README.md

CPDAG Estimation using PC-Algorithm

Overview

This package provides functions to estimate a skeleton graph and a completed partially acyclic graph (CPDAG) from a data matrix that describes the appearance patterns of the graph nodes. The detailed algorithm is written in [Kalisch2007].

Code

The source code is available at https://github.com/keiichishima/pcalg

Bug Reports

README.md

Tests + Coverage Python passing | Test Benchmarks (Smoke Tests) passing | Test CUDA Device Functions passing | Lint Python passing | Lint CUDA passing

GPUCSL

`GPUCSL` is a python library for constraint-based causal structure learning using Graphics Processing Units (GPUs). In particular, it utilizes CUDA for GPU acceleration to speed up the well-known PC algorithm.

Features

- Performant GPU implementation of the PC algorithm (covers discrete, and multivariate normal distributed data), see General Notes;
- Multi-GPU support (experimental; for now, only for gaussian CI kernel);
- Easy to install and use, see Usage;
- A Command Line Interface (CLI);

- Took only 18 seconds to read in the entire first chromosome and generate a CPDAG (31 column x 4644 row dataset)
- Using Google Colab's Tesla T4 GPU with 16GB of memory.

  Result: DiGraph with 31 nodes and 73 edges

  [(0, 4), (0, 8), (0, 12), (0, 26), (0, 30), (1, 3), (1, 7), (1, 15), (1, 19), (1, 23), (1, 27), (2, 6), (2, 10), (2, 16), (2, 18), (2, 26), (3, 7), (3, 9), (3, 11), (3, 15), (3, 19), (3, 23), (3, 27), (4, 8), (4, 10), (4, 16), (4, 28), (4, 30), (5, 9), (5, 19), (5, 23), (5, 27), (6, 12), (6, 22), (7, 11), (7, 27), (8, 16), (8, 20), (8, 26), (9, 11), (9, 19), (9, 23), (9, 27), (10, 18), (10, 22), (11, 15), (11, 19), (11, 23), (11, 27), (12, 14), (12, 16), (12, 20), (13, 19), (13, 23), (13, 27), (14, 18), (14, 20), (14, 22), (15, 19), (15, 27), (16, 24), (16, 28), (17, 23), (18, 22), (20, 24), (20, 26), (20, 28), (21, 23), (22, 26), (23, 27), (25, 27), (26, 30), (28, 30)]

- *shortened to column index. Still have access to nodes corresponding to names

- Wanted to build off last week’s hypothesis
- Unable to figure out how to one hot encode a column of a ndarray (python)
- Chose to do all preprocessing in R and save the .csv
- No longer works

(new non-working array)	(old working array)
class: ndarray	class: ndarray
shape: (4644, 4658)	shape: (4644, 31)
strides: (37264, 8)	strides: (248, 8)
itemsize: 8	itemsize: 8
aligned: True	aligned: True
contiguous: True	contiguous: True
fortran: False	fortran: False
data pointer: 0x583a7dbdc2e0	data pointer: 0x5c5677b39a20
byteorder: little	byteorder: little
byteswap: False	byteswap: False
type: float64	type: float64

```
---> 34         result = f(*args, **kw)
     35         end = timer()
     36         duration = end - start

/usr/local/lib/python3.10/dist-packages/gpucsl/pc/gaussian_device_manager.py in compute_skeleton(self, 
    188             final_skeleton = self.execute_ci_workers_in_parallel(worker_function)
    189         else:
--> 190             worker_function(0, self)
    191             final_skeleton = self.get_skeleton_for_device_index(0).get()
    192

/usr/local/lib/python3.10/dist-packages/gpucsl/pc/discover_skeleton_gaussian.py in gaussian_ci_worker(d
     96                 )
     97
---> 98                 stream.synchronize()
     99                 log_time(
    100                     f"[T: {device_index} D: {device}]    Level {level} time",

cupy/cuda/stream.pyx in cupy.cuda.stream._BaseStream.synchronize()

cupy_backends/cuda/api/runtime.pyx in cupy_backends.cuda.api.runtime.streamSynchronize()

cupy_backends/cuda/api/runtime.pyx in cupy_backends.cuda.api.runtime.check_status()

CUDARuntimeError: cudaErrorIllegalAddress: an illegal memory access was encountered
```