PythiaPlotter Documentation

Release 0.5.0

Robin Aggleton

February 13, 2017

Contents:

1 PythiaPlotter **3**
- 1.1 What Is It . . . 3
- 1.2 What Can I Give It To Plot? . . . 4
- 1.3 What Do I Need: . . . 4
- 1.4 How Do I Get It: . . . 5
- 1.5 How Do I Use It: . . . 5
  - 1.5.1 Advanced: particle representations . . . 5
- 1.6 Full documentation: . . . 6

2 Usage **7**
- 2.1 Input Parsers . . . 7
  - 2.1.1 Pythia8 STDOUT `PYTHIA` . . . 7
  - 2.1.2 LHE `LHE` . . . 7
  - 2.1.3 HepMC `HEPMC` . . . 7
  - 2.1.4 CMSSW ParticleListDrawer `CMSSW` . . . 7
  - 2.1.5 Heppy `HEPPY` . . . 7
  - 2.1.6 Common Parser options . . . 7
- 2.2 Output Printers . . . 8
  - 2.2.1 dot `DOT` . . . 8
  - 2.2.2 web `WEB` . . . 8
  - 2.2.3 Common Printer Options . . . 8
- 2.3 Configuring Parsers & Printers . . . 8

3 Some Theory: Graphs and Particle Representations **9**
- 3.1 Basics . . . 9
- 3.2 Graphs and the Representations of Particles . . . 10
- 3.3 Conversion between representations . . . 11
  - 3.3.1 Edge to Node (easier) . . . 11
  - 3.3.2 Node to Edge (harder) . . . 11

4 Notes for developers **13**
- 4.1 Setup . . . 13
- 4.2 Design Notes . . . 13
- 4.3 Testing . . . 14
- 4.4 Conventions / Style . . . 14

5 Indices and tables **15**

PythiaPlotter plots diagrams of particle decay trees from HEP Monte Carlo (MC) events. It comes in handy to figure out what is actually going on in your MC!

CHAPTER 1

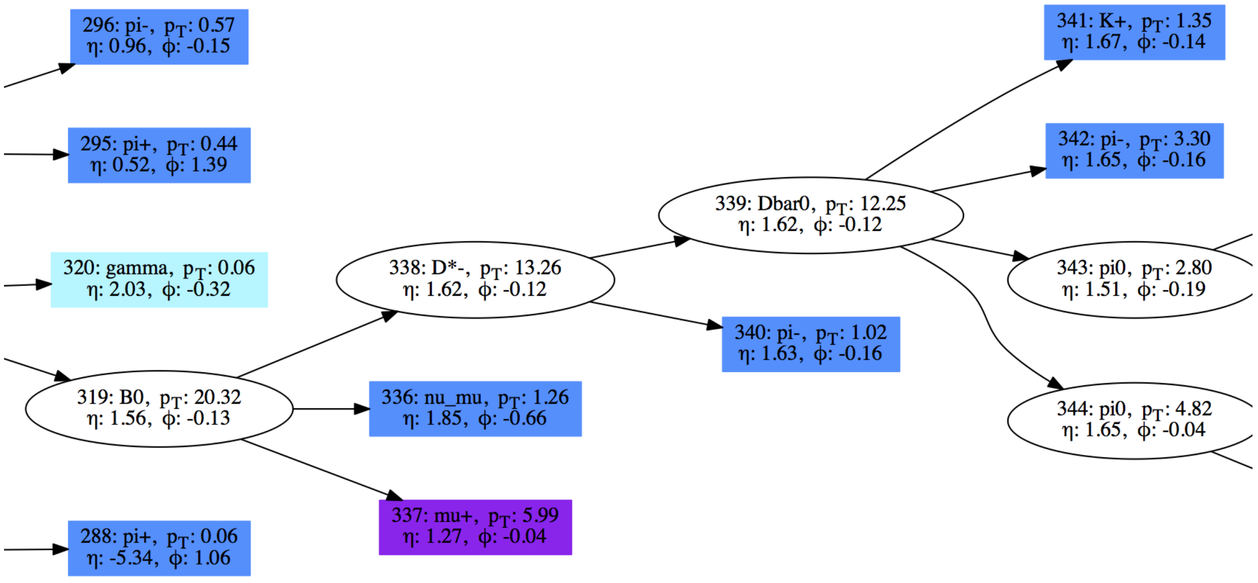
PythiaPlotter

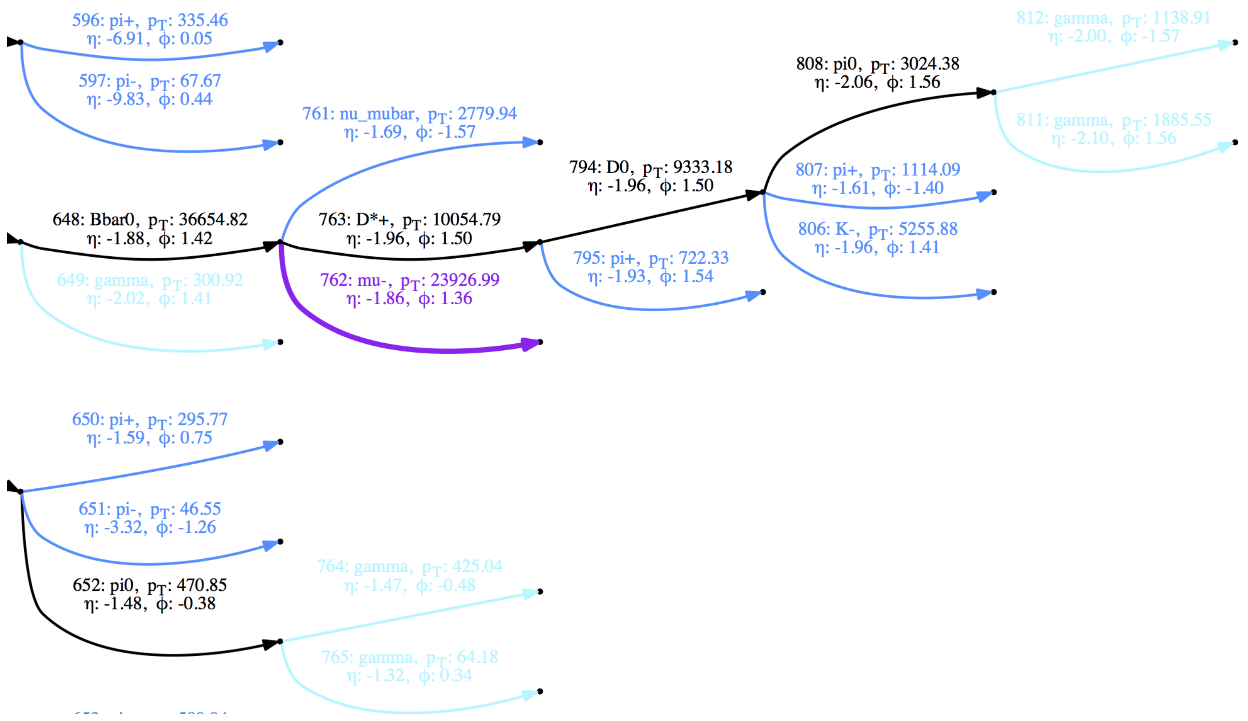
1.1 What Is It

Plots diagrams of particle decay trees from HEP Monte Carlo (MC) events. Very handy to figure out what is actually going on in your MC!

Click here for an interactive example.

And some snippets from the PDF examples in the example folder:


296: pi-, pT: 0.57 η: 0.96, ϕ: -0.15
295: pi+, pT: 0.44 η: 0.52, ϕ: 1.39
320: gamma, pT: 0.06 η: 2.03, ϕ: -0.32
319: B0, pT: 20.32 η: 1.56, ϕ: -0.13
288: pi+, pT: 0.06 η: -5.34, ϕ: 1.06
338: D*-, pT: 13.26 η: 1.62, ϕ: -0.12
336: nu_mu, pT: 1.26 η: 1.85, ϕ: -0.66
337: mu+, pT: 5.99 η: 1.27, ϕ: -0.04
339: Dbar0, pT: 12.25 η: 1.62, ϕ: -0.12
340: pi-, pT: 1.02 η: 1.63, ϕ: -0.16
341: K+, pT: 1.35 η: 1.67, ϕ: -0.14
342: pi-, pT: 3.30 η: 1.65, ϕ: -0.16
343: pi0, pT: 2.80 η: 1.51, ϕ: -0.19
344: pi0, pT: 4.82 η: 1.65, ϕ: -0.04


596: pi+, p_T: 335.46 η: -6.91, ϕ: 0.05
597: pi-, p_T: 67.67 η: -9.83, ϕ: 0.44
648: Bbar0, p_T: 36654.82 η: -1.88, ϕ: 1.42
649: gamma, p_T: 300.92 η: -2.02, ϕ: 1.41
761: nu_mubar, p_T: 2779.94 η: -1.69, ϕ: -1.57
763: D*+, p_T: 10054.79 η: -1.96, ϕ: 1.50
762: mu-, p_T: 23926.99 η: -1.86, ϕ: 1.36
794: D0, p_T: 9333.18 η: -1.96, ϕ: 1.50
795: pi+, p_T: 722.33 η: -1.93, ϕ: 1.54
808: pi0, p_T: 3024.38 η: -2.06, ϕ: 1.56
807: pi+, p_T: 1114.09 η: -1.61, ϕ: -1.40
806: K-, p_T: 5255.88 η: -1.96, ϕ: 1.41
812: gamma, p_T: 1138.91 η: -2.00, ϕ: -1.57
811: gamma, p_T: 1885.55 η: -2.10, ϕ: 1.56
650: pi+, p_T: 295.77 η: -1.59, ϕ: 0.75
651: pi-, p_T: 46.55 η: -3.32, ϕ: -1.26
652: pi0, p_T: 470.85 η: -1.48, ϕ: -0.38
764: gamma, p_T: 425.04 η: -1.47, ϕ: -0.48
765: gamma, p_T: 64.18 η: -1.32, ϕ: 0.34

1.2 What Can I Give It To Plot?

PythiaPlotter currently supports:

- Pythia 8 STDOUT (i.e. the big particle event table) (e.g. example/example_pythia8.txt)
- HepMC files (e.g. example/example_hepmc.hepmc)
- LHE files (e.g. example/example_lhe.lhe)
- ParticleListDrawer STDOUT as output by CMSSW (e.g. example/example_cmssw.txt)
- Heppy ROOT files - requires a special module to add mother/daughter relationships (e.g. example/example_heppy.root)

1.3 What Do I Need:

- Currently, both python 2.7 and python >=3.4 are supported (although the Heppy input is only supported by python 2.7 due to ROOT limitations)
- graphviz (If the command `which dot` returns a filepath, you will be fine)
- PyROOT if you want to parse Heppy ROOT NTuples. Note that if you are in a virtualenv, you will need to enable global site packages. If ROOT cannot be found, then the `--inputFormat HEPPY` option will be disabled.
- All other required python packages will be installed automatically

1.4 How Do I Get It:

- The easiest way is to use `pip`:

```
pip install git+https://github.com/raggleton/PythiaPlotter.git
```

- It can also be cloned from Github and installed locally from the base directory:

```
make install
```

- If you *really* can't install it, it is also possible to clone and run it in the main PythiaPlotter directory by doing:

```
python -m pythiaplotter
```

However this is the least recommended way, as you must be in the specific *PythiaPlotter* directory.

1.5 How Do I Use It:

Example usage (requires downloading the example input files in the example directory)

```
PythiaPlotter example/example_pythia8.txt --open --inputFormat PYTHIA --printer WEB
PythiaPlotter example/example_hepmc.hepmc --open --inputFormat HEPMC --printer DOT

# to show all options:
PythiaPlotter --help
```

There are various input (**parser**) and output (**printer**) options. You should specify the input format using `--inputFormat` (although it does try to guess), and and output printer using `--printer` (defaults to `DOT`). There are also other options for specifying the output filename and format.

MC generators often internally make several sequential copies of a particle, updating it as the event evolves. For our purposes, these are redundant particles that add no information, and just make things more complicated. Therefore they are removed by default. To keep these redundant particles use the `--redundants` flag.

1.5.1 Advanced: particle representations

Briefly, particles can be represented as **nodes** (graphically represented as a dot or blob) or **edges** (a line). In a generic graph, edges join together nodes, and may or may not have a direction. Here, we make use of the directionality of edges.

- **Node representation**: edges indicate a relationship between particles, where the direction may be read as "produces" or "decays into". For example, `a ->- b` represents `a` decaying into `b`.
- **Edge representation**: edges represent particles, like in a Feynman diagram. Nodes therefore join connected particles, such that all **incoming** edges into a node may be seen a "producing" or "decaying into" all **outgoing** edges.

This difference is that input formats naturally fall into one of the two representations. Pythia8, LHE, Heppy are all in the **node representation**, whilst HepMC is in the **edge representation**. Included in this program is the possibility to convert from the default representation into the other representation using the `-r {NODE, EDGE}` option. This can be useful to help elucidate what's going on in an event.

Note that redundant particle removal is done *after* representation conversion.

1.6 Full documentation:

See readthedocs

CHAPTER 2

Usage

Running the program requires the user to (1) select a suitable parser for their input, and (2) a printer.

To show all options, do `PythiaPlotter -h`.

2.1 Input Parsers

There are a variety of acceptable input sources. The input format should be specified using the `--inputFormat <FORMAT>` flag.

2.1.1 Pythia8 STDOUT `PYTHIA`

This is the full event listing output to screen when running Pythia 8. The user should pipe this into a file, and then pass that file to PythiaPlotter.

2.1.2 LHE `LHE`

2.1.3 HepMC `HEPMC`

2.1.4 CMSSW ParticleListDrawer `CMSSW`

This is the output from the ParticleListDrawer module used in CMSSW. The user should pipe the output into a file, and then pass that file to PythiaPlotter.

2.1.5 Heppy `HEPPY`

This reads in a ROOT file, with branches produced by a custom analyzer module to add in genparticle and mother/daughter info.

2.1.6 Common Parser options

- `-n, --eventNumber`: specify the index of the event to parse in the input file. By default, it will parse the first event (0).

2.2 Output Printers

Currently supported printers (specify via `-p, --printer`):

2.2.1 dot `DOT`

This prints a static document using Graphviz. By default it makes a PDF using the `dot` layout program, however the user is free to specify the layout program (via `--layout <LAYOUT>`) and the output format (via `--outputFormat <FORMAT>`).

2.2.2 web `WEB`

This creates an interactive webpage using Graphviz + vis.js. By default, it uses the `dot` layout program, however the user can change this via the `--layout <LAYOUT>` flag.

2.2.3 Common Printer Options

There are several options that apply to both printers.

- `-O, --output <FILENAME>`: specify output filename.
- `--open`: automatically open the output file once done.
- `--title <TITLE>`: can optionally put a title on the plot. Note that by default the input file and event number are automatically included.
- `--redundants`: by default the program removes chains of the same particle that are used internally by the MC generators. To keep these chains, use this flag.
- `--saveGraphviz`: this allows you to save the graph in a format suitable for parsing by graphviz. The user can then modify settings, etc in the file and ismply run graphviz over it, without having to rerun the entire program.
- `-r, --representation {NODE, EDGE}`: specify the output particle representation. For more info, see Graphs and representations.

2.3 Configuring Parsers & Printers

Although many options can be configured on the command line, some require more complex settings, or alternatively can be "set-and-forget". This is done using a config file.

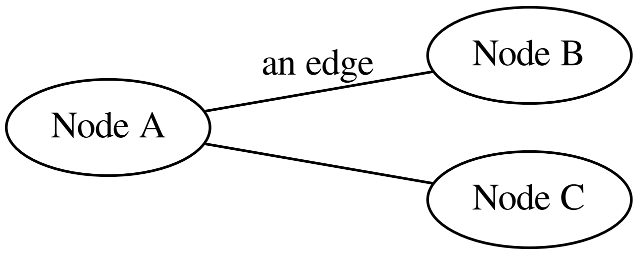
CHAPTER 3

Some Theory: Graphs and Particle Representations

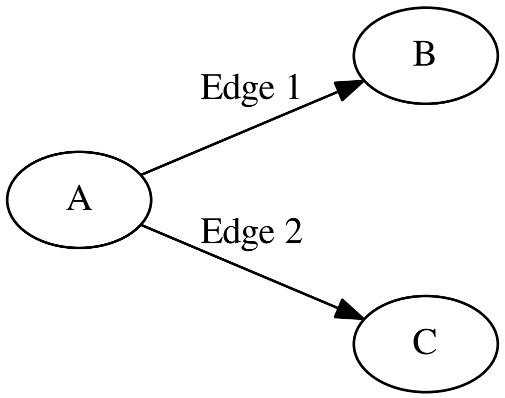
This is a brief introduction to graphs, some terminology, and how particle event trees can be represented by a graph.

3.1 Basics

A graph is composed of a set of **nodes**, along with **edges** which connect those nodes. An example of a simple graph is shown below.



Edges may have a **direction** associated with them; in this case we have a **directed graph** or digraph. An example of a digraph is shown below.

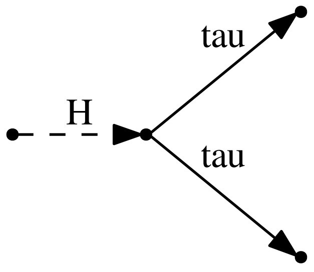


A directed edge has an **outgoing** node (the one it leaves) and an **incoming** node (the one it is going into). In the example, Edge 1 has outgoing node A, and incoming node B, whilst Edge 2 has outgoing node A and incoming node C. This implies some sort of "relationship" between nodes.

3.2 Graphs and the Representations of Particles

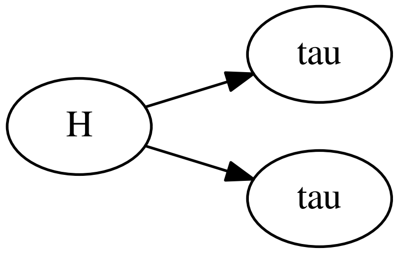
A particle event is characterised by the production and decay of particles; for example two gluons combine to make a Higgs boson, or a pion decays to two photons. There are **parents** (historically "mothers") such as the gluons or pion, and **children** ("daughters"), the Higgs or photons, and the parents "produce" the children. Therefore, there are **relationships** between these particles, which we can denote using graphs in 2 distinct ways.

Physicists are already used to this concept without realising it: a Feynman diagram (see below) is a graph, where each particle is represented by an edge, and a node with incoming and outgoing particles indicates some interaction (a **vertex**).



This association of particles to edges is deemed the **edge representation**. In this representation, nodes act as endpoints of edges, and a node with several incoming and outgoing particles may be interpreted as "all incoming particles are parents to all outgoing particles (children)". That is, the incoming Higgs is parent to the two tau children.

This is not the only possible assignment of particles to a direct graph however. Whilst natural from a theoretical particle physics persepctive, one can pose an alternate formulation. If we choose to represent each particle as a node, then each parent-daughter relationships can be denoted with a directed edge. In the example above of Higgs decaying to a pair of taus, there are two (parent, child) relationships, (Higgs, tau), so we can denote this as:



This is the **node representation**. Given a list of parent-child relationships, this is the natural representation, and is therefore the default for Pythia8, LHE, Heppy graphs.

3.3 Conversion between representations

Naturally, one might ask if it is possible to convert between the two representations. Spoiler: yes! Although it is sometimes non-trivial.

3.3.1 Edge to Node (easier)

Due to the definition of a node, this conversion is easy:

1. On our new graph, for each edge particle, create a node for that particle
2. For each vertex, iterate through all combinations of (incoming, outgoing) edges; for each draw a directed line between the corresponding pair of nodes.

And that's it!

A simple example is shown:

3.3.2 Node to Edge (harder)

Unlike edge to node, this is not always trivial. Consider the very simple example below.

CHAPTER 4

Notes for developers

Note: There is a makefile with handy targets for running tests, etc - use `make list` to show these.

4.1 Setup

One should install PythiaPlotter in "editable" mode using pip. This is easily done using `make installe` (note the **e**), or `make reinstalle` to uninstall and reinstall.

Developers should install the packages in `tests/requirements.txt` for running the tests, and `docs/requirements.txt` for making the docs:

```
pip install docs/requirements.txt
pip install tests/requirements.txt
```

4.2 Design Notes

The program is designed to be easy to extend in the event that more support needs to be added for different input or output formats. Because of this, there is a strong divide between the **parsing** part of the program, and the **plotting/printing** part.

These two parts communicate by exchanging an `Event` object, which contains a NetworkX `MultiDiGraph` object that holds the graph structure and particles.

In addition, there is a divide between the physical objects in an event, and the structure that connects them. This offers several advantages:

- uniform way to access particle relationships regardless of input or output mode
- separation of physical particle attributes from graph structure and attributes (e.g. parent nodes, etc)
- separation of visual attributes from physical attributes.

The last point is important: this way, we can easily assign a `NodeAttr` object to each node, and an `EdgeAttr` to each edge, to hold the display options and decouple the visual attributes from the physical attributes. By sub-classing those `*Attr` objects for different output formats, we can easily implement a new output format without interference with other output formats.

4.3 Testing

There are various make targets to run tests easily:

- `make test`: run unit tests
- `make test-examples`: run full execution of the program, for a variety of input options
- `make cov`: run coverage.py and make a HTML report
- `make benchmark`: run performance metrics (mostly timing of components)

There are also various targets for linting, etc:

- `make lint`: run pylint
- `make lint-py3`: run pylint's python3 checker **Note this is not perfect!**
- `make flake`: run flake8

Docs can be made locally by doing `make docs`.

4.4 Conventions / Style

Lines should be < 100 lines, but sometimes going over makes more sense than horrible linebreaks. Try and fix all linter errors/warnings, but sometimes they are silly. Use Numpy-style for docstrings. All code should be compatible with python 2.7 and >=3.4. The compatibilty with 2.7 is because a significant proportion of HEP is still forced to use it, but at the same time we should forsee migration to python 3.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search