LHCb Starterkit 2024: Practice session*

29 November 2024

This is the version with solutions. Solutions are marked in this font.

Goals of the session

It has been a long week of lectures, now it's time for you to apply the obtained knowledge on practice! During this session, you will need some knowledge from each of the lessons you have attended this week.

Today, we put you in a real-life situation: you wish to start a brand new analysis of a given decay mode. What are your *first analysis steps*? How do you get your tuples with all the variables of interest?

This document will guide you through the list of short problems and questions. For each step, take your time to think and try to solve it on your own. Questions in *italic* are optional and can be skipped in the lack of time. Don't hesitate to ask helpers if you have any doubts! For certain problems, the instructor will guide you towards the proposed solution. Caution: your solution might be easier and nicer than the proposed one, don't hesitate to comment if this is the case!

Now, let's take off!

1 Creating a gitlab repository with your scripts

First step: let's create a repository where we will store our scripts! As a rule in LHCb, everyone has to preserve the code used to produce the tuples for each analysis.

- Create a new repository on your personal gitlab
- When we will prepare our scripts, regularly commit them
- When the scripts are ready, add a little `README` explaining how to run them, and which software versions to use.

2 Introduction

Today we will study the decay $\Lambda_b^0 \to pK^- J/\psi$.

Before we start, let's discuss several physics questions which are crucial for our further steps:

- Which type of the transition is this decay? (Strong, weak, electromagnetic)?

  Weak.

- Which of the final state particles are stable?

  Proton, kaon.

- If there are unstable particles, which decay mode(s) could be used to reconstruct them?

  J/ψ ***is unstable, can be reconstructed through its decays to two muons, two electrons, hadronic decays etc.***

*Found a mistake in this document? Please contact lhcb-starterkit-organisers@cern.ch

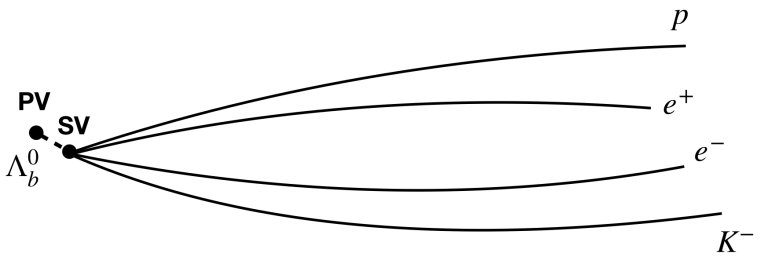

Figure 1: ***Sketch of the decay*** $\Lambda_b^0 \to pK^- J/\psi(\to e^+e^-)$***.*** J/ψ ***decays instantly in the secondary vertex (SV).***

- Are there unstable particles flying measurable distance before the decay?

  The mother particle Λ_b^0 ***flies before decaying.***

- *Why is it interesting to study such a decay mode?*

  This is up to your taste, but one reason is that pentaquarks were discovered in this channel.

For pedagogical reasons (we do not search for easy ways, eh?) we will consider J/ψ decaying to e^+e^-. From now on, we will denote our decay mode of interest as $\Lambda_b^0 \to pK^- J/\psi(\to e^+e^-)$.

- Based on your answers to previous questions, draw the schematic topology of this decay.

  Fig. 1 gives an idea.

2.1 Dataset and Stripping

In the Run II of LHCb, proton-proton collision data can be accessed in two ways: via the `Stripping` or `Turbo` frameworks, depending on the way it was collected. In this exercise, we will use the data from the `Stripping`.

To make things simple, we will consider only 2018 data sample in this exercise.

- Find out which is the most recent Stripping version for 2018 proton-proton data. Hint: check the list of Stripping versions here `http://lhcbdoc.web.cern.ch/lhcbdoc/stripping/`[1].

  The most recent stripping versions for 2018 proton-proton data taking are Stripping 34r0p3 (incremental) and Stripping34 (full).

- There are full and incremental restrippings. What is the difference between the two?

  Full restrippings include the entire set of stripping lines. Incremental restrippings include only those lines which needed to be modified compared to the latest full re-stripping.

- We are happy to use stripping34r0p1. Click on the name of the stripping version, which opens the list of stripping lines.

  See *http://lhcbdoc.web.cern.ch/lhcbdoc/stripping/config/stripping34r0p1/index.html*.

Now, we have to define which stripping line to use for our analysis. First, the full dataset is split in several streams. The stream relevant for decay modes with leptons in the final state is called `Leptonic`. Now, you can see there is a huge number of stripping lines in this stream.

How to figure out which line should one use? Normally, if you do not know the answer by yourself (and don't want to spend a long evening checking contents of every single line), the best solution is to ask your

[1] The links to all LHCb software documentation can be found here: `http://lhcb.web.cern.ch/lhcb/computing/index.html`

supervisor, or the `Stripping liaison` of your Working Group (the person responsible for maintaining the stripping lines).

Let's assume you have asked them, and received the following reply: for your decay channel, you should use the stripping line called `Bu2LLK_eeLine2`.

- Find this line in the list, and explore its contents. We want to check whether our decay mode is indeed present in this stripping line. To do that, open the properties of the `CombineParticles` algorithm.

  ***See** `http://lhcbdoc.web.cern.ch/lhcbdoc/stripping/config/stripping28r1/leptonic/strippingbu2llk_eeline2.html`.*

- Check the `DecayDescriptors` property. It lists the decay modes which are selected by this stripping line. Does it contain our mode of interest? Hint: intermediate particles may be present in our decay chain.

  Indeed, by the first look, there is no** $\Lambda_b^0 \to pK^- J/\psi(\to e^+e^-)$ **channel listed in the list of decay descriptors. However, the hint tells us to check whether this channel could proceed through any intermediate particles. There are two decay modes of the** Λ_b^0 **baryon listed:

  ```
  '[ Lambda_b0 -> J/psi(1S) Lambda0 ]cc' , '[ Lambda_b0 -> J/psi(1S) Lambda(1520)0 ]cc' ]
  ```

  The next question asks us to investigate which decay modes are considered for the** `Lambda0` **and** `Lambda(1520)0`**, and find which of those includes our channel of interest.

- Check the `GaudiSequencer/SeqMergeBu2LLK_ee2` contents to make sure if our decay mode of interest is indeed present in this stripping line.

  Here are the items which we have to explore:

  ```
  GaudiSequencer/SEQ:LambdasLLForBu2LLK
  GaudiSequencer/SEQ:LambdasDDForBu2LLK
  GaudiSequencer/SEQ:LambdastarsForBu2LLK
  ```

  Exploring their contents, we can figure out that the latter takes as inputs

  ```
  Phys/StdLooseLambdastar2pK/Particles
  ```

  This container covers the decay** `[Lambda(1520)0 -> p+ K-]cc` **which leads to our final state. As a conclusion, indeed, this stripping line contains our channel of interest, which is defined as** $\Lambda_b^0 \to \Lambda(1520) J/\psi(\to e^+e^-)$**.

2.2 Simulation

Besides the sample of real LHCb data, we will also need a signal simulation sample for our analysis. This will allow to understand the properties of our signal decay before actually searching for it in the data.

- Find the event type for our decay $\Lambda_b^0 \to pK^- J/\psi(\to e^+e^-)$. Hint: the complete list of event types is present here: `http://lhcbdoc.web.cern.ch/lhcbdoc/decfiles/`

  The event type is 15154001.

3 Bookkeeping

Now, you have enough information to search the Bookkeeping for the relevant data and simulation samples.

- Find the bookkeeping paths for the data and simulation samples of interest. Reminder: you can find the bookkeeping online at `https://lhcb-portal-dirac.cern.ch/DIRAC/`.

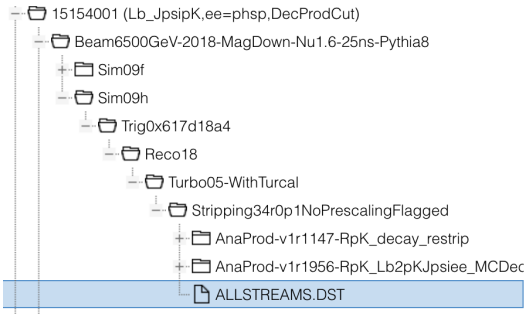

Figure 2: ***The simulation sample which we will use.***

- We will use sim09h version of the simulation. For instance, the another sample available, the `Sim09f` one, was simulated with wrong ECAL thresholds. Differences between versions can be checked here `https://lhcb-simulation.web.cern.ch/WPP/Sim09Differences.html`.
- Download local DSTs (one for the data, one for the simulation) which can be used to test our DaVinci scripts locally. Note: downloading DST files can take a significant amount of time, so it is wiser to do that in a screen session.

  Get your grid proxy (`lhcb-proxy-init`) and use the following command:

  ```
  lb-dirac dirac-dms-get-file blabla.DST .
  ```

4 Local DaVinci version

First, we should define which DaVinci version to use (see the complete list here `http://lhcbdoc.web.cern.ch/lhcbdoc/davinci/releases/`). The rule of thumb is that one should use the latest DaVinci[2], unless there is a good reason not to do so. Note that versions > 50 are only to be used to produce Run 3 tuples. So, we will use `DaVinci v46r12`.

Optional: Since in one of the following steps we plan to add new TupleTools to our DaVinci, we have to create a local version which we can modify afterwards.

- Create a local DaVinci and build it without any modifications. Hint: follow the instructions in the "Developing the LHCb software" lesson.

  Use the following commands:

  ```
  lb-dev DaVinci/v46r12
  cd ./DaVinciDev_v46r12
  git lb-use DaVinci
  make
  ```

- Make sure you can successfully run it.

  ```
  ./run gaudirun.py
  ```

5 Finding TES locations

- Using the script from the lesson "Interactively exploring the DST" (`https://lhcb.github.io/starterkit-lessons/first-analysis-steps/interactive-dst.html`), figure out the TES location in our simulation sample (the one you have downloaded locally).

  Assuming the script which explores the DST is called `explore.py`, one has to run the following command:

[2]Excluding the versions for upgrade studies (v50rX), which should not be used for Run1-Run2 analyses.

```
lb-run DaVinci/v46r12 ipython -i explore.py file.dst
```

This will printout the following list of TES locations:

```
/Event
/Event/Gen
/Event/PrevPrev
/Event/Prev
/Event/Next
/Event/NextNext
/Event/MC
/Event/Link
/Event/pSim
/Event/Rec
/Event/Rec/Header
/Event/Rec/Status
/Event/Rec/Summary
/Event/Calo
/Event/Unstripped
/Event/HC
/Event/Tracker
/Event/PersistReco
/Event/Velo
/Event/Muon
/Event/Rich
/Event/Trigger
/Event/Trigger/RawEvent
/Event/pRec
/Event/Turbo
/Event/Strip
/Event/AllStreams
/Event/DAQ
/Event/DAQ/ODIN
```

The TES location we are interested in is `/Event/AllStreams` which corresponds to Flagged DSTs. In case a filter was applied, AllStreams has to be substituted by the name of the filter. Therefore, the complete TES location to be used in the inputs of the DecayTreeTuple for your simulation, has the following structure:

```
/Event/AllStreams/Phys/Bu2LLK_eeLine2/Particles
```

6 Minimal DaVinci job

Now, we can start preparing our DaVinci scripts! Let's start from a minimal setup (you can re-use the one from the relevant Starterkit lesson), and as soon as it works, we will add more complications in.

Ideally, we should use the same exact script for the real data and simulation, with some switch inside for selecting between the two, passed as an option.

- Write the decay descriptor for our decay of interest. Hint: it should match one of the decay channels selected by our stripping line.

  We have to match the descriptor in the stripping line:

  ```
  theTuple.Decay = '[ Lambda_b0 -> ^(J/psi(1S) -> ^e+ ^e-) ^(Lambda(1520)0 -> ^p+ ^K-)]CC'
  ```

- Recall which TES locations to be used for the real data and simulation. Think how to take into account properly the fact that the data has the MDST format, while simulation sample relies on the DST.

Simulation: as discussed above. It is convenient to have a switch (let's call it `isSim`) to define whether we deal with simulation or data candidates. We can define our inputs this way:

```
line = 'Bu2LLK_eeLine2'
if isSim:
    theTuple.Inputs = ['/Event/AllStreams/Phys/{0}/Particles'.format(line)]
```

Data: use the `RootInTES` property for the MDST format:

```
else:
    DaVinci().RootInTES = "/Event/Leptonic"
    theTuple.Inputs = ['Phys/'+line+'/Particles']
```

- Now, write a minimal DaVinci job.

We will write a function taking `isSim` as an argument, so that it is neat to switch between the data and simulation. All the items shown below are the standard ones, the only complication is the extraction of the database tags. For data, the default tags should be used, therefore we can skip defining them. However, for the simulation, the tags used for its production have to be used. Following the procedure described in the lesson "Running a minimal DaVinci job locally", we can extract the tags from the bookkeeping: `DDDB: dddb-20170721-3 Condition DB: sim-20190430-vc-mu100`. Here is the minimal DaVinci job:

```
from Configurables import DaVinci, DecayTreeTuple
from DaVinci.Configuration import *
from DecayTreeTuple.Configuration import *
from GaudiConf import IOHelper

def addNtupleToDaVinci(isSim):
    theTuple = DecayTreeTuple("LbTuple")
    theTuple.Decay = '[ Lambda_b0 -> ^(J/psi(1S) -> ^e+ ^e-) ^(Lambda(1520)0 -> ^p+ ^K-)]CC'

    theTuple.addBranches ({
        "Lb" : "[ Lambda_b0 -> (J/psi(1S) -> e+ e-) (Lambda(1520)0 -> p+ K-)]CC",
        "Jpsi":"[ Lambda_b0 -> ^(J/psi(1S) -> e+ e-) (Lambda(1520)0 -> p+ K-)]CC",
        "E1":"[ Lambda_b0 -> (J/psi(1S) -> ^e+ e-) (Lambda(1520)0 -> p+ K-)]CC",
        "E2":"[ Lambda_b0 -> (J/psi(1S) -> e+ ^e-) (Lambda(1520)0 -> p+ K-)]CC",
        "Proton":"[ Lambda_b0 -> (J/psi(1S) -> e+ e-) (Lambda(1520)0 -> ^p+ K-)]CC",
        "Kaon":"[ Lambda_b0 -> (J/psi(1S) -> e+ e-) (Lambda(1520)0 -> p+ ^K-)]CC",
        })

    line = 'Bu2LLK_eeLine2'
    if isSim:
 theTuple.Inputs = ['/Event/AllStreams/Phys/{0}/Particles'.format(line)]
    else:
 DaVinci().RootInTES = "/Event/Leptonic"
        theTuple.Inputs = ['Phys/'+line+'/Particles']

    DaVinci().UserAlgorithms += [theTuple]

####
DaVinci().Simulation = True    # or false for data
DaVinci().InputType = "MDST" # for data
DaVinci().Lumi = True #for data
if DaVinci().Simulation:
    DaVinci().InputType = "DST" # for MC
    DaVinci().Lumi = False # for MC
```

```
Year = '2018'
MagnetPolarity = 'MD'

DaVinci().DataType = Year
DaVinci().EvtMax = 1000 #    -1 to run over the full sample
DaVinci().TupleFile = "Lambdab.root"

if DaVinci().Simulation:
    DaVinci().DDDBtag = "dddb-20170721-3"
    CondDB_tag  = "sim-20190430-vc"
    if MagnetPolarity == 'MU':
        DaVinci().CondDBtag = CondDB_tag + "-mu100"
    elif MagnetPolarity == 'MD':
        DaVinci().CondDBtag = CondDB_tag + "-md100"

addNtupleToDaVinci(DaVinci().Simulation)

if DaVinci().Simulation:
    IOHelper().inputFiles(["00068049_00000016_1.bu2kll_nopid.strip.dst"],clear=True)
else:
    IOHelper().inputFiles(["00069527_00000003_1.leptonic.mdst"],clear=True)
```

- Test locally whether your DaVinci script runs and the proper output file is produced.
- *Add and instance of* `EventPrefilters` *filtering on the stripping line decision.*

  To do that, we will use the LOKI functor *HLT_PASS_RE* ***which checks for a given stripping (or trigger) decision. Add the following lines to you script:***

```
from PhysConf.Filters import LoKi_Filters
fltrs = LoKi_Filters (
      STRIP_Code = "HLT_PASS_RE('StrippingBu2LLK_eeLine2Decision')"
      )
DaVinci().EventPreFilters = fltrs.filters('Filters')
```

In the following steps, we will work mostly with the simulation sample, as it is sufficient to run quickly on 1000 events there in order to get enough statistics to see the mass peak.

7 Adding more variables

Now, let's add more useful variables to our tuple!

First, let's add few more TupleTools:

- `TupleToolTrackInfo` which fills the variables related to the track quality;

  ```
  theTuple.addTupleTool('TupleToolTrackInfo')
  ```

- `TupleToolBremInfo` which fills the information on the bremasstrahlung photons emitted by our electrons;

  ```
  theTuple.addTupleTool('TupleToolBremInfo')
  ```

- `TupleToolMCTruth` which provides truth-level variables (*i.e.* the generated values before propagation through the detector). This TupleTool should be added only when running on the simulation sample, as it makes no sense for the real data.

  ```
  if isSim:
      theTuple.addTupleTool('TupleToolMCTruth')
  ```

- `TupleToolTISTOS` which fills the trigger decision variables. The syntax of adding specific trigger lines is `theTuple.TupleToolTISTOS.TriggerList = MyTriggerList`, where `MyTriggerList` is a python list containing set of lines. Let's add the following ones there:
  - `L0ElectronDecision, L0HadronDecision`
  - `Hlt1TrackMVADecision`
  - `Hlt2Topo2BodyDecision,Hlt2Topo3BodyDecision,Hlt2Topo4BodyDecision`

```
TriggerList = [
    'L0ElectronDecision','L0HadronDecision',
    'Hlt1TrackMVADecision',
    'Hlt2Topo2BodyDecision','Hlt2Topo3BodyDecision','Hlt2Topo4BodyDecision'
    ]
theTuple.addTupleTool('TupleToolTISTOS')
theTuple.TupleToolTISTOS.TriggerList = TriggerList
theTuple.TupleToolTISTOS.Verbose = True
```

- `TupleToolSubMass` which fills invariant masses for all possible (2,...,N-1)-body combinations of particles in an N-body decay; and allows to replace the mass hypotheses. It should be added to the head of the decay. It has the property `Substitution` which takes a list of mass hypotheses to be tested, with the following syntax: `["pi+ => K+"]`; as well as an the property `DoubleSubstitution` which takes the following syntax: `["K-/pi+ => pi+/K-"]`. Let's add the following substitutions there:
  - Replace the proton mass hypothesis by kaon and pion
  - Swap the proton and kaon mass hypotheses

```
theTuple.Lb.addTupleTool('TupleToolSubMass')
theTuple.Lb.TupleToolSubMass.Substitution += ["p+ => K+"]
theTuple.Lb.TupleToolSubMass.DoubleSubstitution += ["K-/p+ => p+/K-"]
```

As you might have noticed, each of the TupleTools adds a predefined list of variables, and some of them might be not really useful for your analysis. The flexible way to add specific single variables is to use LOKI functors.

Let's add some!

- First, a simple one. Add the variables for the pseudorapidity (functor `ETA`) and polar angle (`PHI`).
- Now, add the variable providing the vertex quality (χ^2) of the Λ_b^0. Compare the distribution of your variable to the existing variable `Lb_ENDVERTEX_CHI2`.

```
LokiVariables = theTuple.addTupleTool('LoKi::Hybrid::TupleTool/LokiVariables')
LokiVariables.Variables = {
  "ETA" : "ETA",
  "PHI" : "PHI",
  "VCHI2NDOF" : "VFASPF(VCHI2/VDOF)"
    }
```

Try to understand which variables in your tuple were added by which TupleTool or LOKI functor. Go through the DaVinci printout and try to understand

8 DTF and mass constraints

If you check the distribution of the invariant mass of the Λ_b^0, you will notice that the resolution of the mass peak is very large (compare it to the resolution on the sample you used yesterday!).

- Explain this. Hint: check the invariant mass resolution on the J/ψ peak.

Electrons are very light particles and they emit bremsstrahlung radiation in interactions with the detector material. These lost bremsstrahlung photons degrade the resolution of the invariant mass of the dielectron (J/ψ) and therefore also of the Λ_b^0 candidate.

Now let's see if we can improve the resolution. We will use the constraints coming from having J/ψ and Λ_b^0 with precisely known masses.

- Create an instance of the DecayTreeFitter algorithm.
- Add mass constraints on the J/ψ and Λ_b^0 to the head of the decay. Specify the option to add DTF variables also to the daughter particles.

```
theTuple.Lb.addTupleTool( 'TupleToolDecayTreeFitter', name = "DTF" )
theTuple.Lb.addTupleTool( theTuple.Lb.DTF.clone( "DTF_PV",
                                    constrainToOriginVertex = True ) )
theTuple.Lb.ToolList += [ "TupleToolDecayTreeFitter/DTF_PV" ]
theTuple.Lb.addTupleTool( theTuple.Lb.DTF.clone( "DTF_JPsi",
                                    constrainToOriginVertex = False,
                                    daughtersToConstrain = [ "J/psi(1S)" ] ) )
theTuple.Lb.ToolList += [ "TupleToolDecayTreeFitter/DTF_JPsi" ]
theTuple.Lb.addTupleTool( theTuple.Lb.DTF.clone( "DTF_Lb",
                                    constrainToOriginVertex = False,
                                    daughtersToConstrain = [ "Lambda_b0" ],
                                    Verbose=True) )
theTuple.Lb.ToolList += [ "TupleToolDecayTreeFitter/DTF_Lb" ]
```

The option "Verbose" propagates the DTF variables also to daughter particles.

- Run your script again on at least 2000 events.
- Check the Λ_b^0 mass shape with J/ψ mass constraint applied, and compare to the nominal Λ_b^0 invariant mass shape. Explain the difference.

  Requiring a mass constraint on the J/ψ eliminates the largest part of the effect of the bremsstrahlung radiation, if looking at the Λ_b^0 invariant mass. The resolution improves substantially, showing the power of the DTF algorithm and mass constraints.

- Now, check the J/ψ mass with the Λ_b^0 mass constraint. Does it work as well?
- Imagine we want to study the decay $\Lambda_b^0 \to J/\psi\,\Lambda^*(1520)$, where $\Lambda^*(1520)$ is a strongly decaying (to the pK^- final state) resonance having a mass about 1520 MeV and the width about 16 MeV. Can we apply a mass constraint on the $\Lambda^*(1520)$ mass? Justify.

  No, this makes no sense from the physics point of view. Applying a mass constrain basically forces the width of a given intermediate particle to zero. This is fine if the width is way smaller than the resolution (as for the J/ψ), but is unacceptable for particles having significant width of tens MeV, such as Λ^* resonances. Applying such a constraint will bias the kinematics of the decay and will not improve the resolution.

We can also add a DTF instance using the LOKI functor `DTF_FUN`.

- Add a mass constraint on the J/ψ using LOKI functors.

  You should have already some variable defined through LOKI functors in Question 7, so let's just add one line to that list:

  ```
  "LOKI_MASS_JpsiConstr" : "DTF_FUN ( M , True , 'J/psi(1S)' )" ,
  ```

- Compare the distributions of the resulting Λ_b^0 invariant mass variables with the J/ψ mass constraint, obtained through the TupleTool or LOKI. Take into account that the decay tree fit may fail for some events, then the invariant mass will get an unphysical negative value. When plotting the mass, apply a cut so that it is larger than zero.

9 Adding a new TupleTool

Some people write their customs TupleTools to fill quickly certain information in the tuples. Some working groups have examples of useful TupleTools which are used widely within the working group, but for whatever reasons are not added to the core DaVinci. Let's take one of such TupleTools and add it to our local DaVinci version.

We will use the TupleToolHelicity defined here: `https://gitlab.cern.ch/vlisovsk/ttps/tree/master`. It defines certain helicity variables for three- and four-body decays of a similar kind.

- Download the TupleTool (both `.cpp` and `.h` files) to your location.
- Now, go back to the folder with your local DaVinci version. To modify the list of TupleTools, you have to checkout the `DecayTreeTuple`. It is located in the `Analysis` project. Hint: it is strongly advised to checkout the version of the Analysis project used to prepare this DaVinci release, which you can deduce from the release notes of your DaVinci version.

  As one can see in the release notes of our DaVinci version (`http://lhcbdoc.web.cern.ch/lhcbdoc/davinci/releases/v46r12/`), the version `Analysis v22r12` was used to build this DaVinci version. Therefore, let's do the following:

  ```
  cd DaVinciDev_v46r12/
  git lb-use Analysis
  git lb-checkout Analysis/v22r12 Phys/DecayTreeTuple
  ```

- Now, navigate to the `Phys/DecayTreeTuple/src` folder. You should see a list of the TupleTools already present there. Copy your new TupleTool here.
- Now, go back to the head of your DaVinci folder, and build your DaVinci version again.

  ```
  cd ../../../
  make
  ```

  Note that some of "custom" TupleTools will not compile with the latest DaVinci versions, which rely on recent C++ compiler versions. One has to go carefully through the errors and correct the C++ code.

- In your DaVinci options file, add the new TupleTool to your list, and run your script again.

  ```
  theTuple.addTupleTool('TupleToolHelicity')
  ```

- Figure out which new variables should have been added to your tuple by the new TupleTool. Check whether these variables were indeed created, and whether they are filled properly.

  You should get variables called `Jpsi_cosThetaH`, `Lb_phiH` etc.

10 *Retrieving RelatedInfo*

Unlike our simulation sample, which has the DST format and contains full event information, the sample of our real data is stored in a format of the MDST. This means only the information about the particles of our decay is stored, but not about other tracks in the event. However, sometimes it is useful to know what happens with other tracks. For example, many analyses rely on so-called *isolation* variables, which quantify the presence of extra tracks (or neutral objects) in the cone or cylinder around a given track.

- Why this could be useful? ***Isolation variables help to reduce the backgrounds which have the same final state as your signal, but with additional charged or neutral particles.***

When the dataset is stored in the MDST format, the isolation information is by default lost. However, some stripping lines are configured in the way so that the necessary information is preserved. It is possible to flag certain full-event information to be stored in your MDST while designing a stripping line. In this case this information is stored in so-called *RelatedInfo*.

- Go back to the webpage describing our stripping line. Find the `RelatedInfo` sections.

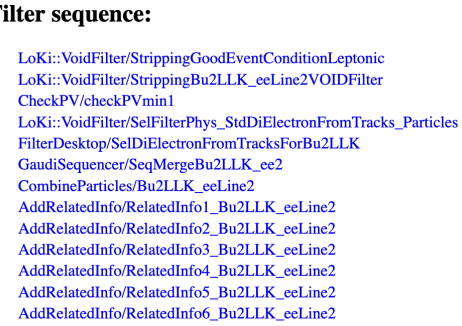

Figure 3: ***There are six different RelInfo tools defined for this stripping line.***

You can see there are several kinds of information stored there: for example, vertex isolation, track isolation *etc.*. For simplicity, we will work with the vertex isolation variables only.

The `RelatedInfo` information can be accessed via the LOKI functor `RELINFO`, which has the following syntax (in this case to extract a variable `VTXISONUMVTX` from the `VertexIsoInfo`):

`RELINFO('/Event/Leptonic/Phys/Bu2LLK_eeLine2/VertexIsoInfo', 'VTXISONUMVTX', -999.)`.

- Add the following variables to you list (only for the real data): `VTXISONUMVTX`, `VTXISODCHI2ONETRACK` and `VTXISODCHI2TWOTRACK`.
- For the simulation, the same information can be accessed via the `TupleToolVtxIsoln`. Add it to your simulation sample.

```
if isSim:
    theTuple.addTupleTool("TupleToolVtxIsoln")
else:
#isolation variables from RelInfo of microDST (data)
  lineQ = "'Bu2LLK_eeLine2'"
  LoKi_Cone =  LoKi__Hybrid__TupleTool("LoKi_Cone")
  LoKi_Cone.Variables = {
  #vertex variables
 "NumVtxWithinChi2WindowOneTrack":  "RELINFO('/Event/Leptonic/Phys/'"+lineQ+
 "'/VertexIsoInfo', 'VTXISONUMVTX', -999.)",
 "SmallestDeltaChi2OneTrack":  "RELINFO('/Event/Leptonic/Phys/'"+lineQ+
 "'/VertexIsoInfo', 'VTXISODCHI2ONETRACK', -999.)",
 "SmallestDeltaChi2MassOneTrack":  "RELINFO('/Event/Leptonic/Phys/'"+lineQ+
 "'/VertexIsoInfo', 'VTXISODCHI2MASSONETRACK', -999.)",
 "SmallestDeltaChi2TwoTracks":  "RELINFO('/Event/Leptonic/Phys/'"+lineQ+
 "'/VertexIsoInfo', 'VTXISODCHI2TWOTRACK', -999.)",
 "SmallestDeltaChi2MassTwoTracks":  "RELINFO('/Event/Leptonic/Phys/'"+lineQ+
 "'/VertexIsoInfo', 'VTXISODCHI2MASSTWOTRACK', -999.)",
       }

  theTuple.Lb.addTool(LoKi_Cone , name = "LoKi_Cone" )
  theTuple.Lb.ToolList   += [ "LoKi::Hybrid::TupleTool/LoKi_Cone"]
```

- Make sure all the variables are filled properly.

11 *MCDecayTreeTuple*

Sometimes it is useful to be able to access the information of your simulated sample at the generated stage, before propagation through the detector and adding any reconstruction effects. For example, this

allows to study the reconstruction efficiency. The method allowing to access the generator-level tuple is called `MCDecayTreeTuple`. It has rather similar syntax to the usual `DecaytreeTuple` algorithm, but instead of `TupleTools` the corresponding `MCTupleTools` should be used. Of course, the information provided by them is limited to the quantities defined at the generation level.

The main difference in the syntax between the DecayTreeTuple and MCDecayTreeTuple is the arrows in the decay descriptor: they should look like this `==>` for the MCDecayTreeTuple[3].

Another important difference is that MCDecayTreeTuple does not know anything about your stripping, – it only knows which decay channel was generated. It means, if for the DecayTreeTuple we had to take into account how our decay looks like in the stripping line, here we have to write it the same way as it was generated (so, $\Lambda_b^0 \to pK^- J/\psi\,(\to e^+e^-)$).

Finally, MCDecayTreeTuple does not require any inputs (like we did for `DecayTreeTuple.Inputs()`).

- Create a separate options file for your MCDecayTreeTuple.
- Fill it by analogy to the minimal DaVinci job, but taking into account what was discussed above.
- Add the following TupleTools: `MCTupleToolKinematic`, `MCTupleToolHierarchy`
- Add the following variables via LOKI functors: `TRUEETA` (functor `MCETA`), `TRUEPHI` (functor `MCPHI`).

```
from Configurables import DaVinci, DecayTreeTuple
from DaVinci.Configuration import *
from DecayTreeTuple.Configuration import *
from GaudiConf import IOHelper
from Configurables import MCDecayTreeTuple

def addNtupleToDaVinci():
    MCTuple = MCDecayTreeTuple("MCDecayTreeTuple")

    MCTuple.Decay = "[Lambda_b0 ==> ^(J/psi(1S) ==> ^e+ ^e-)  ^p+ ^K-]CC"
    MCTuple.Branches = {
        "Lb":" [Lambda_b0 ==>  (J/psi(1S) ==> e+ e-)  p+  K- ]CC",
        "L1":" [Lambda_b0 ==>  (J/psi(1S) ==> ^e+ e-)  p+  K- ]CC",
        "L2":" [Lambda_b0 ==>  (J/psi(1S) ==> e+ ^e-)  p+  K- ]CC",
        "Jpsi":" [Lambda_b0 ==>  ^(J/psi(1S) ==> e+ e-)  p+  K- ]CC",
        "Proton":" [Lambda_b0 ==>  (J/psi(1S) ==> e+ e-) ^p+  K- ]CC",
        "Kaon": "[Lambda_b0 ==>  (J/psi(1S) ==> e+ e-) p+  ^K- ]CC",
        }

    MCTuple.ToolList += [ "MCTupleToolHierarchy" ]
    MCTuple.ToolList += [ "MCTupleToolKinematic" ]
    MCTuple.ToolList += [ "MCTupleToolPID" ]


    from Configurables import TupleToolDecay
    MCTuple.addTool(TupleToolDecay, name = 'Lb')
    MCTuple.Lb.ToolList += ["MCTupleToolPID"]

    MCTuple.ToolList += [ "MCTupleToolReconstructed","TupleToolEventInfo","MCTupleToolPrimaries" ]

    from Configurables import LoKi__Hybrid__MCTupleTool
    MCTuple.addTool( LoKi__Hybrid__MCTupleTool, name = "MCLoKiHybrid" )
    MCTuple.ToolList += [ "LoKi::Hybrid::MCTupleTool/MCLoKiHybrid" ]
```

[3]See here `https://twiki.cern.ch/twiki/bin/view/LHCb/FAQ/LoKiNewDecayFinders#Arrows` for the explanation of different arrow types.

```
    MCTuple.MCLoKiHybrid.Variables = {
        "TRUEETA" : "MCETA",
        "TRUEPHI" : "MCPHI"
        }

    DaVinci().UserAlgorithms += [MCTuple]

Year = "2018"
MagnetPolarity= "MU"
DaVinci().Simulation = True
DaVinci().InputType = "DST"
DaVinci().Lumi = False
DaVinci().DDDBtag = "dddb-20170721-3"
CondDB_tag  = "sim-20190430-vc"
if MagnetPolarity == 'MU':
    DaVinci().CondDBtag = CondDB_tag + "-mu100"
elif MagnetPolarity == 'MD':
    DaVinci().CondDBtag = CondDB_tag + "-md100"

addNtupleToDaVinci()

DaVinci().DataType = Year
DaVinci().EvtMax = -1
DaVinci().PrintFreq = 1000
DaVinci().TupleFile = "MCTuple.root"

IOHelper().inputFiles(["00068049_00000016_1.bu2kll_nopid.strip.dst"],clear=True)
```

12 Preparing your Analysis Productions

After you have tested your DaVinci script on a local DST for both real data and simulation, we can prepare the scripts to submit jobs to the grid via Analysis Productions.

- Prepare one single script `info.yaml` which allows to submit your jobs for both real data and simulation.
- Note that in case you set `automatically_configure: true`, DataType, Simulation and condition tags attributes in the DaVinci options file aren't needed anymore.
- Push the scripts to the Analysis Productions repository (remember to follow the steps explained in the AP session) and check that the pipelines are passing. **Remember to add the *Starterkit* label!**.

```
defaults:
    wg: b2cc
    application: DaVinci/v46r12
    automatically_configure: true
    inform:
        - name.surname@cern.ch

data_stripping34r0p1_2018_MagUp:
  input:
    bk_query: /LHCb/Collision18/Beam6500GeV-VeloClosed-MagUp/Real Data/Reco18/Stripping34r0p1/
    90000000/LEPTONIC.MDST
  options:
    - DV_ntuple.py
```

```
  output: LB2PKJPSIEE.ROOT

MC_stripping34r0p1_2018_MagUp:
  input:
    bk_query: Beam6500GeV-2018-MagUp-Nu1.6-25ns-Pythia8/Sim09h/Trig0x617d18a4/Reco18/
    Turbo05-WithTurcal/Stripping34r0p1NoPrescalingFlagged/15154001/ALLSTREAMS.DST
  options:
    - DV_ntuple.py
  output: LB2PKJPSIEE.ROOT
```