Table of Contents

Decay-tree-fitter

Introduction

The traditional method to fit a decay tree is 'leaf-by-leaf': one starts by fitting the vertices most downstream in the decay and builds up the tree by propagating information upstream. The advantage of this approach is that it is fast. It also models most closely how we reconstruct decay chains. The disadvantage is that it is relatively hard to propagate information from a 'mother' vertex to its daughters downstream, in particular if the decay tree spans more than one generation. For some vertex fits, like the decay of Ks->pi0pi0, traditional fits simply don't work: There are no tracks to form a Ks vertex from downstream particles. The 'origin' of the mother particle and the known mass constraints of the resonant daughters are essential ingredients in reconstructing the downstream vertex.

The 'decay tree fitter' was developed in BaBar to deal with Ks->pi0pi0 and similar decays, where essential information of the vertex is obtained by including the origin of the decaying particle and by adding mass constraints that include information from photons.The algorithm takes a complete decay chain, parameterizes it in terms of vertex positions, decay lengths and momentum parameters, and then fits these parameters simultaneously, taking into account the relevant constraints, such as the measured parameters of the final state tracks and photons, 4-momentum conservation at each vertex etc. To perform the fit efficiently a Kalman filter is used. The procedure is described in this NIM paper.

The LHCb package Phys/DecayTreeFitter

The BaBar algorithm has been ported to the LHCb analysis framework and can be found in Phys/DecayTreeFitter. The interface is not yet 'Gaudi-like' in the sense that there are no tools or algorithms yet. (For now it is a 'linker' library and not yet a 'component' library.) However, it is already reasonably complete and should be easy to use.

The fitter can be used in any Gaudi algorithm by adding

```
   use   DecayTreeFitter   v*   Phys
```

to the requirement file an including the header file of the interface:

```
    #include "DecayTreeFitter/Fitter.h"
```

The fitter can also be used in GaudiPython, for example:

```
dtf=gbl.DecayTreeFitter.Fitter(cand,bpv)
dtf.setMassConstraint(gbl.LHCb.ParticleID(443), True)
dtf.fit()
chi2dtf=dtf.chiSquare()
ndofdtf=dtf.nDof()
mdtf=dtf.fitParams().momentum().m().value()
ctaudtf=dtf.fitParams().ctau().value()
```

It may happen that a candidate has been reconstructed with the wrong PID, and a mass-constrained fit will give wrong results. A well known example is the Ds that is given the PID of a D+ in StdLooseDplus. This can be fixed by changing the PID before fitting.

```
    #correct pid if it is a Ds that has been called D+
    for d in cand.daughters():
      if abs(d.particleID().pid())==411:
        nK=0
        for dd in d.daughters():
```

```
    if abs(dd.particleID().pid())==321:nK=nK+1
  if nK==2:
    if d.particleID().pid()==411:d.setParticleID(gbl.LHCb.ParticleID(431))
    if d.particleID().pid()==-411:d.setParticleID(gbl.LHCb.ParticleID(-431))
```

Extracting the lifetime of a long-lived daughter

As an example consider the decay B+->D0 pi, with D0 -> K pi. Suppose that you are interested in extracting the decay length of the D0 meson. Given a pointer to the B+ particle, you can obtain the D0 decay length as follows:

```
// create the fiter object
DecayTreeFitter::Fitter myfitter( myB ) ;

// eventually add mass constraints
// myfitter.setMassConstraint( myD0 ) ;
// myfitter.setMassConstraint( myB ) ;

// fit
myfitter.fit() ;

// D0 must be the D0 daughter the B points to
LHCb::VtxFitParams myD0Params = myfitter.fitParams( myD0 ) ;

// get the decay length with err
LHCb::VtxDoubleErr len = myD0Params.decayLengthErr() ;
```

The `fitparams(const LHCb::Particle&)` method returns an `LHCb::VtxFitParams` object that contains all the (possible) fitted parameters of a particle in the decay tree, namely

- **position**. For composite particles this is the decay vertex. For tracks or photons it is the production vertex;
- **4-momentum**. For composite particles with a mass constraint, and for tracks and photons, there are only 3 independent components.
- **decaylength**. This is the distance between the production and decay vertex. It is only non-zero for composites that are not treated as a resonance.

The `LHCb::VtxFitParams` also contains the full 8x8 covariance matrix. (The only thing that it does not store is correlations between different particles. For that, we need to extend the `Fitter` interface.) It also has a method to compute the proper decay time, with or without assuming a known mass for the decaying particle. (See below.)

Fitting with an origin constraint to extract the B proper time

There also exists a Fitter constructor that takes an additional 'origin' vertex for your candidate. This allows to extract the proper decay time:

```
 // create the fiter object with an associated primary vertex
 DecayTreeFitter::Fitter myfitter( myB, primaryVertex ) ;
 // fit
 myfitter.fit() ;
 // extract the parameters of the B
 LHCb::VtxFitParams myBParams = myfitter.fitParams( myB) ;
 // get the decay time in units of distance
 LHCb::VtxDoubleErr ctau = myBParams.ctau() ;
// get the decay time in units of time
 LHCb::VtxDoubleErr tau = myBParams.properDecayTime() ;
```

The error in the proper time is evaluated by propagating the full uncertainty in the B 4-momentum. Sometimes it is useful to fix the mass of the mother particle to the known mass. It is not necessary to perform a mass constrained fit for that, because the VtxFitParams object can do the math internally:

```
 // get the pdg mass of the B
 const double pdgmass = ... ;
 // get the decay time in units of distance
 LHCb::VtxDoubleErr ctau = myBParams.ctau(pdgmass) ;
// get the decay time in units of time
 LHCb::VtxDoubleErr tau = myBParams.properDecayTime(pdgmass) ;
```

In fact, the VtxFitparams has a member 'setMass' which performs the mass constrained fit in the most optimal way, AKAIK:

```
 // get the pdg mass of the B
 const double pdgmass = ... ;
 // fix the mass to a known mass (identical to performing a mass constrained fit)
 fitParams.setMass( pdgmass ) ;
```

Extracting the decay angles in Bs --> J/psi phi

The extraction of the decay angles in Bs -> J/psi phi depends on knowledge of the 4-momenta of the four final state particles. The unceraint in observables like the invariant mass and the decay angle are usually dominated by the most poorly known momentum. That is why mass constraints help to reduce the uncertainty in such observables. A simple example of the B invariant mass on Bs --> J/psi phi: with a J/psi mass constraint the B invariant mass resolution is considerably (50%) better than without. Something similar holds for the decay angles.

With the decay tree fitter you can easily extract the 4-momenta of the final state particles:

```
 #include "DecayTreeFitter/Fitter.h"
 #include "DecayTreeFitter/PidPdg.h"

  // fit the B with mass constraints on the B and the J/psi
 DecayTreeFitter::Fitter myfitter( myB ) ;
 myfitter.setMassConstraint( myB ) ; // sets mass constrained using a pointer to particle
 myfitter.setMassConstraint( LHCb::ParticleID( LHCb::PidPdg::J_psi ) )  ; // alernatively: use
 myfitter.fit() ;

 // extract 4 momenta of all final state particles
 Gaudi::LorentzVector p4 = myfitter.fitParams( myKaon ).p4() ; // myKaon is a pointer to the m
 [...]
```

Although this yields all information necessary, it is maybe more convenient to use the `IP2VVPartAngleCalculator`, which takes a pointer the head of the decay tree as argument. However, the following solution

```
 LHCb::Particle fittedB = myfitter.getFitted( myB ) ;
 // call the decay angle tool. WRONG: B daughters not updated.
double thetaTr(0), phiTr(0), thetaV(0) ;
m_anglecalculator->decayAngles( myFittedB, thetaTr, phiTr, thetaV ) ;
```

will **NOT** work.

The problem is that the fitter cannot just return a new particle for the B with daughter particles that are 'fitted' (or 'updated', in my terminology).This has to do with a technical problem: LHCb::Particles do not own their decay tree. The daughters are in containers in the event store. The daughter particles can be shared with the other B candidates, so one shouldn't change them. The solution to this problem is to copy the daughters before updating them. However, that still leaves us with the problem of assigning an owner to the newly created particles.

I will try to convince the DaVinci experts that the most convenient solution is to change the Particle model. Until that time, we will use a solution proposed by Vanya: The fitter can create a `DecayTreeFitter::Tree` object which takes care of cloning a decay tree and destructing it. You can now call the decay angle tool as follows:

```
#include "DecayTreeFitter/Tree.h"

// create a tree
DecayTreeFitter::Tree decaytree = fitter.getFittedTree() ;
// get access to the head of the decay tree. all particles in this decay tree have updated mom
const LHCb::Particle* myFittedB = decaytree->head() ;
// call the decay angle tool
double thetaTr(0), phiTr(0), thetaV(0) ;
m_anglecalculator->decayAngles( myFittedB, thetaTr, phiTr, thetaV ) ;
```

By calling `Tree::release()` you can assume ownership of the tree and store it in the desktop, for persistence in your micro dst, etc. If you do not call release, the particles will be properly destructed by the `Tree` destructor.

```
DecayTreeFitter::Tree decaytree = fitter.getFittedTree() ;
if ( this is such a cool candidate, I want to keep it! ) {
   // take owner ship. Tree will no longer delete the particles
   LHCb::Particle* head = decaytree.release() ;
   // now add 'head' to you favourite container in the Event store
} else {
  // no need to do anything. Tree::~Tree takes care of deletion of all particles in the decay
}
```

Fitting with missing particles

DecayTreeFitter can also deal with final state particles that do not have any external information, for example a neutrino. Of course, if all you have is a single decay vertex, then the parameters of that particle would not be constrained. So, how meaningful the fit is depends on the topology.

As an example, take B->KD0 with D0 decaying to Kmunu. The D0 flies and has a well reconstructed vertex. The B vertex has only one track. The constraint of the D0 direction of flight to that track fixes one parameter of the neutrino direction. Hence, you may hope to constrain one of three neutrino momentum parameters. With a PV constraint for the B, you can constrain yet another two. So, by exploiting the D0 and B lifetime, you can constrain the neutrino momentum even without any mass constraints.

Now, there is one big caveat to 'fitting' with missing particles: unless the decay is fully constrained, there is more than one solution. The fit will find the solution closest to its starting point. So, if you use DTF for estimating the parameters of a missing particle, makes sure those parameters are initialized as well as possible to the parameters that you would estimate with a back of the envelop calculation.

To use DTF with missing particles, you need to first create the decay. Currently, you cannot specify missing particles in decay descriptors. That makes this a bit complicated: you need to create the decay chain yourself. The following (untested, uncompiled) code snippet serves as an example:

```
// get these from somewhere else
const LHCb::Particle* myKFromD0 = ... ;
const LHCb::Particle* myMuFromD0 = ... ;
const LHCb::Particle* myKFromB0 = ... ;

// create a particle that represents neutrino. make sure to give it the right PID
LHCb::Particle* myNeutrino = new LHCb::Particle() ;
myNeutrino.setParticleID ( muonNeutrinoId ) ;
```

```
   // initialize its momentum. one solution, reasonably suitable for this case:
   // choose its momentum vector parallel to the KMu momentum. now choose
   // momentum such that  you get mass of B. Of all solutions consistent with the B mass, this is
   const auto muKp4 = myKFromD0->momentum() + myMuFromD0 ->momentum()  ;
   const auto muKKp4 = muKp4 +  myKFromB0->momemtum() ;
   const auto dir =   muKp4.Vect().Unit() ;
   const double dM2 = pdgBMass*pdgBMass - muKKp4.M2() ;
   const double E = 0.5 * dM2 / (muKKp4.E() - muKKp4.Vect().Dot(dir)) ;
   myNeutrino.setMomentum( Gaudi::LorentzVector( E* dir.x(), E*dir.y(), E*dir.z(), E) ) ;

   // create the D0 particle. you could use a particle combiner here, but I'll do it by hand.
   LHCb::Particle* myD0 = new LHCb::Particle() ;
   myD0->setParticleID ( ...)
   myD0->setMomentum( myNeutrino.momentum() + muKp4 ) ;
   myD0->addToDaughters( myKFromD0 ) ;
   myD0->addToDaughters( myMuFromD0) ;
   myD0->addToDaughters( myNeutrino ) ;

   // you probably want to create a vertex, too.
   LHCb::Vertex* vd0 = new LHCb::Vertex() ;
   myD0->setEndVertex( vd0) ;
  // it would be best to initialize this with the poca of the kaon and muon, but I'll leave that

   // now create the B+ in the same way
   LHCb::Particle* myB = new LHCb::Particle() ;
   myB->setParticleID ( ...)
   myB->setMomentum( myD0.momentum() + muKp4 ) ;
   myB->addToDaughters( myMuFromB) ;
   myB->addToDaughters( myD0 ) ;

   // now the fitting part
   DecayTreeFit::Fitter fitter( *myB, myPV, myTrackStateProvider ) ;

   // make sure to give the neutrino a mass constraint
   fitter.setMassConstraint( myNeutrino ) ;

   // eventually do the same for the B or the D

   // now fit, and extract parameters as in examples above
   fitter.fit()
```

-- WouterHulsbergen - 13 May 2009

This topic: LHCb > DecayTreeFitter
Topic revision: r11 - 2015-12-28 - BarbaraStoraci