Table of Contents

4.5 MC Truth Matching Tools

Complete:
Detailed Review status

Goals of this page:

This page is intended to familiarize you with the tools to match the reconstructed object to generated particle.

Contents

Purpose

Provide common tool to make easier writing Monte Carlo generator *truth* matching.

Introduction

The matching tools assume we use the new format for HepMCProduct using Particle Candidates.

The Monte Carlo truth matching of a composite object (for instance, a reconstructed $Z \rightarrow \mu^+\mu^-$) is done in two steps:

1. matching of final state particles to final state generator particles (in this case the muons)
2. automatic matching of reconstructed composite objects to composite MC parents (in this case the Z)

The output product of the first stage is a one-to-one AssociationMap that can be stored in the event, and works as input for the second step.

These matching tools are based on:

- Compact generator collection
- Simplified association map type

Matching Format

Using `AssociationMap`

Object matchings are stored in the event with the format of an one-to-one AssociationMap. For each matched object of a collection `A`, a unique object in a collection `B` is stored. For convenience, the following typedef is defined in `DataFormats/Candidate/interface/CandMatchMap.h`:

```
namespace reco {
  typedef edm::AssociationMap<
            edm::OneToOne<reco::CandidateCollection, reco::CandidateCollection>
          > CandMatchMap;
}
```

An example of code accessing an association is the following:

```
  Handle<CandMatchMap> match;
  event.getByLabel( "zToMuMuGenParticlesMatch", match );

  CandidateRef cand = ...; // get your reference to a candidate
  CandidateRef mcMatch = (*match)[ cand ];
```

You can access those maps in FWLite. Internally, the association map stores indices to the matched objects in the two source collections in the data member `map_`.

For instance, to plot the reconstructed muon p_t versus the true p_t, from generator particles, you can use the following interactive ROOT command:

```
Events.Draw("allMuons.data_[allMuonsGenParticlesMatch.map_.first].pt():
  genParticleCandidates.data_[allMuonsGenParticlesMatch.map_.second].pt()")
#edit the above two lines to be a single line
```

In the above example, the following branches have been used:

- `allMuons`: the collection of muon candidates
- `genParticleCandidates`: the collection of generator level particles,
- `allMuonsGenParticlesMatch`: the name of the module used to create the match map of muon candidates to generator particles. The ROOT branch containing the map will have an alias identical to the module name.

Some problems with ROOT to manage dictionary properly has been experienced, so some expected interactive use patterns may give problems. Please, report them to the Analysis Tools group.

Using `Association`

It is also possible to use object matchings with the format of an one-to-one Association. For each matched object of any type, a unique object in a collection of `GenParticle` objects is stored. For convenience, the following typedef is defined in `DataFormats/HepMCCandidate/interface/GenParticleFwd.h`:

```
namespace reco {
  typedef edm::Association<GenParticleCollection> GenParticleMatch;
}
```

An example of code accessing an association is the following:

```
  Handle<GenParticleMatch> match;
  event.getByLabel( "zToMuMuGenParticlesMatch", match );
```

```
CandidateRef cand = ...; // get your reference to a candidate
GenParticleRef mcMatch = (*match)[cand];
```

Accessing those kinds of maps in FWLite is possible. It is trivial when objects from a single collection are matched (you basically store internally a vector of matched indices), but interpreting the data may require non trivial unpacking if more than one collection is matched. Work is ongoing on the EDM side to review in general association maps.

ΔR Matching Modules

Using `MCTruthDeltaRMatcher`

The module `MCTruthDeltaRMatcher` defined in `CMS.PhysicsTools/HepMCCandAlgos` matches candidate collection to their MC truth parent based on a maximum ΔR are provided. Optionally, only particles with a given PDG id are matched. One example of configuration is the following:

```
process.selectedMuonsGenParticlesMatch = cms.EDProducer( "MCTruthDeltaRMatcher",
                                                 src = cms.InputTag("selectedLayer1Muons"),
                                                 matched = cms.InputTag("genParticleCandidates"),
                                                 distMin = cms.double(0.15),
                                                 matchPDGId = cms.vint32(13)
                                                 )
```

This example will create a map between the selected PAT layer 1 muons and the generator level muons that are matched within a cone of 0.15.

The follwing `cfi.py` are provided in the `CMS.PhysicsTools/HepMCCandAlgos` directory to match pre-defined candidate sequences:

- `allMuonsGenParticlesMatch_cfi`
- `allElectronsGenParticlesMatch_cfi`
- `allTracksGenParticlesMatch_cfi`
- `allTracksGenParticlesLeptonMatch_cfi`
- `allSuperClustersGenParticlesLeptonMatch_cfi`
- `allStandAloneMuonTracksGenParticlesLeptonMatch_cfi`

Using `MCTruthDeltaRMatcherNew`

The module `MCTruthDeltaRMatcherNew` defined in `CMS.PhysicsTools/HepMCCandAlgos` matches any collection matching `edm::View<Candidate>` to their MC truth parent based on a maximum ΔR are provided. Optionally, only particles with a given PDG id are matched. One example of configuration is the following:

```
process.selectedMuonsGenParticlesMatchNew = cms.EDProducer( "MCTruthDeltaRMatcherNew",
                                                 src = cms.InputTag("selectedLayer1Muons"),
                                                 matched = cms.InputTag("genParticleCandidates"),
                                                 distMin = cms.double(0.15),
                                                 matchPDGId = cms.vint32(13)
                                                 )
```

Composite Object Matching

Using `MCTruthCompositeMatcher`

Once you have done RECO-MC truth matching for final state particles, you may want to reconstruct

composite objects, like $Z \rightarrow \mu^+\mu^-$ or $H \rightarrow ZZ \rightarrow \mu^+\mu^-e^+e^-$, and then find the corresponding parent match (if the Z or Higgs are correctly reconstructed) in the Monte Carlo truth.

The module `MCTruthCompositeMatcher` creates an association map of reconstructed composite objects to their corresponding generator parent, based on the association of their daughters.

The following example, taken from Electro-Weak $Z \rightarrow \mu^+\mu^-$ skim matches $Z \rightarrow \mu^+\mu^-$ candidate to MC truth based on the matching of muon daughters:

```
process.zToMuMuGenParticlesMatch = cms.EDProducer( "MCTruthCompositeMatcher",
                  src = cms.InputTag("zToMuMu"),
                  matchMaps = cms.VInputTag("selectedLayer1MuonsGenParticlesMatch")
                                           )
```

Using `MCTruthCompositeMatcherNew`

The module `MCTruthCompositeMatcherNew` creates an association map of reconstructed composite objects to their corresponding generator parent, based on the association of their daughters.

One example of matching of $Z \rightarrow \mu^+\mu^-$ given a match map of muons to generator particles, is the following:

```
process.zToMuMuMCMatch = cms.EDProducer( "MCTruthCompositeMatcherNew",
               src = cms.InputTag("zToMuMu"),
               matchMaps = cms.VInputTag("selectedMuonsGenParticlesMatchNew")
                                    )
```

Merging MC match maps

The new map type allow very simple way to merge them, whatever is the type of input collection that was matched. A single merged map can be saved instead of many single maps, if needed. The way to match then is trivial, following the example below:

```
process.mergedMCMatch = cms.EDProducer( "GenParticleMatchMerger",
             src = cms.VInputTag("muonMCMatch", "electronMCMatch",
             "trackMCMatch", "zToMuMuMCMatch", "zToEEMCMatch",
             "HTo4lMCMatch")
                                           )
```

Warning: merged maps are non trivial to inspect via FWLite. Work is ongoing on the EDM side to review association maps.

Composite Matching Utility

Using `MCCandMatcher<C1, C2>`

The utility `MCCandMatcher<C1, C2>` defined in `CMS.PhysicsTools/HepMCCandAlgos` does the matching within your analyzer, if you prefer not to run a framework module. `C1` and `C2` could be either `reco::CandidateCollection`, or any collection of objects inheriting from `reco::Candidate`, like jets, electrons, muons, etc. or `edm::View<reco::Candidate>`. The utility takes as input the one-to-one match map containing the final state matches, e.g.: the one produced with the module `MCTruthDeltaRMatcher` described above.

The usage is the following:

```
// get the previously produced final state match
Handle<CandMatchMap> mcMatchMap;
evt.getByLabel( matchMap_, mcMatchMap );

// create the extended matcher that includes automatic parent matching
MCCandMatcher<reco::CandidateCollection, reco::CandidateCollection> match( * mcMatchMap );

// get your candidate
const Candidate & cand = ...
// find match reference
CandidateRef mc = match( cand );
// access matched parent if non null
if ( mc.isNonnull() ) {
  int pdgId = pdgId( * mc );
  double mass = mc->mass();
}
```

The map can also find matches of final state particles, just looking at the input final state match map.

Using `utilsNew::CandMatcher<GenParticleCollection>`

The utility `MCCandMatcher` should be replaced by `utilsNew::CandMatcher<GenParticleCollection>`, defined in: `CMS.PhysicsTools/CandUtils`.

An example of usage is reported below:

```
using namespace edm;
using namespace std;
using namespace reco;
// get your collection of composite objects
Handle<CandidateView> cands;
evt.getByLabel(src_, cands);

// get your match maps for final state
// (electrons, muons, tracks, ...)
size_t nMaps = matchMaps_.size();
std::vector<const GenParticleMatch *> maps;
maps.reserve( nMaps );
for( size_t i = 0; i != nMaps; ++ i ) {
  Handle<reco::GenParticleMatch> matchMap;
  evt.getByLabel(matchMaps_[i], matchMap);
  maps.push_back(& * matchMap);
}
// create cand matcher utility passing the input maps
utilsNew::CandMatcher<GenParticleCollection> match(maps);

int size = cands->size();
for( int i = 0; i != size; ++ i ) {
  const Candidate & cand = (* cands)[i];
  // get MC match for specific candidate
  GenParticleRef mc = match[cand];
}
```

Complete Running Examples

An example of complete analysis using MC matching tools is the *Z* reconstruction skim from the EWK Analysis Group. See:

- Z Reconstruction for Analysis Skims

Generic Candidate Matching

Final state MC truth matching has been written in such a way that it could be extended to different types of matching, even not specifically MC truth matching. The "generality" could be further extended, if needed.

A generic match module is defined in the package `CMS.PhysicsTools/CandAlgos` by the following template, defined in the namespace `reco::modules`:

```
template<typename S, typename D = DeltaR<reco::Candidate> >
class CandMatcher;
```

where `S` is a (typically, but not only RECO-MC) pair selector type (see below), and `D` is an utility to measure the match "distance" of two Candidates. By default, the distance is measured as **ΔR** (the utility `DeltaR` is defined in `CMS.PhysicsTools/CandUtils`), but other criteria could be adopted and easily plugged-in by the user.

An example of RECO-MC pair selection `S` is defined in the package `CMS.PhysicsTools/HepMCCandAlgos`, and checks that the Monte Carlo particles belong to the final state (`status = 1`) and have the same charge as the particle to be matched:

```
struct MCTruthPairSelector {
  explicit MCTruthPairSelector( const edm::ParameterSet & ) { }
  bool operator()( const reco::Candidate & c, const reco::Candidate & mc ) const {
    if ( reco::status( mc ) != 1 ) return false;
    if ( c.charge() != mc.charge() ) return false;
    return true;
  }
};
```

This selection is applied before applying the **ΔR** cut.

The actual selector module is defined as `CMS.PhysicsTools/HepMCCandAlgos`:

```
typedef reco::modules::CandMatcher<
          helpers::MCTruthPairSelector
        > MCTruthDeltaRMatcher;
```

More details and usage examples can be found in:

- SWGuide section on Candidate Matching Modules

Physics Object Matching

An extended version of the `CandMatcher` has been created for the physics object toolkit. The `PhysObjectMatcher` allows a more general matching and an ambiguity resolution for multiple matches. It can be configured using 5 template arguments:

C1	The collection to be matched (e.g., a `CandidateView` of reconstructed objects)
C2	The target collection (e.g., a collection of `GenParticles`)
S	A preselector for the match (e.g., a selection on PDG id or status)
D	The class determining a match between two objects (default: deltaR)
Q	The ranking of matches (default: by increasing deltaR)

The module produces an `Association` from C1 to C2. Configuration parameters are

src	InputTag for C1
matched	InputTag for C2
resolveAmbiguities	bool to enable / disable the resolution of ambiguities. If false each object in C1 is associated to the best match in C2, but several objects in C1 can point to the same object in C2.
resolveByMatchQuality	bool to choose the type of ambiguity resolution. If true multiple associations of the same object in C2 are resolved by choosing the best match, otherwise the match with the lowest index in C1 is chosen.

Options for the helper classes used by `PhysObjectMatcher` are

S	`MCMatchSelector`	Preselection for MC matches	
		checkCharge	bool: use / ignore electrical charge
		mcPdgId	vint32: MC particle codes
		mcStatus	vint32: MC status codes
	`DummyMatchSelector`	no preselection	
D	`MatchByDR`	deltaR match	
		maxDeltaR	cut on deltaR
	`MatchByDRDPt`	match by deltaR and relative deltaPt	
		maxDeltaR	cut on deltaR
		maxDPtRel	cut on fabs(pt2-pt1)/pt2
Q	`reco::helper::LessByMatchDistance<D,C1,C2>`	ranking by distance (e.g., `DeltaR`)	
	`MatchLessByDPt`	ranking by relative deltaPt	

Some concrete matching modules are defined in `PhysicsTools/HepMCCandAlgos/plugins/MCTruthMatchers.cc` :

- `MCMatcher` : deltaR + deltaPt match between a `CandidateView` and a `GenParticleCollection`; ranking by deltaR
- `MCMatcherByPt` : as above, but ranking by deltaPt
- `GenJetMatcher` : deltaR match between a `CandidateView` and `GenJetCollection`; ranking by deltaR

Warning: Matching in Dense Environments

Matching by ΔR may not work reliably in dense environments, such as jets. For studies needing high quality matching of reconstructed tracks with true tracks, it is possible to base the matching either on the number of hits that they share in common, or on a comparison of the 5 helix parameters describing the track. How to do this is described here, but unfortunately can only be done on FEVT data, since it requires the presence of TrackingParticles that are not stored on RECO. (These are truth tracks, which contain links to the GEANT-produced SimTracks and generator-produced GenParticles that they correspond to).

Review status

Reviewer/Editor and Date (copy from screen)	Comments
XuanChen - 31 Jul 2014	changed the links from cvs to github
WolfgangAdam - 21 Apr 2008	added PhysObjectMatcher
LucaLista - 10 Dec 2007	author
JennyWilliams - 11 Dec 2006	Moved to workbook, added contents, added review info
JennyWilliams - 24 Jul 2007	fixed latex errors on equations

IanTomalin - 16 Oct 2009	Added reference to track matching by hit or helix

Responsible: LucaLista
Last reviewed by: PetarMaksimovic - 28 Feb 2008

This topic: CMSPublic > WorkBookMCTruthMatch
Topic revision: r42 - 2014-07-31 - XuanChen