QCompute-QAPP Documentation

Release 0.0.1

Institute for Quantum Computing, Baidu Inc.

API DOCS

	QCompute-QAPP Package			
	1.1	<pre>qcompute_qapp.algorithm</pre>	1	
	1.2	<pre>qcompute_qapp.application</pre>	5	
	1.3	<pre>qcompute_qapp.circuit</pre>	7	
	1.4	<pre>qcompute_qapp.optimizer</pre>	13	
	1.5	<pre>qcompute_qapp.utils</pre>	16	
Python Module Index				
In	dex		19	

QCOMPUTE-QAPP PACKAGE

1.1 qcompute_qapp.algorithm

The variational quantum algorithm library

class qcompute_qapp.algorithm.**VQE**(num: int, hamiltonian: list, ansatz: ParameterizedCircuit, optimizer: BasicOptimizer, backend: str, measurement: str = 'default')

Bases: object

Variational Quantum Eigensolver class

The constructor of the VQE class

Parameters

- **num** (int) Number of qubits
- hamiltonian (list) Hamiltonian whose minimum eigenvalue is to be solved
- ansatz (ParameterizedCircuit) Ansatz used to search for the ground state of the Hamiltonian
- optimizer (BasicOptimizer) Optimizer used to optimize the parameters in the ansatz
- **backend** (*str*) Backend to be used in this task. Please refer to https://quantum-hub.baidu. com/quickGuide for details
- **measurement** (*str*) Method chosen from 'default', 'ancilla', and 'SimMeasure' for measuring the expectation value, defaults to 'default'

```
get_measure(shots: int = 1024) \rightarrow dict
```

Returns the measurement results

Parameters

shots (int) – Number of measurement shots, defaults to 1024

Returns

Measurement results in bitstrings with the number of counts

Return type

dict

```
get\_gradient(shots: int = 1024) \rightarrow ndarray
```

Calculates the gradient with respect to current parameters in circuit

Parameters

shots (int) – Number of measurement shots, defaults to 1024

Returns

Gradient with respect to current parameters

Return type

np.ndarray

```
get_loss(shots: int = 1024) \rightarrow float
```

Calculates the loss with respect to current parameters in circuit

Parameters

shots (int) – Number of measurement shots, defaults to 1024

Returns

Loss with respect to current parameters

Return type

float

run(*shots:* int = 1024) \rightarrow None

Searches for the minimum eigenvalue of the input Hamiltonian with the given ansatz and optimizer

Parameters

shots (int) – Number of measurement shots, defaults to 1024

property minimum_eigenvalue: Union[str, float]

The optimized minimum eigenvalue from last run

Returns

Optimized minimum eigenvalue from last run

Return type

Union[str, float]

 $set_backend(backend: str) \rightarrow None$

Sets the backend to be used

Parameters

backend (str) – Backend to be used

class qcompute_qapp.algorithm.SSVQE(num: int, ex_num: int, hamiltonian: list, ansatz:

ParameterizedCircuit, *optimizer*: BasicOptimizer, *backend*: *str*, *measurement*: *str* = 'default')

Bases: object

Subspace-Search Variational Quantum Eigensolver class

Please see https://journals.aps.org/prresearch/abstract/10.1103/PhysRevResearch.1.033062 for details on this algorithm.

The constructor of the SSVQE class

- **num** (*int*) Number of qubits
- **ex_num** (*int*) Number of extra eignevalues to be solved. When ex_num = 0, only compute the minimum eigenvalue
- hamiltonian (list) Hamiltonian whose eigenvalues are to be solved
- ansatz (ParameterizedCircuit) Ansatz used to search for the eigenstates of the Hamiltonian
- **optimizer** (BasicOptimizer) Optimizer used to optimize the parameters in the ansatz

- backend (str) Backend to be used in this task. Please refer to https://quantum-hub.baidu. com/quickGuide for details
- **measurement** (*str*) Method chosen from 'default', 'ancilla', and 'SimMeasure' for measuring the expectation value, defaults to 'default'

```
get\_gradient(shots: int = 1024) \rightarrow ndarray
```

Calculates the gradient with respect to current parameters in circuit

Parameters

shots (int) – Number of measurement shots, defaults to 1024

Returns

Gradient with respect to current parameters

Return type

np.ndarray

```
get_loss(shots: int = 1024) \rightarrow float
```

Calculates the loss with respect to current parameters in circuit

Parameters

shots (int) – Number of measurement shots, defaults to 1024

Returns

Loss with respect to current parameters

Return type

float

run(*shots:* int = 1024) \rightarrow None

Searches for the minimum eigenvalue of the input Hamiltonian with the given ansatz and optimizer

Parameters

shots (int) – Number of measurement shots, defaults to 1024

property minimum_eigenvalues: Union[str, list]

The optimized minimum eigenvalue from last run

Returns

Optimized minimum eigenvalues from last run

Return type

Union[str, list]

 $set_backend(backend: str) \rightarrow None$

Sets the backend to be used

Parameters

backend (str) – Backend to be used

class qcompute_qapp.algorithm.QAOA(num: int, hamiltonian: list, ansatz: QAOAAnsatz, optimizer:

BasicOptimizer, backend: str, measurement: str = 'default', delta: float = 0.1)

Bases: object

Quantum Approximate Optimization Algorithm class

The constructor of the QAOA class

Parameters

• **num** (*int*) – Number of qubits

- hamiltonian (list) Hamiltonian used to construct the QAOA ansatz
- ansatz (QAOAAnsatz) QAOA ansatz used to search for the maximum eigenstate of the Hamiltonian optimizer (BasicOptimizer): Optimizer used to optimize the parameters in the ansatz
- **backend** (*str*) Backend to be used in this task. Please refer to https://quantum-hub.baidu. com/quickGuide for details
- **measurement** (*str*) Method chosen from 'default', 'ancilla', and 'SimMeasure' for measuring the expectation value, defaults to 'default'
- **delta** (*float*) Parameter used to calculate gradients, defaults to 0.1

```
get_measure(shots: int = 1024) \rightarrow dict
```

Returns the measurement results

Parameters

shots (*int*) – Number of measurement shots, defaults to 1024

Returns

Measurement results in bitstrings with the number of counts

Return type

float

 $get_gradient(shots: int = 1024) \rightarrow ndarray$

Calculates the gradient with respect to current parameters in circuit

Parameters

shots (int) – Number of measurement shots, defaults to 1024

Returns

Gradient with respect to current parameters

Return type

np.ndarray

```
get_loss(shots: int = 1024) \rightarrow float
```

Calculates the loss with respect to current parameters in circuit

Parameters

shots (*int*) – Number of measurement shots, defaults to 1024

Returns

Loss with respect to current parameters

Return type

float

run(shots: int = 1024) \rightarrow None

Searches for the maximum eigenvalue of the input Hamiltonian with the given ansatz and optimizer

Parameters

shots (int) – Number of measurement shots, defaults to 1024

property maximum_eigenvalue: Union[str, float]

The optimized maximum eigenvalue from last run

Returns

Optimized maximum eigenvalue from last run

Return type

Union[str, float]

 $set_backend(backend: str) \rightarrow None$

Sets the backend to be used

Parameters

backend - Backend to be used

class qcompute_qapp.algorithm.KernelClassifier(backend: str, $encoding_style$: str = 'IQP', $kernel_type$: str = 'qke', shots: int = 1024)

Bases: object

Kernel Classifier class

The constructor of the KernelClassifier class

Parameters

- **backend** (*str*) Backend to be used in this task. Please refer to https://quantum-hub.baidu.com/quickGuide for details
- **encoding_style** (*str*) Encoding scheme to be used, defaults to 'IQP', which uses the default encoding scheme
- **kernel_type** (*str*) Type of kernel to be used, defaults to 'qke', i.e., <x1|x2>
- **shots** (*int*) Number of measurement shots, defaults to 1024

fit(X: ndarray, y: ndarray) \rightarrow None

Trains the classifier with known data

Parameters

- X (np.ndarray) Set of classical data vectors as the training data
- **y** (*np.ndarray*) Known labels of the training data

 $predict(x: ndarray) \rightarrow ndarray$

Predicts labels of new data

Parameters

x (np.ndarray) – Set of data vectors with unknown labels

Returns

Predicted labels of the input data

Return type

np.ndarray

1.2 qcompute_qapp.application

1.2.1 qcompute_qapp.application.chemistry

The function about quantum chemistry

```
class qcompute_qapp.application.chemistry.MolecularGroundStateEnergy(num\_qubits: int = 0, hamiltonian: Optional[list] = None)
```

Bases: object

Molecular Ground State Energy class

The constructor of the MolecularGroundStateEnergy class

Parameters

- num_qubits (int) Number of qubits, defaults to 0
- hamiltonian (list) Hamiltonian of the molecular system, defaults to None

property num_qubits: int

The number of qubits used to encoding this molecular system

Returns

Number of qubits

Return type

int

property hamiltonian: list

The Hamiltonian of this molecular system

Returns

Hamiltonian of this molecular system

Return type

list

$compute_ground_state_energy() \rightarrow float$

Analytically computes the ground state energy

Returns

minimum real part of eigenvalues

Return type

float

load_hamiltonian_from_file($filename: str, separator: str = ', ') \rightarrow None$

Loads Hamiltonian from a file

Parameters

- filename(str) Path to the file storing the Hamiltonian in Pauli terms
- **separator** (str) Delimiter between coefficient and Pauli string, defaults to ', '

1.2.2 gcompute_gapp.application.optimization

The function about optimization

```
class qcompute_qapp.application.optimization.MaxCut(num\_qubits: int = 0, hamiltonian: Optional[list] = None)
```

Bases: object

Max Cut Problem class

The constructor of the MaxCut class

- num_qubits (int) Number of qubits, defaults to 0
- hamiltonian (list) Hamiltonian of the target graph of the Max Cut problem, defaults to None

property num_qubits: int

The number of qubits used to encoding this target graph

Returns

Number of qubits used to encoding this target graph

Return type

int

property hamiltonian: list

The Hamiltonian of this target graph

Returns

Hamiltonian of this target graph

Return type

list

$graph_to_hamiltonian(graph: Graph) \rightarrow None$

Constructs Hamiltonian from the target graph of the Max Cut problem

Parameters

graph (nx. Graph) – Undirected graph without weights

$decode_bitstring(bitstring: str) \rightarrow dict$

Decodes the measurement result into problem solution, i.e., set partition

Parameters

bitstring (*str*) – Measurement result with the largest probability

Returns

Solution to the Max Cut problem

Return type

dict

1.3 qcompute_qapp.circuit

The pre-defined quantum circuit

```
class qcompute_qapp.circuit.BasicCircuit(num: int)
```

Bases: ABC

Basic Circuit class

The constructor of the BasicCircuit class

Parameters

num (int) – Number of qubits

abstract add_circuit(q: QRegPool) \rightarrow None

Adds circuit to the register.

Parameters

q (QRegPool) – Quantum register to which this circuit is added

class qcompute_qapp.circuit.IQPEncodingCircuit(num: int, inverse: bool = False)

Bases: BasicCircuit

IQP Encoding Circuit class

The constructor of the IQPEncodingCircuit class

Parameters

- **num** (*int*) Number of qubits
- inverse (boo1) Whether the encoding circuit will be inverted, i.e. U^dagger(x) if True, defaults to False

 $add_circuit(q: QRegPool, x: ndarray) \rightarrow None$

Adds the encoding circuit used to map a classical data vector into its quantum feature state

Parameters

- q (QRegPool) Quantum register to which this circuit is added
- **x** (np.ndarray) Classical data vector to be encoded

class qcompute_qapp.circuit.BasisEncodingCircuit(num: int, bit_string: str)

Bases: BasicCircuit

Basis Encoding Circuit class

The constructor of the BasisEncodingCircuit class

Parameters

- **num** (int) Number of qubits
- **bit_string** (*str*) Bit string to be encoded as a quantum state

 $add_circuit(q: QRegPool) \rightarrow None$

Adds the basis encoding circuit to the register

Parameters

q (QRegPool) – Quantum register to which this circuit is added

class qcompute_qapp.circuit.KernelEstimationCircuit(num: int, encoding_style: str)

Bases: BasicCircuit

Kernel Estimation Circuit class

The constructor of the KernelEstimationCircuit class

Parameters

- **num** (*int*) Number of qubits
- encoding_style (str) Encoding circuit, only accepts 'IQP' for now

 $add_circuit(q: QRegPool, x1: ndarray, x2: ndarray) \rightarrow None$

Adds the kernel estimation circuit used to evaluate the kernel entry value between two classical data vectors

- q (QRegPool) Quantum register to which this circuit is added
- x1 (np.ndarray) First classical vector
- **x2** (np.ndarray) Second classical vector

class qcompute_qapp.circuit.ParameterizedCircuit(num: int, parameters: ndarray)

Bases: BasicCircuit

Parameterized Circuit class

The constructor of the BasicCircuit class

Parameters

- **num** (int) Number of qubits
- parameters (np.ndarray) Parameters of parameterized gates

property parameters: ndarray

Parameters of the circuit

Returns

Parameters of the circuit

Return type

np.ndarray

 $set_parameters(parameters: ndarray) \rightarrow None$

Sets parameters of the circuit

Parameters

parameters (*np.ndarray*) – New parameters of the circuit

abstract add_circuit(q: QRegPool) \rightarrow None

Adds the circuit to the register

Parameters

q (QRegPool) – Quantum register to which this circuit is added

class qcompute_qapp.circuit.PauliMeasurementCircuit(num: int, pauli_terms: str)

Bases: BasicCircuit

Pauli Measurement Circuit class

The constructor of the PauliMeasurementCircuit class

Parameters

- **num** (*int*) Number of qubits
- pauli_terms (str) Pauli terms to be measured

 $add_circuit(q: QRegPool, pauli_str: str) \rightarrow None$

Adds the pauli measurement circuit to the register

Parameters

- q (QRegPool) Quantum register to which this circuit is added
- **pauli_str** (*str*) Pauli string to be measured

 $get_expectation(preceding_circuits: List[BasicCircuit], shots: int, backend: str) <math>\rightarrow$ float

Computes the expectation value of the Pauli terms

- **preceding_circuits** (*List* [BasicCircuit]) Circuit precedes the measurement circuit
- **shots** (*int*) Number of measurement shots

• backend (str) – Backend to be used in this task

Returns

Expectation value of the Pauli terms

Return type

float

class qcompute_qapp.circuit.PauliMeasurementCircuitWithAncilla(num: int, pauli_terms: str)

Bases: BasicCircuit

Pauli Measurement Circuit with Ancilla class

The constructor of the PauliMeasurementCircuitWithAncilla class

Parameters

- **num** (*int*) Number of qubits
- pauli_terms (str) Pauli terms to be measured

 $add_circuit(q: QRegPool, pauli_str: str) \rightarrow None$

Adds the pauli measurement circuit to the register

Parameters

- **q** (QRegPool) Quantum register to which this circuit is added
- pauli_str (str) Pauli string to be measured

 $get_expectation(preceding_circuits: List[BasicCircuit], shots: int, backend: str) \rightarrow float$

Computes the expectation value of the Pauli terms

Parameters

- **preceding_circuits** (*List* [BasicCircuit]) Circuit precedes the measurement circuit
- shots(int) Number of measurement shots
- **backend** (str) Backend to be used in this task

Returns

Expectation value of the Pauli terms

Return type

float

class qcompute_qapp.circuit.SimultaneousPauliMeasurementCircuit(num: int, pauli_terms: list)

Bases: BasicCircuit

Simultaneous Pauli Measurement Circuit for Qubitwise Commute Pauli Terms

The constructor of the SimultaneousPauliMeasurementCircuit class

Parameters

- **num** (*int*) Number of qubits
- pauli_terms (list) Pauli terms to be measured

 $add_circuit(q: QRegPool, clique: list) \rightarrow None$

Adds the simultaneous pauli measurement circuit to the register

Parameters

• **q** (*QRegPool*) – Quantum register to which this circuit is added

• clique (list) – Clique of Pauli terms to be measured together

 $get_expectation(preceding_circuits: List[BasicCircuit], shots: int, backend: str) <math>\rightarrow$ float

Computes the expectation value of the Pauli terms

Parameters

- preceding_circuits (List[BasicCircuit]) Circuit precedes the measurement circuit
- **shots** (*int*) Number of measurement shots
- backend (str) Backend to be used in this task

Returns

Expectation value of the Pauli terms

Return type

float

class qcompute_qapp.circuit.QAOAAnsatz(num: int, parameters: ndarray, hamiltonian: list, layer: int)

Bases: ParameterizedCircuit

QAOA Ansatz class

The constructor of the QAOAAnsatz class

Parameters

- **num** (*int*) Number of qubits in this ansatz
- parameters (np.ndarray) Parameters of parameterized gates in this ansatz
- hamiltonian (list) Hamiltonian used to construct the QAOA ansatz
- layer (int) Number of layers for this Ansatz

 $add_circuit(q: QRegPool) \rightarrow None$

Adds circuit to the register according to the given hamiltonian

Parameters

q (QRegPool) – Quantum register to which this circuit is added

class qcompute_qapp.circuit.UniversalCircuit(num: int, parameters: ndarray)

Bases: ParameterizedCircuit

Universal Circuit class

The constructor of the UniversalCircuit class

Parameters

- **num** (*int*) Number of qubits in this ansatz
- **parameters** (*np.ndarray*) Parameters of parameterized gates in this circuit, whose shape should be (3,) for single-qubit cases and should be (15,) for 2-qubit cases

 $add_circuit(q: QRegPool) \rightarrow None$

Adds the universal circuit to the register. Only support single-qubit and 2-qubit cases

Parameters

q (QRegPool) – Quantum register to which this circuit is added

class qcompute_qapp.circuit.RealEntangledCircuit(num: int, layer: int, parameters: ndarray)

Bases: ParameterizedCircuit

Real Entangled Circuit class

The constructor of the RealEntangledCircuit class

Parameters

- **num** (*int*) Number of qubits in this ansatz
- layer (int) Number of layers for this ansatz
- parameters (np.ndarray) Parameters of parameterized gates in this circuit, whose shape should be (num * layer,)

 $add_circuit(q: QRegPool) \rightarrow None$

Adds the real entangled circuit to the register

Parameters

q (QRegPool) – Quantum register to which this circuit is added

class qcompute_qapp.circuit.ComplexEntangledCircuit(num: int, layer: int, parameters: ndarray)

Bases: ParameterizedCircuit

Complex Entangled Circuit class

The constructor of the ComplexEntangledCircuit class

Parameters

- **num** (*int*) Number of qubits in this Ansatz
- layer (int) Number of layer for this Ansatz
- parameters (np.ndarray) Parameters of parameterized gates in this circuit, whose shape should be (num * layer * 2,)

 $add_circuit(q: QRegPool) \rightarrow None$

Adds the complex entangled circuit to the register

Parameters

q (QRegPool) – Quantum register to which this circuit is added

Bases: ParameterizedCircuit

Real Alternating Layered Circuit class

The constructor of the RealAlternatingLayeredCircuit class

Parameters

- **num** (*int*) Number of qubits in this Ansatz
- layer (int) Number of layer for this Ansatz
- parameters (np.ndarray) Parameters of parameterized gates in this circuit, whose shape should be ((2 * num 2) * layer,)

 $add_circuit(q: QRegPool) \rightarrow None$

Adds the real alternating layered circuit to the register

Parameters

q (*QRegPool*) – Quantum register to which this circuit is added

Bases: ParameterizedCircuit

Complex Alternating Layered Circuit class

The constructor of the ComplexAlternatingLayeredCircuit class

Parameters

- **num** (*int*) Number of qubits in this Ansatz
- layer (int) Number of layer for this Ansatz
- parameters (np.ndarray) Parameters of parameterized gates in this circuit, whose shape should be ((4 * num 4) * layer,)

 $add_circuit(q: QRegPool) \rightarrow None$

Adds the complex alternating layered circuit to the register

Parameters

q (QRegPool) – Quantum register to which this circuit is added

1.4 qcompute_qapp.optimizer

The provided optimizer

class qcompute_qapp.optimizer.BasicOptimizer(iterations: int, circuit: ParameterizedCircuit)

Bases: ABC

Basic Optimizer class

The constructor of the BasicOptimizer class

Parameters

- **iterations** (*int*) Number of iterations
- circuit (ParameterizedCircuit) Circuit whose parameters are to be optimized

set_circuit(*circuit*: ParameterizedCircuit) → None

Sets the parameterized circuit to be optimized

Parameters

circuit (ParameterizedCircuit) - Parameterized Circuit to be optimized

abstract minimize($shots: int, loss_func: Callable[[ndarray, int], float], grad_func: Callable[[ndarray, int], ndarray]) <math>\rightarrow$ None

Minimizes the given loss function

- **shots** (*int*) Number of measurement shots
- loss_func (Callable[[np.ndarray, int], float]) Loss function to be minimized
- grad_func (Callable[[np.ndarray, int], np.ndarray]) Function for calculating gradients

class qcompute_qapp.optimizer.**SGD**(iterations: int, circuit: BasicCircuit, learning_rate: float)

Bases: BasicOptimizer

SGD Optimizer class

The constructor of the SGD class

Parameters

- **iterations** (*int*) Number of iterations
- circuit (BasicCircuit) Circuit whose parameters are to be optimized
- learning_rate (float) Learning rate

minimize(shots: int, loss_func: Callable[[ndarray, int], float], grad_func: Callable[[ndarray, int], ndarray]) \rightarrow None

Minimizes the given loss function

Parameters

- **shots** (*int*) Number of measurement shots
- loss_func (Callable[[np.ndarray, int], float]) Loss function to be minimized
- grad_func (Callable[[np.ndarray, int], np.ndarray]) Function for calculating gradients

class qcompute_qapp.optimizer.SLSQP(iterations: int, circuit: BasicCircuit)

Bases: BasicOptimizer

SLSQP Optimizer class

The constructor of the SLSQP class

Parameters

- **iterations** (*int*) Number of iterations
- **circuit** (BasicCircuit) Circuit whose parameters are to be optimized

minimize(shots: int, loss_func: Callable[[ndarray, int], float], grad_func: Callable[[ndarray, int], ndarray]) \rightarrow None

Minimizes the given loss function

Parameters

- **shots** (*int*) Number of measurement shots
- loss_func (Callable[[np.ndarray, int], float]) Loss function to be minimized
- grad_func (Callable[[np.ndarray, int], np.ndarray]) Function for calculating gradients

class qcompute_qapp.optimizer.SPSA(iterations: int, circuit: BasicCircuit, a: float = 1.0, c: float = 1.0)

Bases: BasicOptimizer

SPSA Optimizer class

The constructor of the SPSA class

Parameters

• **iterations** (*int*) – Number of iterations

- **circuit** (BasicCircuit) Circuit whose parameters are to be optimized
- a (float) Scaling parameter for step size, defaults to 1.0
- c (float) Scaling parameter for evaluation step size, defaults to 1.0

 $minimize(shots: int, loss_func: Callable[[ndarray, int], float], grad_func: Callable[[ndarray, int], ndarray]) <math>\rightarrow$ None

Minimizes the given loss function

Parameters

- **shots** (*int*) Number of measurement shots
- loss_func (Callable[[np.ndarray, int], float]) Loss function to be minimized
- grad_func (Callable[[np.ndarray, int], np.ndarray]) Function for calculating gradients

class qcompute_qapp.optimizer.SMO(iterations: int, circuit: BasicCircuit)

Bases: BasicOptimizer

SMO Optimizer class

Please see https://arxiv.org/abs/1903.12166 for details on this optimization method.

The constructor of the SMO class

Parameters

- **iterations** (*int*) Number of iterations
- circuit (BasicCircuit) Circuit whose parameters are to be optimized

minimize(shots: int, loss_func: Callable[[ndarray, int], float], grad_func: Callable[[ndarray, int], ndarray]) \rightarrow None

Minimizes the given loss function

Parameters

- **shots** (*int*) Number of measurement shots
- loss_func (Callable[[np.ndarray, int], float]) Loss function to be minimized
- grad_func (Callable[[np.ndarray, int], np.ndarray]) Function for calculating gradients

class qcompute_qapp.optimizer.Powell(iterations: int, circuit: BasicCircuit)

Bases: BasicOptimizer

Powell Optimizer class

The constructor of the Powell class

Parameters

- **iterations** (*int*) Number of iterations
- circuit (BasicCircuit) Circuit whose parameters are to be optimized

minimize(shots: int, loss_func: Callable[[ndarray, int], float], grad_func: Callable[[ndarray, int], ndarray]) \rightarrow None

Minimizes the given loss function

Parameters

- **shots** (*int*) Number of measurement shots
- loss_func (Callable[[np.ndarray, int], float]) Loss function to be minimized
- grad_func (Callable[[np.ndarray, int], np.ndarray])) Function for calculating gradients

1.5 qcompute_qapp.utils

Some auxiliary functions

qcompute_qapp.utils.grouping_hamiltonian(hamiltonian: list, coloring_strategy: $str = 'largest_first') \rightarrow List[List[str]]$

Finds the minimum clique cover of the Hamiltonian graph, which is used for simultaneous Pauli measurement

Parameters

- **hamiltonian** (*list*) Hamiltonian of the target system
- **coloring_strategy** (*str*) Graph coloring strategy chosen from the following: 'largest_first', 'random_sequential', 'smallest_last', 'independent_set', 'connected_sequential_bfs', 'connected_sequential_dfs', 'connected_sequential', 'saturation_largest_first', and 'DSATUR'; defaults to 'largest_first'

Returns

List of cliques consisting of Pauli strings to be measured together

Return type

List[List[str]]

 $\verb|qcompute_qapp.utils.pauli_terms_to_matrix| (pauli_terms: list) \rightarrow \verb|ndarray|$

Converts Pauli terms to a matrix

Parameters

pauli_terms (list) – Pauli terms whose matrix is to be computed

Returns

Matrix form of the Pauli terms

Return type

np.ndarray

PYTHON MODULE INDEX

q

```
qcompute_qapp.algorithm, 1
qcompute_qapp.application.chemistry, 5
qcompute_qapp.application.optimization, 6
qcompute_qapp.circuit, 7
qcompute_qapp.optimizer, 13
qcompute_qapp.utils, 16
```

18 Python Module Index

INDEX

Α	В		
<pre>add_circuit() (qcompute_qapp.circuit.BasicCircuit</pre>	BasicCircuit (class in qcompute_qapp.circuit), 7		
method), 7	<pre>BasicOptimizer (class in qcompute_qapp.optimizer),</pre>		
add_circuit() (qcom-	13		
pute_qapp.circuit.BasisEncodingCircuit	BasisEncodingCircuit (class in qcom-		
method), 8	pute_qapp.circuit), 8		
add_circuit() (qcom-	^		
pute_qapp.circuit.ComplexAlternatingLayeredC	irsuit		
method), 13	ComplexAlternatingLayeredCircuit (class in qcom-		
add_circuit() (qcom-	pute_qapp.circuit), 13		
<pre>pute_qapp.circuit.ComplexEntangledCircuit method), 12</pre>	ComplexEntangledCircuit (class in qcom-		
add_circuit() (qcom-	pute_qapp.circuit), 12		
pute_qapp.circuit.IQPEncodingCircuit	compute_ground_state_energy() (qcom-		
method), 8	pute_qapp.application.chemistry.MolecularGroundStateEnergy		
add_circuit() (qcom-	method), 6		
pute_qapp.circuit.KernelEstimationCircuit	D		
method), 8	_		
add_circuit() (qcom-	decode_bitstring() (qcom-		
pute_qapp.circuit.ParameterizedCircuit	pute_qapp.application.optimization.MaxCut		
method), 9	method), 7		
add_circuit() (qcom-	F		
pute_qapp.circuit.PauliMeasurementCircuit	•		
method), 9	<pre>fit() (qcompute_qapp.algorithm.KernelClassifier method), 5</pre>		
add_circuit() (qcom-			
pute_qapp.circuit.PauliMeasurementCircuitWith	nA ns illa		
method), 10	ant armostation()		
add_circuit() (qcompute_qapp.circuit.QAOAAnsatz	<pre>get_expectation() (qcom- pute_qapp.circuit.PauliMeasurementCircuit</pre>		
method), 11	method), 9		
add_circuit() (qcom-			
pute_qapp.circuit.RealAlternatingLayeredCircu	pute_qapp.circuit.PauliMeasurementCircuitWithAncilla		
method), 12 add_circuit() (qcom-	method), 10		
<pre>add_circuit()</pre>	get_expectation() (qcom-		
method), 12	pute_qapp.circuit.SimultaneousPauliMeasurementCircuit		
add_circuit() (qcom-	method), 11		
pute_qapp.circuit.SimultaneousPauliMeasureme			
method), 10	method), 4		
add_circuit() (qcom-	<pre>get_gradient() (qcompute_qapp.algorithm.SSVQE</pre>		
pute_qapp.circuit.UniversalCircuit method),	method), 3		
11	<pre>get_gradient() (qcompute_qapp.algorithm.VQE</pre>		
	method), 1		

```
get_loss() (qcompute_qapp.algorithm.QAOA method), minimum_eigenvalue (qcompute_qapp.algorithm.VQE
                                                             property), 2
                   (qcompute qapp.algorithm.SSVQE
get_loss()
                                                    minimum_eigenvalues
                                                             pute_qapp.algorithm.SSVQE property), 3
        method), 3
get_loss() (qcompute_qapp.algorithm.VQE method), 2
                                                    module
                    (qcompute_qapp.algorithm.QAOA
get_measure()
                                                         qcompute_qapp.algorithm, 1
        method), 4
                                                         qcompute_qapp.application.chemistry, 5
                     (qcompute_qapp.algorithm.VQE
                                                         qcompute_qapp.application.optimization, 6
get_measure()
        method), 1
                                                         qcompute_qapp.circuit, 7
graph_to_hamiltonian()
                                                         qcompute_qapp.optimizer, 13
                                            (qcom-
        pute_qapp.application.optimization.MaxCut
                                                         qcompute_qapp.utils, 16
        method), 7
                                                    MolecularGroundStateEnergy
                                                                                    (class in
                                                                                                 acom-
                                                             pute_qapp.application.chemistry), 5
                                             qcom-
grouping_hamiltonian()
                             (in
                                   module
        pute_qapp.utils), 16
                                                    Ν
Н
                                                    num\_qubits(qcompute\_qapp.application.chemistry.MolecularGroundState)
hamiltonian (qcompute_qapp.application.chemistry.MolecularGround@nateEnergy
                                                    num_qubits(qcompute_qapp.application.optimization.MaxCut
        property), 6
\verb|hamiltonian| (q compute\_q app.application.optimization.MaxCut|
                                                             property), 7
        property), 7
                                                    P
                                                    ParameterizedCircuit
                                                                                 (class
                                                                                                  acom-
                                                             pute_qapp.circuit), 8
IQPEncodingCircuit (class in qcompute_qapp.circuit),
                                                    \verb"parameters" (q compute\_q app.circuit. Parameterized Circuit
                                                             property), 9
K
                                                    pauli_terms_to_matrix()
                                                                                  (in
                                                                                        module
                                                                                                  qcom-
                                                             pute_qapp.utils), 16
KernelClassifier
                         (class
                                    in
                                             qcom-
                                                    PauliMeasurementCircuit
                                                                                   (class
                                                                                            in
                                                                                                  qcom-
        pute_qapp.algorithm), 5
                                                             pute_qapp.circuit), 9
KernelEstimationCircuit
                              (class
                                             qcom-
                                       in
                                                    PauliMeasurementCircuitWithAncilla (class in
        pute_qapp.circuit), 8
                                                             qcompute_qapp.circuit), 10
                                                    Powell (class in qcompute_qapp.optimizer), 15
                                                    predict() (gcompute gapp.algorithm.KernelClassifier
load_hamiltonian_from_file()
                                            (qcom-
        pute_qapp.application.chemistry.MolecularGroundStateEnergy method), 5
        method), 6
M
                                                    QAOA (class in qcompute_qapp.algorithm), 3
                                                    QAOAAnsatz (class in geompute gapp.circuit), 11
MaxCut
                 (class
                                 in
                                             qcom-
                                                    qcompute_qapp.algorithm
        pute_qapp.application.optimization), 6
                                                         module, 1
maximum_eigenvalue
                                            (qcom-
                                                    qcompute_qapp.application.chemistry
        pute gapp.algorithm.QAOA property), 4
                                                         module, 5
minimize() (qcompute_qapp.optimizer.BasicOptimizer
                                                    qcompute_qapp.application.optimization
        method), 13
                                                         module, 6
minimize() (qcompute_qapp.optimizer.Powell method),
                                                    qcompute_qapp.circuit
                                                         module, 7
minimize() (qcompute_qapp.optimizer.SGD method),
                                                    qcompute_qapp.optimizer
                                                         module, 13
minimize() (qcompute qapp.optimizer.SLSQP method),
                                                    qcompute_qapp.utils
                                                         module, 16
minimize() (qcompute_qapp.optimizer.SMO method),
                                                    R
minimize() (qcompute_qapp.optimizer.SPSA method),
                                                    RealAlternatingLayeredCircuit (class in gcom-
        15
                                                             pute_qapp.circuit), 12
```

20 Index

```
RealEntangledCircuit
                            (class
                                       in
                                             qcom-
        pute_qapp.circuit), 11
run() (qcompute_qapp.algorithm.QAOA method), 4
run() (qcompute_qapp.algorithm.SSVQE method), 3
run() (qcompute_qapp.algorithm.VQE method), 2
S
set_backend()
                    (qcompute_qapp.algorithm.QAOA
        method), 5
set_backend()
                    (qcompute_qapp.algorithm.SSVQE
        method), 3
set_backend()
                      (qcompute_qapp.algorithm.VQE
        method), 2
set_circuit()
                                             (qcom-
        pute_qapp.optimizer.BasicOptimizer method),
         13
set_parameters()
                                             (qcom-
        pute_qapp.circuit.ParameterizedCircuit
        method), 9
SGD (class in qcompute_qapp.optimizer), 13
SimultaneousPauliMeasurementCircuit (class in
         qcompute_qapp.circuit), 10
SLSQP (class in qcompute_qapp.optimizer), 14
SMO (class in qcompute_qapp.optimizer), 15
SPSA (class in qcompute_qapp.optimizer), 14
SSVQE (class in qcompute_qapp.algorithm), 2
U
UniversalCircuit (class in qcompute_qapp.circuit),
V
VQE (class in qcompute_qapp.algorithm), 1
```

Index 21