# Hierarchy Tools

August 25, 2022

## 1 TLDR

To make use of the Hierarchy Helper to perform matching between reconstruction and truth you can refer to the code snippet in Listing 1 (note that visualisation and monitoring classes already exist, this section is intended to show how you can get match information if you want to perform non-standard tasks).

```
1  const CaloHitList *pCaloHitList(nullptr);
2  PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::GetList(*this,
       m_caloHitListName, pCaloHitList));
3  const MCParticleList *pMCParticleList(nullptr);
4  PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::GetCurrentList(*this,
       pMCParticleList));
5  const PfoList *pPfoList(nullptr);
6  PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::GetList(*this,
       m_pfoListName, pPfoList));
7
8  LArHierarchyHelper::FoldingParameters foldParameters;
9  if (m_foldToPrimaries)
10     foldParameters.m_foldToTier = true;
11 else if (m_foldDynamic)
12     foldParameters.m_foldDynamic = true;
13
14 LArHierarchyHelper::MCHierarchy mcHierarchy;
15 LArHierarchyHelper::FillMCHierarchy(*pMCParticleList, *pCaloHitList, foldParameters,
       mcHierarchy);
16 LArHierarchyHelper::RecoHierarchy recoHierarchy;
17 LArHierarchyHelper::FillRecoHierarchy(*pPfoList, foldParameters, recoHierarchy);
18 LArHierarchyHelper::MatchInfo matchInfo;
19 LArHierarchyHelper::MatchHierarchies(mcHierarchy, recoHierarchy, matchInfo);
```

Listing 1: "Matching reco and truth with the Hierarchy Helper."

Lines 1-6 perform the usual set up of lists for CaloHits, MCParticles and ParticleFlowObjects. Lines 8-12 set up the folding parameters, where you can choose the type of MC folding option. By default no folding is performed, but specifying folding to tiers will fold hierarchies back to primary particles, as has historically been the case for Pandora metrics, while the new dynamic folding method retains information about parent/child relationships. Lines 14-17 fill the respective truth and reconstructed hierarchies according to the specified folding rules, while 18-19 performs the match between those hierarchies and stores the matching results in matchInfo.

## 2 The Hierarchy Tools Concept

Hierarchy Tools is conceptually simple; when assessing reconstruction performance it is necessary to relate reconstructed quantities to true quantities, and this is what Hierarchy Tools provides. Prior to the introduction of Hierarchy Tools this was a rather manual process with a substantial coding overhead and potentially error-prone. Different classes existed for the validation of neutrino and test beam hierarchies for example and support for multiple neutrino interactions, as in NDLAr were unsupported. Hierarchy Tools introduces a set of helper classes that provide a common interface to query the relationship between reconstructed and true quantities in a way that is common to any kind of particle hierarchy, and allows different rules to be applied for how reconstructed and true hierarchies should be compared and what the thresholds are for reconstruction to be considered correct. A Python environment to gather summary statistics and produce standard plots based on the ROOT tree output that can be produced by the Hierarchy Tools is in development.

The Hierarchy Tools are composed of a few core classes (not exhaustive); `MCHierarchy`, `RecoHierarchy`, `MCMatches`, `MatchInfo`, and `FoldingParameters`. As you might imagine, the first two describe the true particle hierarchy and the reconstructed hierarchy given the list of `CaloHits`, `MCParticles` and `ParticleFlowObjects`, most typically at the end of Pandora reconstruction, but these hierarchies can be populated at any point where

the respective lists exist (though the usefulness of mid-reconstruction hierarchies is questionable). `MCMatches` links together each true particle and all of the corresponding reconstructed particles that can be matched to it (hopefully 1, but if the true particle is fragmented by the reconstruction there can be more than one match, and if the reconstruction fails to construct a matching particle then the true particle will have an empty list of matched reconstructed particles). The `MatchInfo` class performs the actual matching procedure given true and reconstructed hierarchies, creating a collection of `MCMatches` objects for a given event and provides an interface for accessing the match information. A key aspect of the Hierarchy Tools is the ability to determine how true and reconstructed particles should be matched.

Historically, two options existed, either matching every individual true particle, or identifying the final state particles and combining all subsequent downstream activity from each final state particle with that final state particle to form a single object to be matched. Each of these approaches is somewhat problematic. In the former case, trying to match every nuance of the true interaction is often impractical, because the true particle hierarchy is frequently very complex and retains details of, for example, tiny elastic scatters that the reconstruction will never be able to distinguish, resulting in unreasonable matching failures, and can also result in many particles with very few hits, which it is also unreasonable to expect to be reconstructed. The practical solution to this was to combine together final state particles with their downstream interactions, such that small elastic scatters are no longer relevant, as all hits going into and out of the non-primary interaction vertex are rolled into a single object, and similarly low hit count particles are subsumed into larger objects. This creates a much more reasonable matching environment, but suffers from the problem that details of the interaction are lost, so errors in the particle hierarchy can be hidden.

Hierarchy Tools provides the `FoldingParameters` class to help address such issues without altering the process for requesting hierarchy matching or querying the results of that matching procedure. By specifying a small set of folding parameters it is possible to choose from several different options for matching the hierarchies. It remains possible to request that particles are folded back to the final state particles, and to request the full, unfolded matching process. However, in addition, it is now possible to request folding to a particular tier in the hierarchy, that is, final state particles are tier 1, their immediate child particles are tier 2 and so on, so if you wanted to isolate final state particles, but then fold all of the downstream interaction details back to the second, you can. Folding to tier 3 would result in tier 1 and tier 2 particles retaining independence, but tier 3, 4, etc would be combined. There is also a 'dynamic folding' option, which attempts to intelligently combine particles together such that a degree of detail is retained in the particle hierarchy, but small elastic scatters result in folding of parent/child/grandchild/etc particles. The dynamic folding approach aims to find a balance between folding to final state particles and fully unfolded hierarchies.

# 3 Using the Hierarchy Tools

This section will describe a few ways to use the Hierarchy Tools. The first two approaches represent the simplest case, in which you can take advantage of existing algorithms to perform standard validation tasks, or visualise matching information. The third approach looks in a little more detail at how to run the matching and the functions available to query the results if you are looking to explore specific aspects of the matching result. Listing 2 shows an example algorithm to produce event-level validation output, using dynamic folding and writing the information to a ROOT file.

```
<algorithm type = "LArHierarchyValidation">
    <ValidateEvent>true</ValidateEvent>
    <WriteTree>true</WriteTree>
    <FileName>hierarchy_validation_event.root</FileName>
    <TreeName>events</TreeName>
    <FoldDynamic>true</FoldDynamic>
</algorithm>
```

Listing 2: "Running the standard event-level reconstruction validation."

Event-level validation provides information about the interaction type, the number of nodes (a node is similar to a particle, but accounts for folding such that a single particle and a combination of folded particles are both considered one node), the number of well matched nodes, poorly matched nodes and the number of unmatched nodes. It also has equivalent counts specifically for tier 1 nodes and extends this to consider track-like and shower-like nodes specifically. Event-level information also incorporates information about the presence of a leading lepton, and whether or not this is correctly matched. Finally, the delta between the true and reconstructed neutrino vertex is recorded.

Listing 3 shows the corresponding algorithm to produce particle-level validation output, using dynamic folding and writing the information to a ROOT file.

```
<algorithm type = "LArHierarchyValidation">
    <ValidateMC>true</ValidateMC>
```

```
    <WriteTree>true</WriteTree>
    <FileName>hierarchy_validation_mc.root</FileName>
    <TreeName>mc</TreeName>
    <FoldDynamic>true</FoldDynamic>
</algorithm>
```

Listing 3: "Running the standard MC particle-level reconstruction validation."

Particle/Node-level validation, as the name suggests, provides detailed node-level information. Each MC node records the event number, the interaction number (e.g. for many neutrino interactions in a single NDLAr spill), a unique true node identifier, the PDG code of the leading (in the case of folding) particle, the tier of the node and the number of hits. It further identifies the nature of the interaction, that is, whether it is part of a neutrino interaction hierarchy, cosmic ray or test beam particle. Whether the node is a leading lepton or Michel electron is also noted. The number of matched reconstructed nodes is recorded, along with a list of unique identifiers for each reconstructed node and their respective reconstructed slice. The number of hits in each reconstructed node is recorded, along with a corresponding list of how many of those hits are shared with the true node. Purity and completeness lists are recorded to match each reconstructed node. Purity and completeness have a number of different variants; an overall metric for all hits from all views, corresponding metrics where each hit is weighted by its ADC count and equivalent lists for the separate U, V and W views. Finally, vertex delta information is recorded for each node.

Another pre-existing algorithm provides visualisation of matches. Listing 4 shows example usage. This visualisation tool has a number of options to display the individual reconstructed and true hierarchies based on specific matching criteria, such as dynamic folding. Rather than visualising individual particles, nodes are visualised, and parameters can be specified to show which reconstructed nodes are matched to which true nodes, distinct nodes can be rendered, or you can simply choose to colour code hits according to whether or not they are matched. In addition, you can choose to render only the collection plane to avoid overly complex displays.

```
<algorithm type = "LArHierarchyMonitoring">
    <VisualizeMC>true</VisualizeMC>
    <VisualizeDistinct>true</VisualizeDistinct>
    <CollectionOnly>true</CollectionOnly>
    <PerformMatching>true</PerformMatching>
</algorithm>
```

Listing 4: "Running the matching visualisation."

Finally, we can look in more detail at how you go about constructing these hierarchies and performing matching if you want to go beyond the pre-existing monitoring and validation options.

To make use of the tools you first need to get access to the Pandora object lists that are relevant to your requirements. This can use the standard approach, as shown in Listing 5. Note that you may not need all of these lists depending on your requirements, but they're included here for completeness, and it's hopefully clear how you would need to modify this for your specific needs.

```
const CaloHitList *pCaloHitList(nullptr);
PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::GetList(*this,
    m_caloHitListName, pCaloHitList));
const MCParticleList *pMCParticleList(nullptr);
PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::GetCurrentList(*this,
    pMCParticleList));
const PfoList *pPfoList(nullptr);
PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::GetList(*this,
    m_pfoListName, pPfoList));
```

Listing 5: "Getting the Pandora objects you require."

One decision you'll have to make is which folding parameters to use. Listing 6 provides an example using the dynamic folding. Other alternatives are `m_foldToTier`, or to set both of these parameters to `false` to leave the hierarchy unfolded.

```
LArHierarchyHelper::FoldingParameters foldParameters;
foldParameters.m_foldDynamic = true;
```

Listing 6: "Setting folding parameters."

When folding to a tier, the default is to fold to tier 1, i.e. primary particles, which effectively matches the behaviour of the `NeutrinoEventValidation` algorithm. You can specify a different tier via the `m_tier` parameter. The final parameter of note is the `m_cosAngleTolerance`, which defaults to `0.9962f` and is used by dynamic folding to determine when scattering events should be ignored and treated as a continuation of an existing particle.

Listing 7 shows how to construct a truth hierarchy. You simply need to provide the list of MC particles and calo hits (the full 2D list, individual views won't work), along with the previously specified folding parameters and the MC hierarchy variable to hold the result.

```
LArHierarchyHelper::MCHierarchy mcHierarchy;
LArHierarchyHelper::FillMCHierarchy(*pMCParticleList, *pCaloHitList, foldParameters,
    mcHierarchy);
```

Listing 7: "Constructing a truth hierarchy."

The `MCHierarchy` class provides the ability to query the type of hierarchy you are dealing with (e.g. neutrino or test beam) and access the neutrino, final state nodes (recall nodes and particles are not necessarily the same thing) or a flattened set of all nodes. Once you have access to a `Node` in the hierarachy you can establish whether or not that node is reconstructable, determine which `MCParticle` is the leading particle of the node, access the child nodes, the hits, the PDG code, its tier, whether it's a leading lepton, etc. You can look at the doxygen comments for the class for a complete list of functions.

Listing 8 shows how to construct a reco hierarchy. The structure closely mirrors that of the truth hierarchy, based on a list of PFOs and calo hits, again with the previously specified folding parameters and the reco hierarchy variable to hold the result.

```
LArHierarchyHelper::RecoHierarchy recoHierarchy;
LArHierarchyHelper::FillRecoHierarchy(*pPfoList, foldParameters, recoHierarchy);
```

Listing 8: "Constructing a reco hierarchy."

The `RecoHierarchy` class provides a similar interface to the `MCHierarchy` class, though of course, there are fewer functions because we lack truth information to query. Nonetheless, you can access the root neutrinos (for example), final state nodes and a flattened set of all nodes. Once you have access to a `Node` in the hierarachy you can access the child nodes, the hits, the particle ID (track or shower), etc. Again, you can look at the doxygen comments for the class for a complete list of functions.

If you want to perform matching between these hierarchies, Listing 9 shows how to do this. Simply pass the truth and reco hierarchies to the `MatchHierarchies` function, along with the output `MatchInfo` variable. This object provides access to `GetMatches` and `GetUnmatchedReco`. In the latter case, this is simply a vector of reconstructed nodes that haven't been matched to truth, while the former is a vector of `MCMatches`, which provides links between truth and reconstruction for each true node. The relevant truth node can be access via `GetMC` and the corresponding vector of matched reconstructed nodes (hopefully one, but this can be empty or have multiple matches) via `GetRecoMatches`. It's also possible to retrieve the number of shared hits and get purity and completeness metrics, as well as check whether the match passes quality cuts. Again, refer to the doxygen comments for details.

```
LArHierarchyHelper::MatchInfo matchInfo;
LArHierarchyHelper::MatchHierarchies(mcHierarchy, recoHierarchy, matchInfo);
```

Listing 9: "Matching hierarchies."