OpenFermionProjectQ Documentation

Release 0.2

openfermionprojectq

Aug 17, 2018

Contents

Contents

CHAPTER 1

Code Documentation

1.1 openfermionprojectq

class openfermionprojectq.**Graph**

An undirected graph of nodes of arbitrary connectivity

A generic graph class for undirected graphs that holds Nodes and edges that connect them.

nodes
list – A list of Node objects containing the nodes of the graph

node_uids
list – A list of unique IDs (uids) for each node

uid_to_index
dict – A dictionary that maps UIDs currently present in the graph to their node index

neigbors
list of sets – A list of sets that enumerate the neighbors of each node. For example the neighbors of node i are the set neighbors[i]

next_uid
int – The next unique ID to be assigned to a node on addition

__init__()
Set up an empty graph with no nodes

add_edge(*node_id1*, *node_id2*)
Add an edge between node 1 and node2

Parameters

- **node_id1** (*int*) – Index of first node on edge
- **node_id2** (*int*) – Index of second node on edge

add_node(*node=<openfermionprojectq._graph.Node instance>*)
Add a Node to the graph.

Parameters **node** (Node) – A Node object to be added to the graph

Returns(int): The unique identified that was given to the node

find_index(*value*, *starting_node=0*)
Find the index of the first node that matches value in a BFS

Performs a breadth-first search of the graph starting at node index starting_node. Returns the index or None if no match is found

Parameters

- **value** (*Node Value*) –
- **starting_node** (*int*) –

get_neighbors(*node_id*)
Return list of neighbors of the specified node

Parameters **node_id** – Index of node to examine the neighbors of

Returns(list): List of current node IDs that are neighbors of node_id.

is_adjacent(*node_id1*, *node_id2*)
Test for adjacency between node1 and node2

Parameters

- **node_id1** (*int*) – Index of first node
- **node_id2** (*int*) – Index of second node

node_count()
Number of nodes in the graph

Returns(int): Number of nodes currently in the graph

remove_edge(*node_id1*, *node_id2*)
Remove an edge between node1 and node2

Parameters

- **node_id1** (*int*) – Index of first node on edge
- **node_id2** (*int*) – Index of second node on edge

remove_node(*node_id*)
Remove a graph node

This removes a node from the graph and leverages the unique ID system used internally to avoid having to modify the edges for all nodes in the graph.

Parameters **node_id** (*int*) – Index of the node to be removed.

shortest_path(*node_id1*, *node_id2*)
Find the shortest path between node1 and node2 on the graph

Parameters

- **node_id1** (*int*) – Index of first node
- **node_id2** (*int*) – Index of second node

Returns(list): List of nodes from node_id1 to node_id2 that constitute the shortest possible path in the graph between those two nodes.

class openfermionprojectq.**Node**(*value=None*)
Graph node

These graph nodes may be used to store data or other attributes within the Graph structure.

__init__(*value=None*)
Build a graph node initialized with generic value

openfermionprojectq.**TimeEvolution**(*time*, *hamiltonian*)
Converts the Hamiltonian to an instance of the ProjectQ QubitOperator and then returns an instance of the ProjectQ TimeEvolution gate. This gate is the unitary time evolution propagator: exp(-i * H * t), where H is the Hamiltonian of the system and t is the time. Note that -i factor is stored implicitly.

Example

wavefuction = eng.allocate_qureg(5) hamiltonian = 0.5 * QubitOperator("X0 Z1 Y5") # Apply exp(-i * H * t) to the wavefunction: TimeEvolution(time=2.0, hamiltonian=hamiltonian) | wavefunction

Parameters

- **time** (*float, int*) – time t
- **hamiltonian** (*QubitOperator*) – hamiltonaian H

Returns Instance of ProjectQ TimeEvolution gate.

Raises

- TypeError – If time is not a numeric type and hamiltonian is not a QubitOperator.
- NotHermitianOperatorError – If the input hamiltonian is not hermitian (only real coefficients).

openfermionprojectq.**uccsd_generator**(*single_amplitudes*, *double_amplitudes*, *anti_hermitian=True*)
Create a fermionic operator that is the generator of uccsd.

This a the most straight-forward method to generate UCCSD operators, however it is slightly inefficient. In particular, it parameterizes all possible excitations, so it represents a generalized unitary coupled cluster ansatz, but also does not explicitly enforce the uniqueness in parametrization, so it is redundant. For example there will be a linear dependency in the ansatz of single_amplitudes[i,j] and single_amplitudes[j,i].

Parameters

- **single_amplitudes** (*list or ndarray*) – list of lists with each sublist storing a list of indices followed by single excitation amplitudes i.e. [[[i,j],t_ij], . . .] OR [NxN] array storing single excitation amplitudes corresponding to t[i,j] * (a_i^dagger a_j - H.C.)
- **double_amplitudes** (*list or ndarray*) – list of lists with each sublist storing a list of indices followed by double excitation amplitudes i.e. [[[i,j,k,l],t_ijkl], . . .] OR [NxNxNxN] array storing double excitation amplitudes corresponding to t[i,j,k,l] * (a_i^dagger a_j a_k^dagger a_l - H.C.)
- **anti_hermitian** (*Bool*) – Flag to generate only normal CCSD operator rather than unitary variant, primarily for testing

Returns Anti-hermitian fermion operator that is the generator for the uccsd wavefunction.

Return type uccsd_generator(FermionOperator)

openfermionprojectq.**uccsd_singlet_evolution**(*packed_amplitudes*, *n_qubits*, *n_electrons*, *fermion_transform=<function jordan_wigner>*)

Create a ProjectQ evolution operator for a UCCSD singlet circuit

Parameters

- **packed_amplitudes** (*ndarray*) – Compact array storing the unique single and double excitation amplitudes for a singlet UCCSD operator. The ordering lists unique single excitations before double excitations.
- **n_qubits** (*int*) – Number of spin-orbitals used to represent the system, which also corresponds to number of qubits in a non-compact map.
- **n_electrons** (*int*) – Number of electrons in the physical system
- **fermion_transform** (*openfermion.transform*) – The transformation that defines the mapping from Fermions to QubitOperator.

Returns

The unitary operator that constructs the UCCSD singlet state.

Return type evoution_operator(TimeEvolution)

openfermionprojectq.**uccsd_singlet_generator**(*packed_amplitudes*, *n_qubits*, *n_electrons*, *anti_hermitian=True*)

Create a singlet UCCSD generator for a system with n_electrons

This function generates a FermionOperator for a UCCSD generator designed to act on a single reference state consisting of n_qubits spin orbitals and n_electrons electrons, that is a spin singlet operator, meaning it conserves spin.

Parameters

- **packed_amplitudes** (*list*) – List storing the unique single and double excitation amplitudes for a singlet UCCSD operator. The ordering lists unique single excitations before double excitations.
- **n_qubits** (*int*) – Number of spin-orbitals used to represent the system, which also corresponds to number of qubits in a non-compact map.
- **n_electrons** (*int*) – Number of electrons in the physical system.
- **anti_hermitian** (*Bool*) – Flag to generate only normal CCSD operator rather than unitary variant, primarily for testing

Returns

Generator of the UCCSD operator that builds the UCCSD wavefunction.

Return type generator(FermionOperator)

openfermionprojectq.**uccsd_singlet_paramsize**(*n_qubits*, *n_electrons*)

Determine number of independent amplitudes for singlet UCCSD

Parameters

- **n_qubits** (*int*) – Number of qubits/spin-orbitals in the system
- **n_electrons** (*int*) – Number of electrons in the reference state

Returns Number of independent parameters for singlet UCCSD with a single reference.

openfermionprojectq.**uccsd_trotter_engine**(*compiler_backend=<projectq.backends._sim._simulator.Simulator object>*, *qubit_graph=None*, *opt_size=None*)

Define a ProjectQ compiler engine that is common for use with UCCSD

This defines a ProjectQ compiler engine that decomposes time evolution gates using a first order Trotter decomposition on non-commuting gates down to a base gate decomposition.

Parameters

- **compiler_backend** (*projectq.backend*) – Define the backend on the circuit compiler, so that it may either simulate gates numerically or alternatively print a gate sequence, e.g. using projectq.backends.CommandPrinter()
- **qubit_graph** (Graph) – Graph object specifying connectivity of qubits. The values of the nodes of this unique qubit ids. If None, all-to-all connectivity is assumed.
- **opt_size** (*int*) – Number for ProjectQ local optimizer to determine size of blocks optimized over.

Returns

projectq.cengine that is the compiler engine set up with these rules and decompostions.

Python Module Index

o

Index

Symbols

A

F

G

I

N

O

R

S

T

U