dwave-system Documentation

Release 0.2.3

D-Wave Systems Inc

Apr 02, 2018

Contents

Note: This is an alpha release of this package.

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

CHAPTER 1

Documentation

1.1 Reference Documentation

Release 0.2.3

Date Apr 02, 2018

1.1.1 Samplers

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

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

Release 0.2.3

Date Apr 02, 2018

D-Wave Sampler

<<<<<<< HEAD Class =====

Overview

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

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

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

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

Parameters

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

Examples

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

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

Sampler Properties

`DWaveSampler.properties`	*dict* – D-Wave solver properties as returned by a SAPI query.
`DWaveSampler.parameters`	*dict[str, list]* – D-Wave solver parameters in the form of a dict, where keys are

dwave.system.samplers.DWaveSampler.properties

`DWaveSampler.`**`properties`**

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

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

Examples

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

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

dwave.system.samplers.DWaveSampler.parameters

`DWaveSampler.`**`parameters`**

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

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

Examples

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

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

Structured Sampler Properties

`DWaveSampler.nodelist`	*list* – List of active qubits for the D-Wave solver.
`DWaveSampler.edgelist`	*list* – List of active couplers for the D-Wave solver.
`DWaveSampler.adjacency`	*dict[variable, set]* – The adjacency structure.

Continued on next page

Table 1.2 – continued from previous page

DWaveSampler.structure	A namedtuple Structure(nodelist, edgelist, adjacency)

dwave.system.samplers.DWaveSampler.nodelist

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

Examples

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

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

dwave.system.samplers.DWaveSampler.edgelist

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

Examples

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

```
>>> from dwave.system.samplers import DWaveSampler
>>> sampler = DWaveSampler()
>>> sampler.edgelist
[(0, 4),
 (0, 5),
 (0, 6),
 (0, 7),
 (0, 128),
 (1, 4),
 (1, 5),
 (1, 6),
 (1, 7),
 (1, 129),
 (2, 4),
# Snipped above response for brevity
```

dwave.system.samplers.DWaveSampler.adjacency

`DWaveSampler.`**`adjacency`**

dict[variable, set] – The adjacency structure.

Examples

```
>>> class StructuredObject(dimod.Structured):
...     @property
...     def nodelist(self):
...         return [0, 1, 2]
...
...     @property
...     def edgelist(self):
...         return [(0, 1), (1, 2)]
>>> test_obj = StructuredObject()
>>> for u, v in test_obj.edgelist:
...     assert u in test_obj.adjacency[v]
...     assert v in test_obj.adjacency[u]
```

dwave.system.samplers.DWaveSampler.structure

`DWaveSampler.`**`structure`**

A `namedtuple` `Structure(nodelist, edgelist, adjacency)`

Methods

`DWaveSampler.sample`(bqm, **parameters)	Samples from the given bqm using the instantiated sample method.
`DWaveSampler.sample_ising`(h, J, **kwargs)	Sample from the provided Ising model.
`DWaveSampler.sample_qubo`(Q, **kwargs)	Sample from the provided QUBO.

dwave.system.samplers.DWaveSampler.sample

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

Samples from the given bqm using the instantiated sample method.

dwave.system.samplers.DWaveSampler.sample_ising

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

Sample from the provided Ising model.

Parameters

- **h** (*`list/dict`*) – Linear biases of the Ising model. If a list, the list's indices are used as variable labels.
- **J** (*`dict[(int, int)`*) – float]): Quadratic biases of the Ising model.

- ****kwargs** – Optional keyword arguments for the sampling method, specified per solver in *DWaveSampler.parameters*

Returns dimod.Response

Examples

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

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

dwave.system.samplers.DWaveSampler.sample_qubo

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

Sample from the provided QUBO.

Parameters

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

Returns dimod.Response

Examples

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

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

1.1.2 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.

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

For example, the D-Wave system is Chimera-structured (a particular architecture of sparsely connected qubits) and so any arbitrarily posed binary quadratic problem 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.

Release 0.2.3

Date Apr 02, 2018

EmbeddingComposite

Class

Because the D-Wave System is Chimera-structured but most problems of application interest are not, it is convenient to be able to map from a structured sampler to an unstructured one.

A structured sampler is one that can only solver problems that map to a specific graph (see structured)

The `EmbeddingComposite` uses the minorminer library to map unstructured problems to a structured sampler.

class EmbeddingComposite(*child_sampler*)

Composite to map unstructured problems to a structured sampler.

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

Sampler Properties

`EmbeddingComposite.properties`	*dict* – Contains one key `'child_properties'` which has a copy of the child sampler's properties.
`EmbeddingComposite.parameters`	*dict[str, list]* – The keys are the keyword parameters accepted by the child sampler.

dwave.system.composites.EmbeddingComposite.properties

EmbeddingComposite.**properties**

dict – Contains one key `'child_properties'` which has a copy of the child sampler's properties.

dwave.system.composites.EmbeddingComposite.parameters

EmbeddingComposite.**parameters**

dict[str, list] – The keys are the keyword parameters accepted by the child sampler.

Composite Properties

`EmbeddingComposite.children`	*list* – Contains the single wrapped structured sampler.
`EmbeddingComposite.child`	The first child in `children`.

dwave.system.composites.EmbeddingComposite.children

EmbeddingComposite.**children**

list – Contains the single wrapped structured sampler.

dwave.system.composites.EmbeddingComposite.child

EmbeddingComposite.**child**

The first child in `children`.

Methods

`EmbeddingComposite.sample`(bqm, **parameters)	Samples from the given bqm using the instantiated sample method.
`EmbeddingComposite.sample_ising`(h, J, ...)	Sample from the provided unstructured Ising model.
`EmbeddingComposite.sample_qubo`(Q, **parameters)	Samples from the given QUBO using the instantiated sample method.

dwave.system.composites.EmbeddingComposite.sample

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

Samples from the given bqm using the instantiated sample method.

dwave.system.composites.EmbeddingComposite.sample_ising

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

Sample from the provided unstructured Ising model.

Parameters

- **h** (*list/dict*) – Linear terms of the model.
- **J** (*dict of (int, int)*) – float): Quadratic terms of the model.
- ****parameters** – Parameters for the sampling method, specified by the child sampler.

Returns `dimod.Response`

dwave.system.composites.EmbeddingComposite.sample_qubo

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

Samples from the given QUBO using the instantiated sample method.

TilingComposite

Class

Tiles many smaller problems across a larger Chimera-structured sampler.

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

Composite to tile a small problem across a Chimera-structured sampler. A problem that can fit on a small Chimera graph can be replicated across a larger Chimera graph to get samples from multiple areas of the system in one call. For example, a 2x2 Chimera lattice could be tiled 64 times (8x8) on a fully-yielded D-Wave 2000Q system (16x16).

Parameters

- **sampler** (`dimod.Sampler`) – A structured dimod sampler to be wrapped.
- **sub_m** (*int*) – The number of rows in the sub-Chimera lattice.
- **sub_n** (*int*) – The number of columns in the sub-Chimera lattice.
- **t** (*int*) – The size of the shore within each Chimera cell.

Sampler Properties

TilingComposite.properties	*dict* – Contains one key `'child_properties'` which has a copy of the child sampler's properties.
TilingComposite.parameters	*dict[str, list]* – The keys are the keyword parameters accepted by the child sampler.

dwave.system.composites.TilingComposite.properties

TilingComposite.**properties = None**

dict – Contains one key `'child_properties'` which has a copy of the child sampler's properties.

dwave.system.composites.TilingComposite.parameters

TilingComposite.**parameters = None**

dict[str, list] – The keys are the keyword parameters accepted by the child sampler.

Composite Properties

TilingComposite.children	*list* – Contains the single wrapped structured sampler.
TilingComposite.child	The first child in `children`.

dwave.system.composites.TilingComposite.children

TilingComposite.**children = None**

list – Contains the single wrapped structured sampler.

dwave.system.composites.TilingComposite.child

TilingComposite.**child**
The first child in `children`.

Structure Properties

TilingComposite.nodelist	*list* – The nodes available to the sampler.
TilingComposite.edgelist	*list* – The edges available to the sampler.
TilingComposite.adjacency	*dict[variable, set]* – The adjacency structure.
TilingComposite.structure	A `namedtuple` `Structure(nodelist, edgelist, adjacency)`

dwave.system.composites.TilingComposite.nodelist

TilingComposite.**nodelist = None**
list – The nodes available to the sampler.

dwave.system.composites.TilingComposite.edgelist

TilingComposite.**edgelist = None**
list – The edges available to the sampler.

dwave.system.composites.TilingComposite.adjacency

TilingComposite.**adjacency**
dict[variable, set] – The adjacency structure.

Examples

```
>>> class StructuredObject(dimod.Structured):
...     @property
...     def nodelist(self):
...         return [0, 1, 2]
...
...     @property
...     def edgelist(self):
...         return [(0, 1), (1, 2)]
>>> test_obj = StructuredObject()
>>> for u, v in test_obj.edgelist:
...     assert u in test_obj.adjacency[v]
...     assert v in test_obj.adjacency[u]
```

dwave.system.composites.TilingComposite.structure

TilingComposite.**structure**
A `namedtuple` `Structure(nodelist, edgelist, adjacency)`

Methods

`TilingComposite.sample`(bqm, **parameters)	Samples from the given bqm using the instantiated sample method.
`TilingComposite.sample_ising`(h, J, **kwargs)	Sample from the sub-Chimera lattice.
`TilingComposite.sample_qubo`(Q, **parameters)	Samples from the given QUBO using the instantiated sample method.

dwave.system.composites.TilingComposite.sample

`TilingComposite.sample`(*bqm*, ***parameters*)

Samples from the given bqm using the instantiated sample method.

dwave.system.composites.TilingComposite.sample_ising

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

Sample from the sub-Chimera lattice.

Parameters

- **h** (*list/dict*) – Linear terms of the model.
- **J** (*dict of (int, int)*) – float): Quadratic terms of the model.
- ****kwargs** – Parameters for the sampling method, specified per solver.

Returns `dimod.Response`

dwave.system.composites.TilingComposite.sample_qubo

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

Samples from the given QUBO using the instantiated sample method.

VirtualGraphComposite

Class

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

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

Apply the VirtualGraph composite layer to the given solver.

Parameters

- **sampler** (*DWaveSampler*) – A dimod `dimod.Sampler`. Normally `DWaveSampler`, or a derived composite sampler. Other samplers in general will not work or will not make sense with this composite layer.
- **embedding** (*dict[hashable, iterable]*) – A mapping from a source graph to the given sampler's graph (the target graph).

- **chain_strength** (*float, optional, default=None*) – The desired chain strength. If None, will use the maximum available from the processor.
- **flux_biases** (*list/False/None, optional, default=None*) – The per-qubit flux bias offsets. If given, should be a list of lists. Each sublist should be of length 2 and is the variable and the flux bias offset associated with the variable. If *flux_biases* evaluates False, then no flux bias is applied or calculated. If None if given, the flux biases are pulled from the database or calculated empirically.
- **flux_bias_num_reads** (*int, optional, default=1000*) – The number of samples to collect per flux bias value.
- **flux_bias_max_age** (*int, optional, default=3600*) – The maximum age (in seconds) allowed for a previously calculated flux bias offset.

Sampler Properties

`VirtualGraphComposite.properties`	*dict* – Contains one key `'child_properties'` which has a copy of the child sampler's properties.
`VirtualGraphComposite.parameters`	The same parameters as are accepted by the child sampler with an additional parameter 'apply_flux_bias_offsets'.

dwave.system.composites.VirtualGraphComposite.properties

`VirtualGraphComposite.`**`properties = None`**

dict – Contains one key `'child_properties'` which has a copy of the child sampler's properties.

dwave.system.composites.VirtualGraphComposite.parameters

`VirtualGraphComposite.`**`parameters = None`**

The same parameters as are accepted by the child sampler with an additional parameter 'apply_flux_bias_offsets'.

Composite Properties

`VirtualGraphComposite.children`	*list* – A list containig the wrapped sampler.
`VirtualGraphComposite.child`	The first child in `children`.

dwave.system.composites.VirtualGraphComposite.children

`VirtualGraphComposite.`**`children = None`**

list – A list containig the wrapped sampler.

dwave.system.composites.VirtualGraphComposite.child

`VirtualGraphComposite.`**`child`**

The first child in `children`.

Structure Properties

`VirtualGraphComposite.nodelist`	*list* – The nodes available to the sampler.
`VirtualGraphComposite.edgelist`	*list* – The edges available to the sampler.
`VirtualGraphComposite.adjacency`	*dict[variable, set]* – The adjacency structure.
`VirtualGraphComposite.structure`	A namedtuple `Structure(nodelist, edgelist, adjacency)`

dwave.system.composites.VirtualGraphComposite.nodelist

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

list – The nodes available to the sampler.

dwave.system.composites.VirtualGraphComposite.edgelist

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

list – The edges available to the sampler.

dwave.system.composites.VirtualGraphComposite.adjacency

`VirtualGraphComposite.`**`adjacency = None`**

dict[variable, set] – The adjacency structure.

Examples

```
>>> class StructuredObject(dimod.Structured):
...     @property
...      def nodelist(self):
...         return [0, 1, 2]
...
...     @property
...     def edgelist(self):
...         return [(0, 1), (1, 2)]
>>> test_obj = StructuredObject()
>>> for u, v in test_obj.edgelist:
...     assert u in test_obj.adjacency[v]
...     assert v in test_obj.adjacency[u]
```

dwave.system.composites.VirtualGraphComposite.structure

`VirtualGraphComposite.`**`structure`**

A namedtuple `Structure(nodelist, edgelist, adjacency)`

Methods

VirtualGraphComposite.sample(bqm, **parameters)	Samples from the given bqm using the instantiated sample method.
VirtualGraphComposite.sample_ising(h, J[, ...])	Sample from the given Ising model.
VirtualGraphComposite.sample_qubo(Q, ...)	Samples from the given QUBO using the instantiated sample method.

dwave.system.composites.VirtualGraphComposite.sample

VirtualGraphComposite.**sample**(*bqm*, ***parameters*)
Samples from the given bqm using the instantiated sample method.

dwave.system.composites.VirtualGraphComposite.sample_ising

VirtualGraphComposite.**sample_ising**(*h*, *J*, *apply_flux_bias_offsets=True*, ***kwargs*)
Sample from the given Ising model.

Parameters

- **h** (*list/dict*) – Linear terms of the model.
- **J** (*dict of (int, int)*) – float): Quadratic terms of the model.
- **apply_flux_bias_offsets** (*bool, optional*) – If True, use the calculated flux_bias offsets (if available).
- ****kwargs** – Parameters for the sampling method, specified by the child sampler.

dwave.system.composites.VirtualGraphComposite.sample_qubo

VirtualGraphComposite.**sample_qubo**(*Q*, ***parameters*)
Samples from the given QUBO using the instantiated sample method.

1.2 Installation

Installation from PyPI:

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

Installation from source:

```
pip install -r requirements.txt --extra-index-url https://pypi.dwavesys.com/simple
python setup.py
```

Downloaded with this package is a dependency called dwave-system-tuning that has a restricted license. To view the license details:

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

To uninstall the proprietary components:

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

1.3 License

Apache License

Version 2.0, January 2004

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

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

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

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

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

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

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

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

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

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

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

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

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.
5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without

1.4 D-Wave

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

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

1.5 Ocean Overview

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

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

1.6 Contributing to Ocean

D-Wave welcomes contributions to Ocean projects.

See how to contribute at Ocean Contributors.

1.7 Glossary

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

Learn the relevant terminology at Ocean Glossary.

Python Module Index

d

Index