dwave-system Documentation

Release 0.7.2

D-Wave Systems Inc

Mar 19, 2019

Contents

1 Documentation 3

Python Module Index 67

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 Introduction

dwave-system enables easy incorporation of the D-Wave system as a sampler—the component used to find variable values that minimize the binary quadratic model (BQM) representing a problem—in the typical Ocean problem-solving procedure:

1. Formulate the problem as a BQM.
2. Solve the BQM with a sampler.

1.1.1 Example

This example solves a small example of a known graph problem, minimum vertex cover. It uses the NetworkX graphic package to create the problem, Ocean's dwave_networkx to formulate the graph problem as a BQM, and dwave-system's `DWaveSampler()` to use a D-Wave system as the sampler. (Access to a D-Wave system has been set up in a configuration file that is used implicitly.) dwave-system's `EmbeddingComposite()` handles mapping between the problem graph to the D-Wave system's numerically indexed qubits, a mapping known as minor-embedding.

```
>>> import networkx as nx
>>> import dwave_networkx as dnx
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import EmbeddingComposite
...
>>> s5 = nx.star_graph(4)  # a star graph where node 0 is hub to four other nodes
>>> sampler = EmbeddingComposite(DWaveSampler())
>>> print(dnx.min_vertex_cover(s5, sampler))
[0]
```

1.1.2 Samplers

Samplers are processes that sample from low energy states of a problem's objective function. A 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.

Ocean software provides a variety of dimod samplers, which all support 'sample_qubo' and 'sample_ising' methods as well as the generic BQM sampler method. In addition to `DWaveSampler()`, classical solvers, which run on CPU or GPU, are available and useful for developing code or on a small versions of a problem to verify code.

1.1.3 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.

Examples of composites are `EmbeddingComposite()`, used in the example above, and `VirtualGraphComposite()`, both of which handle the mapping known as minor-embedding.

1.1.4 Using the D-Wave System as a Sampler

The dimod API makes it possible to easily interchange samplers in your code. For example, you might develop code using dwave_neal, Ocean's classical simulated annealing sampler, and then swap in a D-Wave system composed sampler.

Using a D-Wave System explains how you set up access to a D-Wave system.

D-Wave System Documentation describes the D-Wave system, its features, parameters, and properties. The documentation provides guidance on programming the D-Wave system, including how to formulate problems and configure parameters.

Below one example attribute of the D-Wave system is described. For others and further information, see the D-Wave System Documentation.

Minor-Embedding

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.

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.

In addition to composites that handle minor-embedding, dwave-system provides the related functionality described in *Embedding*.

1.2 Reference Documentation

Release 0.7.2

Date Mar 19, 2019

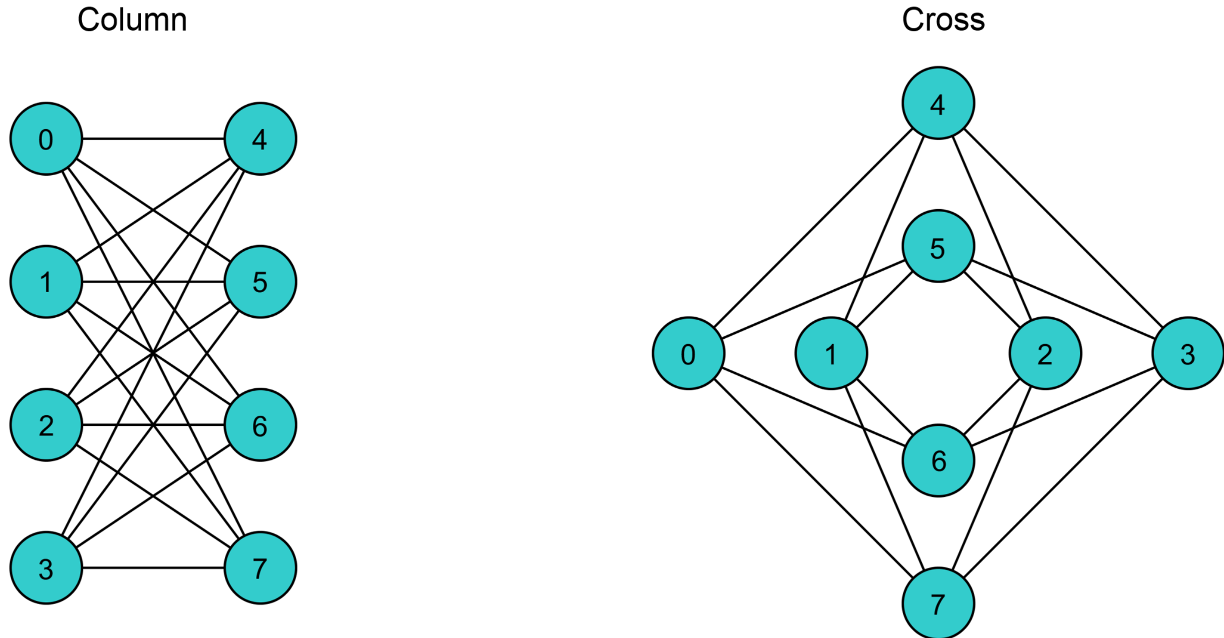

Fig. 1: Chimera unit cell.

1.2.1 Samplers

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

Release 0.7.2

Date Mar 19, 2019

D-Wave Sampler

A dimod sampler for the D-Wave system.

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

Class

class DWaveSampler(***config*)

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).

Typically you store the parameters used to identify and communicate with your D-Wave system in a configuration file, the D-Wave Cloud Client configuration file, which may include more than one profile, and are selected when not overridden by explicit input arguments or environment variables. For more information, see D-Wave Cloud Client.

Parameters

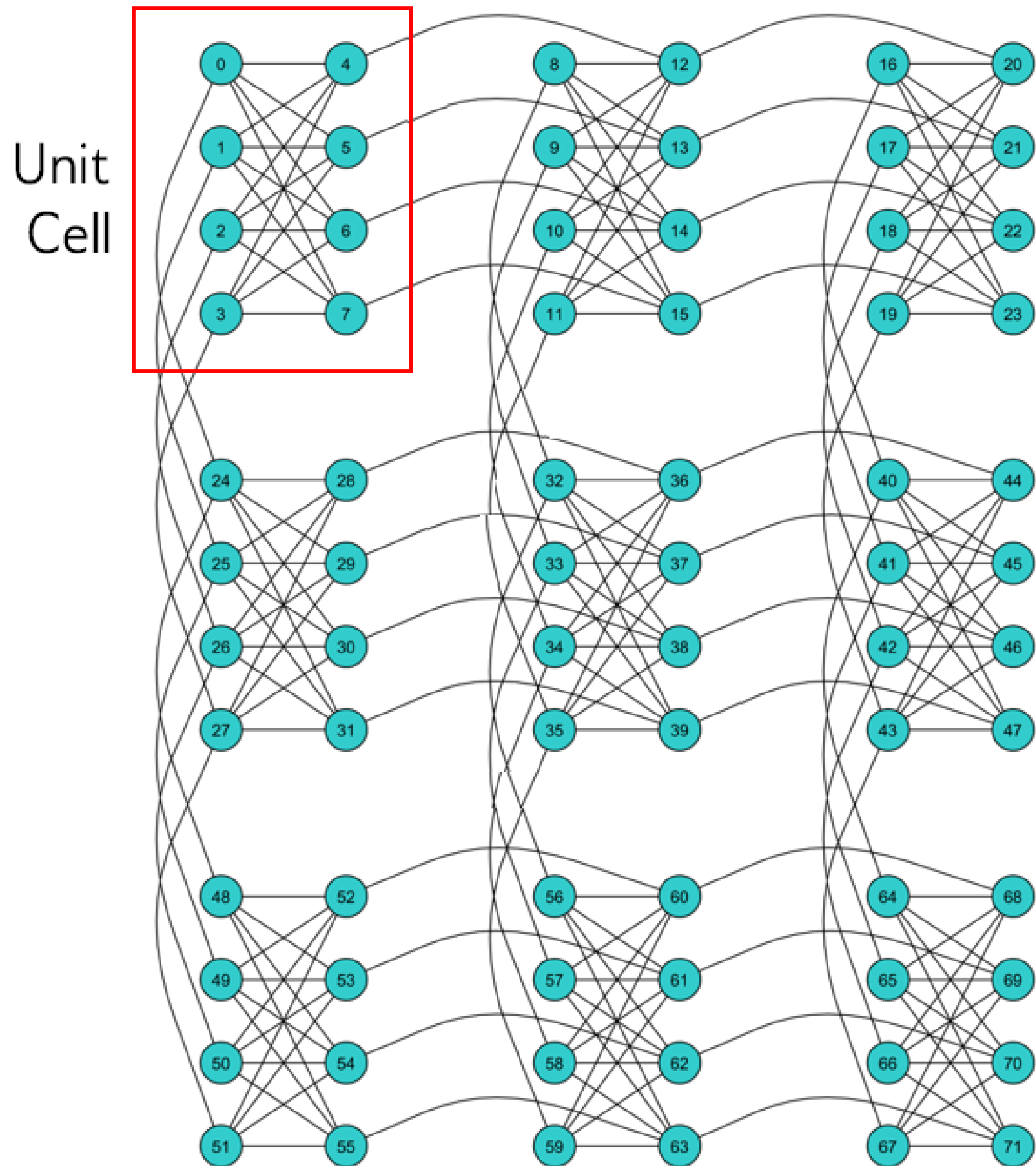

Fig. 2: A 3x3 Chimera graph, denoted C3. Qubits are arranged in 9 unit cells.

- **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 the configuration file.
- **endpoint** (*str, optional*) – D-Wave API endpoint URL.
- **token** (*str, optional*) – Authentication token for the D-Wave API to authenticate the client session.
- **solver** (*dict/str, optional*) – Solver (a D-Wave system on which to run submitted problems) to select given as a set of required features. Supported features and values are described in `get_solvers()`. For backward compatibility, solver name, formatted as a string, is accepted.
- **proxy** (*str, optional*) – Proxy URL to be used for accessing the D-Wave API.
- ****config** – Keyword arguments passed directly to `from_config()`.

Examples

This example submits a two-variable Ising problem mapped directly to qubits 0 and 1 on the example system.

Example configuration file /home/susan/.config/dwave/dwave.conf:

```
[defaults]
endpoint = https://url.of.some.dwavesystem.com/sapi
client = qpu
[2000q]
solver = {"num_qubits__gte": 2000}
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}
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

Sampler Properties

D-Wave System Documentation lists and describes the parameters and properties of D-Wave systems.

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.
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 1 – 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.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 prints properties retrieved from a D-Wave system 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
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

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 prints the parameters retrieved from a D-Wave system selected by the user's default D-Wave Cloud Client configuration file.

```
>>> from dwave.system.samplers import DWaveSampler
>>> sampler = DWaveSampler()
>>> sampler.parameters
```

(continues on next page)

(continued from previous page)

```
{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
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

dwave.system.samplers.DWaveSampler.nodelist

DWaveSampler.**nodelist**

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

Examples

This example prints the active qubits retrieved from a D-Wave system 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,
# Snipped above response for brevity
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

dwave.system.samplers.DWaveSampler.edgelist

DWaveSampler.**edgelist**

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

Examples

This example prints the active couplers retrieved from a D-Wave system 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),
# Snipped above response for brevity
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

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

Sample 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 specified Ising model.
`DWaveSampler.sample_qubo`(Q, **kwargs)	Sample from the specified QUBO.

dwave.system.samplers.DWaveSampler.sample

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

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

dwave.system.samplers.DWaveSampler.sample_ising

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

Sample from the specified 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 A *dimod* `SampleSet` object.

Return type `dimod.SampleSet`

Examples

This example submits a two-variable Ising problem mapped directly to qubits 0 and 1 on a D-Wave system selected by the user's default D-Wave Cloud Client configuration file.

```
>>> 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}
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

dwave.system.samplers.DWaveSampler.sample_qubo

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

Sample from the specified 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 A *dimod* `SampleSet` object.

Return type `dimod.SampleSet`

Examples

This example submits a two-variable Ising problem mapped directly to qubits 0 and 4 on a D-Wave system selected by the user's default D-Wave Cloud Client configuration file.

```
>>> 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}
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

Methods

DWaveSampler.validate_anneal_schedule(...)	Raise an exception if the specified schedule is invalid for the sampler.

dwave.system.samplers.DWaveSampler.validate_anneal_schedule

`DWaveSampler.`**`validate_anneal_schedule`**(*anneal_schedule*)

Raise an exception if the specified schedule is invalid for the sampler.

Parameters **anneal_schedule** (*list*) – An anneal schedule variation is defined by a series of pairs of floating-point numbers identifying points in the schedule at which to change slope. The first element in the pair is time t in microseconds; the second, normalized persistent current s in the range [0,1]. The resulting schedule is the piecewise-linear curve that connects the provided points.

Raises

- `ValueError` – If the schedule violates any of the conditions listed below.
- `RuntimeError` – If the sampler does not accept the *anneal_schedule* parameter or if it does not have *annealing_time_range* or *max_anneal_schedule_points* properties.

An anneal schedule must satisfy the following conditions:

- Time t must increase for all points in the schedule.
- For forward annealing, the first point must be (0,0) and the anneal fraction s must increase monotonically.
- For reverse annealing, the anneal fraction s must start and end at s=1.
- In the final point, anneal fraction s must equal 1 and time t must not exceed the maximum value in the *annealing_time_range* property.
- The number of points must be >=2.

- The upper bound is system-dependent; check the *max_anneal_schedule_points* property. For reverse annealing, the maximum number of points allowed is one more than the number given by this property.

Examples

This example sets a quench schedule on a D-Wave system selected by the user's default D-Wave Cloud Client configuration file.

```
>>> from dwave.system.samplers import DWaveSampler
>>> sampler = DWaveSampler()
>>> quench_schedule=[[0.0, 0.0], [12.0, 0.6], [12.8, 1.0]]
>>> DWaveSampler().validate_anneal_schedule(quench_schedule)
>>>
```

1.2.2 Composites

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

Release 0.7.2

Date Mar 19, 2019

EmbeddingComposite

Class

class EmbeddingComposite(*child_sampler*)

Composite that maps problems to a structured sampler.

Enables quick incorporation of the D-Wave system as a sampler by handling minor-embedding of the problem into the D-Wave system's Chimera graph. Minor-embedding is calculated using the heuristic minorminer library each time one of its sampling methods is called.

Parameters **sampler** (`dimod.Sampler`) – Structured dimod sampler such as a `DWaveSampler()`.

Examples

This example submits a simple Ising problem to a D-Wave solver selected by the user's default D-Wave Cloud Client configuration file. `EmbeddingComposite` maps the problem's variables 'a' and 'b' to qubits on the D-Wave system.

```
>>> from dwave.system import DWaveSampler, 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}
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

Properties

EmbeddingComposite.child	First child in `children`.
EmbeddingComposite.children	*list [child_sampler]* – List containing the structured sampler.
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.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

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

        raise NotImplementedError
```

dwave.system.composites.EmbeddingComposite.children

EmbeddingComposite.**children**

list [child_sampler] – List containing the structured sampler.

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 views properties of a composed sampler using a D-Wave system selected by the user's default D-Wave Cloud Client configuration file.

```
>>> 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
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

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 and parameters added by the composite such as those related to chains.

Examples

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

```
>>> 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
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

Methods

EmbeddingComposite.sample(bqm[, ...])	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*, *chain_break_fraction=True*, ***parameters*)

Sample from the provided binary quadratic model.

Also set parameters for handling a chain, the set of vertices in a target graph that represents a source-graph vertex; when a D-Wave system is the sampler, it is a set of qubits that together represent a variable of the binary quadratic model being minor-embedded.

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. The energy penalty of chain breaks is 2 * *chain_strength*.
- **chain_break_fraction** (*bool, optional, default=True*) – If True, the unembedded response contains a 'chain_break_fraction' field that reports the fraction of chains broken before unembedding.
- ****parameters** – Parameters for the sampling method, specified by the child sampler.

Returns A *dimod* SampleSet object.

Return type dimod.SampleSet

Examples

This example submits an triangle-structured Ising problem to a D-Wave solver, selected by the user's default D-Wave Cloud Client configuration file, by minor-embedding the problem's variables to physical qubits.

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import EmbeddingComposite
>>> import dimod
...
>>> sampler = EmbeddingComposite(DWaveSampler())
>>> bqm = dimod.BinaryQuadraticModel.from_ising({}, {'ab': 0.5, 'bc': 0.5, 'ca':␣
↪0.5})
>>> response = sampler.sample(bqm, chain_strength=2)
>>> response.first:
Sample(sample={'a': -1, 'b': 1, 'c': 1}, energy=-0.5,
       num_occurrences=1, chain_break_fraction=0.0)
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

dwave.system.composites.EmbeddingComposite.sample_ising

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

Samples from an Ising model using an implemented sample method.

dwave.system.composites.EmbeddingComposite.sample_qubo

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

Samples from a QUBO using an implemented sample method.

FixedEmbeddingComposite

Class

class FixedEmbeddingComposite(*child_sampler*, *embedding=None*, *source_adjacency=None*)

Composite that uses a specified minor-embedding to map problems to a structured sampler.

Enables incorporation of the D-Wave system as a sampler, given a precalculated minor-embedding.

Parameters

- **sampler** (*dimod.Sampler*) – Structured dimod sampler such as a D-Wave system.
- **embedding** (*dict[hashable, iterable]*) – Mapping from a source graph to the specified sampler's graph (the target graph).
- **source_adjacency** (*dict[hashable, iterable]*) – Dictionary to describe source graph. Ex. {node: {node neighbours}}

Examples

This example submits an triangle-structured Ising problem to a D-Wave solver, selected by the user's default D-Wave Cloud Client configuration file, using a given minor-embedding of the problem's variables to physical qubits.

```
>>> 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.

See *EmbeddingComposite.properties* for detailed information.

dwave.system.composites.FixedEmbeddingComposite.parameters

`FixedEmbeddingComposite.`**`parameters = None`**

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

See `EmbeddingComposite.parameters` for detailed information.

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.

See `EmbeddingComposite.children` for detailed information.

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.

Examples

```
>>> fixedsampler = FixedEmbeddingComposite(DWaveSampler(), {'a': [0, 4], 'b': [1,␣
↪5], 'c': [2, 6]})
>>> fixedsampler.nodelist
['a', 'b', 'c']
```

dwave.system.composites.FixedEmbeddingComposite.edgelist

`FixedEmbeddingComposite.`**`edgelist = None`**

list – Edges available to the composed sampler.

Examples

```
>>> fixedsampler = FixedEmbeddingComposite(DWaveSampler(), {'a': [0, 4], 'b': [1,␣
↪5], 'c': [2, 6]})
>>> fixedsampler.edgelist
[('a', 'b'), ('a', 'c'), ('b', 'c')]
```

dwave.system.composites.FixedEmbeddingComposite.adjacency

`FixedEmbeddingComposite.`**`adjacency = None`**

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

Examples

```
>>> fixedsampler = FixedEmbeddingComposite(DWaveSampler(), {'a': [0, 4], 'b': [1,␣
↪5], 'c': [2, 6]})
>>> fixedsampler.adjacency
{'a': {'b', 'c'}, 'b': {'a', 'c'}, 'c': {'a', 'b'}}
```

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[, ...])	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*, *chain_strength=1.0*, *chain_break_fraction=True*, ***parameters*)

Sample from the provided binary quadratic model.

Also set parameters for handling a chain, the set of vertices in a target graph that represents a source-graph vertex; when a D-Wave system is the sampler, it is a set of qubits that together represent a variable of the binary quadratic model being minor-embedded.

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. The energy penalty of chain breaks is 2 * *chain_strength*.
- **chain_break_fraction** (*bool, optional, default=True*) – If True, the unembedded response contains a 'chain_break_fraction' field that reports the fraction of chains broken before unembedding.
- ****parameters** – Parameters for the sampling method, specified by the child sampler.

Returns A *dimod* `SampleSet` object.

Return type `dimod.SampleSet`

Examples

This example submits an triangle-structured problem to a D-Wave solver, selected by the user's default D-Wave Cloud Client configuration file, using a specified minor-embedding of the problem's variables to physical qubits.

```
>>> 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]})
>>> response = sampler.sample_ising({}, {'ab': 0.5, 'bc': 0.5, 'ca': 0.5}, chain_
↪strength=2)
>>> response.first
Sample(sample={'a': 1, 'b': -1, 'c': 1}, energy=-0.5, num_occurrences=1, chain_
↪break_fraction=0.0)
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

dwave.system.composites.FixedEmbeddingComposite.sample_ising

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

Samples from an Ising model using an implemented sample method.

dwave.system.composites.FixedEmbeddingComposite.sample_qubo

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

Samples from a QUBO using an implemented sample method.

LazyFixedEmbeddingComposite

Class

class LazyFixedEmbeddingComposite(*sampler*)

Takes an unstructured problem and maps it to a structured problem. This mapping is stored and gets reused for all following sample(..) calls.

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

Examples

```
>>> from dwave.system import LazyFixedEmbeddingComposite, DWaveSampler
...
>>> sampler = LazyFixedEmbeddingComposite(DWaveSampler())
>>> sampler.nodelist is None  # no structure yet
True
>>> __ = sampler.sample_ising({}, {('a', 'b'): -1})
>>> sampler.nodelist  # has structure based on given problem
['a', 'b']
```

Properties

LazyFixedEmbeddingComposite.parameters	
LazyFixedEmbeddingComposite.properties	
LazyFixedEmbeddingComposite.nodelist	
LazyFixedEmbeddingComposite.edgelist	
LazyFixedEmbeddingComposite.adjacency	
LazyFixedEmbeddingComposite.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.LazyFixedEmbeddingComposite.parameters

LazyFixedEmbeddingComposite.**parameters = None**

dwave.system.composites.LazyFixedEmbeddingComposite.properties

LazyFixedEmbeddingComposite.**properties = None**

dwave.system.composites.LazyFixedEmbeddingComposite.nodelist

LazyFixedEmbeddingComposite.**nodelist = None**

dwave.system.composites.LazyFixedEmbeddingComposite.edgelist

LazyFixedEmbeddingComposite.**edgelist = None**

dwave.system.composites.LazyFixedEmbeddingComposite.adjacency

LazyFixedEmbeddingComposite.**adjacency = None**

dwave.system.composites.LazyFixedEmbeddingComposite.structure

LazyFixedEmbeddingComposite.**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

LazyFixedEmbeddingComposite.sample(bqm[, . . .])	Sample the binary quadratic model.
LazyFixedEmbeddingComposite.sample_ising(h, . . .)	Samples from an Ising model using an implemented sample method.
LazyFixedEmbeddingComposite.sample_qubo(Q, . . .)	Samples from a QUBO using an implemented sample method.

dwave.system.composites.LazyFixedEmbeddingComposite.sample

LazyFixedEmbeddingComposite.**sample**(*bqm*, *chain_strength=1.0*, *chain_break_fraction=True*, ***parameters*)

Sample the binary quadratic model.

Note: At the initial sample(..) call, it will find a suitable embedding and initialize the remaining attributes before sampling the bqm. All following sample(..) calls will reuse that initial embedding.

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*.
- **chain_break_fraction** (*bool, optional, default=True*) – If True, a 'chain_break_fraction' field is added to the unembedded response which report what fraction of the chains were broken before unembedding.

- `**parameters` – Parameters for the sampling method, specified by the child sampler.

Returns `dimod.SampleSet`

dwave.system.composites.LazyFixedEmbeddingComposite.sample_ising

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

dwave.system.composites.LazyFixedEmbeddingComposite.sample_qubo

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

TilingComposite

Class

`class TilingComposite(sampler, sub_m, sub_n, t=4)`
Composite to tile a small problem across a Chimera-structured sampler.

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

Notation *CN* refers to a Chimera graph consisting of an NxN grid of unit cells, where each unit cell is 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 such as a `DWaveSampler()`.
- **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 submits a two-variable QUBO problem representing a logical NOT gate to a D-Wave system selected by the user's default D-Wave Cloud Client configuration file. The QUBO—two nodes with biases of -1 that are coupled with strength 2—needs only any two coupled qubits and so is easily minor-embedded in a single unit cell. Composite `TilingComposite` tiles it multiple times 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)
>>> response.first
Sample(sample={1: 0, 2: 1}, energy=-1.0, num_occurrences=1, chain_break_
↪fraction=0.0)
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

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.

See `EmbeddingComposite.properties` for detailed information.

dwave.system.composites.TilingComposite.parameters

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

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

See `EmbeddingComposite.parameters` for detailed information.

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.

See `EmbeddingComposite.children` for detailed information.

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

```
>>> 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

```
>>> sampler_tile = TilingComposite(DWaveSampler(), 1, 2, 4)
>>> len(sampler_tile.edgelist)
36
```

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)]
>>> 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 specified 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 specified 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.SampleSet

Examples

This example submits a simple Ising problem of just two variables on a D-Wave system selected by the user's default D-Wave Cloud Client configuration file. Because the problem fits in a single Chimera unit cell, it is tiled across the solver's entire Chimera graph, resulting in multiple samples (the exact number depends on the working Chimera graph of the D-Wave system).

```
>>> from dwave.system.samplers import DWaveSampler
>>> from dwave.system.composites import EmbeddingComposite
>>> from dwave.system.composites import EmbeddingComposite, TilingComposite
...
>>> sampler = EmbeddingComposite(TilingComposite(DWaveSampler(), 1, 1, 4))
>>> response = sampler.sample_ising({},{('a', 'b'): 1})
>>> len(response)
246
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

dwave.system.composites.TilingComposite.sample_ising

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

Samples from an Ising model using an implemented sample method.

dwave.system.composites.TilingComposite.sample_qubo

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

Samples from a QUBO using an implemented sample method.

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.

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

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.

> **Attention:** D-Wave's *virtual graphs* feature can require many seconds of D-Wave system time to calibrate qubits to compensate for the effects of biases. If your account has limited D-Wave system access, consider using *FixedEmbeddingComposite()* instead.

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}
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

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
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

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
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

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 18 – 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']
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

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')]
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

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'}}
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

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[, ...])	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*, *apply_flux_bias_offsets=True*, ***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}
```

See Ocean Glossary for explanations of technical terms in descriptions of Ocean tools.

dwave.system.composites.VirtualGraphComposite.sample_ising

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

Samples from an Ising model using an implemented sample method.

dwave.system.composites.VirtualGraphComposite.sample_qubo

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

Samples from a QUBO using an implemented sample method.

CutOffComposite

Composites that remove interactions with biases smaller than a cutoff. Isolated variables (after the cutoff) are also removed.

CutOffComposite

class CutOffComposite(*child_sampler, cutoff, cutoff_vartype=<Vartype.SPIN: frozenset({1, -1})>, comparison=<built-in function lt>*)

Composite to cut off small interactions.

Removes interactions smaller than a given cutoff. Isolated variables (after the cutoff) are also removed.

Note that if the problem had isolated variables before the cutoff, they will also be affected.

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

cutoff (number): The lower bound for interaction bias magnitudes. Interactions with biases less than cutoff are removed. Isolated variables are also not sent to the child sampler.

cutoff_vartype (`Vartype`/str/set, default='SPIN'): Variable space to do the cutoff in. Accepted input values:

- `Vartype.SPIN`, `'SPIN'`, `{-1, 1}`
- `Vartype.BINARY`, `'BINARY'`, `{0, 1}`

comparison (function, optional): A comparison operator for comparing the bias magnitude to the cutoff value. Defaults to `operator.lt()`.

Properties

CutOffComposite.child	First child in `children`.
CutOffComposite.children	list[`Sampler`] – List of samplers (or composed samplers).
CutOffComposite.properties	*dict* – A dict containing any additional information about the sampler.
CutOffComposite.parameters	*dict* – A dict where keys are the keyword parameters accepted by the sampler methods and values are lists of the properties relevent to each parameter.

dwave.system.composites.cutoffcomposite.CutOffComposite.child

CutOffComposite.**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
```

dwave.system.composites.cutoffcomposite.CutOffComposite.children

CutOffComposite.**children**
list[Sampler] – List of samplers (or composed samplers).

This abstract property must be implemented.

Examples

These examples define the supported samplers for the composed sampler upon instantiation.

```
class MyComposite(dimod.Composite):
    def __init__(self, *children):
        self._children = list(children)

    @property
    def children(self):
        return self._children
```

```
class AnotherComposite(dimod.Composite):
    self.children = None
    def __init__(self, child_sampler):
        self.children = [child_sampler]
```

dwave.system.composites.cutoffcomposite.CutOffComposite.properties

`CutOffComposite.`**`properties`**
dict – A dict containing any additional information about the sampler.

dwave.system.composites.cutoffcomposite.CutOffComposite.parameters

`CutOffComposite.`**`parameters`**
dict – A dict where keys are the keyword parameters accepted by the sampler methods and values are lists of the properties relevent to each parameter.

Methods

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

dwave.system.composites.cutoffcomposite.CutOffComposite.sample

`CutOffComposite.`**`sample`**(*bqm*, ***parameters*)
Cutoff and sample from the provided binary quadratic model.

Removes interactions smaller than a given cutoff. Isolated variables (after the cutoff) are also removed.

Note that if the problem had isolated variables before the cutoff, they will also be affected.

Parameters

- **bqm** (`dimod.BinaryQuadraticModel`) – Binary quadratic model to be sampled from.
- ****parameters** – Parameters for the sampling method, specified by the child sampler.

Returns `dimod.SampleSet`

dwave.system.composites.cutoffcomposite.CutOffComposite.sample_ising

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

dwave.system.composites.cutoffcomposite.CutOffComposite.sample_qubo

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

PolyCutOffComposite

class PolyCutOffComposite(*child_sampler*, *cutoff*, *cutoff_vartype=<Vartype.SPIN: frozenset({1, -1})>*, *comparison=<built-in function lt>*)
Composite to cut off small interactions.

Removes interactions smaller than a given cutoff. Isolated variables (after the cutoff) are also removed.

Note that if the problem had isolated variables before the cutoff, they will also be affected.

Parameters **sampler** (`dimod.PolySampler`) – A dimod binary polynomial sampler

cutoff (number): The lower bound for interaction bias magnitudes. Interactions with biases less than cutoff are removed. Isolated variables are also not sent to the child sampler.

cutoff_vartype (`Vartype`/str/set, default='SPIN'): Variable space to do the cutoff in. Accepted input values:

- `Vartype.SPIN`, `'SPIN'`, `{-1, 1}`
- `Vartype.BINARY`, `'BINARY'`, `{0, 1}`

comparison (function, optional): A comparison operator for comparing the bias magnitude to the cutoff value. Defaults to `operator.lt()`.

Properties

PolyCutOffComposite.child	First child in `children`.
PolyCutOffComposite.children	list[`Sampler`] – List of samplers (or composed samplers).
PolyCutOffComposite.properties	*dict* – A dict containing any additional information about the sampler.
PolyCutOffComposite.parameters	*dict* – A dict where keys are the keyword parameters accepted by the sampler methods and values are lists of the properties relevant to each parameter.

dwave.system.composites.cutoffcomposite.PolyCutOffComposite.child

PolyCutOffComposite.**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):
```

(continues on next page)

(continued from previous page)

```
    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
```

dwave.system.composites.cutoffcomposite.PolyCutOffComposite.children

PolyCutOffComposite.**children**

list[Sampler] – List of samplers (or composed samplers).

This abstract property must be implemented.

Examples

These examples define the supported samplers for the composed sampler upon instantiation.

```
class MyComposite(dimod.Composite):
    def __init__(self, *children):
        self._children = list(children)

    @property
    def children(self):
        return self._children
```

```
class AnotherComposite(dimod.Composite):
    self.children = None
    def __init__(self, child_sampler):
        self.children = [child_sampler]
```

dwave.system.composites.cutoffcomposite.PolyCutOffComposite.properties

PolyCutOffComposite.**properties**

dict – A dict containing any additional information about the sampler.

dwave.system.composites.cutoffcomposite.PolyCutOffComposite.parameters

PolyCutOffComposite.**parameters**
dict – A dict where keys are the keyword parameters accepted by the sampler methods and values are lists of the properties relevant to each parameter.

Methods

PolyCutOffComposite.sample_poly(poly, **kwargs)	Cutoff and sample from the provided binary polynomial.
PolyCutOffComposite.sample_hising(h, J, **kwargs)	Sample from a higher-order Ising model.
PolyCutOffComposite.sample_hubo(H, **kwargs)	Sample from a higher-order unconstrained binary optimization problem.

dwave.system.composites.cutoffcomposite.PolyCutOffComposite.sample_poly

PolyCutOffComposite.**sample_poly**(*poly*, ***kwargs*)
Cutoff and sample from the provided binary polynomial.

Removes interactions smaller than a given cutoff. Isolated variables (after the cutoff) are also removed.

Note that if the problem had isolated variables before the cutoff, they will also be affected.

Parameters

- **poly** (dimod.BinaryPolynomial) – Binary polynomial to be sampled from.
- ****parameters** – Parameters for the sampling method, specified by the child sampler.

Returns dimod.SampleSet

dwave.system.composites.cutoffcomposite.PolyCutOffComposite.sample_hising

PolyCutOffComposite.**sample_hising**(*h*, *J*, ***kwargs*)
Sample from a higher-order Ising model.

Convert the given higher-order Ising model to a BinaryPolynomial and invoke *sample_poly*.

Parameters

- **h** (*dict*) – The variable biases of the Ising problem.
- **J** (*dict*) – The interaction biases of the Ising problem.
- ****kwargs** – See *sample_poly* for additional keyword definitions.

Returns SampleSet

dwave.system.composites.cutoffcomposite.PolyCutOffComposite.sample_hubo

PolyCutOffComposite.**sample_hubo**(*H*, ***kwargs*)
Sample from a higher-order unconstrained binary optimization problem.

Convert the given higher-order unconstrained binary optimization problem to a BinaryPolynomial and invoke *sample_poly*.

Parameters

- **H** (*dict*) – The coefficients of the HUBO.
- ****kwargs** – See *sample_poly* for additional keyword definitions.

Returns SampleSet

1.2.3 Embedding

Provides functions that map binary quadratic models and samples between a source graph and a target graph.

Example

A sampler may not natively support a given problem graph. For example, the D-Wave system does not natively support K_3 graphs. The Boolean AND gate ($x_3 \Leftrightarrow x_1 \wedge x_2$ where x_3 is the AND gate's output and x_1, x_2 the inputs) may be represented as penalty model

$$x_1x_2 - 2(x_1 + x_2)x_3 + 3x_3.$$

This penalty model can in turn be represented as the QUBO,

$$E(a_i, b_{i,j}; x_i) = 3x_3 + x_1x_2 - 2x_1x_3 - 2x_2x_3,$$

which is a fully connected K_3 graph.

Sampling this problem on a D-Wave system, therefore, requires minor-embedding. Embedding in this case is accomplished by an edge contraction operation on the target graph: two nodes (qubits) are chained to represent a single node.

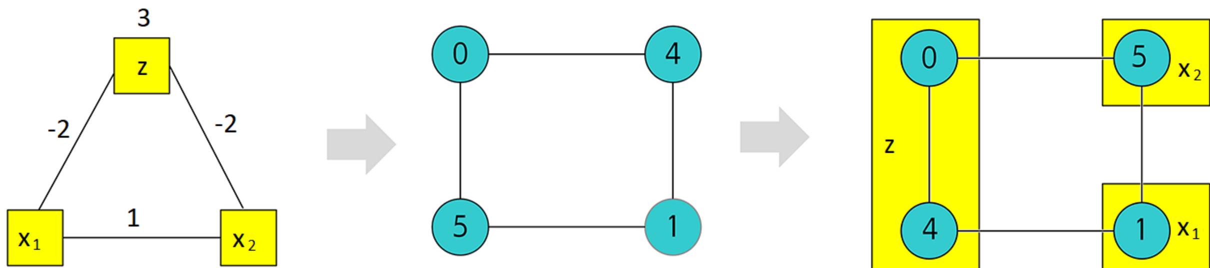

Fig. 3: Embedding an AND gate represented by a K_3 graph onto the D-Wave system's graph. The leftmost graph is the source graph, which is the QUBO representing the AND gate; the middle one is the target graph, representing the D-Wave system; and in the rightmost graph, qubits 0 and 4 of the D-Wave system's graph are chained to represent the single node z of the source graph.

Generating Embeddings

MinorMiner

minorminer is a heuristic tool for minor embedding: given a minor and target graph, it tries to find a mapping that embeds the minor into the target.

minorminer.find_embedding(S, T, **params)	Heuristically attempt to find a minor-embedding of a graph representing an Ising/QUBO into a target graph.

minorminer.find_embedding

find_embedding(*S*, *T*, ***params*)

Heuristically attempt to find a minor-embedding of a graph representing an Ising/QUBO into a target graph.

Parameters

- **S** – an iterable of label pairs representing the edges in the source graph
- **T** – an iterable of label pairs representing the edges in the target graph
- ****params** (*optional*) – see below

Returns

When return_overlap = False (the default), returns a dict that maps labels in S to lists of labels in T

When return_overlap = True, returns a tuple consisting of a dict that maps labels in S to lists of labels in T and a bool indicating whether or not a valid embedding was foun

When interrupted by Ctrl-C, returns the best embedding found so far

Note that failure to return an embedding does not prove that no embedding exists

Optional parameters:

```
max_no_improvement: Maximum number of failed iterations to improve the
    current solution, where each iteration attempts to find an embedding
    for each variable of S such that it is adjacent to all its
    neighbours. Integer >= 0 (default = 10)

random_seed: Seed for the random number generator that find_embedding
    uses. Integer >=0 (default is randomly set)

timeout: Algorithm gives up after timeout seconds. Number >= 0 (default
    is approximately 1000 seconds, stored as a double)

tries: Number of restart attempts before the algorithm stops. On
    D-WAVE 2000Q, a typical restart takes between 1 and 60 seconds.
    Integer >= 0 (default = 10)

inner_rounds: the algorithm takes at most this many iterations between
    restart attempts; restart attempts are typically terminated due to
    max_no_improvement. Integer >= 0 (default = effectively infinite)

chainlength_patience: Maximum number of failed iterations to improve
    chainlengths in the current solution, where each iteration attempts
    to find an embedding for each variable of S such that it is adjacent
    to all its neighbours. Integer >= 0 (default = 10)

max_fill: Restricts the number of chains that can simultaneously
    incorporate the same qubit during the search. Integer >= 0 (default
    = effectively infinite)

threads: Maximum number of threads to use. Note that the
```

(continues on next page)

(continued from previous page)

```
    parallelization is only advantageous where the expected degree of
    variables is significantly greater than the number of threads.
    Integer >= 1 (default = 1)

return_overlap: This function returns an embedding whether or not qubits
    are used by multiple variables. Set this value to 1 to capture both
    return values to determine whether or not the returned embedding is
    valid. Logical 0/1 integer (default = 0)

skip_initialization: Skip the initialization pass. Note that this only
    works if the chains passed in through initial_chains and
    fixed_chains are semi-valid. A semi-valid embedding is a collection
    of chains such that every adjacent pair of variables (u,v) has a
    coupler (p,q) in the hardware graph where p is in chain(u) and q is
    in chain(v). This can be used on a valid embedding to immediately
    skip to the chainlength improvement phase. Another good source of
    semi-valid embeddings is the output of this function with the
    return_overlap parameter enabled. Logical 0/1 integer (default = 0)

verbose: Level of output verbosity. Integer < 4 (default = 0).
    When set to 0, the output is quiet until the final result.
    When set to 1, output looks like this:

        initialized
        max qubit fill 3; num maxfull qubits=3
        embedding trial 1
        max qubit fill 2; num maxfull qubits=21
        embedding trial 2
        embedding trial 3
        embedding trial 4
        embedding trial 5
        embedding found.
        max chain length 4; num max chains=1
        reducing chain lengths
        max chain length 3; num max chains=5

    When set to 2, outputs the information for lower levels and also
        reports progress on minor statistics (when searching for an
        embedding, this is when the number of maxfull qubits decreases;
        when improving, this is when the number of max chains decreases)
    When set to 3, report before each before each pass. Look here when
        tweaking `tries`, `inner_rounds`, and `chainlength_patience`
    When set to 4, report additional debugging information. By default,
        this package is built without this functionality. In the c++
        headers, this is controlled by the CPPDEBUG flag
    Detailed explanation of the output information:
        max qubit fill: largest number of variables represented in a qubit
        num maxfull: the number of qubits that has max overfill
        max chain length: largest number of qubits representing a single variable
        num max chains: the number of variables that has max chain size

initial_chains: Initial chains inserted into an embedding before
    fixed_chains are placed, which occurs before the initialization
    pass. These can be used to restart the algorithm in a similar state
    to a previous embedding; for example, to improve chainlength of a
    valid embedding or to reduce overlap in a semi-valid embedding (see
    skip_initialization) previously returned by the algorithm. Missing
```

(continues on next page)

(continued from previous page)

```
    or empty entries are ignored. A dictionary, where initial_chains[i]
    is a list of qubit labels.

fixed_chains: Fixed chains inserted into an embedding before the
    initialization pass. As the algorithm proceeds, these chains are not
    allowed to change. Missing or empty entries are ignored. A
    dictionary, where fixed_chains[i] is a list of qubit labels.

restrict_chains: Throughout the algorithm, we maintain the condition
    that chain[i] is a subset of restrict_chains[i] for each i, except
    those with missing or empty entries. A dictionary, where
    restrict_chains[i] is a list of qubit labels.

suspend_chains: This is a metafeature that is only implemented in the Python
    interface.  suspend_chains[i] is an iterable of iterables; for example
        suspend_chains[i] = [blob_1, blob_2],
    with each blob_j an iterable of target node labels.
    this enforces the following:
        for each suspended variable i,
            for each blob_j in the suspension of i,
                at least one qubit from blob_j will be contained in the
                chain for i

    we accomplish this trhough the following problem transformation
    for each iterable blob_j in suspend_chains[i],
        * add an auxiliary node Zij to both source and target graphs
        * set fixed_chains[Zij] = [Zij]
        * add the edge (i,Zij) to the source graph
        * add the edges (q,Zij) to the target graph for each q in blob_j
```

Chimera

Functionality for minor-embedding in Chimera-structured target graphs.

chimera.find_clique_embedding(k, m[, n, t, ...])	Find an embedding for a clique in a Chimera graph.
chimera.find_biclique_embedding(a, b, m[, ...])	Find an embedding for a biclique in a Chimera graph.
chimera.find_grid_embedding(dim, m[, n, t])	Find an embedding for a grid in a Chimera graph.

dwave.embedding.chimera.find_clique_embedding

find_clique_embedding(*k*, *m*, *n=None*, *t=None*, *target_edges=None*)

Find an embedding for a clique in a Chimera graph.

Given a target Chimera graph size, and a clique (fully connect graph), attempts to find an embedding.

Parameters

- **k** (*int/iterable*) – Clique to embed. If k is an integer, generates an embedding for a clique of size k labelled [0,k-1]. If k is an iterable, generates an embedding for a clique of size len(k), where iterable k is the variable labels.
- **m** (*int*) – Number of rows in the Chimera lattice.

- **n** (*int, optional, default=m*) – Number of columns in the Chimera lattice.
- **t** (*int, optional, default 4*) – Size of the shore within each Chimera tile.
- **target_edges** (*iterable[edge]*) – A list of edges in the target Chimera graph. Nodes are labelled as returned by `chimera_graph()`.

Returns An embedding mapping a clique to the Chimera lattice.

Return type dict

Examples

The first example finds an embedding for a K_4 complete graph in a single Chimera unit cell. The second for an alphanumerically labeled K_3 graph in 4 unit cells.

```
>>> from dwave.embedding.chimera import find_clique_embedding
...
>>> embedding = find_clique_embedding(4, 1, 1)
>>> embedding
{0: [4, 0], 1: [5, 1], 2: [6, 2], 3: [7, 3]}
```

```
>>> from dwave.embedding.chimera import find_clique_embedding
...
>>> embedding = find_clique_embedding(['a', 'b', 'c'], m=2, n=2, t=4)
>>> embedding
{'a': [20, 16], 'b': [21, 17], 'c': [22, 18]}
```

dwave.embedding.chimera.find_biclique_embedding

find_biclique_embedding(*a, b, m, n=None, t=None, target_edges=None*)

Find an embedding for a biclique in a Chimera graph.

Given a target Chimera graph size, and a biclique (a bipartite graph where every vertex in a set in connected to all vertices in the other set), attempts to find an embedding.

Parameters

- **a** (*int/iterable*) – Left shore of the biclique to embed. If a is an integer, generates an embedding for a biclique with the left shore of size a labelled [0,a-1]. If a is an iterable, generates an embedding for a biclique with the left shore of size len(a), where iterable a is the variable labels.
- **b** (*int/iterable*) – Right shore of the biclique to embed.If b is an integer, generates an embedding for a biclique with the right shore of size b labelled [0,b-1]. If b is an iterable, generates an embedding for a biclique with the right shore of size len(b), where iterable b provides the variable labels.
- **m** (*int*) – Number of rows in the Chimera lattice.
- **n** (*int, optional, default=m*) – Number of columns in the Chimera lattice.
- **t** (*int, optional, default 4*) – Size of the shore within each Chimera tile.
- **target_edges** (*iterable[edge]*) – A list of edges in the target Chimera graph. Nodes are labelled as returned by `chimera_graph()`.

Returns

A 2-tuple containing:

dict: An embedding mapping the left shore of the biclique to the Chimera lattice.

dict: An embedding mapping the right shore of the biclique to the Chimera lattice

Return type tuple

Examples

This example finds an embedding for an alphanumerically labeled biclique in a single Chimera unit cell.

```
>>> from dwave.embedding.chimera import find_biclique_embedding
...
>>> left, right = find_biclique_embedding(['a', 'b', 'c'], ['d', 'e'], 1, 1)
>>> print(left, right)
{'a': [4], 'b': [5], 'c': [6]} {'d': [0], 'e': [1]}
```

dwave.embedding.chimera.find_grid_embedding

find_grid_embedding(*dim*, *m*, *n=None*, *t=4*)

Find an embedding for a grid in a Chimera graph.

Given a target Chimera graph size, and grid dimensions, attempts to find an embedding.

Parameters

- **dim** (*iterable[int]*) – Sizes of each grid dimension. Length can be between 1 and 3.
- **m** (*int*) – Number of rows in the Chimera lattice.
- **n** (*int, optional, default=m*) – Number of columns in the Chimera lattice.
- **t** (*int, optional, default 4*) – Size of the shore within each Chimera tile.

Returns An embedding mapping a grid to the Chimera lattice.

Return type dict

Examples

This example finds an embedding for a 2x3 grid in a 12x12 lattice of Chimera unit cells.

```
>>> from dwave.embedding.chimera import find_grid_embedding
...
>>> embedding = find_grid_embedding([2, 3], m=12, n=12, t=4)
>>> embedding
{(0, 0): [0, 4],
 (0, 1): [8, 12],
 (0, 2): [16, 20],
 (1, 0): [96, 100],
 (1, 1): [104, 108],
 (1, 2): [112, 116]}
```

Pegasus

Functionality for minor-embedding in Pegasus-structured target graphs.

pegasus.find_clique_embedding(k[, m, ...])	Find an embedding of a k-sized clique on a Pegasus graph (target_graph).

dwave.embedding.pegasus.find_clique_embedding

find_clique_embedding(*k*, *m=None*, *target_graph=None*)

Find an embedding of a k-sized clique on a Pegasus graph (target_graph).

This clique is found by transforming the Pegasus graph into a K2,2 Chimera graph and then applying a Chimera clique finding algorithm. The results are then converted back in terms of Pegasus coordinates.

Note: If target_graph is None, m will be used to generate a m-by-m Pegasus graph. Hence m and target_graph cannot both be None.

Parameters

- **k** (int/iterable/`networkx.Graph`) – Number of members in the requested clique; list of nodes; a complete graph that you want to embed onto the target_graph
- **m** (*int*) – Number of tiles in a row of a square Pegasus graph
- **target_graph** (`networkx.Graph`) – A Pegasus graph

Returns A dictionary representing target_graphs's clique embedding. Each dictionary key represents a node in said clique. Each corresponding dictionary value is a list of pegasus coordinates that should be chained together to represent said node.

Return type dict

Embedding Utilities

Utilities

embed_bqm(source_bqm, embedding, ...[, ...])	Embed a binary quadratic model onto a target graph.
embed_ising(source_h, source_J, embedding, ...)	Embed an Ising problem onto a target graph.
embed_qubo(source_Q, embedding, target_adjacency)	Embed a QUBO onto a target graph.
unembed_sampleset(target_sampleset, ...[, ...])	Unembed the samples set.
chain_break_frequency(samples_like, embedding)	Determine the frequency of chain breaks in the given samples.

dwave.embedding.embed_bqm

embed_bqm(*source_bqm*, *embedding*, *target_adjacency*, *chain_strength=1.0*, *smear_vartype=None*)

Embed a binary quadratic model onto a target graph.

Parameters

- **source_bqm** (`BinaryQuadraticModel`) – Binary quadratic model to embed.
- **embedding** (*dict*) – Mapping from source graph to target graph as a dict of form {s: {t,

…}, …}, where s is a source-model variable and t is a target-model variable.

- **target_adjacency** (dict/networkx.Graph) – Adjacency of the target graph as a dict of form {t: Nt, …}, where t is a variable in the target graph and Nt is its set of neighbours.
- **chain_strength** (*float, optional*) – 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*.
- **smear_vartype** (Vartype, optional, default=None) – When a single variable is embedded, it's linear bias is 'smeared' evenly over the chain. This parameter determines whether the variable is smeared in SPIN or BINARY space. By default the embedding is done according to the given source_bqm.

Returns Target binary quadratic model.

Return type BinaryQuadraticModel

Examples

This example embeds a fully connected K_3 graph onto a square target graph. Embedding is accomplished by an edge contraction operation on the target graph: target-nodes 2 and 3 are chained to represent source-node c.

```
>>> import dimod
>>> import networkx as nx
>>> # Binary quadratic model for a triangular source graph
>>> bqm = dimod.BinaryQuadraticModel.from_ising({}, {('a', 'b'): 1, ('b', 'c'): 1,
↪ ('a', 'c'): 1})
>>> # Target graph is a graph
>>> target = nx.cycle_graph(4)
>>> # Embedding from source to target graphs
>>> embedding = {'a': {0}, 'b': {1}, 'c': {2, 3}}
>>> # Embed the BQM
>>> target_bqm = dimod.embed_bqm(bqm, embedding, target)
>>> target_bqm.quadratic[(0, 1)] == bqm.quadratic[('a', 'b')]
True
>>> target_bqm.quadratic
{(0, 1): 1.0, (0, 3): 1.0, (1, 2): 1.0, (2, 3): -1.0}
```

This example embeds a fully connected K_3 graph onto the target graph of a dimod reference structured sampler, *StructureComposite*, using the dimod reference *ExactSolver* sampler with a square graph specified. Target-nodes 2 and 3 are chained to represent source-node c.

```
>>> import dimod
>>> # Binary quadratic model for a triangular source graph
>>> bqm = dimod.BinaryQuadraticModel.from_ising({}, {('a', 'b'): 1, ('b', 'c'): 1,
↪ ('a', 'c'): 1})
>>> # Structured dimod sampler with a structure defined by a square graph
>>> sampler = dimod.StructureComposite(dimod.ExactSolver(), [0, 1, 2, 3], [(0, 1),
↪ (1, 2), (2, 3), (0, 3)])
>>> # Embedding from source to target graph
>>> embedding = {'a': {0}, 'b': {1}, 'c': {2, 3}}
>>> # Embed the BQM
>>> target_bqm = dimod.embed_bqm(bqm, embedding, sampler.adjacency)
>>> # Sample
>>> samples = sampler.sample(target_bqm)
>>> samples.record.sample
array([[-1, -1, -1, -1],
```

(continues on next page)

(continued from previous page)

```
        [ 1, -1, -1, -1],
        [ 1,  1, -1, -1],
        [-1,  1, -1, -1],
        [-1,  1,  1, -1],
>>> # Snipped above samples for brevity
```

dwave.embedding.embed_ising

embed_ising(*source_h*, *source_J*, *embedding*, *target_adjacency*, *chain_strength=1.0*)
Embed an Ising problem onto a target graph.

Parameters

- **source_h** (*dict[variable, bias]/list[bias]*) – Linear biases of the Ising problem. If a list, the list's indices are used as variable labels.
- **source_J** (*dict[(variable, variable), bias]*) – Quadratic biases of the Ising problem.
- **embedding** (*dict*) – Mapping from source graph to target graph as a dict of form {s: {t, . . . }, . . . }, where s is a source-model variable and t is a target-model variable.
- **target_adjacency** (dict/networkx.Graph) – Adjacency of the target graph as a dict of form {t: Nt, . . . }, where t is a target-graph variable and Nt is its set of neighbours.
- **chain_strength** (*float, optional*) – Magnitude of the quadratic bias (in SPIN-space) applied between variables to form a chain. Note that the energy penalty of chain breaks is 2 * *chain_strength*.

Returns

A 2-tuple:

dict[variable, bias]: Linear biases of the target Ising problem.

dict[(variable, variable), bias]: Quadratic biases of the target Ising problem.

Return type tuple

Examples

This example embeds a fully connected K_3 graph onto a square target graph. Embedding is accomplished by an edge contraction operation on the target graph: target-nodes 2 and 3 are chained to represent source-node c.

```
>>> import dimod
>>> import networkx as nx
>>> # Ising problem for a triangular source graph
>>> h = {}
>>> J = {('a', 'b'): 1, ('b', 'c'): 1, ('a', 'c'): 1}
>>> # Target graph is a square graph
>>> target = nx.cycle_graph(4)
>>> # Embedding from source to target graph
>>> embedding = {'a': {0}, 'b': {1}, 'c': {2, 3}}
>>> # Embed the Ising problem
>>> target_h, target_J = dimod.embed_ising(h, J, embedding, target)
>>> target_J[(0, 1)] == J[('a', 'b')]
True
```

(continues on next page)

(continued from previous page)

```
>>> target_J
{(0, 1): 1.0, (0, 3): 1.0, (1, 2): 1.0, (2, 3): -1.0}
```

This example embeds a fully connected K_3 graph onto the target graph of a dimod reference structured sampler, *StructureComposite*, using the dimod reference *ExactSolver* sampler with a square graph specified. Target-nodes 2 and 3 are chained to represent source-node c.

```
>>> import dimod
>>> # Ising problem for a triangular source graph
>>> h = {}
>>> J = {('a', 'b'): 1, ('b', 'c'): 1, ('a', 'c'): 1}
>>> # Structured dimod sampler with a structure defined by a square graph
>>> sampler = dimod.StructureComposite(dimod.ExactSolver(), [0, 1, 2, 3], [(0, 1),
↪ (1, 2), (2, 3), (0, 3)])
>>> # Embedding from source to target graph
>>> embedding = {'a': {0}, 'b': {1}, 'c': {2, 3}}
>>> # Embed the Ising problem
>>> target_h, target_J = dimod.embed_ising(h, J, embedding, sampler.adjacency)
>>> # Sample
>>> samples = sampler.sample_ising(target_h, target_J)
>>> for sample in samples.samples(n=3, sorted_by='energy'):
...     print(sample)
...
{0: 1, 1: -1, 2: -1, 3: -1}
{0: 1, 1: 1, 2: -1, 3: -1}
{0: -1, 1: 1, 2: -1, 3: -1}
```

dwave.embedding.embed_qubo

embed_qubo (*source_Q*, *embedding*, *target_adjacency*, *chain_strength=1.0*)
Embed a QUBO onto a target graph.

Parameters

- **source_Q** (*dict[(variable, variable), bias]*) – Coefficients of a quadratic unconstrained binary optimization (QUBO) model.
- **embedding** (*dict*) – Mapping from source graph to target graph as a dict of form {s: {t, . . . }, . . . }, where s is a source-model variable and t is a target-model variable.
- **target_adjacency** (dict/networkx.Graph) – Adjacency of the target graph as a dict of form {t: Nt, . . . }, where t is a target-graph variable and Nt is its set of neighbours.
- **chain_strength** (*float, optional*) – Magnitude of the quadratic bias (in SPIN-space) applied between variables to form a chain. Note that the energy penalty of chain breaks is 2 * *chain_strength*.

Returns Quadratic biases of the target QUBO.

Return type dict[(variable, variable), bias]

Examples

This example embeds a square source graph onto fully connected K_5 graph. Embedding is accomplished by an edge deletion operation on the target graph: target-node 0 is not used.

```
>>> import dimod
>>> import networkx as nx
>>> # QUBO problem for a square graph
>>> Q = {(1, 1): -4.0, (1, 2): 4.0, (2, 2): -4.0, (2, 3): 4.0,
...      (3, 3): -4.0, (3, 4): 4.0, (4, 1): 4.0, (4, 4): -4.0}
>>> # Target graph is a fully connected k5 graph
>>> K_5 = nx.complete_graph(5)
>>> 0 in K_5
True
>>> # Embedding from source to target graph
>>> embedding = {1: {4}, 2: {3}, 3: {1}, 4: {2}}
>>> # Embed the QUBO
>>> target_Q = dimod.embed_qubo(Q, embedding, K_5)
>>> (0, 0) in target_Q
False
>>> target_Q
{(1, 1): -4.0,
 (1, 2): 4.0,
 (2, 2): -4.0,
 (2, 4): 4.0,
 (3, 1): 4.0,
 (3, 3): -4.0,
 (4, 3): 4.0,
 (4, 4): -4.0}
```

This example embeds a square graph onto the target graph of a dimod reference structured sampler, *StructureComposite*, using the dimod reference *ExactSolver* sampler with a fully connected K_5 graph specified.

```
>>> import dimod
>>> import networkx as nx
>>> # QUBO problem for a square graph
>>> Q = {(1, 1): -4.0, (1, 2): 4.0, (2, 2): -4.0, (2, 3): 4.0,
...      (3, 3): -4.0, (3, 4): 4.0, (4, 1): 4.0, (4, 4): -4.0}
>>> # Structured dimod sampler with a structure defined by a K5 graph
>>> sampler = dimod.StructureComposite(dimod.ExactSolver(), list(K_5.nodes),␣
↪list(K_5.edges))
>>> sampler.adjacency
{0: {1, 2, 3, 4},
 1: {0, 2, 3, 4},
 2: {0, 1, 3, 4},
 3: {0, 1, 2, 4},
 4: {0, 1, 2, 3}}
>>> # Embedding from source to target graph
>>> embedding = {0: [4], 1: [3], 2: [1], 3: [2], 4: [0]}
>>> # Embed the QUBO
>>> target_Q = dimod.embed_qubo(Q, embedding, sampler.adjacency)
>>> # Sample
>>> samples = sampler.sample_qubo(target_Q)
>>> for datum in samples.data():
...     print(datum)
...
Sample(sample={1: 0, 2: 1, 3: 1, 4: 0}, energy=-8.0)
Sample(sample={1: 1, 2: 0, 3: 0, 4: 1}, energy=-8.0)
Sample(sample={1: 1, 2: 0, 3: 0, 4: 0}, energy=-4.0)
Sample(sample={1: 1, 2: 1, 3: 0, 4: 0}, energy=-4.0)
Sample(sample={1: 0, 2: 1, 3: 0, 4: 0}, energy=-4.0)
Sample(sample={1: 1, 2: 1, 3: 1, 4: 0}, energy=-4.0)
>>> # Snipped above samples for brevity
```

dwave.embedding.unembed_sampleset

unembed_sampleset(*target_sampleset*, *embedding*, *source_bqm*, *chain_break_method=None*, *chain_break_fraction=False*)

Unembed the samples set.

Construct a sample set for the source binary quadratic model (BQM) by unembedding the given samples from the target BQM.

Parameters

- **target_sampleset** (`dimod.SampleSet`) – SampleSet from the target BQM.
- **embedding** (*dict*) – Mapping from source graph to target graph as a dict of form {s: {t, . . . }, . . . }, where s is a source variable and t is a target variable.
- **source_bqm** (`dimod.BinaryQuadraticModel`) – Source binary quadratic model.
- **chain_break_method** (*function, optional*) – Method used to resolve chain breaks. See `dwave.embedding.chain_breaks`.
- **chain_break_fraction** (*bool, optional, default=False*) – If True, a 'chain_break_fraction' field is added to the unembedded samples which report what fraction of the chains were broken before unembedding.

Returns

Return type `SampleSet`

Examples

```
>>> import dimod
...
>>> # say we have a bqm on a triangle and an embedding
>>> J = {('a', 'b'): -1, ('b', 'c'): -1, ('a', 'c'): -1}
>>> bqm = dimod.BinaryQuadraticModel.from_ising({}, J)
>>> embedding = {'a': [0, 1], 'b': [2], 'c': [3]}
...
>>> # and some samples from the embedding
>>> samples = [{0: -1, 1: -1, 2: -1, 3: -1},  # [0, 1] is unbroken
               {0: -1, 1: +1, 2: +1, 3: +1}]  # [0, 1] is broken
>>> energies = [-3, 1]
>>> embedded = dimod.SampleSet.from_samples(samples, dimod.SPIN, energies)
...
>>> # unembed
>>> samples = dwave.embedding.unembed_sampleset(embedded, embedding, bqm)
>>> samples.record.sample
array([[-1, -1, -1],
       [ 1,  1,  1]], dtype=int8)
```

dwave.embedding.chain_break_frequency

chain_break_frequency(*samples_like*, *embedding*)

Determine the frequency of chain breaks in the given samples.

Parameters

- **samples_like** (samples_like/dimod.SampleSet) – A collection of raw samples. 'samples_like' is an extension of NumPy's array_like. See `dimod.as_samples()`.
- **embedding** (*dict*) – Mapping from source graph to target graph as a dict of form {s: {t, ...}, ...}, where s is a source-model variable and t is a target-model variable.

Returns Frequency of chain breaks as a dict in the form {s: f, ...}, where s is a variable in the source graph, and frequency, a float, is the fraction of broken chains.

Return type dict

Examples

This example embeds a single source node, 'a', as a chain of two target nodes (0, 1) and uses `chain_break_frequency()` to show that out of two synthetic samples, one ([-1, +1]) represents a broken chain.

```
>>> import dimod
>>> import numpy as np
>>> samples = np.array([[-1, +1], [+1, +1]])
>>> embedding = {'a': {0, 1}}
>>> print(dimod.chain_break_frequency(samples, embedding)['a'])
0.5
```

This example embeds a single source node (0) as a chain of two target nodes (a, b) and uses `chain_break_frequency()` to show that out of two samples in a dimod response, one ({'a': 1, 'b': 0}) represents a broken chain.

```
>>> import dimod
...
>>> response = dimod.SampleSet.from_samples([{'a': 1, 'b': 0}, {'a': 0, 'b': 0}],
...                                         {'energy': [1, 0]}, {}, dimod.BINARY)
>>> embedding = {0: {'a', 'b'}}
>>> print(dimod.chain_break_frequency(response, embedding)[0])
0.5
```

Diagnostics

diagnose_embedding(emb, source, target)	A detailed diagnostic for minor embeddings.
is_valid_embedding(emb, source, target)	A simple (bool) diagnostic for minor embeddings.
verify_embedding(emb, source, target[, ...])	A simple (exception-raising) diagnostic for minor embeddings.

dwave.embedding.diagnose_embedding

diagnose_embedding(*emb*, *source*, *target*)

A detailed diagnostic for minor embeddings.

This diagnostic produces a generator, which lists all issues with *emb*. The errors are yielded in the form

ExceptionClass, arg1, arg2,...

where the arguments following the class are used to construct the exception object. User-friendly variants of this function are `is_valid_embedding()`, which returns a bool, and `verify_embedding()` which raises

the first observed error. All exceptions are subclasses of `EmbeddingError`.

Parameters

- **emb** (*dict*) – Dictionary mapping source nodes to arrays of target nodes.
- **source** (list/`networkx.Graph`) – Graph to be embedded as a NetworkX graph or a list of edges.
- **target** (list/`networkx.Graph`) – Graph being embedded into as a NetworkX graph or a list of edges.

Yields *One of* – `MissingChainError`, snode: a source node label that does not occur as a key of *emb*, or for which emb[snode] is empty

`ChainOverlapError`, tnode, snode0, snode0: a target node which occurs in both *emb[snode0]* and *emb[snode1]*

`DisconnectedChainError`, snode: a source node label whose chain is not a connected subgraph of *target*

`InvalidNodeError`, tnode, snode: a source node label and putative target node label which is not a node of *target*

`MissingEdgeError`, snode0, snode1: a pair of source node labels defining an edge which is not present between their chains

dwave.embedding.is_valid_embedding

is_valid_embedding(*emb*, *source*, *target*)

A simple (bool) diagnostic for minor embeddings.

See `diagnose_embedding()` for a more detailed diagnostic / more information.

Parameters

- **emb** (*dict*) – a dictionary mapping source nodes to arrays of target nodes
- **source** (*graph or edgelist*) – the graph to be embedded
- **target** (*graph or edgelist*) – the graph being embedded into

Returns True if *emb* is valid.

Return type bool

dwave.embedding.verify_embedding

verify_embedding(*emb*, *source*, *target*, *ignore_errors=()*)

A simple (exception-raising) diagnostic for minor embeddings.

See `diagnose_embedding()` for a more detailed diagnostic / more information.

Parameters

- **emb** (*dict*) – a dictionary mapping source nodes to arrays of target nodes
- **source** (*graph or edgelist*) – the graph to be embedded
- **target** (*graph or edgelist*) – the graph being embedded into

Raises

- `EmbeddingError` – a catch-all class for the below
- `MissingChainError` – in case a key is missing from *emb*, or the associated chain is empty
- `ChainOverlapError` – in case two chains contain the same target node
- `DisconnectedChainError` – in case a chain is disconnected
- `InvalidNodeError` – in case a chain contains a node label not found in *target*
- `MissingEdgeError` – in case a source edge is not represented by any target edges

Returns True (if no exception is raised)

Return type bool

Chain-Break Resolution

Generators

chain_breaks.discard(samples, chains)	Discard broken chains.
chain_breaks.majority_vote(samples, chains)	Use the most common element in broken chains.
chain_breaks.weighted_random(samples, chains)	Determine the sample values of chains by weighed random choice.

dwave.embedding.chain_breaks.discard

discard(*samples*, *chains*)

Discard broken chains.

Parameters

- **samples** (*array_like*) – Samples as a nS x nV array_like object where nS is the number of samples and nV is the number of variables. The values should all be 0/1 or -1/+1.
- **chains** (*list[array_like]*) – List of chains of length nC where nC is the number of chains. Each chain should be an array_like collection of column indices in samples.

Returns

A 2-tuple containing:

`numpy.ndarray`: An array of unembedded samples. Broken chains are discarded. The array has dtype 'int8'.

`numpy.ndarray`: The indicies of the rows with unbroken chains.

Return type tuple

Examples

This example unembeds two samples that chains nodes 0 and 1 to represent a single source node. The first sample has an unbroken chain, the second a broken chain.

```
>>> import dimod
>>> import numpy as np
...
>>> chains = [(0, 1), (2,)]
>>> samples = np.array([[1, 1, 0], [1, 0, 0]], dtype=np.int8)
>>> unembedded, idx = dwave.embedding.discard(samples, chains)
>>> unembedded
array([[1, 0]], dtype=int8)
>>> idx
array([0])
```

dwave.embedding.chain_breaks.majority_vote

majority_vote(*samples*, *chains*)

Use the most common element in broken chains.

Parameters

- **samples** (*array_like*) – Samples as a nS x nV array_like object where nS is the number of samples and nV is the number of variables. The values should all be 0/1 or -1/+1.
- **chains** (*list[array_like]*) – List of chains of length nC where nC is the number of chains. Each chain should be an array_like collection of column indices in samples.

Returns

A 2-tuple containing:

`numpy.ndarray`: A nS x nC array of unembedded samples. The array has dtype 'int8'. Where there is a chain break, the value is chosen to match the most common value in the chain. For broken chains without a majority, the value is chosen arbitrarily.

`numpy.ndarray`: Equivalent to `np.arange(nS)` because all samples are kept and no samples are added.

Return type tuple

Examples

This example unembeds samples from a target graph that chains nodes 0 and 1 to represent one source node and nodes 2, 3, and 4 to represent another. Both samples have one broken chain, with different majority values.

```
>>> import dimod
>>> import numpy as np
...
>>> chains = [(0, 1), (2, 3, 4)]
>>> samples = np.array([[1, 1, 0, 0, 1], [1, 1, 1, 0, 1]], dtype=np.int8)
>>> unembedded, idx = dwave.embedding.majority_vote(samples, chains)
>>> unembedded
array([[1, 0],
       [1, 1]], dtype=int8)
>>> idx
array([0, 1])
```

dwave.embedding.chain_breaks.weighted_random

weighted_random(*samples*, *chains*)
Determine the sample values of chains by weighed random choice.

Parameters

- **samples** (*array_like*) – Samples as a nS x nV array_like object where nS is the number of samples and nV is the number of variables. The values should all be 0/1 or -1/+1.
- **chains** (*list[array_like]*) – List of chains of length nC where nC is the number of chains. Each chain should be an array_like collection of column indices in samples.

Returns

A 2-tuple containing:

`numpy.ndarray`: A nS x nC array of unembedded samples. The array has dtype 'int8'. Where there is a chain break, the value is chosen randomly, weighted by frequency of the chain's value.

`numpy.ndarray`: Equivalent to `np.arange(nS)` because all samples are kept and no samples are added.

Return type tuple

Examples

This example unembeds samples from a target graph that chains nodes 0 and 1 to represent one source node and nodes 2, 3, and 4 to represent another. The sample has broken chains for both source nodes.

```
>>> import dimod
>>> import numpy as np
...
>>> chains = [(0, 1), (2, 3, 4)]
>>> samples = np.array([[1, 0, 1, 0, 1]], dtype=np.int8)
>>> unembedded, idx = dwave.embedding.weighted_random(samples, chains)
>>> unembedded
array([[1, 1]], dtype=int8)
>>> idx
array([0, 1])
```

Callable Objects

chain_breaks.MinimizeEnergy(bqm, embedding)	Determine the sample values of broken chains by minimizing local energy.

dwave.embedding.chain_breaks.MinimizeEnergy

class MinimizeEnergy(*bqm*, *embedding*)
Determine the sample values of broken chains by minimizing local energy.

Parameters

- **bqm** (`BinaryQuadraticModel`) – The binary quadratic model associated with the source graph.

- **embedding** (*dict*) – Mapping from source graph to target graph as a dict of form {s: [t, ...], ...}, where s is a source-model variable and t is a target-model variable.

Examples

This example embeds from a triangular graph to a square graph, chaining target-nodes 2 and 3 to represent source-node c, and unembeds minimizing the energy for the samples. The first two sample have unbroken chains, the second two have broken chains.

```
>>> import dimod
>>> import numpy as np
...
>>> h = {'a': 0, 'b': 0, 'c': 0}
>>> J = {('a', 'b'): 1, ('b', 'c'): 1, ('a', 'c'): 1}
>>> bqm = dimod.BinaryQuadraticModel.from_ising(h, J)
>>> embedding = {'a': [0], 'b': [1], 'c': [2, 3]}
>>> cbm = dwave.embedding.MinimizeEnergy(bqm, embedding)
>>> samples = np.array([[+1, -1, +1, +1],
...                     [-1, -1, -1, -1],
...                     [-1, -1, +1, -1],
...                     [+1, +1, -1, +1]], dtype=np.int8)
>>> chains = [embedding['a'], embedding['b'], embedding['c']]
>>> unembedded, idx = cbm(samples, chains)
>>> unembedded
array([[ 1, -1,  1],
       [-1, -1, -1],
       [-1, -1,  1],
       [ 1,  1, -1]], dtype=int8)
>>> idx
array([0, 1, 2, 3])
```

__call__(*samples*, *chains*)

Parameters

- **samples** (*array_like*) – Samples as a nS x nV array_like object where nS is the number of samples and nV is the number of variables. The values should all be 0/1 or -1/+1.
- **chains** (*list[array_like]*) – List of chains of length nC where nC is the number of chains. Each chain should be an array_like collection of column indices in samples.

Returns

A 2-tuple containing:

`numpy.ndarray`: A nS x nC array of unembedded samples. The array has dtype 'int8'. Where there is a chain break, the value is chosen by greedy energy descent.

`numpy.ndarray`: Equivalent to `np.arange(nS)` because all samples are kept and no samples are added.

Return type tuple

Exceptions

`exceptions.EmbeddingError`	Base class for all embedding exceptions.
`exceptions.MissingChainError`(snode)	Raised if a node in the source graph has no associated chain.
`exceptions.ChainOverlapError`(tnode, snode0, . . .)	Raised if two source nodes have an overlapping chain.
`exceptions.DisconnectedChainError`(snode)	Raised if a chain is not connected in the target graph.
`exceptions.InvalidNodeError`(snode, tnode)	Raised if a chain contains a node not in the target graph.
`exceptions.MissingEdgeError`(snode0, snode1)	Raised when two source nodes sharing an edge to not have a corresponding edge between their chains.

dwave.embedding.exceptions.EmbeddingError

exception EmbeddingError
Base class for all embedding exceptions.

dwave.embedding.exceptions.MissingChainError

exception MissingChainError(*snode*)
Raised if a node in the source graph has no associated chain.

Parameters **snode** – The source node with no associated chain.

dwave.embedding.exceptions.ChainOverlapError

exception ChainOverlapError(*tnode*, *snode0*, *snode1*)
Raised if two source nodes have an overlapping chain.

Parameters

- **tnode** – Location where the chains overlap.
- **snode0** – First source node with overlapping chain.
- **snode1** – Second source node with overlapping chain.

dwave.embedding.exceptions.DisconnectedChainError

exception DisconnectedChainError(*snode*)
Raised if a chain is not connected in the target graph.

Parameters **snode** – The source node associated with the broken chain.

dwave.embedding.exceptions.InvalidNodeError

exception InvalidNodeError(*snode*, *tnode*)
Raised if a chain contains a node not in the target graph.

Parameters

- **snode** – The source node associated with the chain.
- **tnode** – The node in the chain not in the target graph.

dwave.embedding.exceptions.MissingEdgeError

exception MissingEdgeError(*snode0*, *snode1*)

Raised when two source nodes sharing an edge to not have a corresponding edge between their chains.

Parameters

- **snode0** – First source node.
- **snode1** – Second source node.

1.3 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-drivers` (previously also called `dwave-system-tuning`). This dependency has a restricted license and has been made optional as of v0.3.0, but is highly recommended. To view the license details:

```
from dwave.drivers 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 install
```

Note that installing from source installs `dwave-drivers`. To uninstall the proprietary components:

```
pip uninstall dwave-drivers
```

1.4 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.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

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.

1.8 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.

1.9 Leap

Leap, launched in 2018, is the real-time Quantum Application Environment from D-Wave Systems Inc. Leap brings quantum computing to the real world by providing cloud access to our systems, for free, to anyone who wants to give it a try. Learn about the types of problems that the D-Wave quantum computer can solve, run interactive demos and coding examples on the quantum computer, contribute your coding ideas, and join the growing conversation in our community of like-minded users at Leap.

1.10 D-Wave System Documentation

D-Wave System Documentation describes the D-Wave system, its properties and parameters, and provides information on solving problems using a D-Wave system. Learn about the D-Wave quantum computer at D-Wave System Documentation.

Python Module Index

d

dwave.system.composites.cutoffcomposite, 36
dwave.system.composites.virtual_graph, 29
dwave.system.samplers.dwave_sampler, 5

Index

Symbols

__call__() (MinimizeEnergy method), 59

A

adjacency (DWaveSampler attribute), 10
adjacency (FixedEmbeddingComposite attribute), 19
adjacency (LazyFixedEmbeddingComposite attribute), 22
adjacency (TilingComposite attribute), 27
adjacency (VirtualGraphComposite attribute), 34

C

chain_break_frequency() (in module dwave.embedding), 53
ChainOverlapError, 60
child (CutOffComposite attribute), 37
child (EmbeddingComposite attribute), 14
child (FixedEmbeddingComposite attribute), 18
child (PolyCutOffComposite attribute), 39
child (TilingComposite attribute), 25
child (VirtualGraphComposite attribute), 32
children (CutOffComposite attribute), 37
children (EmbeddingComposite attribute), 14
children (FixedEmbeddingComposite attribute), 18
children (PolyCutOffComposite attribute), 40
children (TilingComposite attribute), 25
children (VirtualGraphComposite attribute), 32
CutOffComposite (class in dwave.system.composites.cutoffcomposite), 36

D

diagnose_embedding() (in module dwave.embedding), 54
discard() (in module dwave.embedding.chain_breaks), 56
DisconnectedChainError, 60
dwave.system.composites.cutoffcomposite (module), 36
dwave.system.composites.virtual_graph (module), 29
dwave.system.samplers.dwave_sampler (module), 5
DWaveSampler (class in dwave.system.samplers), 5

E

edgelist (DWaveSampler attribute), 9
edgelist (FixedEmbeddingComposite attribute), 19
edgelist (LazyFixedEmbeddingComposite attribute), 22
edgelist (TilingComposite attribute), 27
edgelist (VirtualGraphComposite attribute), 33
embed_bqm() (in module dwave.embedding), 48
embed_ising() (in module dwave.embedding), 50
embed_qubo() (in module dwave.embedding), 51
EmbeddingComposite (class in dwave.system.composites), 13
EmbeddingError, 60

F

find_biclique_embedding() (in module dwave.embedding.chimera), 46
find_clique_embedding() (in module dwave.embedding.chimera), 45
find_clique_embedding() (in module dwave.embedding.pegasus), 48
find_embedding() (in module minorminer), 43
find_grid_embedding() (in module dwave.embedding.chimera), 47
FixedEmbeddingComposite (class in dwave.system.composites), 17

I

InvalidNodeError, 60
is_valid_embedding() (in module dwave.embedding), 55

L

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

M

majority_vote() (in module dwave.embedding.chain_breaks), 57
MinimizeEnergy (class in dwave.embedding.chain_breaks), 58

MissingChainError, 60
MissingEdgeError, 61

N

nodelist (DWaveSampler attribute), 9
nodelist (FixedEmbeddingComposite attribute), 19
nodelist (LazyFixedEmbeddingComposite attribute), 22
nodelist (TilingComposite attribute), 26
nodelist (VirtualGraphComposite attribute), 33

P

parameters (CutOffComposite attribute), 38
parameters (DWaveSampler attribute), 8
parameters (EmbeddingComposite attribute), 15
parameters (FixedEmbeddingComposite attribute), 18
parameters (LazyFixedEmbeddingComposite attribute), 22
parameters (PolyCutOffComposite attribute), 41
parameters (TilingComposite attribute), 25
parameters (VirtualGraphComposite attribute), 31
PolyCutOffComposite (class in dwave.system.composites.cutoffcomposite), 39
properties (CutOffComposite attribute), 38
properties (DWaveSampler attribute), 8
properties (EmbeddingComposite attribute), 14
properties (FixedEmbeddingComposite attribute), 17
properties (LazyFixedEmbeddingComposite attribute), 22
properties (PolyCutOffComposite attribute), 40
properties (TilingComposite attribute), 25
properties (VirtualGraphComposite attribute), 30

S

sample() (CutOffComposite method), 38
sample() (DWaveSampler method), 11
sample() (EmbeddingComposite method), 16
sample() (FixedEmbeddingComposite method), 20
sample() (LazyFixedEmbeddingComposite method), 23
sample() (TilingComposite method), 28
sample() (VirtualGraphComposite method), 35
sample_hising() (PolyCutOffComposite method), 41
sample_hubo() (PolyCutOffComposite method), 41
sample_ising() (CutOffComposite method), 38
sample_ising() (DWaveSampler method), 11
sample_ising() (EmbeddingComposite method), 16
sample_ising() (FixedEmbeddingComposite method), 21
sample_ising() (LazyFixedEmbeddingComposite method), 24
sample_ising() (TilingComposite method), 29
sample_ising() (VirtualGraphComposite method), 36
sample_poly() (PolyCutOffComposite method), 41
sample_qubo() (CutOffComposite method), 39
sample_qubo() (DWaveSampler method), 11
sample_qubo() (EmbeddingComposite method), 17
sample_qubo() (FixedEmbeddingComposite method), 21
sample_qubo() (LazyFixedEmbeddingComposite method), 24
sample_qubo() (TilingComposite method), 29
sample_qubo() (VirtualGraphComposite method), 36
structure (DWaveSampler attribute), 10
structure (FixedEmbeddingComposite attribute), 20
structure (LazyFixedEmbeddingComposite attribute), 23
structure (TilingComposite attribute), 27
structure (VirtualGraphComposite attribute), 34

T

TilingComposite (class in dwave.system.composites), 24

U

unembed_sampleset() (in module dwave.embedding), 53

V

validate_anneal_schedule() (DWaveSampler method), 12
verify_embedding() (in module dwave.embedding), 55
VirtualGraphComposite (class in dwave.system.composites), 29

W

weighted_random() (in module dwave.embedding.chain_breaks), 58