



LUND UNIVERSITY



For tutorials  
at Summer Schools

# PYTHIA 8 Merging Tutorial

Torbjörn Sjöstrand

Department of Theoretical Physics, Lund University

Stefan Prestel

Theory group, DESY

## 1 Introduction

The objective of this exercise is to teach you the basics of how to use the PYTHIA 8.1 event generator to study multi-jet backgrounds. As you become more familiar you will better understand the tools at your disposal, and can develop your own style to use them. Within this first exercise it is not possible to describe the physics models used in the program; for this we refer to the PYTHIA 8.1 brief introduction [1], to the full PYTHIA 6.4 physics description [2] (and all the further references found in them), and [5, 6, 7] concerning matrix element merging.

PYTHIA 8 is, by today's standards, a small package. As such, it should be noted that PYTHIA8 includes a selection  $2 \rightarrow 1$  and  $2 \rightarrow 2$  processes, as well as a limited variety of  $2 \rightarrow 3$  processes, but does not contain a general matrix element generator. New processes, particularly for two or more additional jets, can be made available in form of Les Houches Event (LHE) files. This means that to estimate backgrounds with many jets, you can use a matrix element generator like e.g. MADGRAPH/MADEVENT to improve the PYTHIA8 description of well-separated jets.

## 2 Getting started: Installing Pythia 8

If you would like to install PYTHIA 8 on your private machine, and you have a C++ compiler, here is how to install the latest PYTHIA 8 version on a Linux/Unix/MacOSX system as a standalone package.

For this tutorial, most installation steps have already been taken. You can find the PYTHIA 8 source code under

```
PYTHIADIR=/opt/share/Pythia8
```

on your virtual machine. PYTHIA 8 has been installed so that you can directly change into one of the tutorial directories

```
INTRODIR=~/.tutorials/intro/pythia
HIGGSDIR=~/.tutorials/higgs/pythia
BOOSTDIR=~/.tutorials/boost/pythia
```

and start directly with Step 6 below. Alternatively, the following steps allow you to install PYTHIA 8 from the source code on your private computer.

While PYTHIA can be run standalone, it can also be interfaced with a set of other libraries. One example is HEPMC, which is the standard format used by experimentalists to store generated events. Since the the location of external libraries is naturally installation-dependent, it is not possible to give a fool-proof linking procedure, but some hints are given below.

1. In a browser, go to

```
http://www.thep.lu.se/~torbjorn/Pythia.html
```

2. Download the (current) program package

```
pythia81xx.tgz
```

to a directory of your choice (e.g. by right-clicking on the link).

3. In a terminal window, cd to where `pythia81xx.tgz` was downloaded, and type

```
tar xvfz pythia81xx.tgz
```

This will create a new (sub)directory `pythia81xx` where all the PYTHIA 8 source files are now ready and unpacked. From now on, we will call this directory<sup>1</sup> `PYTHIADIR`.

4. Move to this directory (`cd $PYTHIADIR`). If you are only interested in directly producing plots from PYTHIA event records, you can directly go to the next step. If you want to produce and store HEPMC event output, configure the program in preparation for the compilation by typing

```
./configure --with-hepmc=$HEPMC_PATH
```

where the directory-tree `$HEPMC_PATH` would depend on your local HEPMC installation. Should `configure` not recognise the version number you can supply that with an optional argument, like

```
./configure --with-hepmc=$HEPMC_PATH--with-hepmcversion=2.06.06
```

For this tutorial, we will use a trick to keep the size of inputs small: We will allow PYTHIA to read compressed Les Houches event files. For this, configure PYTHIA with the options

---

<sup>1</sup>On your virtual machine, you can find the PYTHIA 8 source files under `/opt/share/Pythia8`.

```
./configure --enable-gzip --with-boost=$BOOST_PATH \  
  --with-zlib=$ZLIB_PATH --with-hepmc=$HEPMC_PATH \  
  --with-hepmcversion=2.06.06
```

in order to link to the `gzip` and `boost` libraries. For this school, you can use

```
BOOST_PATH=/opt  
ZLIB_PATH=/opt/lib/i386-linux-gnu  
HEPMC_PATH=/opt
```

5. Do a `make`. This will take 2–3 minutes (computer-dependent). The `PYTHIA 8` libraries are now compiled and ready for physics.
6. For test runs, `cd` to the `examples/` subdirectory, or one of the tutorial directories. An `ls` reveals a list of programs, `mainNN`, with `NN` from 01 through 90. These example programs each illustrate an aspect of `PYTHIA 8`. For a list of what they do, see the `README` file in the same directory or look at the online documentation. Initially only compile one or two of them to check that the installation works. Once you have worked your way through the introductory exercises in the next sections you can return and study the programs and their output in more detail. If you want to produce `HEPMC` output when installing `PYTHIA 8` for the source code, do either of

```
source config.csh  
source config.sh
```

the former when you use the `csh` or `tcsh` shells, otherwise the latter. (Use `echo $SHELL` if uncertain.). If you are not interested in `HEPMC`, this step can be skipped. To execute one of the test programs, do

```
make mainNN  
./mainNN.exe
```

The output is now just written to the terminal, `stdout`. To save the output to a file instead, do `./mainNN.exe > mainNN.out`, after which you can study the test output at leisure by opening `mainNN.out`. See Appendix A for an explanation of the event record that is listed in several of the runs.

7. If you open the file

```
$PYTHIADIR/html/doc/Welcome.html
```

you will gain access to the online manual, where all available methods and parameters are described. Use the left-column index to navigate among the topics, which are then displayed in the larger right-hand field.

To produce a matrix-element improved prediction with `PYTHIA 8`, you will also need to supply Les Houches Event (LHE) files as input. You will have to generate these yourself, with your own cuts, and your preferred matrix element generator. For the sake of this introductory tutorial, we supply sample LHE files. These can be mounted by executing

```
./mountSamples.sh
```

in the `~tutorial` directory. Then, you'll find (links to) Les Houches files, which can be used as input for the programs you will develop in this tutorial. The names of the files reflect the process and the generation settings<sup>2</sup>

```
ggh_cc_tree_0.lhe.gz: pp → H (leading order,  $\mu_f = m_H, \mu_r = m_H$ )
ggh_ll_tree_0.lhe.gz: pp → H (leading order,  $\mu_f = \frac{1}{2}m_H, \mu_r = \frac{1}{2}m_H$ )
ggh_cc_tree_1.lhe.gz: pp → H + 1 parton (leading order,  $\mu_f = m_H, \mu_r = m_H$ )
ggh_hh_tree_1.lhe.gz: pp → H + 1 parton (leading order,  $\mu_f = 2m_H, \mu_r = 2m_H$ )
...
```

These files come with no guarantees and should only be used for this tutorial. For the first part of the tutorial, you can stick to the input events generated with  $\mu_f = m_H$  and  $\mu_r = m_H$ , i.e. the "ggh\_cc"-files.

## 3 Pythia 8 standalone

When using PYTHIA, you are expected to write the main program yourself, for maximal flexibility and power. Several examples of such main programs are included with the code, to illustrate common tasks and help getting started. You will also see how the parameters of a run can be read in from a file, so that the main program can be kept fixed. Many of the provided main programs therefore allow you to create executables that can be used for different physics studies without recompilation, but potentially at the cost of some flexibility.

### 3.1 Simple LHC Events

The focus of the next sessions will be on extracting signals from Standard Model backgrounds. So, to get to know the PYTHIA 8 syntax, we will generate one  $pp \rightarrow H + \text{jets}$  event at the LHC, using PYTHIA standalone to produce all jets.

Open a new file `mymain.cc` in the `INTRODIR` subdirectory with a text editor, e.g. Emacs. Then type the following lines (here with explanatory comments added)<sup>3</sup>:

```
// Headers and Namespaces.
#include "Pythia.h" // Include Pythia headers.
using namespace Pythia8; // Let Pythia8:: be implicit.

int main() { // Begin main program.

    // Set up generation.
    // Declare Pythia object
```

---

<sup>2</sup>The usage of the different input files will be explained in the text.

<sup>3</sup>Experienced PYTHIA 8 users might be surprised by the initialisation on a Les Houches reader object. Using this slightly more complicated initialisation, gzip-compressed files will become readable.

```

Pythia pythia;
// Declare Les Houches event file reader.
LHAupLHEF lhareader("./ggh_cc_tree_0.lhe.gz");
// Initialise pythia on LHE file for gg-> H
pythia.init(&lhareader);

// Generate event(s).
// Generate an(other) event. Fill event record.
pythia.next();

// End main program with error-free return.
return 0;
}

```

Next you need to edit the Makefile (the one in the INTRODIR subdirectory) so it knows what to do with `mymain.cc`. The line

```

# Create an executable for one of the normal test programs
main00 main01 main02 main03 ... main09 main10 main10 \

```

together with the following three lines enumerate the main programs that do not need any external libraries. Edit the last of these lines to include also `mymain`:

```

main40 mymain: \

```

Now you can compile and run your main program by typing

```

make mymain
./mymain.exe > mymain.out

```

You can then study `mymain.out`, especially the example of an event record. At this point you need to turn to Appendix A for a brief overview of the information stored in the event record.

An important part of the event record is that many copies of the same particle may exist, but only those with a positive status code are still present in the final state. For illustration, consider a gluon produced in the first initial state splitting of the  $gg \rightarrow H$  hard interaction. Initially, this parton will have a positive status code. When a shower branching changes properties of this gluon, then the new, changed gluon is added at the bottom of the then-current event record, but the old  $g$  is not removed. It is marked as decayed, however, by negating its status code. At any stage of the shower there is thus only one “current” copy of this gluon. When you understand the basic principles, see if you can find several copies of the a parton produced in the hard interaction, and check the status codes to figure out why each new copy has been added. Also note how the mother/daughter indices tie together the various copies.

## 3.2 A first realistic analysis

We will now gradually expand the skeleton `mymain` program from above, towards what would be needed for a more realistic analysis setup.

- Generally, we wish to generate more than one event. To do this, introduce a loop around `pythia.next()` and `pythia.event.list()`, so the code now reads

```
for (int iEvent = 0; iEvent < 5; ++iEvent) {
    pythia.next();
    pythia.event.list();
}
```

Hereafter, we will call this the **event loop**. The program will now generate and print 5 events; each call to `pythia.next()` resets the event record and fills it with a new event. Once you start generating many events, it might be convenient to remove the `pythia.event.list()` call. By default, PYTHIA 8 will still print a record of the very first event.

- To obtain statistics on the number of events generated of the different kinds, and the estimated cross sections, add

```
pythia.stat();
```

just before the end of the program.

- During the run you may receive problem messages. These come in three kinds:
  - a *warning* is a minor problem that is automatically fixed by the program, at least approximately;
  - an *error* is a bigger problem, that is normally still automatically fixed by the program, by backing up and trying again;
  - an *abort* is such a major problem that the current event could not be completed; in such a rare case `pythia.next()` is `false` and the event should be skipped.

Thus the user need only be on the lookout for aborts. During event generation, a problem message is printed only the first time it occurs. The above-mentioned `pythia.stat()` will then tell you how many times each problem was encountered over the entire run.

- Looking at the `pythia.event.list()` listing for a few events at the beginning of each run is useful to make sure you are generating the right kind of events, at the right energies, etc. For real analyses, however, you need automated access to the event record. The Pythia event record provides many utilities to make this as simple and efficient as possible. To access all the particles in the event record, insert the following loop after `pythia.next()` (but fully enclosed by the **event loop**)

```
for (int i = 0; i < pythia.event.size(); ++i)
    cout << "i = " << i << ", id = " << pythia.event[i].id() <<
```

```
endl;
```

which we will call the **particle loop**. Inside this loop, you can access the properties of each particle `pythia.event[i]`. For instance, the method `id()` returns the PDG identity code of a particle (see Appendix A). The `cout` statement, therefore, will give a list of the PDG code of every particle in the event record. All methods that give particle properties can be found following the “Particle properties” link in the section “Study output” in the left-hand menu in the manual, or in the file `/path/to//Pythia/pythia8160/html/doc/ParticleProperties.html`.

- If you are e.g. only interested in final state partons, the `isFinal()` and `isParton()` methods can be applied to the event record entry:

```
for (int i = 0; i < pythia.event.size(); ++i)
    if(pythia.event[i].isFinal() && pythia.event[i].isParton())
        cout << "i = " << i << ", id = " <<
pythia.event[i].id() << endl;
```

This will only print the PDG code of final state gluons, (anti)quarks and diquarks.

- In addition to the particle properties in the event listing, there are also methods that return many derived quantities for a particle, such as transverse momentum (`pythia.event[i].pT()`). Use this method to find the transverse momentum of the H-boson after the evolution, i.e. the last H-boson in the event record (which is of course cheating a bit, since in an experimental environment, it is a lot more complicated to isolate H-candidates).
- We now want to generate more events, say 1000. Make sure you remove all unnecessary output before you proceed. Then, let’s study the shape the  $p_{T,H}$ -spectrum. Inside PYTHIA is a very simple histogramming class, that can be used for rapid check/debug purposes. To book the histograms, insert before the **event loop**

```
Hist HistPTH("pT of H-boson", 50, 0., 500.);
```

where the last three arguments are the number of bins, the lower edge and the upper edge of the histogram, respectively. As an exercise, fill this histogram for each event with the transverse momentum of the H-boson after the evolution. For this, initialise a variable before the **particle loop**, and find the  $p_T$  inside a **particle loop**:

```
double pT = 0.;
for (int i = pythia.event.size(); i > 0; --i)
    if( pythia.event[i].idAbs() == 25 ) {
        pT = pythia.event[i].pT();
        break;
    }
```

Then, before the end of the **event loop**, insert

```
HistPTH.fill(pT);
```

to fill the histogram. To arrive at a correctly normalised histogram, include

```
HistPTH *= pythia.info.sigmaGen() / pythia.info.nAccepted();
```

after the **event loop** and the `pythia.stat()` call. In this way, the sum of the heights of all histogram bins will give the correct cross section, irrespectively of how many events you requested<sup>4</sup>. Finally, to print the histograms to the terminal, add a line like

```
cout << HistPTH;
```

For comparison with merged results, it might be useful to save the output of this run.

## 4 Tree-level merged predictions

The main program we have constructed in the previous section has one drawback: All radiation will be produced by PYTHIA 8. This will give reliable results for soft and collinear configurations, but less so for multiple hard, well-separated jets. To model both soft/collinear and well-separated jets at the same time, we need to include matrix element calculations – which describe the production of multiple hard jets nicely – into the jet evolution of the parton shower. This can be done by supplying LHE files to PYTHIA 8, which will then internally be processed to perform a smooth transition from  $n$ -jet to  $n+1$ -jet events. Several main programs illustrating matrix element + parton shower merging (MEPS) are included in the PYTHIA 8 distribution, offering a range of different merging schemes:

- Tree-level merging: MLM-style jet matching with MADGRAPH or ALPGEN, CKKW-L merging [5], and unitarised ME+PS merging (UMEPS) [6];
- Next-to-leading order merging:  $NL^3$  merging, unitarised NLO+PS merging (UNLOPS) [7].

Here, we will get to know the CKKW-L, UMEPS and UNLOPS schemes by manipulating some example main programs<sup>5</sup> We will use  $H + \leq 2$  jets as an example.

### 4.1 CKKW-L merging

CKKW-L merging [5] was the first merging scheme available in PYTHIA 8. For this school, we supply the example main program `main-ckkw1.cc`. This program uses the LH files in the examples directory to produce a result that simultaneously describes  $H + 0, 1, 2$  jet

---

<sup>4</sup>Pythia cross sections are given in units of mb. If you instead prefer e.g. pb then multiply by `1e9`.

<sup>5</sup>A more concise tutorial on how to write a CKKW-L main program from scratch can be found at <http://home.thep.lu.se/~torbjorn/pythia8/mergingworksheet8160.pdf>.



observables with leading-order matrix elements, while also including PS resummation (i.e. arbitrarily many PS emissions). CKKW-L is described in the online manual under **Link to other programs** → **CKKW-L merging**.

Take as an example one-jet observables (e.g. the transverse momentum of the jet in events with *exactly* one jet). In this case, we want to take a "hard" jet from the  $pp \rightarrow Hj$  matrix element (ME), while "soft" jets should be modelled by emissions off  $pp \rightarrow H$  states, generated by Pythia. This means that in order to smoothly merge these two samples, we have to know in which measure "hard" is defined, and which value of this measure separates the hard and soft regions. In `main-ckkw1.cc`, these definitions are

```
pythia.readString("Merging:doPTLundMerging = on");
pythia.readString("Merging:TMS = 15");
```

This will enable the merging procedure, with the merging scale defined through the evolution  $p_{\perp}$  of PYTHIA 8. Such a definition fixes what we mean when we talk about "hard" and "soft" jets<sup>6</sup>:

Hard jets:  $\min\{\text{Any relative } k_{\perp} \text{ between sets of partons}\} > t_{\text{MS}}$   
 Soft jets:  $\min\{\text{Any relative } k_{\perp} \text{ between sets of partons}\} < t_{\text{MS}}$

Then, to define that we want to produce results for Higgs production, we set

```
pythia.readString("Merging:Process = pp>h");
```

and allow for full flexibility in the the Higgs decay products, by simply disregarding the decay products in the merging:

```
pythia.readString("Merging:mayRemoveDecayProducts = on");
```

Finally, we want to include the pre-generated ME events for up to two additional jets. This is communicated to PYTHIA by setting

```
pythia.readString("Merging:nJetMax = 2");
```

Then, `main-ckkw1.cc` will start generating events, reading from the LHE file with the highest parton multiplicity first. Each event after the merging comes with a weight

```
double weight = pythia.info.mergingWeight();
```

which contains Sudakov factors to remove the double counting between samples of different multiplicity,  $\alpha_s$ -ratios to incorporate  $\alpha_s$ -running, and ratios of parton distributions to include variable factorisation scales. This weight *must* be used to fill histogram bins. As an exercise, convince yourself that the variation in this weight is moderate.

`main-ckkw1.cc` finishes by normalising the histograms for each sub-sample by the correct cross section

---

<sup>6</sup>There is of course a plethora of possible hardness criteria. For this reason, CKKW-L merging in PYTHIA 8 allows you to define your very own merging scale (please consult the manual for details).

```
SubHistPTH *= 1e9 * pythia.info.sigmaGen() / pythia.info.nAccepted();
```

and then adding the histograms

```
HistPTH += SubHistPTH;
```

You can compile and `main-ckkw1.cc` by issuing the commands

```
make main-ckkw1
./main-ckkw1.exe
```

Now that we understand the main program, let us (a) compare to the full result of simply showering the Higgs production process, (b) check in which  $p_{\perp}$  regions which jet multiplicity contributes, (c) vary the merging scale value `TMS`.

If you would like to get your hands dirty on some CKKW-L details, check out the collection of main programs for CKKW-L merging (`main81.cc-main85.cc`) that PYTHIA 8 offers. Each of these require a different level of knowledge about the merging procedure. `main85.cc` is intended as a black box. However, before you get stuck on CKKW-L, it is useful to know that other, more modern, merging schemes are available in PYTHIA 8.

## 4.2 Unitarised ME+PS merging

One main disadvantage of classical merging schemes is the matrix element calculations that are included into the parton shower might contain large logarithmic contributions that cannot be cancelled by the shower resummation. These terms can, particularly for small merging scales, lead to changes of the inclusive cross section after multi-jet merging. A better shower would ameliorate this issues. Still, such problems reveal that the merging scale value is not a purely technical parameter, but should be chosen in a sensible range.

However, it is possible to demote the merging scale to a technical parameter without having to improve the PS resummation. Rather, improved approximate virtual corrections can be employed, resulting in a scheme called unitarised ME+PS merging, short UMEPS [6]. The basic idea is simple: Subtract what you add, then the inclusive cross section will stay fixed. The way this "add-subtract" scheme is facilitated (without disturbing the description of well-separated jets) closely follows the prescription how (un-merged) parton showers preserve the inclusive cross section. UMEPS is described in the online manual under **Link to other programs** → **UMEPS merging**.

For this school, we supply a sample main program<sup>7</sup> for UMEPS merging, `main-umeps.cc`. The main program starts by setting the merging scale value

```
pythia.readString("Merging:TMS = 15");
```

Note that the actual merging scale definition will be set on a sample-by-sample basis. Then, the core scattering is defined

---

<sup>7</sup>The standard UMEPS sample main program is `main86.cc`, which already yields HepMC event output. Since we don't need these features for this tutorial, we use a stripped-down main program.

```

pythia.readString("Merging:Process = pp>h");
pythia.readString("Merging:mayRemoveDecayProducts = on");

```

and the maximal number of additional partons in the matrix element calculation is set

```

pythia.readString("Merging:nJetMax = 2");

```

The event generation code starts after these settings. Note that this code is split into an "additive" part (the loop over `njetcounterL0`), followed by a "subtractive" part (the loop over `njetcounterCT`).

In the additive part of UMEPS, the merging scale is defined by

```

pythia.readString("Merging:doUMEPSTree = on");

```

This is currently the only merging scale definition for UMEPS, and corresponds to separating hard and soft jets according to  $\min\{p_{\perp evol}\}$ . The setting

```

pythia.readString("Merging:nRecluster = 0");

```

is also necessary to ensure that the algorithm works as expected (see below). Histograms bins for observables in this additive part should, as in CKKW-L, be filled with the weight containing Sudakov-,  $\alpha_s$  and PDF-factors,

```

double weight = pythia.info.mergingWeight();

```

and the resulting sub-sample histogram should be added to the to-be merged histogram.

In the subtractive part of UMEPS, we define the merging scale by

```

pythia.readString("Merging:doUMEPSSubt = on");
pythia.readString("Merging:nRecluster = 1");

```

This enables the trick that is at the very heart of UMEPS: In a parton shower, approximate virtual corrections  $V_{ps}$  and approximate real emissions  $R_{ps}$  cancel because  $V_{ps} = -\int R_{ps}$ . Thus, the inclusive cross section is not changed by the parton shower. This cancellation is however spoiled when merging a real emission matrix element  $R_{me}$ , since  $R_{me} \neq R_{ps}$ . The subtractive part of UMEPS puts  $V_{umeps} = -\int R_{me}$ , thus explicitly enforcing the parton-shower-like cancellation. The necessary integration is enabled by putting `Merging:nRecluster = 1`. As always, the result should be histogrammed with the weight

```

double weight = pythia.info.mergingWeight();

```

Now, the resulting sub-sample histograms should be subtracted from the to-be merged result.

You can compile and `main-umeps.cc` by issuing the commands

```
make main-umeps
./main-umeps.exe
```

Once we have understood `main-umeps.cc`, let us find the average weight (for one particular number of additional ME partons) of the additive parts of UMEPS, and compare this to the average weight (of the same parton multiplicity) in CKKW-L. Further, change the merging scale value in UMEPS, and try to understand the changes. Compare the merged results between CKKW-L and UMEPS.

If you have finished these challenges, you should take the opportunity to try a few other options. Below are given some examples, but feel free to pick something else that you would be more interested in.

In the next session, you will move one step further in the history of merging schemes. To assess the theoretical uncertainties of current merging methods, you will use the best method we have so far in PYTHIA 8: NLO multi-jet merging in the UNLOPS scheme.

## 5 Next-to-leading order merging

*This part of the tutorial is optional, and will indeed partly be covered elsewhere.*

The most recent development in combining fixed-order calculations and parton shower resummation is next-to-leading order merging. This is different from NLO matching (e.g. POWHEG and MC@NLO) because it allows to describe different parton multiplicities simultaneously at NLO accuracy. NLO merging schemes are the successor of tree-level merging schemes.

In this tutorial, we will assess the uncertainties of the unitarised NLO+PS merging method (UNLOPS, [7]). For this, we will depart from our usual practise of simply printing histograms to the terminal. Instead, we will use a sample main program that uses HEPMC output, which can then be analysed with RIVET. We will briefly describe the main program used for this study, and come back to the actual uncertainty estimates below.

### 5.1 main88.cc

UNLOPS merging is the direct extension of UMEPS to NLO accuracy. For this school, we would like our simulation to describe  $H + 0$  and  $H + 1$  jet simultaneously with NLO accuracy, describe  $H + 2$  jets with tree-level accuracy, and take all further jets from the PS approximation. This means we have to consider two types of input calculations: Tree-level and NLO inputs. For this school, we supply tree-level event samples produced with MadGraph, and NLO event samples produced with POWHEG-BOX. UNLOPS then processes these samples in the following way:

- (1) Use tree-level matrix elements for  $n$  partons as "seeds" for higher-order corrections (of  $\mathcal{O}(\alpha_s^{n+2})$  and beyond), making sure that no  $\mathcal{O}(\alpha_s^{n+1})$  are produced.

- (2) Add the NLO samples, making sure that no higher orders (of  $\mathcal{O}(\alpha_s^{n+2})$  and beyond) are produced.
- (3) Unitarise everything, making sure that no unwanted  $\mathcal{O}(\alpha_s^{n+1})$  terms are produced, i.e. ensure that the inclusive cross section is given by the H + 0 NLO result.

This is done internally in `main88.cc`, the sample main program we will use to generate NLO merged results. In the event generation phase, `main88.cc` uses the tree-level input file twice (once for Step (1) and once for Step (3)), as well as using the NLO (POWHEG) inputs twice (once for Step (2) and once for Step (3)). `main88.cc` is described in detail in the online manual under **Link to other programs** → **NLO merging**<sup>8</sup>.

`main88.cc` reads a settings file (e.g. `main88.cmd`) for the necessary settings. It is important to set the switches

```

// Definition core process for merging
Merging:Process                = ?
Merging:mayRemoveDecayProducts = ?
// Maximal number of additional LO jets.
Merging:nJetMax                = ?
// Maximal number of additional NLO jets.
Merging:nJetMaxNLO             = ?
// Merging scale value.
Merging:TMS                    = ?
// Values of (fixed) scales in the matrix element calculation.
Merging:muFacInME              = ?
Merging:muRenInME              = ?
// Values of (fixed) scales for the PS lowest multiplicity process.
Merging:muFac                  = ?
Merging:muRen                  = ?

```

in such an input file. The process definition, maximal number of additional tree-level jets and merging scale value have already been discussed in the CKKW-L section. In addition, you now need to set the maximal number of additional jets for which an NLO event sample is available, and the renormalisation and factorisation scales with which the event samples have been produced. Finally, it is also necessary to set the scales which would be used in default PYTHIA 8 to evaluate the core scattering (i.e. for H production, the mass  $m_H$ ). In the case of wimpy showers, the value of `Merging:muFac` further sets the shower starting scale in UNLOPS.

In general, you also need to generate tree-level- and POWHEG event samples as input for `main88.cc`. For this school, we have included some pre-calculated samples in the `HIGGSDIR` directory. The main program `main88.cc` assumes, to allow for a streamlined file parsing, a particular naming convention. All POWHEG event samples should be called

---

<sup>8</sup>In fact, we have slightly changed `main88.cc` to be able to read compressed LHE files.

```
myLHEF_powheg_njets.lhe.gz
```

where `myLHEF` is a free process identifier, which is assumed to be identical for all samples belonging to one particular process. `njets` should give the number of additional partons that are described at NLO accuracy (i.e. *not* counting real-emission partons). POWHEG events for  $H + j$ @NLO could for example be called `higgs_powheg_1.lhe.gz`. Tree-level inputs are called

```
myLHEF_tree_njets.lhe.gz
```

A legitimate name for a tree-level sample with two additional jets could e.g. be `higgs_tree_2.lhe.gz`.

To use `main88.cc`, go into the `HIGGS` directory and compile the main program

```
cd $HIGGS
make main88
```

and run by issuing a command of the form

```
./main88.exe myInputFile myLHEF myHEPMCoutput
```

`main88.cc` will then, consecutively, read tree-level and POWHEG event samples, and produce HEPMC events.

To get used to the program, produce only small number of events, and run

```
./main88.exe main88.cmd ggh_cc myHEPMCoutput
```

Can you identify the Steps (1)-(3) through the terminal output? Why is step (3) applied both to tree-level and POWHEG samples?

## 5.2 Scale uncertainty estimates

In this part, we will try to estimate the scale uncertainty of the UNLOPS method. For this, we will vary  $\mu_f$  and  $\mu_r$  in the matrix element calculation, while keeping the scales in the parton shower resummation unchanged. In the `HIGGS` directory, you can find event files produced with three different scale settings:

- Files whose name contains `ll` have been generated with  $\mu_f = \mu_r = \frac{1}{2}m_H$
- Files whose name contains `cc` have been generated with  $\mu_f = \mu_r = m_H$
- Files whose name contains `hh` have been generated with  $\mu_f = \mu_r = 2m_H$

Let us now investigate the scale uncertainty of the  $(H+0, 1@NLO)+(H+2@LO)$  prediction. We will use the analysis `MC_HJETS` of RIVET to do this.

For each (fixed-order) scale setting, you will need to change `Merging:muFacInME` and `Merging:muRenInME` in your input file and use the correct LHE samples.

We use HEPMC events to parse the PYTHIA 8 output to RIVET. However, HEPMC event files quickly become prohibitively large. This is why RIVET allows the usage of `fifo` pipes to pass the events at generator run time, without having to store large intermediate files. PYTHIA 8 will write a single event to the pipe, and RIVET will read and analyse this event. To make this as simple as possible, you can use the BASH script `run.sh`. This script allows to run both PYTHIA 8 and RIVET in the background of one common terminal. To use `run.sh`, make it executable

```
chmod +x run.sh
```

and infer it by

```
./run.sh
```

`main88.cc` needs to be compiled before running the script. You will have to change the name of LHE files, log-files and aida output files in the script, and the scales in the settings file, if you want to generate histograms with different scale choices.

Now, try to do the following

- Run `main88.cc`, for the central scales, with only  $(H + 0@NLO)+(H + 1, 2@LO)$ , and with  $H + 0, 1@NLO)+(H + 2@LO)$ . Check the exclusive jet multiplicity of the `MC_HJETS` analysis. Can you explain the differences?
- Run `main88.cc`, with  $H + 0, 1@NLO)+(H + 2@LO)$  for the three scale choices
  - (a)  $\mu_f = \mu_r = \frac{1}{2}m_H = 62.5$  GeV
  - (b)  $\mu_f = \mu_r = m_H = 125$  GeV
  - (c)  $\mu_f = \mu_r = 2m_H = 250$  GeV

Display the results together, and try to understand the differences.

- Switch between wimpy and power showers by changing the settings `TimeShower:pTmaxMatch` and `SpaceShower:pTmaxMatch`.

Finally, compare the UNLOPS results to the results of the other Event Generators. This is most conveniently done if you display the variations as uncertainty band. You can modify the script

```
~/tutorials/higgs/plotit.sh
```

for this purpose. Type

```
cd ~/tutorials/higgs/  
./plotit.sh -help
```

to find out about the script.

This concludes the generator-specific part of the tutorials. Don't hesitate to contact us [8] if you have further questions.

## Bonus: Some interesting settings

You are now free to play with further options in the input file (or use the `pythia.readString` method directly in your code), and make changes such as:

- `Tune:pp = 5` (or other values between 1 and 7)  
Different combined tunes, in particular to radiation and multiple interactions parameters. In part this reflects that no generator is perfect, and also not all data is perfect, so different emphasis will result in different optima. Currently, the default tune for LHC is Tune 4C, (which can also be explicitly set by `Tune:pp = 5`). What happens if you switch to Tune A2, which is a recent tune that can be used by putting `Tune:pp = 7`?
- `PartonLevel:FSR = off`  
switch off final-state radiation.
- `PartonLevel:ISR = off`  
switch off initial-state radiation.
- `PartonLevel:MPI = off`  
switch off multiple interactions.
- `TimeShower:pTmaxMatch = 1`  
wimpy final state showers (default).
- `SpaceShower:pTmaxMatch = 1`  
wimpy initial state showers (non-default).

For debugging your code for instance, it might be reasonable to switch off multiple interactions, so that you can produce plots more quickly.

The philosophy of PYTHIA is to make every parameter available to the user. A complete list of changeable settings can be found in the online manual. Have a look at the switches related to MEPS merging. As additional challenge, think about which switches are dangerous, i.e. will maximally corrupt your predictions. Can you isolate a singularly bad setting? Why does the prediction deteriorate with the changes?



# A The Event Record

The event record is set up to store every step in the evolution from an initial low-multiplicity partonic process to a final high-multiplicity hadronic state, in the order that new particles are generated. The record is a vector of particles, that expands to fit the needs of the current event (plus some additional pieces of information not discussed here). Thus `event[i]` is the *i*'th particle of the current event, and you may study its properties by using various `event[i].method()` possibilities.

The `event.list()` listing provides the main properties of each particles, by column:

- `no`, the index number of the particle (*i* above);
- `id`, the PDG particle identity code (method `id()`);
- `name`, a plain text rendering of the particle name (method `name()`), within brackets for initial or intermediate particles and without for final-state ones;
- `status`, the reason why a new particle was added to the event record (method `status()`);
- `mothers` and `daughters`, documentation on the event history (methods `mother1()`, `mother2()`, `daughter1()` and `daughter2()`);
- `colours`, the colour flow of the process (methods `col()` and `acol()`);
- `p_x`, `p_y`, `p_z` and `e`, the components of the momentum four-vector  $(p_x, p_y, p_z, E)$ , in units of GeV with  $c = 1$  (methods `px()`, `py()`, `pz()` and `e()`);
- `m`, the mass, in units as above (method `m()`).

For a complete description of these and other particle properties (such as production and decay vertices, rapidity,  $p_{\perp}$ , etc), open the program's online documentation in a browser (see Section 2, point 6, above), scroll down to the "Study Output" section, and follow the "Particle Properties" link in the left-hand-side menu. For brief summaries on the less trivial of the ones above, read on.

## A.1 Identity codes

A complete specification of the PDG codes is found in the Review of Particle Physics [3]. An online listing is available from

[http://pdg.lbl.gov/2008/mcdata/mc\\_particle\\_id\\_contents.html](http://pdg.lbl.gov/2008/mcdata/mc_particle_id_contents.html)

A short summary of the most common id codes would be

1	d	11	$e^-$	21	g	211	$\pi^+$	111	$\pi^0$	213	$\rho^+$	2112	n
2	u	12	$\nu_e$	22	$\gamma$	311	$K^0$	221	$\eta$	313	$K^{*0}$	2212	p
3	s	13	$\mu^-$	23	$Z^0$	321	$K^+$	331	$\eta'$	323	$K^{*+}$	3122	$\Lambda^0$
4	c	14	$\nu_{\mu}$	24	$W^+$	411	$D^+$	130	$K_L^0$	113	$\rho^0$	3112	$\Sigma^-$
5	b	15	$\tau^-$	25	$H^0$	421	$D^0$	310	$K_S^0$	223	$\omega$	3212	$\Sigma^0$
6	t	16	$\nu_{\tau}$			431	$D_s^+$			333	$\phi$	3222	$\Sigma^+$

Antiparticles to the above, where existing as separate entities, are given with a negative sign.

Note that simple meson and baryon codes are constructed from the constituent (anti)quark codes, with a final spin-state-counting digit  $2s + 1$  ( $K_L^0$  and  $K_S^0$  being exceptions), and with a set of further rules to make the codes unambiguous.

## A.2 Status codes

When a new particle is added to the event record, it is assigned a positive status code that describes why it has been added, as follows:

code range	explanation
11 – 19	beam particles
21 – 29	particles of the hardest subprocess
31 – 39	particles of subsequent subprocesses in multiple interactions
41 – 49	particles produced by initial-state-showers
51 – 59	particles produced by final-state-showers
61 – 69	particles produced by beam-remnant treatment
71 – 79	partons in preparation of hadronization process
81 – 89	primary hadrons produced by hadronization process
91 – 99	particles produced in decay process, or by Bose-Einstein effects

Whenever a particle is allowed to branch or decay further its status code is negated (but it is *never* removed from the event record), such that only particles in the final state remain with positive codes. The `isFinal()` method returns `true/false` for positive/negative status codes.

## A.3 History of parton shower branchings

The two mother and two daughter indices of each particle provide information on the history relationship between the different entries in the event record. The detailed rules depend on the particular physics step being described, as defined by the status code. As an example, in a  $2 \rightarrow 2$  process  $ab \rightarrow cd$ , the locations of  $a$  and  $b$  would set the mothers of  $c$  and  $d$ , with the reverse relationship for daughters. When the two mother or daughter indices are not consecutive they define a range between the first and last entry, such as a string system consisting of several partons fragment into several hadrons.

There are also several special cases. One such is when “the same” particle appears as a second copy, e.g. because its momentum has been shifted by it taking a recoil in the dipole picture of parton showers. Then the original has both daughter indices pointing to the same particle, which in its turn has both mother pointers referring back to the original. Another special case is the description of ISR by backwards evolution, where the mother is constructed at a later stage than the daughter, and therefore appears below in the event listing.

If you get confused by the different special-case storage options, the two `pythia.event.motherList(i)` and `pythia.event.daughterList(i)` methods are able

to return a `vector` of all mother or daughter indices of particle `i`.

## A.4 Colour flow information

The colour flow information is based on the Les Houches Accord convention [4]. In it, the number of colours is assumed infinite, so that each new colour line can be assigned a new separate colour. These colours are given consecutive labels: 101, 102, 103,  $\dots$ . A gluon has both a colour and an anticolour label, an (anti)quark only (anti)colour.

While colours are traced consistently through hard processes and parton showers, the subsequent beam-remnant-handling step often involves a drastic change of colour labels. Firstly, previously unrelated colours and anticolours taken from the beams may at this stage be associated with each other, and be relabelled accordingly. Secondly, it appears that the close space-time overlap of many colour fields leads to reconnections, i.e. a swapping of colour labels, that tends to reduce the total length of field lines.

## References

- [1] T. Sjöstrand, S. Mrenna and P. Skands, *Comput. Phys. Comm.* **178** (2008) 852 [arXiv:0710.3820]
- [2] T. Sjöstrand, S. Mrenna and P. Skands, *JHEP* **05** (2006) 026 [hep-ph/0603175]
- [3] Particle Data Group, C. Amsler et al., *Physics Letters* **B667** (2008) 1
- [4] E. Boos et al., in the Proceedings of the Workshop on Physics at TeV Colliders, Les Houches, France, 21 May - 1 Jun 2001 [hep-ph/0109068]
- [5] L. Lönnblad and S. Prestel, *JHEP* **03** (2012) 019, arxiv:1109.4829 [hep-ph]
- [6] L. Lönnblad and S. Prestel, *JHEP* **02** (2013) 094, arxiv:1211.4827 [hep-ph]
- [7] L. Lönnblad and S. Prestel, *JHEP* **03** (2013) 166, arxiv:1211.7278 [hep-ph]
- [8] For merging related questions, email `stefan.prestel@thep.lu.se`  
In case of general problems, contact us under `pythia8@projects.hepforge.org`