dwave-neal Documentation

Release 0.5.9

D-Wave Systems Inc

Jun 10, 2022

Contents

An implementation of a simulated annealing sampler.

A simulated annealing sampler can be used for approximate Boltzmann sampling or heuristic optimization. This implementation approaches the equilibrium distribution by performing updates at a sequence of increasing beta values, `beta_schedule`, terminating at the target beta. Each spin is updated once in a fixed order per point in the beta_schedule according to a Metropolis- Hastings update. When beta is large the target distribution concentrates, at equilibrium, over ground states of the model. Samples are guaranteed to match the equilibrium for long 'smooth' beta schedules.

For more information, see Kirkpatrick, S.; Gelatt Jr, C. D.; Vecchi, M. P. (1983). "Optimization by Simulated Annealing". Science. 220 (4598): 671–680

CHAPTER 1

Example Usage

```
import neal

sampler = neal.SimulatedAnnealingSampler()

h = {0: -1, 1: -1}
J = {(0, 1): -1}
sampleset = sampler.sample_ising(h, J)
```

CHAPTER 2

Documentation

Note: This documentation is for the latest version of dwave-neal. Documentation for the version currently installed by dwave-ocean-sdk is here: dwave-neal.

2.1 Introduction

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.

The `SimulatedAnnealingSampler` sampler implements the simulated annealing algorithm, based on the technique of cooling metal from a high temperature to improve its structure (annealing). This algorithm often finds good solutions to hard optimization problems.

2.2 Reference Documentation

Release 0.5.9

Date Jun 10, 2022

2.2.1 Simulated Annealing Sampler

A dimod sampler that uses the simulated annealing algorithm.

Class

class SimulatedAnnealingSampler
Simulated annealing sampler.

Also aliased as `Neal`.

Examples

This example solves a simple Ising problem.

```
>>> import neal
>>> sampler = neal.SimulatedAnnealingSampler()
>>> h = {'a': 0.0, 'b': 0.0, 'c': 0.0}
>>> J = {('a', 'b'): 1.0, ('b', 'c'): 1.0, ('a', 'c'): 1.0}
>>> sampleset = sampler.sample_ising(h, J, num_reads=10)
>>> print(sampleset.first.energy)
-1.0
```

Sampler Properties

SimulatedAnnealingSampler.properties	A dict containing any additional information about the sampler.
SimulatedAnnealingSampler.parameters	A dict where keys are the keyword parameters accepted by the sampler methods (allowed kwargs) and values are lists of *SimulatedAnnealingSampler.properties* relevant to each parameter.

neal.sampler.SimulatedAnnealingSampler.properties

SimulatedAnnealingSampler.**properties = None**
A dict containing any additional information about the sampler.

Examples

This example looks at the values set for a sampler property.

```
>>> import neal
>>> sampler = neal.SimulatedAnnealingSampler()
>>> sampler.properties['beta_schedule_options']
('linear', 'geometric', 'custom')
```

Type dict

neal.sampler.SimulatedAnnealingSampler.parameters

SimulatedAnnealingSampler.**parameters = None**
A dict where keys are the keyword parameters accepted by the sampler methods (allowed kwargs) and values are lists of *SimulatedAnnealingSampler.properties* relevant to each parameter.

See *SimulatedAnnealingSampler.sample()* for a description of the parameters.

Examples

This example looks at a sampler's parameters and some of their values.

```
>>> import neal
>>> sampler = neal.SimulatedAnnealingSampler()
>>> for kwarg in sorted(sampler.parameters):
...     print(kwarg)
beta_range
beta_schedule_type
initial_states
initial_states_generator
interrupt_function
num_reads
num_sweeps
num_sweeps_per_beta
seed
>>> sampler.parameters['beta_range']
[]
>>> sampler.parameters['beta_schedule_type']
['beta_schedule_options']
```

Type dict

Methods

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

neal.sampler.SimulatedAnnealingSampler.sample

SimulatedAnnealingSampler.**sample**(*bqm*, *beta_range=None*, *num_reads=None*, *num_sweeps=None*, *num_sweeps_per_beta=1*, *beta_schedule_type='geometric'*, *seed=None*, *interrupt_function=None*, *beta_schedule=None*, *initial_states=None*, *initial_states_generator='random'*, ***kwargs*)

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

Parameters

- **bqm** (dimod.BinaryQuadraticModel) – The binary quadratic model to be sampled.
- **beta_range** (*tuple or list, optional*) – A 2-tuple or list defining the beginning and end of the beta schedule, where beta is the inverse temperature. The schedule is interpolated within this range according to the value specified by beta_schedule_type. Default range is set based on the total bias associated with each node.
- **num_reads** (*int, optional, default=len(initial_states) or 1*) – Number of reads. Each read is generated by one run of the simulated annealing algorithm.

If *num_reads* is not explicitly given, it is selected to match the number of initial states given. If initial states are not provided, only one read is performed.

- **num_sweeps** (*int, optional, default=``len(beta_schedule)*num_sweeps_per_beta`` or 1000*) – Number of sweeps used in annealing. If no value is provided and `beta_schedule` is None the value is defaulted to 1000.
- **num_sweeps_per_beta** (*int, optional, default=1*) – Number of sweeps to perform at each beta. One sweep consists of a sequential Metropolis update of all spins.
- **beta_schedule_type** (*string, optional, default="geometric"*) – Beta schedule type, or how the beta values are interpolated between the given `beta_range`. Supported values are:
  - "linear"
  - "geometric"
  - "custom"

  "custom" is recommended for high-performance applications, which typically require optimizing beta schedules beyond those of the "linear" and "geometric" options, with bounds beyond those provided by default. `num_sweeps_per_beta` and `beta_schedule` fully specify a custom schedule.
- **beta_schedule** (*array-like, optional, default = None*) – Sequence of beta values swept. Format compatible with numpy.array(beta_schedule,dtype=float) required. Values should be non-negative.
- **seed** (*int, optional, default = None*) – Seed to use for the PRNG. Specifying a particular seed with a constant set of parameters produces identical results. If not provided, a random seed is chosen.
- **initial_states** (*samples-like, optional, default=None*) – One or more samples, each defining an initial state for all the problem variables. Initial states are given one per read, but if fewer than `num_reads` initial states are defined, additional values are generated as specified by `initial_states_generator`. See func:`.as_samples` for a description of "samples-like".
- **initial_states_generator** (*str, "none"/"tile"/"random", optional, default="random"*) – Defines the expansion of `initial_states` if fewer than `num_reads` are specified:
  - **"none":** If the number of initial states specified is smaller than `num_reads`, raises ValueError.
  - **"tile":** Reuses the specified initial states if fewer than `num_reads` or truncates if greater.
  - **"random":** Expands the specified initial states with randomly generated states if fewer than `num_reads` or truncates if greater.
- **interrupt_function** (*function, optional*) – If provided, interrupt_function is called with no parameters between each sample of simulated annealing. If the function returns True, then simulated annealing will terminate and return with all of the samples and energies found so far.

Returns `dimod.SampleSet`

Examples

This example runs simulated annealing on a binary quadratic model with some different input parameters.

```
>>> import dimod
>>> import neal
...
>>> sampler = neal.SimulatedAnnealingSampler()
>>> bqm = dimod.BinaryQuadraticModel({'a': .5, 'b': -.5},
...                                  {('a', 'b'): -1}, 0.0,
...                                  dimod.SPIN)
>>> # Run with default parameters
>>> sampleset = sampler.sample(bqm)
>>> # Run with specified parameters
>>> sampleset = sampler.sample(bqm, seed=1234,
...                            beta_range=[0.1, 4.2],
...                            num_sweeps=20,
...                            beta_schedule_type='geometric')
>>> # Reuse a seed
>>> a1 = next((sampler.sample(bqm, seed=88)).samples())['a']
>>> a2 = next((sampler.sample(bqm, seed=88)).samples())['a']
>>> a1 == a2
True
```

neal.sampler.SimulatedAnnealingSampler.sample_ising

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

Sample from an Ising model using the implemented sample method.

This method is inherited from the `Sampler` base class.

Converts the Ising model into a `BinaryQuadraticModel` and then calls *`sample()`*.

Parameters

- **`h`** (*`dict/list`*) – Linear biases of the Ising problem. If a dict, should be of the form *{v: bias, ... }* where is a spin-valued variable and *bias* is its associated bias. If a list, it is treated as a list of biases where the indices are the variable labels.
- **`J`** (*`dict[(variable, variable), bias]`*) – Quadratic biases of the Ising problem.
- **`**kwargs`** – See the implemented sampling for additional keyword definitions.

Returns `SampleSet`

See also:

`sample()`, *`sample_qubo()`*

neal.sampler.SimulatedAnnealingSampler.sample_qubo

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

Sample from a QUBO using the implemented sample method.

This method is inherited from the `Sampler` base class.

Converts the QUBO into a `BinaryQuadraticModel` and then calls *`sample()`*.

Parameters

- **Q** (*dict*) – Coefficients of a quadratic unconstrained binary optimization (QUBO) problem. Should be a dict of the form *{(u, v): bias, ... }* where *u*, *v*, are binary-valued variables and *bias* is their associated coefficient.
- ****kwargs** – See the implemented sampling for additional keyword definitions.

Returns SampleSet

See also:

sample(), *sample_ising()*

Alias

Neal
alias of *neal.sampler.SimulatedAnnealingSampler*

2.3 Installation

To install:

```
pip install dwave-neal
```

To build from source:

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

2.4 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.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search
- Glossary

Python Module Index

S

Index

N

P

S