ising Documentation

Konrad Jałowiecki, Marek Rams, Bartłomiej Gardas

Jan 21, 2019

Contents:

Ising is an open source package for exactly solving abritrary Ising model instances via exhaustive search. It can be used as an excellent tool for benchmarking other solvers or generating low energy spectra. The package is compatible with *NIX systems (and in principle should work on Windows too). **Ising** supports parallel computation via OpenMP or GPU, if it was build with CUDA support.

CHAPTER 1

Installation

1.1 Installing binary wheel from PyPI

If you are running a Linux system and are only interested in non GPU-enabled build, you can install binary wheel from PyPI as usual:

```
pip install ising
```

Unfortunately, we cannot provide GPU-enabled binary wheel due to a manylinux PEP-513 policy, as it is impossible to build **ising** on CentOS 5.

1.2 Building from source

If you are not running Linux and/or are interested in a GPU-enabled build, you need to build **ising** from source. The process is simple and requires running a single command. We highly recommend using virtual environment instead of installing the package into the global scope. Note that otherwise installing the package may require root privileges.

1.2.1 Prerequisites

To build **ising** you need the following:

- Virtually any C and C++ compiler,
- A Fortran compiler. The build script supports PGI, Intel, and gfortran compilers.
- Working CUDA toolkit. For CPU based implementation only its thrust library with `OMP_DEVICE_BACKEND` is used, but `nvcc` is still required for compiling sources. You can get around this requirement and use your local installation of `thrust` if you use GNU fortran compiler.
- `numpy` Python package installed in the same environment as is used to run the build process.
- Python development headers.

In addition, to build a GPU-enabled version you need PGI CUDA Fortran. Our package was tested against CUDA 9.2 and CUDA 10.0.

1.2.2 Building and installing

To build the **ising** package download its source code and run `install.py` script as follows:

```
python install.py --fcompiler=<fortran_compiler> [--usecuda]
```

where `<fortran_compiler>` is one of `pgi`, `intel`, `gfortran`. The `--usecuda` switch can be used to enable GPU support. Note that `--usecuda` requires `--fcompiler=pgi`.

The script should take care of building extensions and installing package, so after running the above command **ising** package should be ready to use.

CHAPTER 2

User guide

2.1 Introduction

The **ising** package allows to find a ground state (or, more generally, low energy spectrum) of an arbitrary Ising model. That is, it allows you to find the minimum of the following energy function

$$H(s_0, \ldots, s_n) = -\sum_{i,j=0}^{n} J_{ij} s_i s_j - \sum_{i=0}^{n} h_i s_i$$

where J_{ij} and h_i are arbitrary real coefficients and variables s_i under optimization are either -1 or 1.

2.2 Basic usage

The main functionallity of **ising** package is wrapped in `ising.search` function. As an example, suppose you want to find 4 lowest energy states of the following Ising model

$$H(s_0, s_1, s_2) = -2s_0 s_1 + 3s_1 s_2 + 2.5 s_2 s_3 - s_0$$

In that case you could run `ising.search` as follows

```
import ising

graph = {(0, 1): 2, (1, 2): -3, (2, 3): 2.5, (0, 0): 1}

result = ising.search(graph, num_states=4)
print(result.energies)
```

Note how the above model is specified as a dictionary:

- J_{ij} are specified as entries with key `(i, j)`.
- h_i are specified as entries with key `(i, i)`.

Read further to learn other input formats that **ising** can handle.

2.3 Supported input formats

There are three formats supported by *ising*:

- The dictionary format already presented in previous section.
- The *coefficients list format*. In this format coefficients are specified as a list of lists, in hich each row is of the form `[i, j, J_ij]` or ‘‘[i, i, h_i]‘.‘
- The *matrix* format. In this format you specify your coefficients as a matrix in which diagonal elements correspond to h_i and off-diagonal elements correspond to J_{ij}. The matrix can either be a list of lists or a *numpy* array.

Putting it in another way, here are equivalent ways of specifying graph from the above basic example

```
# coefficients list format
graph = [[0, 1, 2], [1, 2, -3], [0, 0, 1], [2, 3, 2.5]],
# matrix format: as list of lists or numpy array
graph = [[1, 2, 0, 0], [0, 0, -3, 0], [0, 0, 0, 2.5], [0, 0, 0, 0]],
graph = np.array([[1, 2, 0, 0], [0, 0, -3, 0], [0, 0, 0, 2.5], [0, 0, 0, 0]]),
```

Note that the *matrix* format requires your spins to be labelled with $0, \ldots, n$, other two formats are not restricted in this way.

Also note that since both J_{ij} and J_{ji} can be specified in all the formats it does not matter which one you choose. In fact, if you choose to specify both coefficients, both of them will be used. Therefore, using the following graphs would yield the same result as the previous example:

```
# coefficient list format
graph = [[0, 1, 1], [1, 0, 1], [1, 2, -3], [0, 0, 1], [2, 3, 2.5]],
# matrix format
graph = [[1, 1, 0, 0], [1, 0, -3, 0], [0, 0, 0, 2.5], [0, 0, 0, 0]]
```

2.4 Tweaking execution

You can use the following keyword arguments to `ising.search` to tweak its execution:

- `num_states`: integer specifying how many low-energy states should be found.
- `method`: indicating whether CPU (`method='CPU'`) or GPU (`method='GPU'`) implementation should be used. If not given, CPU implementation is used by default.
- `energies_only`: boolean indicating whether to return only energies (`True`) or also states corresponding to those energies (`False`). Default is `False`, set it to `True` if you don’t need states, as it should shorten the execution time.
- `chunk_size`: **ising** performs search in chunks of the size 2^k, where k is choosen as a largest number such that computations are feasible on the host. You can tweak this value to use other exponent if you choose so.

In addition, for CPU implementation, you can specify how many OMP threads will be used for computations using `OMP_NUM_THREADS` environmental variable.