ising Documentation

Konrad Jałowiecki, Marek Rams, Bartłomiej Gardas

Sep 17, 2019

Contents:

Ising is an open source package to solve abritrary spin-glass Ising models using exhaustive (brute force) search. It can serve as an excellent tool for benchmarking other solvers or generating low energy spectra (desirable e.g. for machine learning related tasks). The package is compatible with *NIX systems (and in principle should work on Windows too). **Ising** supports parallel computation via OpenMP or GPU, provided it has been 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 spin-glass Ising model. That is, with **ising** one can to find the minimum of the following energy function (i.e. Hamiltonian)

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

where J_{ij} and h_i are arbitrary real coefficients (interaction couplings and external biases, respectively) and variables s_i can admit one of two values, either $s_i = -1$ or $s_i = 1$.

2.2 Basic usage

The main functionallity of the **ising** package is wrapped in the `ising.search` function. For instance, suppose one would like to to find four lowest energy states given the following problem Hamiltonian,

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

To that end, one can simply 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 defined using a dictionary:

- Couplings, J_{ij}, are specified as its entries with the corresponding keys being `(i, j)`.
- Similarly, biases h_i are provided as the diagonal entries whose keys are `(i, i)`.

2.3 Other 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, where each row is of the form `[i, j, J_ij]` or `[i, i, h_i]`.
- The *matrix* format. In this format one specifies coefficients as a matrix where its 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.

To summarize, here are three equivalent ways to specify the problem graph

```
# 1) coefficients list format
graph = [[0, 1, 2], [1, 2, -3], [0, 0, 1], [2, 3, 2.5]],
# 2) matrix format: list of lists
graph = [[1, 2, 0, 0], [0, 0, -3, 0], [0, 0, 0, 2.5], [0, 0, 0, 0]],
# 3) matrix format: numpy array
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 spins variables to be labelled with $0, \ldots, n$, other two formats are not restricted in this way.

Since both couplings J_{ij} and J_{ji} can be specified in all three formats, it does not matter which one is chosen. In fact, if one provides both coefficients, both will be used. Therefore, specifing 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

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

- `num_states`: an integer specifying how many low-energy states to find.
- `method`: a flag indicating whether CPU (`method='CPU'`, default) or GPU (`method='GPU'`) implementation to invoke.
- `energies_only`: a boolean indicating whether both the energies and the states should be returned. Default is `False`.
- `chunk_size`: To fit into the host memory, **ising** performs search in fixed chunks of a given size 2^{chunk_size}.

In addition, when executing the CPU implementation, one can specify how many OpenMP threads to use for computations using `OMP_NUM_THREADS` environmental variable.