Table of Contents

Flavor History

Here we present tools to determine the ultimate history of a given jet.

There can be several sources for a jet:

- Gluon splitting.
- Quark excitation.
- Electroweak (or strong) resonance decay.
- Hard scattering.

We present tools to decide which category an event falls into.

Bug Fix History

- 09-March-23: Bug in W+c path description in V09-30-02, was not selecting W+c events. Fixed in V09-30-03.
- 09-July-01: Bug in W+Light flavor path in V09-30-03. Was also adding the "trash" samples for W+bb and W+cc when the partons matched the same jets.
- 10-Dec-20 : Clarified the "trash paths" of the flavor history filter, removed some obsolete recipes.

Definitions of Categories

The categories we use are as follows:

- **Matrix Element:** Status 3 parent with precisely 2 "grandparents" that is outside of the "initial" section (0-5) that has the same ID as the status 2 parton in question. NOTE: This is not the actual ultimate progenitor, but this is the signature of matrix element decays. The ultimate progenitor is the parent of the status 3 parton.

- **Flavor excitation:** Almost the same as the matrix element classification, but has only one outgoing parton product instead of two.

- **Gluon splitting:** Parent is a quark of a different flavor than the parton in question, or a gluon. Can come from either ISR or FSR.

- **True decay:** Decays from a resonance like top, Higgs, etc.

Software

Prescription

In all releases after CMSSW 3.x, the FlavorHistory object is part of the default CMSSW packages. No additional packages are necessary.

Flavor History Object

The Flavor History object is shown here. This object will store several objects:

- **Flavor Source**: matrix element, flavor excitation, gluon splitting, or true decay.
- **Parton**: The parton in question (status 2 quark, typically b or c).

- **Progenitor**: The ultimate progenitor of that parton.
- **Sister**: The sister of that parton.

In doing this, we can see that the cases in question will have the following characteristics:

- **Matrix element**
  - Parton: b or c quark.
  - Progenitor: parton from the hard scatter.
  - Sister: the other daughter of the progenitor with the same flavor.
- **Flavor excitation**
  - Parton: b or c quark.
  - Progenitor: parton from the hard scatter.
  - Sister: None.
- **Gluon splitting**
  - Parton: b or c quark.
  - Progenitor: gluon from ISR or FSR.
  - Sister: the other daughter of the progenitor with the same flavor.
- **True decay**
  - Parton: b or c quark.
  - Progenitor: Higgs, top, etc.
  - Sister: Depends on decay, but for instance a Higgs to b-bbar will have the other b as the sister.

Flavor History Event

This is a data type designed to make the content in Flavor History more accessible. It is linked here.

The interface to examine these objects is:

```
// Get the number of b and c genjets that match to partons
unsigned int         nb() const;
unsigned int         nc() const

// Accessor to maximum delta R between highest flavor constituent genjets
double               deltaR() const;
unsigned int         highestFlavor() const;
```

These are browsable in FWLite, for instance. They are also the values used in the filtering process. It also has an =STL=-style iterator interface.

Note that the kinematic quantities in the `FlavorHistoryEvent` object correspond to the matched `GenJet`, not the parton. This is to ensure that the partons in question will give rise to measurable jets. Therefore, the methods "nb" and "nc" correspond to the number of `GenJets` matched to b and c quarks, respectively, and the "deltaR" method is the delta R between the two `GenJets` matched to the same flavor. In the case of being matched to the same `GenJet`, then delta R returns 0. If there is no match for either this parton, or the sister, then the delta R method returns -1.

Flavor History Producer

The producer to create these objects is shown here. There is one file that will create both b and c parton histories linked here. This will create a FlavorHistoryEvent that the user can then select on.

Flavor History Filter

The filter to use these objects is shown here. This will take the FlavorHistoryEvent produced by the flavor history producer, and perform selection on its constituents. The selection is based on the matched `GenJets`

using the SISCone algorithm with a cone size of 0.5. The matching is simple delta R matching between the `GenJets` and the partons before the hadronization string in `PYTHIA`.

At the present, the filter will avoid overlaps based on the "delta R" method. In this method, the event is taken from the matrix element calculation if it contains two matched `GenJets` that are widely separated (i.e. are associated with two different jets). The event is taken from the parton shower calculation if it contains two matched `GenJets` that are very colinear (i.e. are associated with the same jet). The jet cone size is a parameter, and by default is taken to be 0.5.

Filtering in Full Framework

The configuration file to use in the full framework is shown here creates the filter for the users' analyses.

As a new addition, this filter will also produce a single unsigned integer corresponding to the paths as described here.

The hierarchy is:

1. W+bb with >= 2 jets from the ME (dr > 0.5)
2. W+b or W+bb with 1 jet from the ME
3. W+cc from the ME (dr > 0.5)
4. W+c or W+cc with 1 jet from the ME
5. W+bb with 1 jet from the parton shower (dr == 0.0)
6. W+cc with 1 jet from the parton shower (dr == 0.0)
7. W+bb with >= 2 partons but 1 jet from the ME (dr == 0.0)
8. W+cc with >= 2 partons but 1 jet from the ME (dr == 0.0)
9. W+bb with >= 2 partons but 2 jets from the PS (dr > 0.5)
10. W+cc with >= 2 partons but 2 jets from the PS (dr > 0.5)
11. Veto of all the previous (W+ light jets)

Note that paths 1-6, plus sample 11, are the paths you will use in your analysis. Paths 7-10 are not included in any of the samples as this is the double counting that we are trying to eliminate.

The physics content of the samples for your analysis should break down as follows:

- **W+bb** :
  - From the "V+QQ" sample: Paths 1 and 2 (for the matrix element contribution).
  - From the "W+jets" sample: Path 5 (for the gluon splitting contribution).
- **W+cc** :
  - From the "V+QQ" sample: Paths 3 and 4 (for the matrix element contribution).
  - From the "W+Jets" sample: Path 6 (for the gluon splitting contribution).
- **W+c**
  - From the "W+c" sample: Path 4 (for true flavor excitation).
- **W+light flavor**:
  - From the "W+jets" sample: Path 11 (veto of all heavy flavor content).

The remainder of the paths (7-10) are defined for studies and completeness. They contain samples of phase space that are unreliable (such as using the matrix element calculation in the "colinear" case, or using the parton shower in the case of widely separated partons), and hence **the events** should be discarded for the analysis. Note that **the normalization** still must be taken into account when stitching samples back together.

The path is the only integer produced by this filter, so to access it:

- In bare root:

```
Events->SetAlias("path", "uint_flavorHistoryFilter__TEST.obj");
Events->Draw("path");
```

- In FWLite:

```
fwlite::Handle<unsigned int> path;
path        .getByLabel(ev, "flavorHistoryFilter" );
```

This can then be used in assigning physics content as described above.

Filtering in FWLite

As a new development, it is now possible to filter events directly in FWLite without the need for the filter itself. It is slightly long to do so, however, and if you are unsure as to what to do, it is safer to use the full framework "canned" prescription as described above. To use this in FWLite, however, include in your BuildFile:

```
<bin name=run_flavor_history file=run_flavor_history.cc,flavor_history.cc>
 <use name=root>
 <use name=rootcintex>
 <use name=boost>
 <use name=FWCore/FWLite>
 <use name=FWCore/Framework>
 <use name=DataFormats/FWLite>
 <use name=DataFormats/HepMCCandidate>
</bin>
```

The file `run_flavor_history.cc` looks something like:

```
#include "FWCore/FWLite/interface/AutoLibraryLoader.h"
#include "TFile.h"
#include "TSystem.h"
#include "Analysis/FlavorHistory/bin/flavor_history.cc"

#include <iostream>
#include <string>
using namespace std;

int main (int argc, char ** argv)
{


  if ( argc < 2 ) {
    cout << "usage: run_flavor_history <sample>" << endl;
    return 0;
  }


  gSystem->Load("libFWCoreFWLite");
  AutoLibraryLoader::enable();

  string sample(argv[1]);


  flavor_history( sample );


  return 0;
}
```

The file `flavor_history.cc` looks something like:

```
#include "DataFormats/FWLite/interface/Handle.h"
```

```
#include "DataFormats/FWLite/interface/Event.h"
#include <TH1.h>
#include <TFile.h>
#include <TDCacheFile.h>


#if !defined(__CINT__) && !defined(__MAKECINT__)
#include "DataFormats/HepMCCandidate/interface/FlavorHistoryEvent.h"
#include "CMS.PhysicsTools/HepMCCandAlgos/interface/FlavorHistorySelectorUtil.h"
#endif

#include <iostream>
#include <map>
#include <string>

using namespace std;

void flavor_history(std::string sample)
{

  cout << "processing sample " << sample << endl;

  vector<string> files;

  // input your files here

  using namespace std;
  using namespace reco;

  const Int_t NPATHS = 11;

  TH1D * hist_path         = new TH1D("path", "Flavor History Path", NPATHS+1, 0, NPATHS+1 )

  cout << "About to make filter helpers" << endl;

  double dr_ = 0.5;
  bool verbose = false;

  // Set up the boundaries.
  // dr0 = 0.0
  // dr1 = set by user
  // dr2 = infinity
  double dr0 = 0.0;
  double dr1 = dr_;
  double dr2 = 99999.0;


  // These are the processes that can come from the matrix element calculation
  std::vector<int> me_ids;
  me_ids.push_back(2);   // flavor excitation
  me_ids.push_back(3);   // flavor creation

  // These are the processes that can come from the parton shower calculation
  std::vector<int> ps_ids;
  ps_ids.push_back(1);   // gluon splitting


  // To select bb->2 events from matrix element... Path 1
  FlavorHistorySelectorUtil * bb_me  = new FlavorHistorySelectorUtil( 5,
                2,
                me_ids,
                dr1,
                dr2,
                verbose );

  // To select  b->1 events from matrix element... Path 2
  FlavorHistorySelectorUtil *   b_me   = new FlavorHistorySelectorUtil( 5,
```

```
                1,
                me_ids,
                dr0,
                dr0,
                verbose );


// To select cc->2 events from matrix element... Path 3
FlavorHistorySelectorUtil *   cc_me  = new FlavorHistorySelectorUtil( 4,
                2,
                me_ids,
                dr1,
                dr2,
                verbose );

// To select  c->1 events from matrix element... Path 4
FlavorHistorySelectorUtil *   c_me   = new FlavorHistorySelectorUtil( 4,
                1,
                me_ids,
                dr0,
                dr0,
                verbose );

// To select bb->2 events from parton shower ... Path 5
FlavorHistorySelectorUtil *   b_ps   = new FlavorHistorySelectorUtil( 5,
                1,
                ps_ids,
                dr0,
                dr1,
                verbose );


// To select cc->2 events from parton shower ... Path 6
FlavorHistorySelectorUtil *   c_ps   = new FlavorHistorySelectorUtil( 4,
                1,
                ps_ids,
                dr0,
                dr1,
                verbose );

// To select bb->1 events from matrix element... Path 7
FlavorHistorySelectorUtil *   bb_me_comp  = new FlavorHistorySelectorUtil( 5,
                    2,
                    me_ids,
                    dr0,
                    dr1,
                    verbose );

// To select cc->1 events from matrix element... Path 8
FlavorHistorySelectorUtil *   cc_me_comp  = new FlavorHistorySelectorUtil( 4,
                    2,
                    me_ids,
                    dr0,
                    dr1,
                    verbose );

// To select bb->2 events from parton shower ... Path 9
FlavorHistorySelectorUtil *   b_ps_comp   = new FlavorHistorySelectorUtil( 5,
                    2,
                    ps_ids,
                    dr1,
                    dr2,
                    verbose );

// To select cc->1 events from parton shower ... Path 10
FlavorHistorySelectorUtil *   c_ps_comp   = new FlavorHistorySelectorUtil( 4,
                    2,
```

```
                    ps_ids,
                    dr1,
                    dr2,
                    verbose );

// The veto of all of these is               ... Path 11



cout << "About to make chain event" << endl;
fwlite::ChainEvent ev(files);

cout << "About to loop" << endl;

unsigned int count = 0;
for( ev.toBegin();
     ! ev.atEnd();
     ++ev, ++count) {

  if ( count % 1000 == 0 ) cout << "Processing event " << count << endl;

  unsigned int pathi = 0;
  unsigned int * path = &pathi;

  fwlite::Handle<reco::FlavorHistoryEvent > bFlavorHistoryEvent;
  bFlavorHistoryEvent   .getByLabel(ev,"bFlavorHistoryProducer", "bPartonFlavorHistory");

  fwlite::Handle<reco::FlavorHistoryEvent > cFlavorHistoryEvent;
  cFlavorHistoryEvent   .getByLabel(ev,"cFlavorHistoryProducer", "cPartonFlavorHistory");

  if ( ! bFlavorHistoryEvent.isValid() ||
  ! cFlavorHistoryEvent.isValid() ) continue;


  // Get the number of matched b-jets in the event
  unsigned int nb = bFlavorHistoryEvent->nb();
  // Get the number of matched c-jets in the event
  unsigned int nc = cFlavorHistoryEvent->nc();
  // Get the two flavor sources. The highest takes precedence
  // over the rest.
  FlavorHistory::FLAVOR_T bFlavorSource = bFlavorHistoryEvent->flavorSource();
  FlavorHistory::FLAVOR_T cFlavorSource = cFlavorHistoryEvent->flavorSource();
  FlavorHistory::FLAVOR_T flavorSource = FlavorHistory::FLAVOR_NULL;
  // Get the highest flavor in the event
  unsigned int highestFlavor = 0;
  // Get the delta r between the two heavy flavor matched jets.
  double dr = -1;

  // Preference is in increasing priority:
  //  1: gluon splitting
  //  2: flavor excitation
  //  3: flavor creation (matrix element)
  //  4: flavor decay
  if ( bFlavorSource >= cFlavorSource ) {
    flavorSource = bFlavorHistoryEvent->flavorSource();
    highestFlavor = bFlavorHistoryEvent->highestFlavor();
    dr = bFlavorHistoryEvent->deltaR();
  }
  else {
    flavorSource = cFlavorHistoryEvent->flavorSource();
    highestFlavor = cFlavorHistoryEvent->highestFlavor();
    dr = cFlavorHistoryEvent->deltaR();
  }


  *path = 0;
```

```
    // Now make hierarchical determination
    if      ( bb_me     ->select( nb, nc, highestFlavor, flavorSource, dr ) ) *path = 1;
    else if (  b_me     ->select( nb, nc, highestFlavor, flavorSource, dr ) ) *path = 2;
    else if ( cc_me     ->select( nb, nc, highestFlavor, flavorSource, dr ) ) *path = 3;
    else if (  c_me     ->select( nb, nc, highestFlavor, flavorSource, dr ) ) *path = 4;
    else if (  b_ps     ->select( nb, nc, highestFlavor, flavorSource, dr ) ) *path = 5;
    else if (  c_ps     ->select( nb, nc, highestFlavor, flavorSource, dr ) ) *path = 6;
    else if ( bb_me_comp->select( nb, nc, highestFlavor, flavorSource, dr ) ) *path = 7;
    else if ( cc_me_comp->select( nb, nc, highestFlavor, flavorSource, dr ) ) *path = 8;
    else if (  b_ps_comp->select( nb, nc, highestFlavor, flavorSource, dr ) ) *path = 9;
    else if (  c_ps_comp->select( nb, nc, highestFlavor, flavorSource, dr ) ) *path = 10;
    else *path = 11;


    hist_path->Fill( *path );

    if ( *path > NPATHS) continue;

  }


}
```

Full recipes for Summer 08/Fall 08

The above procedure has been checked in the Fall 08 pre-production samples here:

- /VQQ-madgraph/Fall08_IDEAL_V9_v1_pre-production/GEN-SIM-RECO
- /WJets-madgraph/Fall08_IDEAL_V9_v1_pre-production/GEN-SIM-RECO
- /Wc-madgraph/Fall08_IDEAL_V9_reco-v1/GEN-SIM-RECO

An example can be found here:

`CMS.PhysicsTools/HepMCCandAlgos/test/testFlavorHistoryProducer.py`

-- SalvatoreRRappoccio - 06 Aug 2008

This topic: CMSPublic > SWGuideFlavorHistory
Topic revision: r12 - 2010-12-20 - SalvatoreRRappoccio