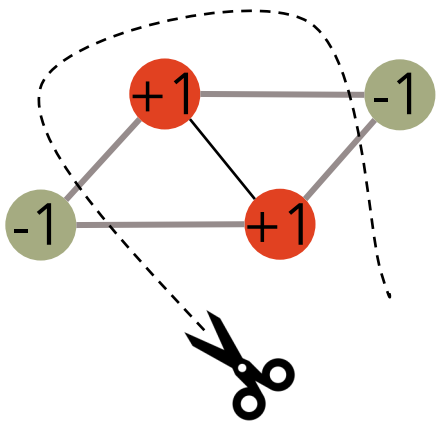
- Maximum cut can be formulated as an optimization problem:

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

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 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:

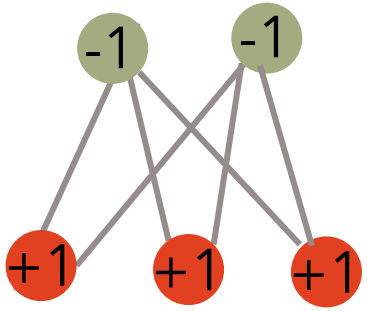


- Maximum cut can be formulated as an optimization problem:

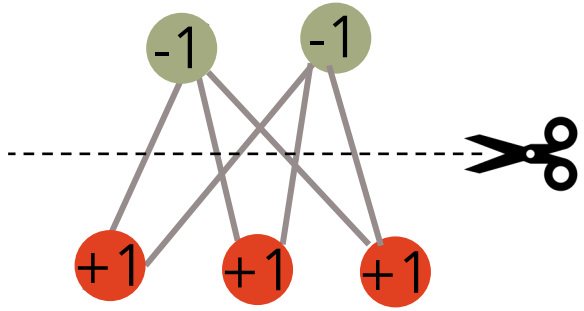
$$\max_{\mathbf{s}} \frac{1}{2} \sum_{ij \in E} (1 - s_i s_j) \qquad s_i \in \{-1, +1\}$$

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

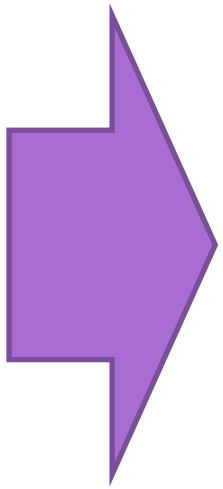
Solving Combinatorial Optimization Problems on a Quantum Computer

CLEMSON UNIVERSITY

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

CLEMSON UNIVERSITY

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 objective	$C(\mathbf{s})$	Expectation of C (energy) in state s $\langle s \vert C \vert s \rangle$
Solution	$\mathbf{s} \in \{-1, +1\}^n$	Highest energy eigenstate $\vert s \rangle$

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
Evaluation of objective	$C(\mathbf{s})$	Expectation of C (energy) in state s $\langle s \vert C \vert s \rangle$
Solution	$\mathbf{s} \in \{-1, +1\}^n$	Highest energy eigenstate $\vert s \rangle$

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

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

How do we construct this Hamiltonian?

Notation Reminder

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

Notation Reminder

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

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

Notation Reminder

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

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

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

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:

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

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,

$$\langle 0|Z|0\rangle = \langle 0|0\rangle = 1$$

$$\langle 1|Z|1\rangle = (-1)\langle 1|1\rangle = -1$$

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,

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

Pauli Z operator

- Acting on the i-th qubit:

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

Pauli Z operator

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

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

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

Constructing MAXCUT Hamiltonian

- MAXCUT objective:

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

- 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:

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

Verifying MAXCUT Hamiltonian

$$\langle x| C |x\rangle = \langle x_0 \dots x_n| \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) |x_0 \dots x_n\rangle$$

$$= \frac{1}{2} \sum_{ij \in E} \langle x_0 \dots x_n| (I - Z_i Z_j) |x_0 \dots x_n\rangle$$

$$= \frac{1}{2} \sum_{ij \in E} (1 - \langle x_0 \dots x_n| Z_i Z_j |x_0 \dots x_n\rangle)$$

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

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

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!

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**

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

- **Now all we need to do is find a quantum state $|x\rangle$ that maximizes $\langle x| C |x\rangle$!**

Q: which are correct formulations of MaxCut problem?

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

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

3. $\max_{\mathbf{x}} \langle x_0 \dots x_n | \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) | x_0 \dots x_n \rangle \quad x_i \in \{0, 1\}$

4. Find eigenvector with the largest eigenvalue of $\frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j)$

Q: which are correct formulations of MaxCut problem?

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

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

3. $\max_{\mathbf{x}} \langle x_0 \dots x_n | \frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j) | x_0 \dots x_n \rangle \quad x_i \in \{0, 1\}$

4. Find eigenvector with the largest eigenvalue of $\frac{1}{2} \sum_{ij \in E} (I - Z_i Z_j)$

5. **All of the above are equivalent (give the same *classical answer)***

Q: what is the size of Hamiltonian C?

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

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

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:

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

Quantum Approximate Optimization Algorithm (QAOA)

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

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

- 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

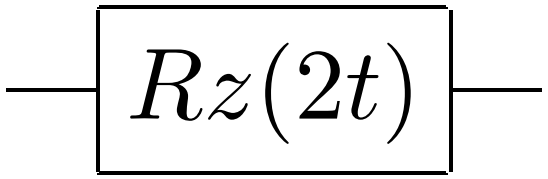
- Let's start with simple operator $e^{-iZt} = R_z(2t)$

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

Implementing QAOA

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

- Circuit:



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

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:

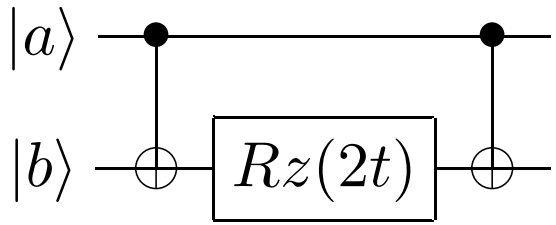
$$e^{-iZ\otimes Zt}|ab\rangle = e^{-i(-1)^{a\oplus b}t}|ab\rangle$$

Implementing QAOA

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

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

- Circuit:

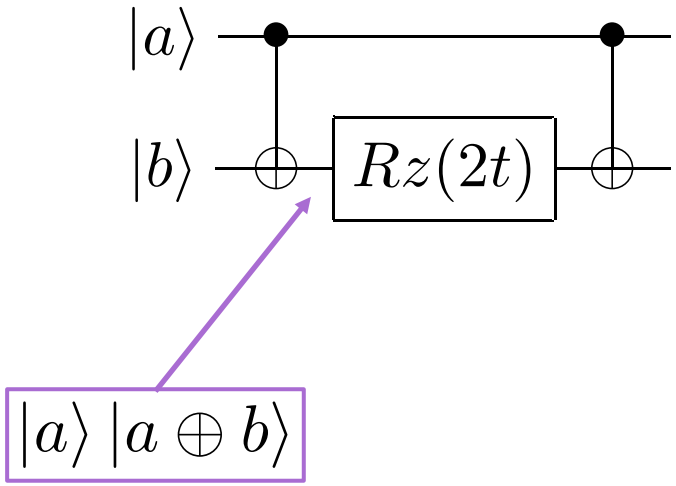


Implementing QAOA

- 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:

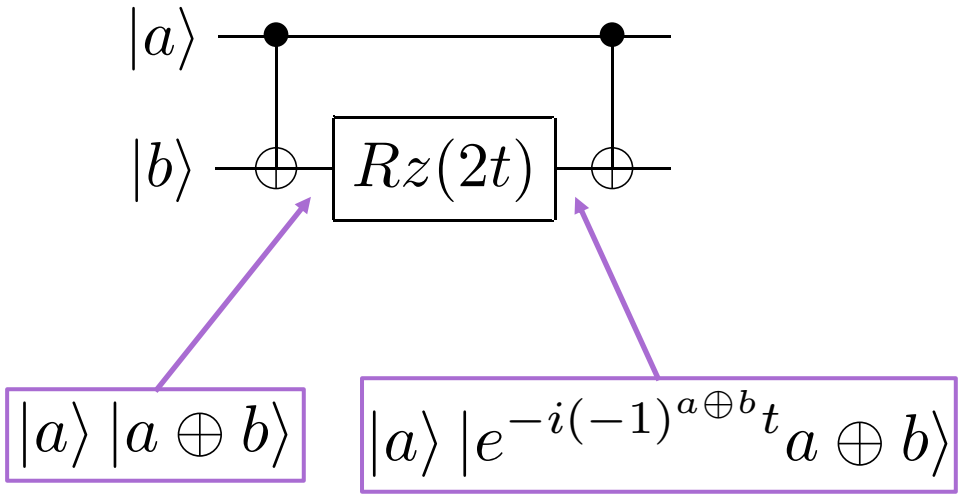


Implementing QAOA

- 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:

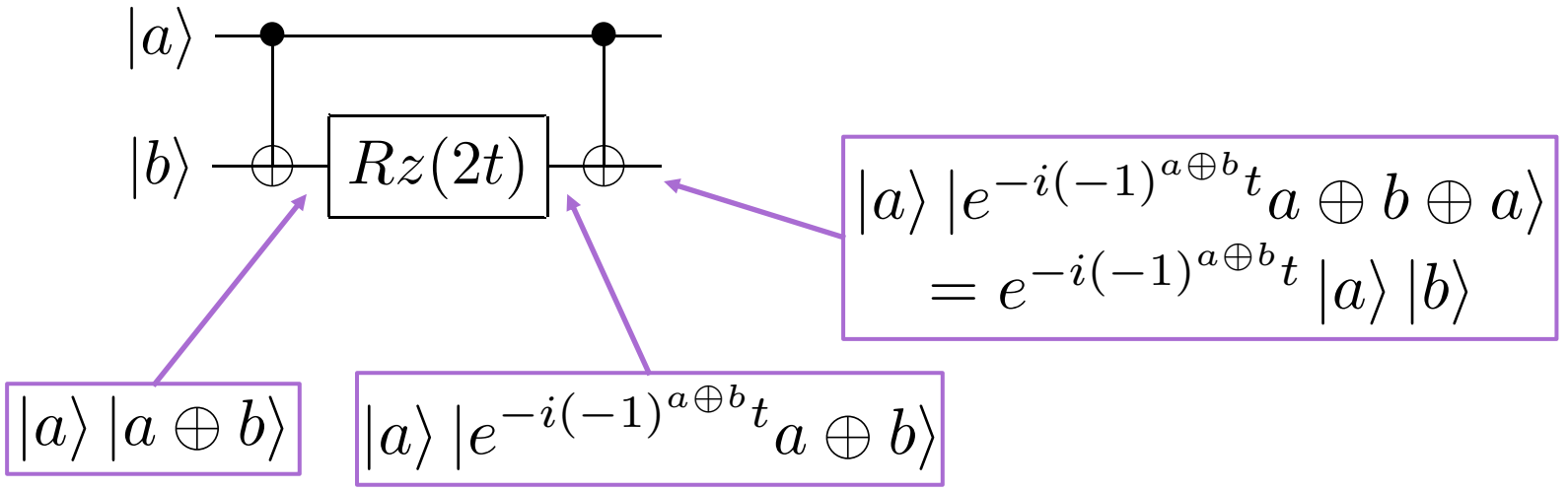


Implementing QAOA

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

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

- Circuit:

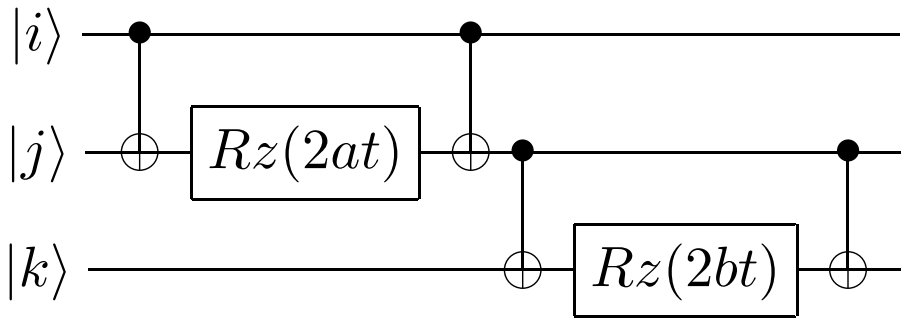


Implementing QAOA

- 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:

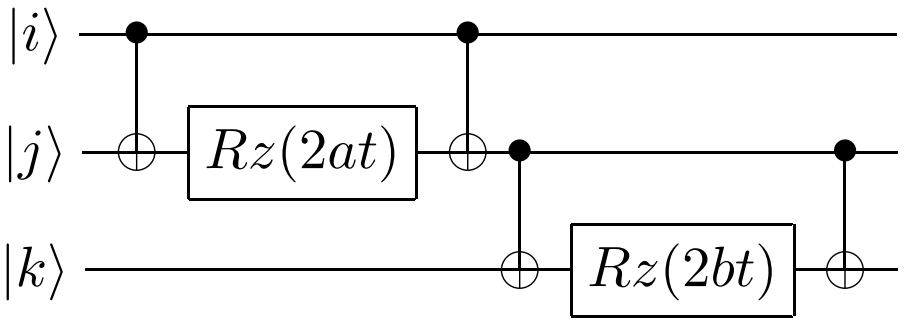


Implementing QAOA

- 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

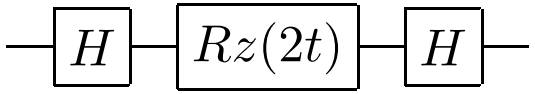
- Mixer operator: $e^{-i\beta B} = e^{-i\beta \sum_j X_j}$
- One term: e^{-iXt}

- Note that

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

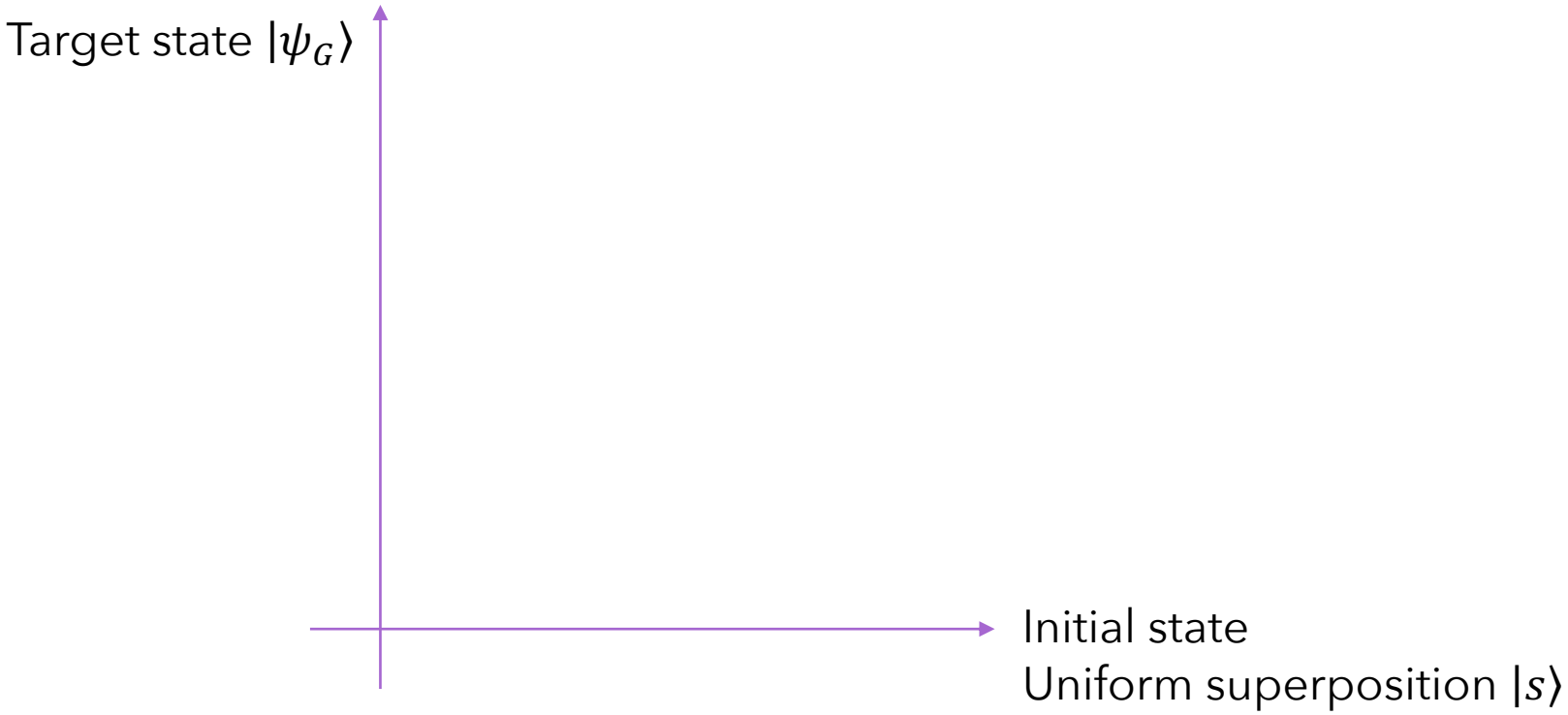
$$\begin{aligned} He^{-iZt}H &= H\left(I - iZt - \tfrac{Z^2t^2}{2!} + \dots\right)H = I - iHZHt - \tfrac{HZHHZHt^2}{2!} + \dots \\ &= I - iXt - \tfrac{X^2t^2}{2!} + \dots = e^{-iXt} \end{aligned}$$

- Circuit:



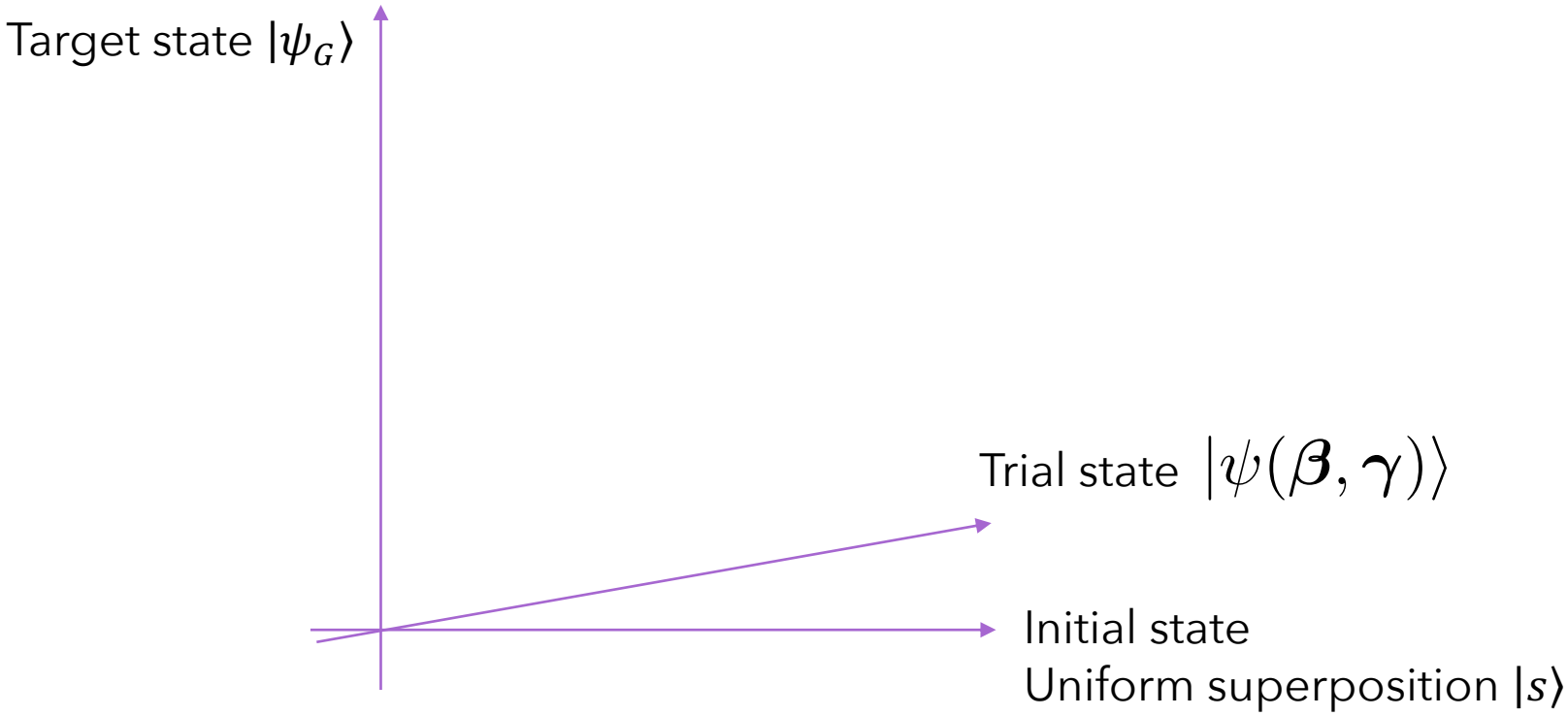
Quantum Approximate Optimization Algorithm (QAOA): Geometric Interpretation

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



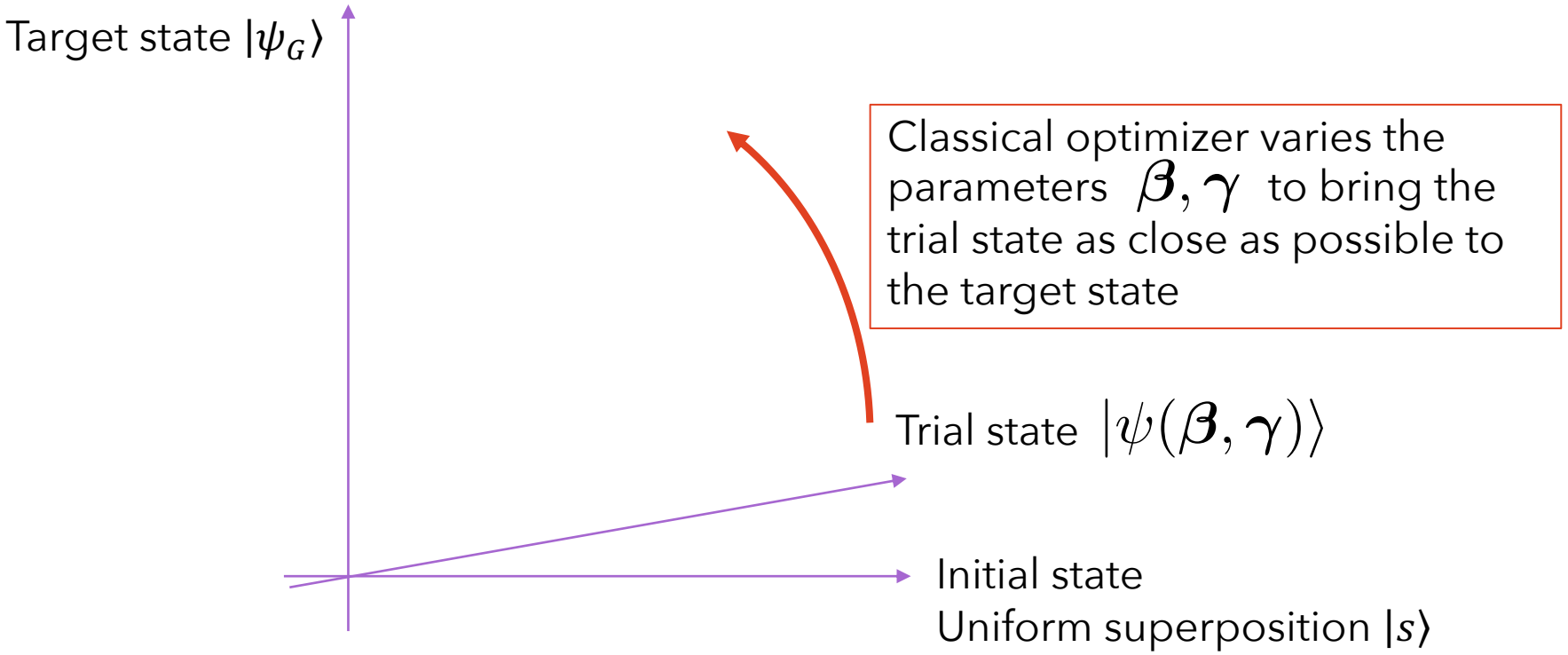
Quantum Approximate Optimization Algorithm (QAOA): Geometric Interpretation

- 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

Q: how does QAOA solve an optimization problem?

1. By efficiently searching 2^n-dimensional solution space
2. By exploring quantum tunneling to reach the ground state of the problem Hamiltonian
3. By preparing parameterized superpositions of solutions and using a classical optimizer to vary parameters to find the optimal solution
4. By preparing a vector close to the highest-eigenvalue eigenvector of the problem Hamiltonian
5. QAOA doesn't actually "solve" the problem, since there are no guarantees that the optimal solution will be reached

Q: how does QAOA solve an optimization problem?

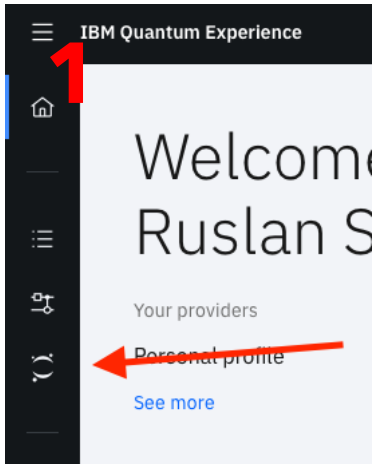
1. By efficiently searching 2^n-dimensional solution space
2. By exploring quantum tunneling to reach the ground state of the problem Hamiltonian
3. **By preparing parameterized superpositions of solutions and using a classical optimizer to vary parameters to find the optimal solution**
4. **By preparing a vector close to the highest-eigenvalue eigenvector of the problem Hamiltonian**
5. **QAOA doesn't actually "solve" the problem, since there are no guarantees that the optimal solution will be reached**

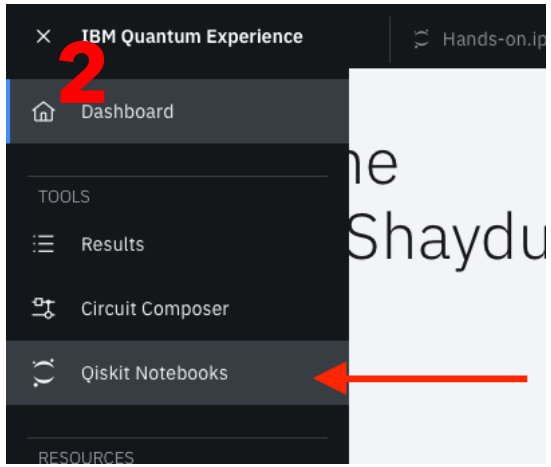
Thanks

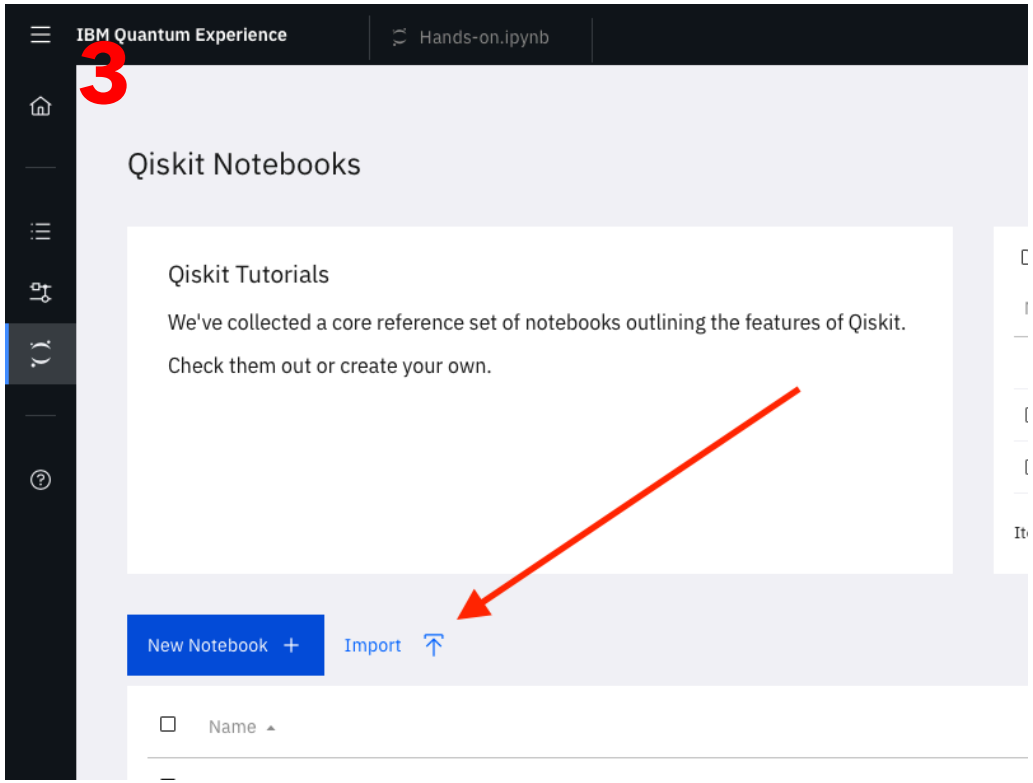
- “Introduction to Quantum Computing” tutorial at SC19 by Eleanor Rieffel and Scott Pakin.
- Paul Bernays lectures by Scott Aaronson
- 2019 Illinois Quantum Computing School lectures by Nathan Wiebe

PART 2:
HANDS-ON

- **Get the latest version of the notebook** from https://github.com/rsln-s/ANL_tutorial_June_15 or https://bit.ly/qtutorial2020
- 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