dwave-system Documentation

Release 0.3.2

D-Wave Systems Inc

Jun 22, 2018

Contents

1 Documentation 3

Python Module Index 41

Note: This is an alpha release of this package.

dwave-system is a basic API for easily incorporating the D-Wave system as a sampler in the D-Wave Ocean software stack. It includes DWaveSampler, a `dimod.Sampler` that accepts and passes system parameters such as system identification and authentication down the stack. It also includes several useful composites—layers of pre- and post-processing—that can be used with DWaveSampler to handle minor-embedding, optimize chain strength, etc.

CHAPTER 1

Documentation

1.1 Reference Documentation

Release 0.3.2

Date Jun 22, 2018

1.1.1 Introduction

Samplers

Samplers are processes that sample from low energy states of a problem's objective function. A binary quadratic model (BQM) sampler samples from low energy states in models such as those defined by an Ising equation or a Quadratic Unconstrained Binary Optimization (QUBO) problem and returns an iterable of samples, in order of increasing energy. A dimod sampler provides 'sample_qubo' and 'sample_ising' methods as well as the generic BQM sampler method.

Composites

Samplers can be composed. The composite pattern allows layers of pre- and post-processing to be applied to binary quadratic programs without needing to change the underlying sampler implementation.

We refer to these layers as *composites*. A composed sampler includes at least one sampler and possibly many composites.

D-Wave System Architecture: Chimera

The D-Wave system is Chimera-structured.

The Chimera architecture comprises sets of connected unit cells, each with four horizontal qubits connected to four vertical qubits via couplers (bipartite connectivity). Unit cells are tiled vertically and horizontally with adjacent qubits connected, creating a lattice of sparsely connected qubits. A unit cell is typically rendered as either a cross or a column.

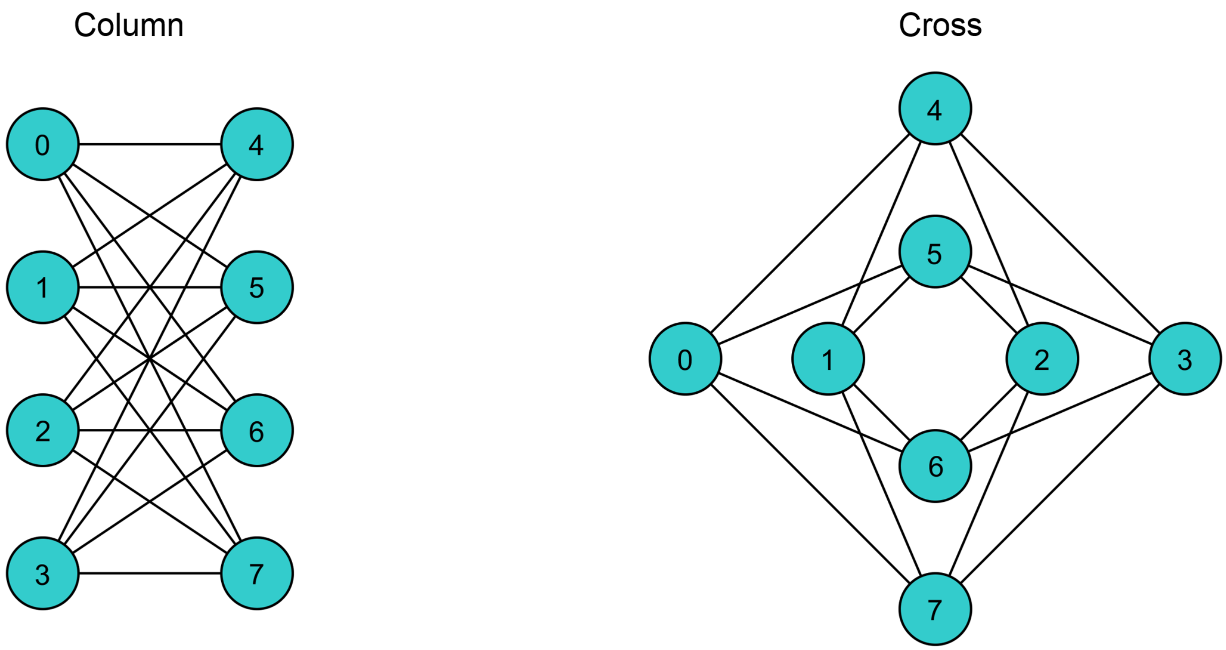

Fig. 1: Chimera unit cell.

Minor-Embedding

To solve an arbitrarily posed binary quadratic problem on a D-Wave system requires mapping, called *minor embedding*, to a Chimera graph that represents the system's quantum processing unit. This preprocessing can be done by a composed sampler consisting of the DWaveSampler and a composite that performs minor-embedding.

1.1.2 Samplers

dwave-system provides dimod samplers for using the D-Wave system.

Release 0.3.2

Date Jun 22, 2018

D-Wave Sampler

A dimod sampler for the D-Wave system.

Class

class DWaveSampler(*config_file=None*, *profile=None*, *endpoint=None*, *token=None*, *solver=None*, *proxy=None*, *permissive_ssl=False*)

A class for using the D-Wave system as a sampler.

Inherits from `dimod.Sampler` and `dimod.Structured`.

Enables quick incorporation of the D-Wave system as a sampler in the D-Wave Ocean software stack. Also enables optional customizing of input parameters to D-Wave Cloud Client (the stack's communication-manager package).

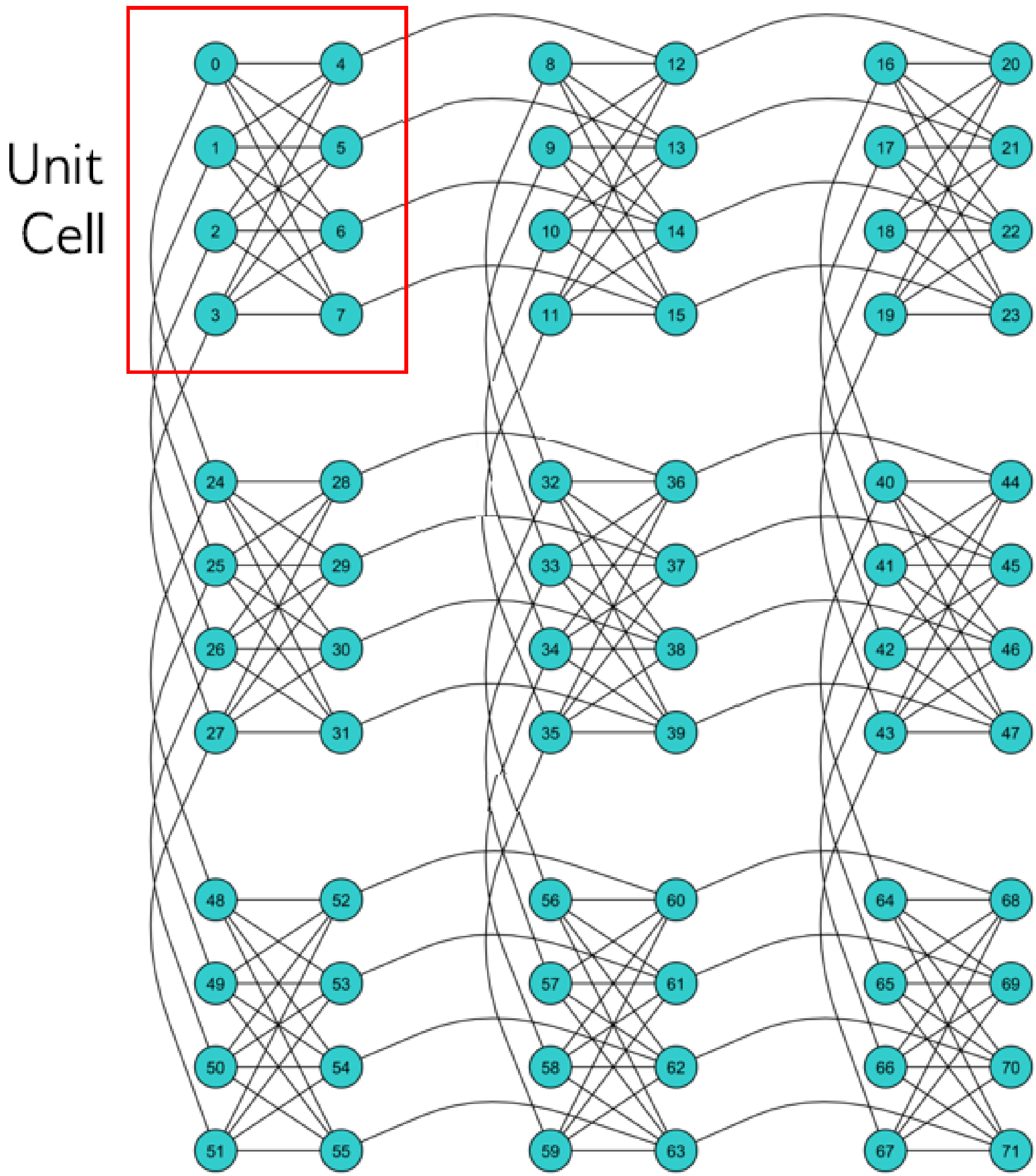

Fig. 2: A 3x3 Chimera graph, denoted C3. Qubits are arranged in 9 unit cells.

Parameters

- **config_file** (*str, optional*) – Path to a D-Wave Cloud Client configuration file that identifies a D-Wave system and provides connection information.
- **profile** (*str, optional*) – Profile to select from a D-Wave Cloud Client configuration file.
- **endpoint** (*str, optional*) – D-Wave API endpoint URL. If specified, used instead of retrieving a value from a D-Wave Cloud Client configuration file.
- **token** (*str, optional*) – Authentication token for the D-Wave API to authenticate the client session. If specified, used instead of retrieving a value from a D-Wave Cloud Client configuration file.
- **solver** (*str, optional*) – Solver (a D-Wave system on which to run submitted problems). If specified, used instead of retrieving a value from a D-Wave Cloud Client configuration file.
- **proxy** (*str, optional*) – Proxy URL to be used for accessing the D-Wave API. If specified, used instead of retrieving a value from a D-Wave Cloud Client configuration file.

Examples

This example creates a `DWaveSampler` based on a fictive user's D-Wave Cloud Client configuration file and submits a simple Ising problem of just two variables that map to qubits 0 and 1 on the example system. (The simplicity of this example obviates the need for an embedding composite—the presence of qubits 0 and 1 on the selected D-Wave system can be verified manually.)

```
>>> # Example configuration file /home/susan/.config/dwave/dwave.conf:
>>> #    [defaults]
>>> #    endpoint = https://url.of.some.dwavesystem.com/sapi
>>> #    client = qpu
>>> #
>>> #    [dw2000]
>>> #    solver = EXAMPLE_2000Q_SYSTEM
>>> #    token = ABC-123456789123456789123456789
>>> from dwave.system.samplers import DWaveSampler
>>> sampler = DWaveSampler()
>>> response = sampler.sample_ising({0: -1, 1: 1}, {})
>>> for sample in response.samples():
...     print(sample)
...
{0: 1, 1: -1}
```

Sampler Properties

`DWaveSampler.properties`	*dict* – D-Wave solver properties as returned by a SAPI query.
`DWaveSampler.parameters`	*dict[str, list]* – D-Wave solver parameters in the form of a dict, where keys are keyword parameters accepted by a SAPI query and values are lists of properties in `DWaveSampler.properties` for each key.

dwave.system.samplers.DWaveSampler.properties

DWaveSampler.**properties**

dict – D-Wave solver properties as returned by a SAPI query.

Solver properties are dependent on the selected D-Wave solver and subject to change; for example, new released features may add properties.

Examples

This example creates a `DWaveSampler` and prints the properties retrieved from a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file.

```
>>> from dwave.system.samplers import DWaveSampler
>>> sampler = DWaveSampler()
>>> sampler.properties
{u'anneal_offset_ranges': [[-0.2197463755538704, 0.03821687759418928],
  [-0.2242514597680286, 0.01718456460967399],
  [-0.20860153999435985, 0.05511969218508182],
# Snipped above response for brevity
```

dwave.system.samplers.DWaveSampler.parameters

DWaveSampler.**parameters**

dict[str, list] – D-Wave solver parameters in the form of a dict, where keys are keyword parameters accepted by a SAPI query and values are lists of properties in `DWaveSampler.properties` for each key.

Solver parameters are dependent on the selected D-Wave solver and subject to change; for example, new released features may add parameters.

Examples

This example creates a `DWaveSampler` and prints the parameters retrieved from a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file.

```
>>> from dwave.system.samplers import DWaveSampler
>>> sampler = DWaveSampler()
>>> sampler.parameters
{u'anneal_offsets': ['parameters'],
u'anneal_schedule': ['parameters'],
u'annealing_time': ['parameters'],
u'answer_mode': ['parameters'],
u'auto_scale': ['parameters'],
# Snipped above response for brevity
```

Structured Sampler Properties

`DWaveSampler.nodelist`	*list* – List of active qubits for the D-Wave solver.
`DWaveSampler.edgelist`	*list* – List of active couplers for the D-Wave solver.

Continued on next page

Table 2 – continued from previous page

DWaveSampler.adjacency	*dict[variable, set]* – Adjacency structure formatted as a dict, where keys are the nodes of the structured sampler and values are sets of all adjacent nodes for each key node.
DWaveSampler.structure	Structure of the structured sampler formatted as a `namedtuple Structure(nodelist, edgelist, adjacency)`, where the 3-tuple values are the `nodelist` and `edgelist` properties and `adjacency()` method.

dwave.system.samplers.DWaveSampler.nodelist

`DWaveSampler.nodelist`

list – List of active qubits for the D-Wave solver.

Examples

This example creates a *DWaveSampler* and prints the active qubits retrieved from a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file.

```
>>> from dwave.system.samplers import DWaveSampler
>>> sampler = DWaveSampler()
>>> sampler.nodelist
[0,
 1,
 2,
 3,
 4,
 5,
# Snipped above response for brevity
```

dwave.system.samplers.DWaveSampler.edgelist

`DWaveSampler.edgelist`

list – List of active couplers for the D-Wave solver.

Examples

This example creates a *DWaveSampler* and prints the active couplers retrieved from a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file.

```
>>> from dwave.system.samplers import DWaveSampler
>>> sampler = DWaveSampler()
>>> sampler.edgelist
[(0, 4),
 (0, 5),
 (0, 6),
 (0, 7),
 (0, 128),
 (1, 4),
```

(continues on next page)

(continued from previous page)

```
 (1, 5),
 (1, 6),
 (1, 7),
 (1, 129),
 (2, 4),
# Snipped above response for brevity
```

dwave.system.samplers.DWaveSampler.adjacency

DWaveSampler.**adjacency**

dict[variable, set] – Adjacency structure formatted as a dict, where keys are the nodes of the structured sampler and values are sets of all adjacent nodes for each key node.

Examples

This example shows the adjacencies for a placeholder structured sampler that samples only from the K4 complete graph, where each of the four nodes connects to all the other nodes.

```
>>> class K4StructuredClass(dimod.Structured):
...     @property
...     def nodelist(self):
...         return [1, 2, 3, 4]
...
...     @property
...     def edgelist(self):
...         return [(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]
>>> K4sampler = K4StructuredClass()
>>> K4sampler.adjacency.keys()
[1, 2, 3, 4]
```

dwave.system.samplers.DWaveSampler.structure

DWaveSampler.**structure**

Structure of the structured sampler formatted as a `namedtuple Structure(nodelist, edgelist, adjacency)`, where the 3-tuple values are the `nodelist` and `edgelist` properties and `adjacency()` method.

Examples

This example shows the structure of a placeholder structured sampler that samples only from the K3 complete graph, where each of the three nodes connects to all the other nodes.

```
>>> class K3StructuredClass(dimod.Structured):
...     @property
...     def nodelist(self):
...         return [1, 2, 3]
...
...     @property
...     def edgelist(self):
```

(continues on next page)

(continued from previous page)

```
...         return [(1, 2), (1, 3), (2, 3)]
>>> K3sampler = K3StructuredClass()
>>> K3sampler.structure.edgelist
[(1, 2), (1, 3), (2, 3)]
```

Methods

DWaveSampler.sample(bqm, **parameters)	Samples from a binary quadratic model using an implemented sample method.
DWaveSampler.sample_ising(h, J, **kwargs)	Sample from the provided Ising model.
DWaveSampler.sample_qubo(Q, **kwargs)	Sample from the provided QUBO.

dwave.system.samplers.DWaveSampler.sample

DWaveSampler.**sample**(*bqm*, ***parameters*)

Samples from a binary quadratic model using an implemented sample method.

Examples

This example implements a placeholder Ising sampler and samples using the mixin binary quadratic model sampler.

```
>>> import dimod
>>> class ImplementIsingSampler(dimod.Sampler):
...     def sample_ising(self, h, J):
...         return dimod.Response.from_dicts([{1: -1, 2: +1}], {'energy': [-1.0]}
↪) # Placeholder
...     @property
...     def properties(self):
...         return self._properties
...     @property
...     def parameters(self):
...         return dict()
...
>>> sampler = ImplementIsingSampler()
>>> model = dimod.BinaryQuadraticModel({0: 1, 1: -1, 2: .5},
...                                    {(0, 1): .5, (1, 2): 1.5},
...                                    1.4,
...                                    dimod.SPIN)
>>> response = sampler.sample(model)
>>> print(response)
[[-1  1]]
```

dwave.system.samplers.DWaveSampler.sample_ising

DWaveSampler.**sample_ising**(*h*, *J*, ***kwargs*)

Sample from the provided Ising model.

Parameters

- **h** (*list/dict*) – Linear biases of the Ising model. If a list, the list's indices are used as variable labels.
- **J** (*dict[(int, int)* – float]): Quadratic biases of the Ising model.
- ****kwargs** – Optional keyword arguments for the sampling method, specified per solver in *DWaveSampler.parameters*

Returns dimod.Response

Examples

This example creates a *DWaveSampler* based on a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file and submits a simple Ising problem of just two variables that map to qubits 0 and 1 on the example system. (The simplicity of this example obviates the need for an embedding composite—the presence of qubits 0 and 1 on the selected D-Wave system can be verified manually.)

```
>>> from dwave.system.samplers import DWaveSampler
>>> sampler = DWaveSampler()
>>> response = sampler.sample_ising({0: -1, 1: 1}, {})
>>> for sample in response.samples():
...     print(sample)
...
{0: 1, 1: -1}
```

dwave.system.samplers.DWaveSampler.sample_qubo

DWaveSampler.**sample_qubo**(*Q*, ***kwargs*)

Sample from the provided QUBO.

Parameters

- **Q** (*dict*) – Coefficients of a quadratic unconstrained binary optimization (QUBO) model.
- ****kwargs** – Optional keyword arguments for the sampling method, specified per solver in *DWaveSampler.parameters*

Returns dimod.Response

Examples

This example creates a *DWaveSampler* based on a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file and submits a simple QUBO problem of just two variables that map to coupled qubits 0 and 4 on the example system. (The simplicity of this example obviates the need for an embedding composite—the presence of qubits 0 and 4, and their coupling, on the selected D-Wave system can be verified manually.)

```
>>> from dwave.system.samplers import DWaveSampler
>>> sampler = DWaveSampler()
>>> Q = {(0, 0): -1, (4, 4): -1, (0, 4): 2}
>>> response = sampler.sample_qubo(Q)
>>> for sample in response.samples():
...     print(sample)
...
{0: 0, 4: 1}
```

1.1.3 Composites

dwave-system provides dimod composites for using the D-Wave system.

Release 0.3.2

Date Jun 22, 2018

EmbeddingComposite

Class

A dimod composite that maps unstructured problems to a structured sampler.

A structured sampler can only solve problems that map to a specific graph: the D-Wave system's architecture is represented by a Chimera graph.

The *EmbeddingComposite* uses the minorminer library to map unstructured problems to a structured sampler such as a D-Wave system.

class EmbeddingComposite(*child_sampler*)

Composite to map unstructured problems to a structured sampler.

Inherits from `dimod.ComposedSampler`.

Enables quick incorporation of the D-Wave system as a sampler in the D-Wave Ocean software stack by handling the minor-embedding of the problem into the D-Wave system's Chimera graph.

Parameters **sampler** (`dimod.Sampler`) – Structured dimod sampler.

Examples

This example uses *EmbeddingComposite* to instantiate a composed sampler that submits a simple Ising problem to a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file. The composed sampler handles minor-embedding of the problem's two generic variables, a and b, to physical qubits on the solver.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import EmbeddingComposite
>>> sampler = EmbeddingComposite(DWaveSampler())
>>> h = {'a': -1., 'b': 2}
>>> J = {('a', 'b'): 1.5}
>>> response = sampler.sample_ising(h, J)
>>> for sample in response.samples():
...     print(sample)
...
{'a': 1, 'b': -1}
```

Sampler Properties

EmbeddingComposite.properties	*dict* – Properties in the form of a dict.
EmbeddingComposite.parameters	*dict[str, list]* – Parameters in the form of a dict.

dwave.system.composites.EmbeddingComposite.properties

`EmbeddingComposite.properties`

dict – Properties in the form of a dict.

For an instantiated composed sampler, contains one key `'child_properties'` that has a copy of the child sampler's properties.

Examples

This example instantiates a composed sampler using a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file and views the solver's properties.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import EmbeddingComposite
>>> sampler = EmbeddingComposite(DWaveSampler())
>>> sampler.properties
{'child_properties': {u'anneal_offset_ranges': [[-0.2197463755538704,
    0.03821687759418928],
   [-0.2242514597680286, 0.01718456460967399],
   [-0.20860153999435985, 0.05511969218508182],
>>> # Snipped above response for brevity
```

dwave.system.composites.EmbeddingComposite.parameters

`EmbeddingComposite.parameters`

dict[str, list] – Parameters in the form of a dict.

For an instantiated composed sampler, keys are the keyword parameters accepted by the child sampler.

Examples

This example instantiates a composed sampler using a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file and views the solver's parameters.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import EmbeddingComposite
>>> sampler = EmbeddingComposite(DWaveSampler())
>>> sampler.parameters
{'anneal_offsets': ['parameters'],
 'anneal_schedule': ['parameters'],
 'annealing_time': ['parameters'],
 'answer_mode': ['parameters'],
 'auto_scale': ['parameters'],
>>> # Snipped above response for brevity
```

Composite Properties

EmbeddingComposite.children	*list* – Children property inherited from `dimod.Composite` class.
EmbeddingComposite.child	First child in `children`.

dwave.system.composites.EmbeddingComposite.children

EmbeddingComposite.**children**

list – Children property inherited from `dimod.Composite` class.

For an instantiated composed sampler, contains the single wrapped structured sampler.

Examples

This example instantiates a composed sampler using a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file and views the solver's parameters.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import EmbeddingComposite
>>> sampler = EmbeddingComposite(DWaveSampler())
>>> sampler.children
[<dwave.system.samplers.dwave_sampler.DWaveSampler at 0x7f45b20a8d50>]
```

dwave.system.composites.EmbeddingComposite.child

EmbeddingComposite.**child**

First child in `children`.

Examples

This example pseudocode defines a composed sampler that uses the first supported sampler in a composite's list of samplers on a binary quadratic model.

```
class MyComposedSampler(Sampler, Composite):

    children = None
    parameters = None
    properties = None

    def __init__(self, child):
        self.children = [child]

        self.parameters = child.parameters.copy()  # propagate parameters
        self.parameters['my_additional_parameter'] = []

        self.properties = child.properties.copy()  # propagate properties

    # Implementation of the composite's functionality
    def sample(self, bqm, my_additional_parameter, **kwargs):
        # Overwrite the abstract sample method.
        # Additional parameters must have defaults
```

(continues on next page)

(continued from previous page)

```
        # Samples are obtained from the sampler by using the `child` property:
        # response = self.child.sample(bqm, **kwargs)

        raise NotImplementedError
```

Methods

EmbeddingComposite.sample(bqm[, chain_strength])	Sample from the provided binary quadratic model.
EmbeddingComposite.sample_ising(h, J, …)	Samples from an Ising model using an implemented sample method.
EmbeddingComposite.sample_qubo(Q, **parameters)	Samples from a QUBO using an implemented sample method.

dwave.system.composites.EmbeddingComposite.sample

EmbeddingComposite.**sample**(*bqm*, *chain_strength=1.0*, ***parameters*)

Sample from the provided binary quadratic model.

Parameters

- **bqm** (dimod.BinaryQuadraticModel) – Binary quadratic model to be sampled from.
- **chain_strength** (*float, optional, default=1.0*) – Magnitude of the quadratic bias (in SPIN-space) applied between variables to create chains. Note that the energy penalty of chain breaks is 2 * *chain_strength*.
- ****parameters** – Parameters for the sampling method, specified by the child sampler.

Returns dimod.Response

Examples

This example uses *EmbeddingComposite* to instantiate a composed sampler that submits an unstructured Ising problem to a D-Wave solver, selected by the user's default D-Wave Cloud Client **configuration_** file, while minor-embedding the problem's variables to physical qubits on the solver.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import EmbeddingComposite
>>> import dimod
>>> sampler = EmbeddingComposite(DWaveSampler())
>>> h = {1: 1, 2: 2, 3: 3, 4: 4}
>>> J = {(1, 2): 12, (1, 3): 13, (1, 4): 14,
...      (2, 3): 23, (2, 4): 24,
...      (3, 4): 34}
>>> bqm = dimod.BinaryQuadraticModel.from_ising(h, J)
>>> response = sampler.sample(bqm)
>>> for sample in response.samples():
...     print(sample)
...
{1: -1, 2: 1, 3: 1, 4: -1}
```

dwave.system.composites.EmbeddingComposite.sample_ising

EmbeddingComposite.**sample_ising**(*h*, *J*, ***parameters*)

Samples from an Ising model using an implemented sample method.

Examples

This example implements a placeholder QUBO sampler and samples using the mixin Ising sampler.

```
>>> import dimod
>>> class ImplementQuboSampler(dimod.Sampler):
...     def sample_qubo(self, Q):
...         return dimod.Response.from_dicts([{1: -1, 2: +1}], {'energy': [-1.0]}
↪) # Placeholder
...     @property
...     def properties(self):
...         return self._properties
...     @property
...     def parameters(self):
...         return dict()
...
>>> sampler = ImplementQuboSampler()
>>> h = {1: 0.5, 2: -1, 3: -0.75}
>>> J = {}
>>> response = sampler.sample_ising(h, J)
>>> print(response)
[[-1  1]]
```

dwave.system.composites.EmbeddingComposite.sample_qubo

EmbeddingComposite.**sample_qubo**(*Q*, ***parameters*)

Samples from a QUBO using an implemented sample method.

Examples

This example implements a placeholder Ising sampler and samples using the mixin QUBO sampler.

```
>>> import dimod
>>> class ImplementIsingSampler(dimod.Sampler):
...     def sample_ising(self, h, J):
...         return dimod.Response.from_dicts([{1: -1, 2: +1}], {'energy': [-1.0]}
↪) # Placeholder
...     @property
...     def properties(self):
...         return self._properties
...     @property
...     def parameters(self):
...         return dict()
...
>>> sampler = ImplementIsingSampler()
>>> Q = {(0, 0): -0.5, (0, 1): 1, (1, 1): -0.75}
>>> response = sampler.sample_qubo(Q)
```

(continues on next page)

(continued from previous page)

```
>>> print(response)
[[0 1]]
```

FixedEmbeddingComposite

Class

class FixedEmbeddingComposite(*child_sampler*, *embedding*)

Composite to alter the structure of a child sampler via an embedding.

Inherits from `dimod.ComposedSampler` and `dimod.Structured`.

Parameters

- **sampler** (*dimod.Sampler*) – Structured dimod sampler.
- **embedding** (*dict[hashable, iterable]*) – Mapping from a source graph to the specified sampler's graph (the target graph).

Examples

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import FixedEmbeddingComposite
...
>>> sampler = FixedEmbeddingComposite(DWaveSampler(), {'a': [0, 4], 'b': [1, 5],
↪'c': [2, 6]})
>>> sampler.nodelist
['a', 'b', 'c']
>>> sampler.edgelist
[('a', 'b'), ('a', 'c'), ('b', 'c')]
>>> resp = sampler.sample_ising({'a': .5, 'c': 0}, {('a', 'c'): -1})
```

Sampler Properties

FixedEmbeddingComposite.properties	*dict* – Properties in the form of a dict.
FixedEmbeddingComposite.parameters	*dict[str, list]* – Parameters in the form of a dict.

dwave.system.composites.FixedEmbeddingComposite.properties

FixedEmbeddingComposite.**properties = None**

dict – Properties in the form of a dict.

For an instantiated composed sampler, `'child_properties'` has a copy of the child sampler's properties and `'embedding'` contains the fixed embedding.

dwave.system.composites.FixedEmbeddingComposite.parameters

FixedEmbeddingComposite.**parameters = None**

dict[str, list] – Parameters in the form of a dict.

The same as the child sampler with the addition of 'chain_strength'

Composite Properties

`FixedEmbeddingComposite.children`	*list* – List containing the wrapped sampler.
`FixedEmbeddingComposite.child`	First child in `children`.

dwave.system.composites.FixedEmbeddingComposite.children

`FixedEmbeddingComposite.children = None`

list – List containing the wrapped sampler.

dwave.system.composites.FixedEmbeddingComposite.child

`FixedEmbeddingComposite.child`

First child in `children`.

Examples

This example pseudocode defines a composed sampler that uses the first supported sampler in a composite's list of samplers on a binary quadratic model.

```
class MyComposedSampler(Sampler, Composite):

    children = None
    parameters = None
    properties = None

    def __init__(self, child):
        self.children = [child]

        self.parameters = child.parameters.copy()  # propagate parameters
        self.parameters['my_additional_parameter'] = []

        self.properties = child.properties.copy()  # propagate properties

    # Implementation of the composite's functionality
    def sample(self, bqm, my_additional_parameter, **kwargs):
        # Overwrite the abstract sample method.
        # Additional parameters must have defaults

        # Samples are obtained from the sampler by using the `child` property:
        # response = self.child.sample(bqm, **kwargs)

        raise NotImplementedError
```

Structured Sampler Properties

FixedEmbeddingComposite.nodelist	*list* – Nodes available to the composed sampler.
FixedEmbeddingComposite.edgelist	*list* – Edges available to the composed sampler.
FixedEmbeddingComposite.adjacency	*dict[variable, set]* – Adjacency structure for the composed sampler.
FixedEmbeddingComposite.structure	Structure of the structured sampler formatted as a `namedtuple` `Structure(nodelist, edgelist, adjacency)`, where the 3-tuple values are the `nodelist` and `edgelist` properties and `adjacency()` method.

dwave.system.composites.FixedEmbeddingComposite.nodelist

FixedEmbeddingComposite.**nodelist = None**

list – Nodes available to the composed sampler.

dwave.system.composites.FixedEmbeddingComposite.edgelist

FixedEmbeddingComposite.**edgelist = None**

list – Edges available to the composed sampler.

dwave.system.composites.FixedEmbeddingComposite.adjacency

FixedEmbeddingComposite.**adjacency = None**

dict[variable, set] – Adjacency structure for the composed sampler.

dwave.system.composites.FixedEmbeddingComposite.structure

FixedEmbeddingComposite.**structure**

Structure of the structured sampler formatted as a `namedtuple` `Structure(nodelist, edgelist, adjacency)`, where the 3-tuple values are the `nodelist` and `edgelist` properties and `adjacency()` method.

Examples

This example shows the structure of a placeholder structured sampler that samples only from the K3 complete graph, where each of the three nodes connects to all the other nodes.

```
>>> class K3StructuredClass(dimod.Structured):
...     @property
...     def nodelist(self):
...         return [1, 2, 3]
...
...     @property
...     def edgelist(self):
...         return [(1, 2), (1, 3), (2, 3)]
>>> K3sampler = K3StructuredClass()
>>> K3sampler.structure.edgelist
[(1, 2), (1, 3), (2, 3)]
```

Methods

FixedEmbeddingComposite.sample(bqm, **kwargs)	Sample from the provided binary quadratic model.
FixedEmbeddingComposite.sample_ising(h, J, ...)	Samples from an Ising model using an implemented sample method.
FixedEmbeddingComposite.sample_qubo(Q, ...)	Samples from a QUBO using an implemented sample method.

dwave.system.composites.FixedEmbeddingComposite.sample

FixedEmbeddingComposite.**sample**(*bqm*, ***kwargs*)

Sample from the provided binary quadratic model.

Parameters

- **bqm** (dimod.BinaryQuadraticModel) – Binary quadratic model to be sampled from.
- **chain_strength** (*float*, *optional*, *default=1.0*) – Magnitude of the quadratic bias (in SPIN-space) applied between variables to create chains. Note that the energy penalty of chain breaks is 2 * *chain_strength*.
- ****parameters** – Parameters for the sampling method, specified by the child sampler.

Returns dimod.Response

Examples

This example uses *FixedEmbeddingComposite* to instantiate a composed sampler that submits an unstructured Ising problem to a D-Wave solver, selected by the user's default D-Wave Cloud Client **configuration_** file, while minor-embedding the problem's variables to physical qubits on the solver.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import FixedEmbeddingComposite
>>> import dimod
>>> sampler = FixedEmbeddingComposite(DWaveSampler(), {'a': [0, 4], 'b': [1, 5],
↪'c': [2, 6]})
>>> resp = sampler.sample_ising({'a': .5, 'c': 0}, {('a', 'c'): -1})
```

dwave.system.composites.FixedEmbeddingComposite.sample_ising

FixedEmbeddingComposite.**sample_ising**(*h*, *J*, ***parameters*)

Samples from an Ising model using an implemented sample method.

Examples

This example implements a placeholder QUBO sampler and samples using the mixin Ising sampler.

```
>>> import dimod
>>> class ImplementQuboSampler(dimod.Sampler):
...     def sample_qubo(self, Q):
...         return dimod.Response.from_dicts([{1: -1, 2: +1}], {'energy': [-1.0]}
↪) # Placeholder
```

(continues on next page)

(continued from previous page)

```
...     @property
...     def properties(self):
...         return self._properties
...     @property
...     def parameters(self):
...         return dict()
...
>>> sampler = ImplementQuboSampler()
>>> h = {1: 0.5, 2: -1, 3: -0.75}
>>> J = {}
>>> response = sampler.sample_ising(h, J)
>>> print(response)
[[-1  1]]
```

dwave.system.composites.FixedEmbeddingComposite.sample_qubo

FixedEmbeddingComposite.**sample_qubo**(*Q*, ***parameters*)

Samples from a QUBO using an implemented sample method.

Examples

This example implements a placeholder Ising sampler and samples using the mixin QUBO sampler.

```
>>> import dimod
>>> class ImplementIsingSampler(dimod.Sampler):
...     def sample_ising(self, h, J):
...         return dimod.Response.from_dicts([{1: -1, 2: +1}], {'energy': [-1.0]}
↪) # Placeholder
...     @property
...     def properties(self):
...         return self._properties
...     @property
...     def parameters(self):
...         return dict()
...
>>> sampler = ImplementIsingSampler()
>>> Q = {(0, 0): -0.5, (0, 1): 1, (1, 1): -0.75}
>>> response = sampler.sample_qubo(Q)
>>> print(response)
[[0 1]]
```

TilingComposite

Class

A dimod composite that tiles small problems multiple times to a Chimera-structured sampler.

The *TilingComposite* takes a problem that can fit on a small Chimera graph and replicates it across a larger Chimera graph to obtain samples from multiple areas of the solver in one call. For example, a 2x2 Chimera lattice could be tiled 64 times (8x8) on a fully-yielded D-Wave 2000Q system (16x16).

class TilingComposite(*sampler*, *sub_m*, *sub_n*, *t=4*)

Composite to tile a small problem across a Chimera-structured sampler.

Inherits from `dimod.Sampler`, `dimod.Composite`, and `dimod.Structured`.

Enables parallel sampling for small problems (problems that are minor-embeddable in a small part of a D-Wave solver's Chimera graph).

The notation *CN* refers to a Chimera graph consisting of an NxN grid of unit cells. Each Chimera unit cell is itself a bipartite graph with shores of size t. The D-Wave 2000Q QPU supports a C16 Chimera graph: its 2048 qubits are logically mapped into a 16x16 matrix of unit cell of 8 qubits (t=4).

A problem that can be minor-embedded in a single unit cell, for example, can therefore be tiled across the unit cells of a D-Wave 2000Q as 16x16 duplicates. This enables sampling 256 solutions in a single call.

Parameters

- **sampler** (`dimod.Sampler`) – Structured dimod sampler to be wrapped.
- **sub_m** (*int*) – Number of rows of Chimera unit cells for minor-embedding the problem once.
- **sub_n** (*int*) – Number of columns of Chimera unit cells for minor-embedding the problem once.
- **t** (*int, optional, default=4*) – Size of the shore within each Chimera unit cell.

Examples

This example instantiates a composed sampler using composite *TilingComposite* to tile a QUBO problem on a D-Wave solver, embedding it with composite *EmbeddingComposite* and selecting the D-Wave solver with the user's default D-Wave Cloud Client configuration file. The two-variable QUBO represents a logical NOT gate (two nodes with biases of -1 that are coupled with strength 2) and is easily minor-embedded in a single Chimera cell (it needs only any two coupled qubits) and so can be tiled multiple times across a D-Wave solver for parallel solution (the two nodes should typically have opposite values).

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import EmbeddingComposite
>>> from dwave.system.composites import TilingComposite
>>> sampler = EmbeddingComposite(TilingComposite(DWaveSampler(), 1, 1, 4))
>>> Q = {(1, 1): -1, (1, 2): 2, (2, 1): 0, (2, 2): -1}
>>> response = sampler.sample_qubo(Q)
>>> for sample in response.samples():
...     print(sample)
...
{1: 0, 2: 1}
{1: 1, 2: 0}
{1: 1, 2: 0}
{1: 1, 2: 0}
{1: 0, 2: 1}
{1: 0, 2: 1}
{1: 1, 2: 0}
{1: 0, 2: 1}
{1: 1, 2: 0}
>>> # Snipped above response for brevity
```

Sampler Properties

`TilingComposite.properties`	*dict* – Properties in the form of a dict.
`TilingComposite.parameters`	*dict[str, list]* – Parameters in the form of a dict.

dwave.system.composites.TilingComposite.properties

`TilingComposite.`**`properties = None`**

dict – Properties in the form of a dict.

For an instantiated composed sampler, contains one key `'child_properties'` that has a copy of the child sampler's properties.

Examples

This example instantiates a `TilingComposite` sampler using a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file and views the solver's properties.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import TilingComposite
>>> sampler_tile = TilingComposite(DWaveSampler(), 1, 1, 4)
>>> sampler_tile.properties
{'child_properties': {u'anneal_offset_ranges': [[-0.2197463755538704,
    0.03821687759418928],
   [-0.2242514597680286, 0.01718456460967399],
   [-0.20860153999435985, 0.05511969218508182],
   [-0.2108920134230625, 0.056392603743884134],
   [-0.21788292874621265, 0.03360435584845211],
   [-0.21700680373359477, 0.005297355417068621],
>>> # Snipped above response for brevity
```

dwave.system.composites.TilingComposite.parameters

`TilingComposite.`**`parameters = None`**

dict[str, list] – Parameters in the form of a dict.

For an instantiated composed sampler, keys are the keyword parameters accepted by the child sampler.

Examples

This example instantiates a `TilingComposite` sampler using a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file and views the solver's parameters.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import TilingComposite
>>> sampler_tile = TilingComposite(DWaveSampler(), 1, 1, 4)
>>> sampler_tile.parameters
{u'anneal_offsets': ['parameters'],
 u'anneal_schedule': ['parameters'],
 u'annealing_time': ['parameters'],
 u'answer_mode': ['parameters'],
```

(continues on next page)

(continued from previous page)

```
u'auto_scale': ['parameters'],
>>> # Snipped above response for brevity
```

Composite Properties

TilingComposite.children	*list* – The single wrapped structured sampler.
TilingComposite.child	First child in `children`.

dwave.system.composites.TilingComposite.children

TilingComposite.**children = None**

list – The single wrapped structured sampler.

dwave.system.composites.TilingComposite.child

TilingComposite.**child**

First child in `children`.

Examples

This example pseudocode defines a composed sampler that uses the first supported sampler in a composite's list of samplers on a binary quadratic model.

```
class MyComposedSampler(Sampler, Composite):

    children = None
    parameters = None
    properties = None

    def __init__(self, child):
        self.children = [child]

        self.parameters = child.parameters.copy()  # propagate parameters
        self.parameters['my_additional_parameter'] = []

        self.properties = child.properties.copy()  # propagate properties

    # Implementation of the composite's functionality
    def sample(self, bqm, my_additional_parameter, **kwargs):
        # Overwrite the abstract sample method.
        # Additional parameters must have defaults

        # Samples are obtained from the sampler by using the `child` property:
        # response = self.child.sample(bqm, **kwargs)

        raise NotImplementedError
```

Structured Sampler Properties

TilingComposite.nodelist	*list* – List of active qubits for the structured solver.
TilingComposite.edgelist	*list* – List of active couplers for the D-Wave solver.
TilingComposite.adjacency	*dict[variable, set]* – Adjacency structure formatted as a dict, where keys are the nodes of the structured sampler and values are sets of all adjacent nodes for each key node.
TilingComposite.structure	Structure of the structured sampler formatted as a `namedtuple Structure(nodelist, edgelist, adjacency)`, where the 3-tuple values are the `nodelist` and `edgelist` properties and `adjacency()` method.

dwave.system.composites.TilingComposite.nodelist

`TilingComposite.`**`nodelist = None`**

list – List of active qubits for the structured solver.

Examples

This example creates a *TilingComposite* for a problem that requires a 2x1 Chimera lattice to solve with a `DWaveSampler` as the sampler. It prints the active qubits retrieved from a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import TilingComposite
>>> sampler_tile = TilingComposite(DWaveSampler(), 2, 1, 4)
>>> sampler_tile.nodelist
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

dwave.system.composites.TilingComposite.edgelist

`TilingComposite.`**`edgelist = None`**

list – List of active couplers for the D-Wave solver.

Examples

This example creates a *TilingComposite* for a problem that requires a 1x2 Chimera lattice to solve with a `DWaveSampler` as the sampler. It prints the active couplers retrieved from a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import TilingComposite
>>> sampler_tile = TilingComposite(DWaveSampler(), 1, 2, 4)
>>> sampler_tile.edgelist
[[0, 4],
[0, 5],
[0, 6],
```

(continues on next page)

(continued from previous page)

```
[0, 7],
[1, 4],
[1, 5],
[1, 6],
[1, 7],
[2, 4],
[2, 5],
[2, 6],
[2, 7],
[3, 4],
[3, 5],
[3, 6],
[3, 7],
[4, 12],
[5, 13],
[6, 14],
[7, 15],
[8, 12],
[8, 13],
[8, 14],
[8, 15],
[9, 12],
[9, 13],
[9, 14],
[9, 15],
[10, 12],
[10, 13],
[10, 14],
[10, 15],
[11, 12],
[11, 13],
[11, 14],
[11, 15]]
```

dwave.system.composites.TilingComposite.adjacency

TilingComposite.**adjacency**

dict[variable, set] – Adjacency structure formatted as a dict, where keys are the nodes of the structured sampler and values are sets of all adjacent nodes for each key node.

Examples

This example shows the adjacencies for a placeholder structured sampler that samples only from the K4 complete graph, where each of the four nodes connects to all the other nodes.

```
>>> class K4StructuredClass(dimod.Structured):
...     @property
...     def nodelist(self):
...         return [1, 2, 3, 4]
...
...     @property
...     def edgelist(self):
...         return [(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]
```

(continues on next page)

(continued from previous page)

```
>>> K4sampler = K4StructuredClass()
>>> K4sampler.adjacency.keys()
[1, 2, 3, 4]
```

dwave.system.composites.TilingComposite.structure

`TilingComposite.structure`
Structure of the structured sampler formatted as a `namedtuple Structure(nodelist, edgelist, adjacency)`, where the 3-tuple values are the `nodelist` and `edgelist` properties and `adjacency()` method.

Examples

This example shows the structure of a placeholder structured sampler that samples only from the K3 complete graph, where each of the three nodes connects to all the other nodes.

```
>>> class K3StructuredClass(dimod.Structured):
...     @property
...     def nodelist(self):
...         return [1, 2, 3]
...
...     @property
...     def edgelist(self):
...         return [(1, 2), (1, 3), (2, 3)]
>>> K3sampler = K3StructuredClass()
>>> K3sampler.structure.edgelist
[(1, 2), (1, 3), (2, 3)]
```

Methods

TilingComposite.sample(bqm, **kwargs)	Sample from the provided binary quadratic model
TilingComposite.sample_ising(h, J, **parameters)	Samples from an Ising model using an implemented sample method.
TilingComposite.sample_qubo(Q, **parameters)	Samples from a QUBO using an implemented sample method.

dwave.system.composites.TilingComposite.sample

`TilingComposite.sample`(*bqm*, ***kwargs*)
Sample from the provided binary quadratic model

Parameters

- **bqm** (`dimod.BinaryQuadraticModel`) – Binary quadratic model to be sampled from.
- ****kwargs** – Optional keyword arguments for the sampling method, specified per solver.

Returns `dimod.Response`

Examples

This example uses `TilingComposite` to instantiate a composed sampler that submits a simple Ising problem of just two variables that map to qubits 0 and 1 on the D-Wave solver selected by the user's default D-Wave Cloud Client configuration file. (The simplicity of this example obviates the need for an embedding composite.) Because the problem fits in a single Chimera unit cell, it is tiled across the solver's entire Chimera graph, resulting in multiple samples.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import EmbeddingComposite, TilingComposite
>>> sampler = TilingComposite(DWaveSampler(), 1, 1, 4)
>>> response = sampler.sample_ising({0: -1, 1: 1}, {})
>>> for sample in response.samples():
...     print(sample)
...
{0: 1, 1: -1}
{0: 1, 1: -1}
{0: 1, 1: -1}
{0: 1, 1: -1}
{0: 1, 1: -1}
{0: 1, 1: -1}
{0: 1, 1: -1}
{0: 1, 1: -1}
>>> # Snipped above response for brevity
```

dwave.system.composites.TilingComposite.sample_ising

`TilingComposite.`**`sample_ising`**(*h*, *J*, ***parameters*)

Samples from an Ising model using an implemented sample method.

Examples

This example implements a placeholder QUBO sampler and samples using the mixin Ising sampler.

```
>>> import dimod
>>> class ImplementQuboSampler(dimod.Sampler):
...     def sample_qubo(self, Q):
...         return dimod.Response.from_dicts([{1: -1, 2: +1}], {'energy': [-1.0]}
↪) # Placeholder
...     @property
...     def properties(self):
...         return self._properties
...     @property
...     def parameters(self):
...         return dict()
...
>>> sampler = ImplementQuboSampler()
>>> h = {1: 0.5, 2: -1, 3: -0.75}
>>> J = {}
>>> response = sampler.sample_ising(h, J)
>>> print(response)
[[-1  1]]
```

dwave.system.composites.TilingComposite.sample_qubo

`TilingComposite.sample_qubo(Q, **parameters)`

Samples from a QUBO using an implemented sample method.

Examples

This example implements a placeholder Ising sampler and samples using the mixin QUBO sampler.

```
>>> import dimod
>>> class ImplementIsingSampler(dimod.Sampler):
...     def sample_ising(self, h, J):
...         return dimod.Response.from_dicts([{1: -1, 2: +1}], {'energy': [-1.0]}
↪) # Placeholder
...     @property
...     def properties(self):
...         return self._properties
...     @property
...     def parameters(self):
...         return dict()
...
>>> sampler = ImplementIsingSampler()
>>> Q = {(0, 0): -0.5, (0, 1): 1, (1, 1): -0.75}
>>> response = sampler.sample_qubo(Q)
>>> print(response)
[[0 1]]
```

VirtualGraphComposite

Class

A dimod composite that uses the D-Wave virtual graph feature for improved minor-embedding.

D-Wave *virtual graphs* simplify the process of minor-embedding by enabling you to more easily create, optimize, use, and reuse an embedding for a given working graph. When you submit an embedding and specify a chain strength using these tools, they automatically calibrate the qubits in a chain to compensate for the effects of biases that may be introduced as a result of strong couplings.

class VirtualGraphComposite(*sampler*, *embedding*, *chain_strength=None*, *flux_biases=None*, *flux_bias_num_reads=1000*, *flux_bias_max_age=3600*)

Composite to use the D-Wave virtual graph feature for minor-embedding.

Inherits from `dimod.ComposedSampler` and `dimod.Structured`.

Calibrates qubits in chains to compensate for the effects of biases and enables easy creation, optimization, use, and reuse of an embedding for a given working graph.

Parameters

- **sampler** (*DWaveSampler*) – A dimod `dimod.Sampler`. Typically a *DWaveSampler* or derived composite sampler; other samplers may not work or make sense with this composite layer.
- **embedding** (*dict[hashable, iterable]*) – Mapping from a source graph to the specified sampler's graph (the target graph).

- **chain_strength** (*float, optional, default=None*) – Desired chain coupling strength. This is the magnitude of couplings between qubits in a chain. If None, uses the maximum available as returned by a SAPI query to the D-Wave solver.
- **flux_biases** (*list/False/None, optional, default=None*) – Per-qubit flux bias offsets in the form of a list of lists, where each sublist is of length 2 and specifies a variable and the flux bias offset associated with that variable. Qubits in a chain with strong negative J values experience a J-induced bias; this parameter compensates by recalibrating to remove that bias. If False, no flux bias is applied or calculated. If None, flux biases are pulled from the database or calculated empirically.
- **flux_bias_num_reads** (*int, optional, default=1000*) – Number of samples to collect per flux bias value.
- **flux_bias_max_age** (*int, optional, default=3600*) – Maximum age (in seconds) allowed for a previously calculated flux bias offset to be considered valid.

Examples

This example uses `VirtualGraphComposite` to instantiate a composed sampler that submits a QUBO problem to a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file. The problem represents a logical AND gate using penalty function $P = xy - 2(x + y)z + 3z$, where variables x and y are the gate's inputs and z the output. This simple three-variable problem is manually minor-embedded to a single Chimera unit cell: variables x and y are represented by qubits 1 and 5, respectively, and z by a two-qubit chain consisting of qubits 0 and 4. The chain strength is set to the maximum allowed found from querying the solver's extended J range. In this example, the ten returned samples all represent valid states of the AND gate.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import VirtualGraphComposite
>>> embedding = {'x': {1}, 'y': {5}, 'z': {0, 4}}
>>> DWaveSampler().properties['extended_j_range']
[-2.0, 1.0]
>>> sampler = VirtualGraphComposite(DWaveSampler(), embedding, chain_strength=2)
>>> Q = {('x', 'y'): 1, ('x', 'z'): -2, ('y', 'z'): -2, ('z', 'z'): 3}
>>> response = sampler.sample_qubo(Q, num_reads=10)
>>> for sample in response.samples():
...     print(sample)
...
{'y': 0, 'x': 1, 'z': 0}
{'y': 1, 'x': 0, 'z': 0}
{'y': 1, 'x': 0, 'z': 0}
{'y': 1, 'x': 1, 'z': 1}
{'y': 0, 'x': 1, 'z': 0}
{'y': 1, 'x': 0, 'z': 0}
{'y': 0, 'x': 1, 'z': 0}
{'y': 0, 'x': 1, 'z': 0}
{'y': 0, 'x': 0, 'z': 0}
{'y': 1, 'x': 0, 'z': 0}
```

Sampler Properties

`VirtualGraphComposite.properties`	*dict* – Properties in the form of a dict.
`VirtualGraphComposite.parameters`	*dict[str, list]* – Parameters in the form of a dict.

dwave.system.composites.VirtualGraphComposite.properties

`VirtualGraphComposite.properties = None`

dict – Properties in the form of a dict.

For an instantiated composed sampler, contains one key `'child_properties'` that has a copy of the child sampler's properties.

Examples

This example uses `VirtualGraphComposite` to instantiate a composed sampler that uses a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file and views the composed sampler's properties.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import VirtualGraphComposite
>>> embedding = {'x': {1}, 'y': {5}, 'z': {0, 4}}
>>> sampler = VirtualGraphComposite(DWaveSampler(), embedding)
>>> sampler.properties
{'child_properties': {u'anneal_offset_ranges': [[-0.2197463755538704,
    0.03821687759418928],
   [-0.2242514597680286, 0.01718456460967399],
   [-0.20860153999435985, 0.05511969218508182],
   [-0.2108920134230625, 0.056392603743884134],
>>>  # Snipped above response for brevity
```

dwave.system.composites.VirtualGraphComposite.parameters

`VirtualGraphComposite.parameters = None`

dict[str, list] – Parameters in the form of a dict.

For an instantiated composed sampler, keys are the keyword parameters accepted by the child sampler with an additional parameter, 'apply_flux_bias_offsets'.

Examples

This example uses `VirtualGraphComposite` to instantiate a composed sampler that uses a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file and views the composed sampler's parameters.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import VirtualGraphComposite
>>> embedding = {'x': {1}, 'y': {5}, 'z': {0, 4}}
>>> sampler = VirtualGraphComposite(DWaveSampler(), embedding)
>>> sampler.parameters
{u'anneal_offsets': ['parameters'],
 u'anneal_schedule': ['parameters'],
 u'annealing_time': ['parameters'],
 u'answer_mode': ['parameters'],
 'apply_flux_bias_offsets': [],
 u'auto_scale': ['parameters'],
>>>  # Snipped above response for brevity
```

Composite Properties

`VirtualGraphComposite.children`	*list* – List containing the FixedEmbeddingComposite-wrapped sampler.
`VirtualGraphComposite.child`	First child in `children`.

dwave.system.composites.VirtualGraphComposite.children

VirtualGraphComposite.**children = None**

list – List containing the FixedEmbeddingComposite-wrapped sampler.

dwave.system.composites.VirtualGraphComposite.child

VirtualGraphComposite.**child**

First child in `children`.

Examples

This example pseudocode defines a composed sampler that uses the first supported sampler in a composite's list of samplers on a binary quadratic model.

```
class MyComposedSampler(Sampler, Composite):

    children = None
    parameters = None
    properties = None

    def __init__(self, child):
        self.children = [child]

        self.parameters = child.parameters.copy()  # propagate parameters
        self.parameters['my_additional_parameter'] = []

        self.properties = child.properties.copy()  # propagate properties

    # Implementation of the composite's functionality
    def sample(self, bqm, my_additional_parameter, **kwargs):
        # Overwrite the abstract sample method.
        # Additional parameters must have defaults

        # Samples are obtained from the sampler by using the `child` property:
        # response = self.child.sample(bqm, **kwargs)

        raise NotImplementedError
```

Structured Sampler Properties

`VirtualGraphComposite.nodelist`	*list* – Nodes available to the composed sampler.
`VirtualGraphComposite.edgelist`	*list* – Edges available to the composed sampler.

Continued on next page

Table 17 – continued from previous page

`VirtualGraphComposite.adjacency`	*dict[variable, set]* – Adjacency structure for the composed sampler.
`VirtualGraphComposite.structure`	Structure of the structured sampler formatted as a `namedtuple` `Structure(nodelist, edgelist, adjacency)`, where the 3-tuple values are the `nodelist` and `edgelist` properties and `adjacency()` method.

dwave.system.composites.VirtualGraphComposite.nodelist

`VirtualGraphComposite.`**`nodelist = None`**

list – Nodes available to the composed sampler.

Examples

This example uses `VirtualGraphComposite` to instantiate a composed sampler that uses a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file. Because qubits 0, 1, 4, 5 are active on the selected D-Wave solver, the three nodes, x, y, and z, specified by the embedding, are all available to problems using this composed sampler.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import VirtualGraphComposite
>>> embedding = {'x': {1}, 'y': {5}, 'z': {0, 4}}
>>> sampler = VirtualGraphComposite(DWaveSampler(), embedding)
>>> sampler.nodelist
['x', 'y', 'z']
```

dwave.system.composites.VirtualGraphComposite.edgelist

`VirtualGraphComposite.`**`edgelist = None`**

list – Edges available to the composed sampler.

Examples

This example uses `VirtualGraphComposite` to instantiate a composed sampler that uses a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file. Because qubits 0, 5, and coupled qubits {0, 4} are all coupled on the selected D-Wave solver, edges between three nodes, x, y, and z, as specified by the embedding, are available to problems using this composed sampler. However, qubit 8 is in an adjacent unit cell on the D-Wave solver and not directly connected to the other four qubits, so node *a* does not share an edge with any other nodes.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import VirtualGraphComposite
>>> embedding = {'x': {1}, 'y': {5}, 'z': {0, 4}, 'a': {8}}
>>> sampler = VirtualGraphComposite(DWaveSampler(), embedding)
>>> sampler.edgelist
[('x', 'y'), ('x', 'z'), ('y', 'z')]
```

dwave.system.composites.VirtualGraphComposite.adjacency

VirtualGraphComposite.**adjacency = None**

dict[variable, set] – Adjacency structure for the composed sampler.

Examples

This example uses *VirtualGraphComposite* to instantiate a composed sampler that uses a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file. Because qubits 0, 5, and coupled qubits {0, 4} are all coupled on the selected D-Wave solver, edges between three nodes, x, y, and z, as specified by the embedding, are available to problems using this composed sampler. However, qubit 8 is in an adjacent unit cell on the D-Wave solver and not directly connected to the other four qubits, so node *a* does not share an edge with any other nodes.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import VirtualGraphComposite
>>> embedding = {'x': {1}, 'y': {5}, 'z': {0, 4}, 'a': {8}}
>>> sampler = VirtualGraphComposite(DWaveSampler(), embedding)
>>> sampler.adjacency
{'a': set(), 'x': {'y', 'z'}, 'y': {'x', 'z'}, 'z': {'x', 'y'}}
```

dwave.system.composites.VirtualGraphComposite.structure

VirtualGraphComposite.**structure**

Structure of the structured sampler formatted as a `namedtuple Structure(nodelist, edgelist, adjacency)`, where the 3-tuple values are the `nodelist` and `edgelist` properties and `adjacency()` method.

Examples

This example shows the structure of a placeholder structured sampler that samples only from the K3 complete graph, where each of the three nodes connects to all the other nodes.

```
>>> class K3StructuredClass(dimod.Structured):
...     @property
...     def nodelist(self):
...         return [1, 2, 3]
...
...     @property
...     def edgelist(self):
...         return [(1, 2), (1, 3), (2, 3)]
>>> K3sampler = K3StructuredClass()
>>> K3sampler.structure.edgelist
[(1, 2), (1, 3), (2, 3)]
```

Methods

VirtualGraphComposite.sample(bqm, **kwargs)	Sample from the given Ising model.
VirtualGraphComposite.sample_ising(h, J, …)	Samples from an Ising model using an implemented sample method.
VirtualGraphComposite.sample_qubo(Q, …)	Samples from a QUBO using an implemented sample method.

dwave.system.composites.VirtualGraphComposite.sample

VirtualGraphComposite.**sample**(*bqm*, ***kwargs*)

Sample from the given Ising model.

Parameters

- **h** (*list/dict*) – Linear biases of the Ising model. If a list, the list's indices are used as variable labels.
- **J** (*dict of (int, int)* – float): Quadratic biases of the Ising model.
- **apply_flux_bias_offsets** (*bool, optional*) – If True, use the calculated flux_bias offsets (if available).
- ****kwargs** – Optional keyword arguments for the sampling method, specified per solver.

Examples

This example uses *VirtualGraphComposite* to instantiate a composed sampler that submits an Ising problem to a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file. The problem represents a logical NOT gate using penalty function $P = xy$, where variable x is the gate's input and y the output. This simple two-variable problem is manually minor-embedded to a single Chimera unit cell: each variable is represented by a chain of half the cell's qubits, x as qubits 0, 1, 4, 5, and y as qubits 2, 3, 6, 7. The chain strength is set to half the maximum allowed found from querying the solver's extended J range. In this example, the ten returned samples all represent valid states of the NOT gate.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import VirtualGraphComposite
>>> embedding = {'x': {0, 4, 1, 5}, 'y': {2, 6, 3, 7}}
>>> DWaveSampler().properties['extended_j_range']
[-2.0, 1.0]
>>> sampler = VirtualGraphComposite(DWaveSampler(), embedding, chain_strength=1)
>>> h = {}
>>> J = {('x', 'y'): 1}
>>> response = sampler.sample_ising(h, J, num_reads=10)
>>> for sample in response.samples():
...     print(sample)
...
{'y': -1, 'x': 1}
{'y': 1, 'x': -1}
{'y': -1, 'x': 1}
{'y': -1, 'x': 1}
{'y': -1, 'x': 1}
{'y': 1, 'x': -1}
{'y': 1, 'x': -1}
{'y': 1, 'x': -1}
{'y': -1, 'x': 1}
{'y': 1, 'x': -1}
```

dwave.system.composites.VirtualGraphComposite.sample_ising

`VirtualGraphComposite.sample_ising(h, J, **parameters)`
Samples from an Ising model using an implemented sample method.

Examples

This example implements a placeholder QUBO sampler and samples using the mixin Ising sampler.

```
>>> import dimod
>>> class ImplementQuboSampler(dimod.Sampler):
...     def sample_qubo(self, Q):
...         return dimod.Response.from_dicts([{1: -1, 2: +1}], {'energy': [-1.0]}
↪) # Placeholder
...     @property
...     def properties(self):
...         return self._properties
...     @property
...     def parameters(self):
...         return dict()
...
>>> sampler = ImplementQuboSampler()
>>> h = {1: 0.5, 2: -1, 3: -0.75}
>>> J = {}
>>> response = sampler.sample_ising(h, J)
>>> print(response)
[[-1  1]]
```

dwave.system.composites.VirtualGraphComposite.sample_qubo

`VirtualGraphComposite.sample_qubo(Q, **parameters)`
Samples from a QUBO using an implemented sample method.

Examples

This example implements a placeholder Ising sampler and samples using the mixin QUBO sampler.

```
>>> import dimod
>>> class ImplementIsingSampler(dimod.Sampler):
...     def sample_ising(self, h, J):
...         return dimod.Response.from_dicts([{1: -1, 2: +1}], {'energy': [-1.0]}
↪) # Placeholder
...     @property
...     def properties(self):
...         return self._properties
...     @property
...     def parameters(self):
...         return dict()
...
>>> sampler = ImplementIsingSampler()
>>> Q = {(0, 0): -0.5, (0, 1): 1, (1, 1): -0.75}
>>> response = sampler.sample_qubo(Q)
```

(continues on next page)

(continued from previous page)

```
>>> print(response)
[[0 1]]
```

1.2 Installation

Installation from PyPI:

```
pip install dwave-system
```

Installation from PyPI with drivers:

Note: Prior to v0.3.0, running `pip install dwave-system` installed a driver dependency called `dwave-system-tuning`. This dependency has a restricted license and has been made optional as of v0.3.0, but is highly recommanded. To view the license details:

```
from dwave.system.tuning import __license__
print(__license__)
```

To install with optional dependencies:

```
pip install dwave-system[drivers] --extra-index-url https://pypi.dwavesys.com/simple
```

Installation from source:

```
pip install -r requirements.txt
python setup.py
```

Note that installing from source installs dwave-system-tuning. To uninstall the proprietary components:

```
pip uninstall dwave-system-tuning
```

1.3 License

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

1.4 D-Wave

D-Wave Systems is the leader in the development and delivery of quantum computing systems and software, and the world's only commercial supplier of quantum computers.

Learn more about D-Wave at D-Wave Systems.

1.5 Ocean Overview

D-Wave Ocean includes various projects/repositories on GitHub that help solve problems on the D-Wave system.

Learn about D-Wave's Ocean and how its projects work together at D-Wave Ocean on Read the Docs.

1.6 Contributing to Ocean

D-Wave welcomes contributions to Ocean projects.

See how to contribute at Ocean Contributors.

1.7 Glossary

The field of quantum computing has many domain-specific terms.

Learn the relevant terminology at Ocean Glossary.

Python Module Index

d

dwave.system.composites.embedding, 12
dwave.system.composites.tiling, 21
dwave.system.composites.virtual_graph, 29
dwave.system.samplers.dwave_sampler, 4

Index

A

adjacency (DWaveSampler attribute), 9
adjacency (FixedEmbeddingComposite attribute), 19
adjacency (TilingComposite attribute), 26
adjacency (VirtualGraphComposite attribute), 34

C

child (EmbeddingComposite attribute), 14
child (FixedEmbeddingComposite attribute), 18
child (TilingComposite attribute), 24
child (VirtualGraphComposite attribute), 32
children (EmbeddingComposite attribute), 14
children (FixedEmbeddingComposite attribute), 18
children (TilingComposite attribute), 24
children (VirtualGraphComposite attribute), 32

D

dwave.system.composites.embedding (module), 12
dwave.system.composites.tiling (module), 21
dwave.system.composites.virtual_graph (module), 29
dwave.system.samplers.dwave_sampler (module), 4
DWaveSampler (class in dwave.system.samplers), 4

E

edgelist (DWaveSampler attribute), 8
edgelist (FixedEmbeddingComposite attribute), 19
edgelist (TilingComposite attribute), 25
edgelist (VirtualGraphComposite attribute), 33
EmbeddingComposite (class in dwave.system.composites), 12

F

FixedEmbeddingComposite (class in dwave.system.composites), 17

N

nodelist (DWaveSampler attribute), 8
nodelist (FixedEmbeddingComposite attribute), 19
nodelist (TilingComposite attribute), 25
nodelist (VirtualGraphComposite attribute), 33

P

parameters (DWaveSampler attribute), 7
parameters (EmbeddingComposite attribute), 13
parameters (FixedEmbeddingComposite attribute), 17
parameters (TilingComposite attribute), 23
parameters (VirtualGraphComposite attribute), 31
properties (DWaveSampler attribute), 7
properties (EmbeddingComposite attribute), 13
properties (FixedEmbeddingComposite attribute), 17
properties (TilingComposite attribute), 23
properties (VirtualGraphComposite attribute), 31

S

sample() (DWaveSampler method), 10
sample() (EmbeddingComposite method), 15
sample() (FixedEmbeddingComposite method), 20
sample() (TilingComposite method), 27
sample() (VirtualGraphComposite method), 35
sample_ising() (DWaveSampler method), 10
sample_ising() (EmbeddingComposite method), 16
sample_ising() (FixedEmbeddingComposite method), 20
sample_ising() (TilingComposite method), 28
sample_ising() (VirtualGraphComposite method), 36
sample_qubo() (DWaveSampler method), 11
sample_qubo() (EmbeddingComposite method), 16
sample_qubo() (FixedEmbeddingComposite method), 21
sample_qubo() (TilingComposite method), 29
sample_qubo() (VirtualGraphComposite method), 36
structure (DWaveSampler attribute), 9
structure (FixedEmbeddingComposite attribute), 19
structure (TilingComposite attribute), 27
structure (VirtualGraphComposite attribute), 34

T

TilingComposite (class in dwave.system.composites), 21

V

VirtualGraphComposite (class in dwave.system.composites), 29