Developing a New Generic Tool for Specifying Generator-Level Cuts in Gaussino

Summer Student: Simona Dubs
Supervisors: Gloria Corti and Adam Morris

Abstract

Gaussino is a modular, experiment-independent simulation framework for high-energy physics, and is built around two main steps: event generation and the transport of particles within the detector. To maximise the number of "useful" events in simulated samples, cuts on generator-level properties are applied. Many cut tools are hard-coded to act on specific properties, but a generic cut tool can apply an arbitrary combination of cuts based on a user-defined cutstring. This enables one to quickly define custom cuts for a specific decay or analysis without needing to modify the underlying C++ code. However, the library used to describe generic cuts at run-time (LoKiGen) is incompatible with the newer LHCb software for Run 3, and a replacement in the form of HepMC3 native functionality has been identified.

This report focuses on the development and implementation of a new interface in Gaussino for specifying generic generator-level cuts using functionality available in HepMC3, namely 'Feature' and 'Filter', and demonstrates that the generic cut tool yields an accurate and consistent result. Furthermore, the porting of an existing cut tool to HepMC3 format is described; two different approaches are tested, showing equivalent performance. The range of HepMC functors is also extended to span a range that is more comparable to that provided by LoKiGen.

CERN, Geneva, Switzerland
20 September, 2024

Contents

Figure 1: A map showing the location of the LHC on the French-Swiss border.

1 Introduction

Located at CERN, the Large Hadron Collider (LHC) is the world's largest and most powerful particle **accelerator**. The purpose of the LHC is to accelerate **hadrons** (i.e. protons) close to the speed of light; collisions between these fast-travelling protons, or with gas injected at one of the collision points, occur at extremely high energies ($\sim$TeV), producing cascades of elementary particles that scatter in all directions. Studying these intricate processes can offer valuable insights into the fundamental laws and structure of the Universe, as well as the conditions under which it was formed. To detect and image the results of proton-proton (pp) collisions, the LHC is equipped with four large **detectors**: ATLAS, CMS, ALICE and LHCb. These are located at different points along the 27-kilometre ring, shown in Fig. 1.

Among the four experiments, **LHCb** is one uniquely dedicated to the study of **heavy-flavour physics** [1]. This primarily focuses on **CP violation** and rare decays of b- and c-hadrons — particles containing a bottom or charm quark, respectively. Hadrons can be further distinguished by the number of quarks they contain; **mesons** have two, while **baryons** have three. Examples of B-mesons observed at LHCb include the B^0 ($d\bar{b}$), B^+ ($u\bar{b}$) and B_s^0 ($s\bar{b}$). A key decay channel is $B^0 \to \pi^+\pi^-$, where the B-meson decays into a pair of pions, and clean mass reconstruction of this final decay state is essential for the experiment. Therefore, it was chosen as a natural starting point for working with cut tools in this work, and this is discussed further in the 'Preliminary Test in Sim10' section.

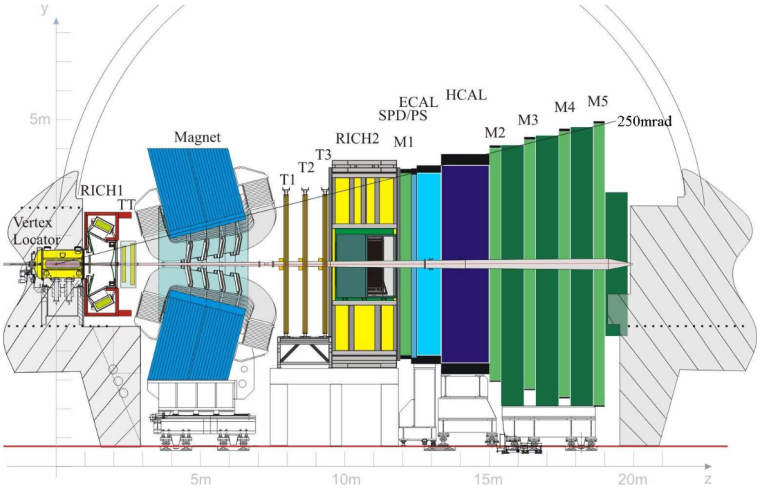

Figure 2: Cross-section view of the LHCb detector.

As shown in Fig. 2, the geometry of the LHCb detector is designed with this goal in mind. It is a single-arm spectrometer with a forward angular coverage, as opposed to the typical cylindrical geometry used by ATLAS, CMS and ALICE. This choice is motivated by the fact that *b*-hadrons are predominantly produced in the forward (or backward) direction at high energies, with a very small cross-section away from the beam axis, so the vast majority of these events are enclosed by the detector geometry. The LHCb detector occupies the cavern formerly used by the DELPHI experiment of the Large Electron-Positron Collider (LEP), so its compact design has also facilitated efficient use of the existing space.

Simulation plays a major role in the design, construction and operation of the any particle physics experiment. It provides an enhanced understanding of the experimental conditions and detector response for different types of events, and allows for comparisons with theoretical predictions to be made [2]. The LHCb simulation process is complex, with many layers of the framework operating in parallel to produce an output that accurately describes the observed reality; one important component of this process is the **generator**, which is further explained in the 'Software' section. To maximise the number of "useful" events in the simulated samples, it is common to apply **cuts** (i.e. restrictions) on certain generator-level properties. This task is performed by various **cut tools**, which are typically implemented as C++ classes. Many cut tools are hard-coded to act on specific properties, but some are fully **generic**, and thus able to apply arbitrary combinations of cuts passed as strings in configuration. This is especially useful for quickly defining cuts for a specific decay or analysis without having to release a new C++ class.

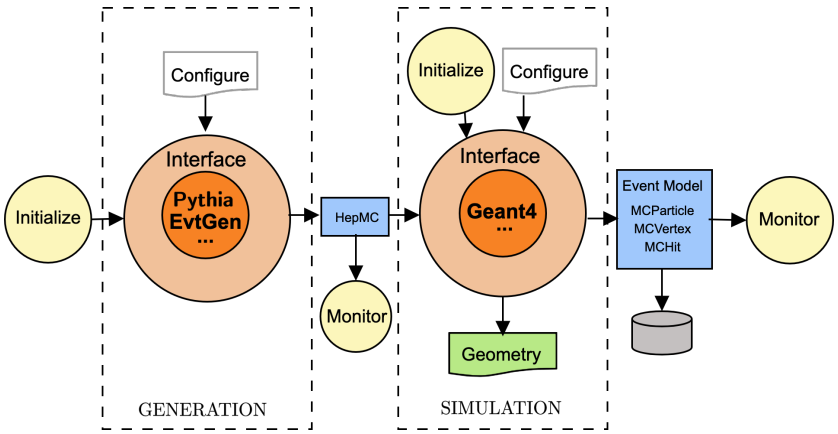

Figure 3: A schematic view of the flow and structure of Gauss.

While the core aim of this work was to develop a new tool for specifying generic generator-level cuts, it was convenient to split the project into several smaller tasks, each with its own contribution towards my learning process and the overall goal. The first step, outlined in the 'Preliminary Test in Sim10' section, was to test a small number of existing cut tools with the current version (Sim10) of the LHCb software. Next, the 'Porting an Existing Tool to Sim11' section details the process of porting one of these tools, such that it is compatible with a newer version (Sim11) of the LHCb software currently being developed. Finally, the implementation and testing of a new generic cut tool are described in the 'Developing a New Generic Cut Tool' section, where it is demonstrated that the tool can be configured correctly from an arbitrary user-defined string that is read and processed at run-time by a **parser**.

2 Software

2.1 Gauss-on-Gaussino

Simulating a collision at the interaction point of the LHCb detector consists of two main steps, and these are handled by an application called **Gauss**. Fig. 3 shows a schematic view of these steps with their interfaces and components. Gauss is based on the **Gaudi** framework, with dependencies on external libraries such as Pythia and EvtGen for event generation, and on GEANT4 for particle transport inside the detector [2]. Both generator-only and full-simulation modes are available in Gauss; for the purposes of this project, only the generation phase was enabled.

- **Event generation**. Full contents of the pp collision, as well as the decays

of some shorter-lived particles. This step is performed by various external packages (e.g. Pythia8, EvtGen, BcVegPy, EPOS) developed by theorists and phenomenologists.

- **Detector simulation**. Propagation of the longer-lived particles through the detector and their interactions with the detector materials. Relative to the generation phase, the simulation phase requires more time per particle and results in a longer event record. Though typically performed by GEANT4, there is a growing suite of alternative techniques (parametric simulation, GPU-accelerated, etc.).

In contrast, **Gaussino** is a simulation framework created by extracting the experiment-independent functionality of Gauss, with modernised features and use of multi-threading [3]. It can be used as either a toolkit upon which a full simulation application, such as Gauss, can be built, or in stand-alone mode for rapid prototyping of new experiments. Gaussino has a modular structure, consisting of four components: generation, detector simulation, geometry and monitoring the output [4]. Note that, at the time of writing, Sim11 is the newest version of Gauss available; this version is based on Gaussino, so it is also known as **Gauss-on-Gaussino**. Since both the Sim10 (Gauss) and Sim11 (Gauss-on-Gaussino) versions were used in this project, they will be referred to as 'Sim10' and 'Sim11', respectively, to avoid confusion.

The **Generation Algorithm** of Gauss is highly modular, meaning that each step of the generation phase is executed by a specific **tool**. Fig. 4 illustrates the sequence of steps in the generation phase, and a brief outline of the purpose of each tool is given below.

- **Pile-Up Tool**. Specifies the number of collisions for the simulated event.
- **Beam Tool**. Provides the kinematics of the colliding particles.
- **Sample Generation Tool**. Controls the type of sample that is generated. In order of increasing restriction, these options are:
  - **minimum bias**, keeping all events;
  - **inclusive**, keeping only the events where a specific category of particle is produced (e.g. a b- or c-hadron);
  - **signal**, keeping only the events where a certain species of particle is produced (e.g. B^+) and forcing it to decay to a specific final state (e.g. $B^+ \to J/\psi\, K^+$) using the Decay Tool.
- **Production Tool**. Calls the external generator (e.g. Pythia8) to produce the collision.
- **Decay Tool**. Ensures accuracy of the decay dynamics, including CP violation/mixing, angular corrections and complex amplitudes.

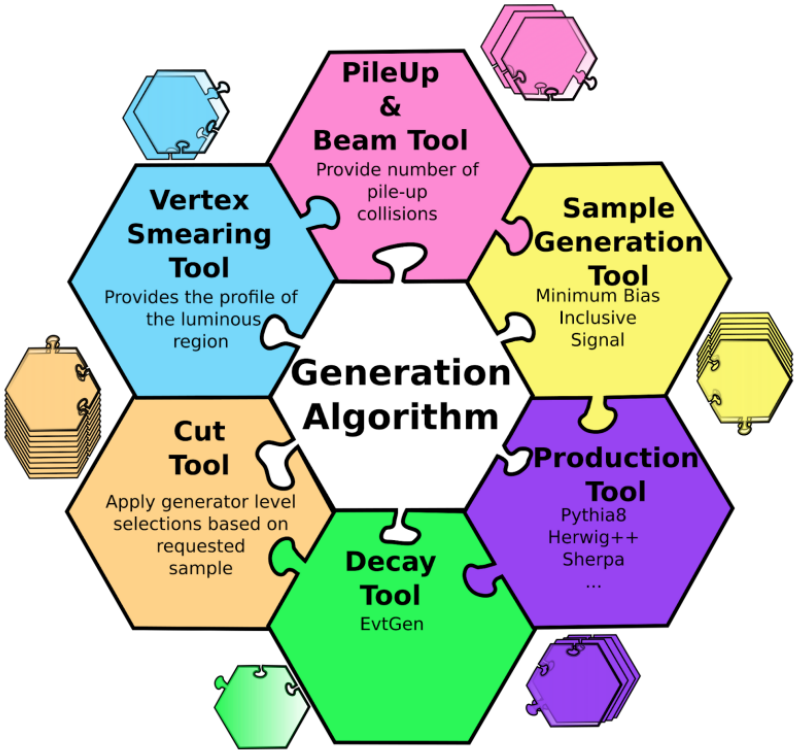

Figure 4: An overview of the Generation Algorithm in Gauss.

- **Cut Tool.** Retains only the "useful" events (e.g. the decays products of the signal particle must be within the acceptance region of the LHCb detector). There are two types:
  - **GenCutTool** applies cuts to the signal decay before the rest of the event is decayed;
  - **FullGenEventCutTool** applies cuts to the full event after all particles have decayed.
- **Vertex Smearing Tool**. Shifts the event by a random distance in space based on the profile of the beam spot (i.e. the volume in which the beams overlap).

2.2 LoKiGen

In Gauss, cut tools are implemented as C++ classes and equipped with functions (e.g. 'applyCut' or 'studyFullEvent') that return a decision on whether to keep or discard an event. These tools have **properties** that can be configured when the application is run, typically via a Python script. For example, setting

numerical values of variables or other options that affect how the tool behaves. As mentioned in the 'Introduction' section, generic cut tools are configured from a user-defined string passed in configuration, also known as a **cutstring**. A library called **LoKiGen** parses the cutstrings, configuring which cuts should be applied to which particles within the decay. The decay itself is given as a **decay descriptor** — a machine-readable string that encodes information about the decay channel. However, the use of LoKiGen in Run 3 of the LHC presents a number of challenges. Firstly, the library is not supported by the newer LHCb software used for Run 3, and porting it would require major work. Furthermore, it is specific to the LHCb experiment, and thus unsuitable for inclusion in Gaussino.

2.3 HepMC3

Since the event record is passed from the generation phase to the simulation phase in HepMC format, a natural replacement in the form of native functionality available in **HepMC3**[1] has been identified and explored in this work, namely the 'Feature' and 'Filter' classes. These classes are part of HepMC3's interface that allows users to search for particles related to other particles and vertices, referred to 'GenParticle' and 'GenVertex', respectively. A **Feature** wraps a function object that can extract either an integer or floating-point attribute from a GenParticle (e.g. energy, transverse momentum, azimuthal angle, PDG ID), while a **Filter** object can be used to select particles whose Features match specific requirements [5]. Practical use of both Feature and Filter will be shown in the 'Porting an Existing Tool to Sim11' section. Furthermore, the 'Selector' and 'StandardSelector' classes are described as an interface to certain "standard" Features, valid for both integral and floating-point comparisons [5]. Selector is very similar to Feature in the sense that Selectors are also constructed from function objects that extract certain attributes from GenParticles. StandardSelector is derived from Selector, and contains a few predefined Selectors (e.g. STATUS, ENERGY, MASS). Importantly, this option has the added benefit of being experiment-independent, making HepMC3 a promising candidate for replacing the suite of functionality currently provided by LoKiGen.

2.4 DaVinci

DaVinci is the offline analysis application used by LHCb; its main purpose is to produce tuples that contain relevant information about the particles involved in a simulated decay [6]. In this work, DaVinci was purely used to read the file

[1] Previous versions of Gauss used HepMC2, but both Gaussino and Sim11 use HepMC3, as this newer version, while very similar to its predeccesor, comes with modernised code and additional functionality (e.g. selection, filtering, standard attributes).

created by Gauss, in which the generator-level history is stored, and convert this into a more usable ROOT file for browsing the decay tree.

2.5 Spirit.Qi

A fully generic cut tool must have the ability to **parse** user-defined strings that specify any arbitrary combination of cuts, and this was implemented using **Spirit.Qi** — a C++ library from Boost designed for building recursive-descent parsers [7]. Spirit is already included in Gaudi, so its use in Gaussino is relatively natural and straightforward. In particular, the ability to define a recursive grammar structure was useful for reading cutstrings with nested expressions, as will be shown in the 'Developing a New Generic Cut Tool' section.

3 Preliminary Test in Sim10

Prior to developing and implementing new tools for Sim11, I first needed to learn and understand how to use the existing tools in Sim10. For this preliminary task, I acquired a range of new skills: configuring decay files (DecFiles), running Gauss and DaVinci, plotting histograms in PyROOT, using Git for version control, and navigating the LHCb software repositories. The objective was to choose a range of cut tools, obtain generator-only samples for each, and observe the resulting distributions of the signal tracks to verify that the cuts were applied correctly. As mentioned in the 'Introduction' section, the $B^0 \to \pi^+\pi^-$ decay channel is among the most important for the LHCb experiment, so it was chosen for this preliminary run. The generation tool, which is normally Pythia8, was set to ParticleGun for testing purposes; while slightly less realistic, this reduced the processing time and allowed for a larger number of events to be run. Three different types of cuts were tested, each with 10000 events.

- **No cuts**. Note that, while no cut tool was specified in the DecFile, it was observed that ParticleGun implicitly generates the signal particles (B^0) within the LHCb acceptance.
- **DaughtersInLHCb**. A standard cut tool that forces the decay products ($\pi^+\pi^-$) to be produced within the acceptance region of the LHCb detector, where the pseudorapidity is roughly in the range $2 \leq \eta \leq 5$.
- **Custom LoKiGen cuts**. For non-standard cuts, additional Python code was inserted between 'InsertPythonCode' and 'EndPythonCode'. In this case, the cuts were set such that the decay products ($\pi^+\pi^-$) are both within the LHCb acceptance and have a PT > 500 MeV.

To produce each sample with the above cuts, the following sequence of steps was executed.

1. Locate and modify the relevant DecFile[2] (.dec) to specify the cut tool.
2. Compile the package to generate an options file (.py) from the DecFile, which will be read by Gauss when it is run. This file is typically named '<EventType>.py', where EventType is a series of numerical flags encoding information about the sample.
3. Set the number of events to generate ('nEvts') in an additional options file, named 'Gauss-Job.py' by default.
4. Run Gauss for the generation phase only, which outputs an .xgen file containing the generator-level history for the sample. The generator (e.g. Pythia8, POWHEG, ParticleGun) is also specified at this step.
5. Run DaVinci to read the .xgen file, producing an MCDecayTreeTuple that can be opened and browsed with ROOT. Another options file, named 'tupleResult.py' by default, must be passed to DaVinci; this contains information such as the decay descriptor, name of the .xgen file, and the year associated with the generated sample.

3.1 $B^0 \to \pi^+\pi^-$

Having executed the above steps, the distributions of the π^+ were observed — the two pions have nearly identical distributions, so it is enough to inspect only the π^+. The pseudorapidity, abbreviated to 'ETA', plotted in Fig. 5 shows a clear difference between running with and without generator-level cuts. When using DaughtersInLHCb or custom LoKiGen cuts, the distribution is visibly concentrated in the acceptance region, as expected. For the transverse momentum, abbreviated to 'PT', Fig. 7 is included to emphasise the effect shown in Fig. 6. The custom LoKiGen cuts required that the π^+ have PT > 500 MeV, and this is exactly what is observed in Fig. 7, with no particles being found below this threshold. Therefore, it can be concluded that the cuts were applied correctly.

3.2 $D^0 \to K^-\pi^+$

As an additional check, I also tested an existing cut tool named 'SignalIsFromBDecay', which requires that the signal particle originate from a b-hadron. Using this cut tool on the $B^0 \to \pi^+\pi^-$ decay channel would be redundant, since the B^0 is itself a b-hadron! Therefore, it made sense to change to a different signal

[2]DecFiles are used for user-side configuration of the EvtGen package, which handles the modelling of decay dynamics in Gauss.

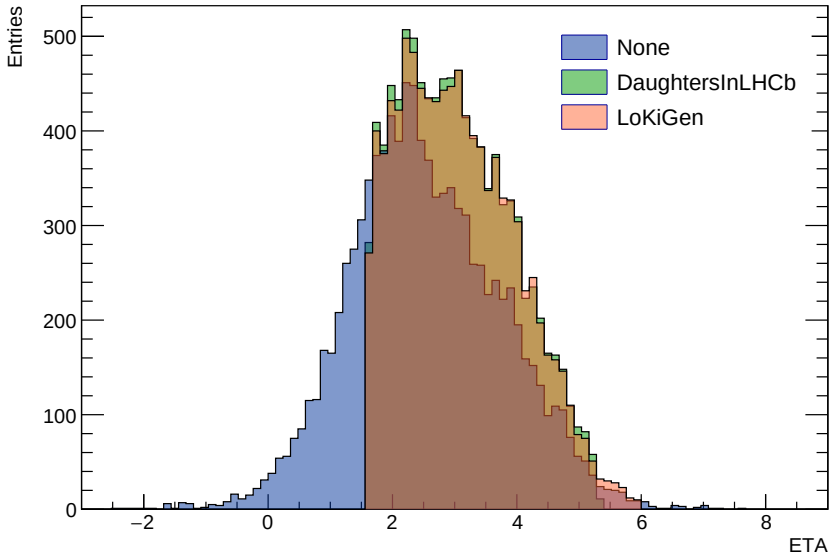

Figure 5: Pseudorapidity distribution of the π^+ in a $B^0 \to \pi^+\pi^-$ signal.

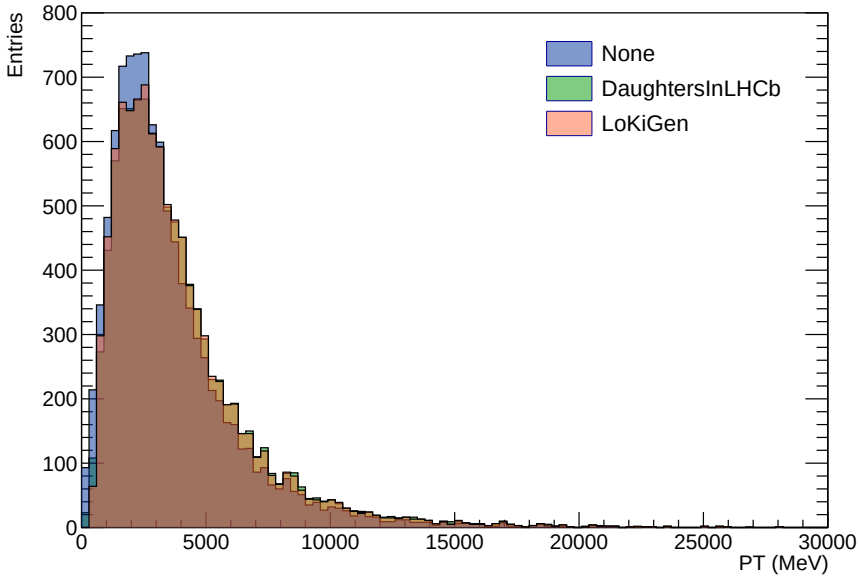

Figure 6: Transverse momentum distribution of the π^+ in a $B^0 \to \pi^+\pi^-$ signal.

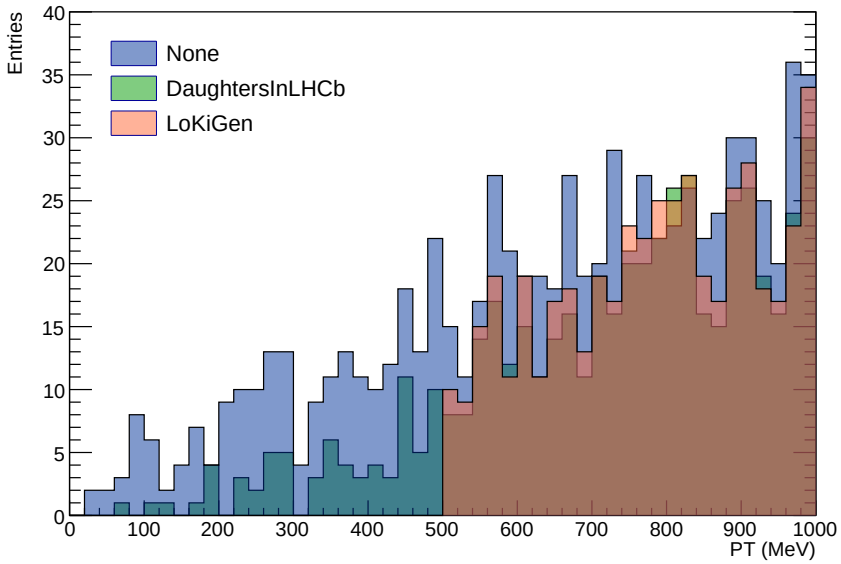

Figure 7: Zoomed-in view of the transverse momentum distribution in Fig. 6.

for this preliminary test — for example, $D^0 \to K^-\pi^+$. This time, Pythia8 replaced ParticleGun as the generation tool while keeping the number of events at 10000, resulting in a slightly longer processing time.

Fig. 8 and 9 show the MC_MOTHER_ID[3] and MC_GD_MOTHER_ID[4] distributions of the D^0, respectively. When running without SignalIsFromBDecay, the MC_MOTHER_ID is generally concentrated around 0, $\pm$400–450 and $\pm$500–550. It can be seen that the effect of applying SignalIsFromBDecay is a significant increase in the $\pm$500–550 range, which coincides with the PDG IDs of a number of B-mesons: B^0 (511), B^{*0} (513), B^+ (521), B^{*+} (523)... A very similar effect was observed in the MC_GD_MOTHER_ID, demonstrating that the cut tool worked as expected.

4 Porting an Existing Tool to Sim11

The version of SignalIsFromBDecay tested in the 'Preliminary Test in Sim10' section had not yet been ported from HepMC2 to HepMC3 format, so this was a natural starting point for the next stage of the project. Aside from general clean-up and modernisation, the source code was updated to make it compatible with

[3]The PDG ID of the first-ancestor (i.e. "mother") particle.

[4]The PDG ID of the second-ancestor (i.e. "grandmother") particle.

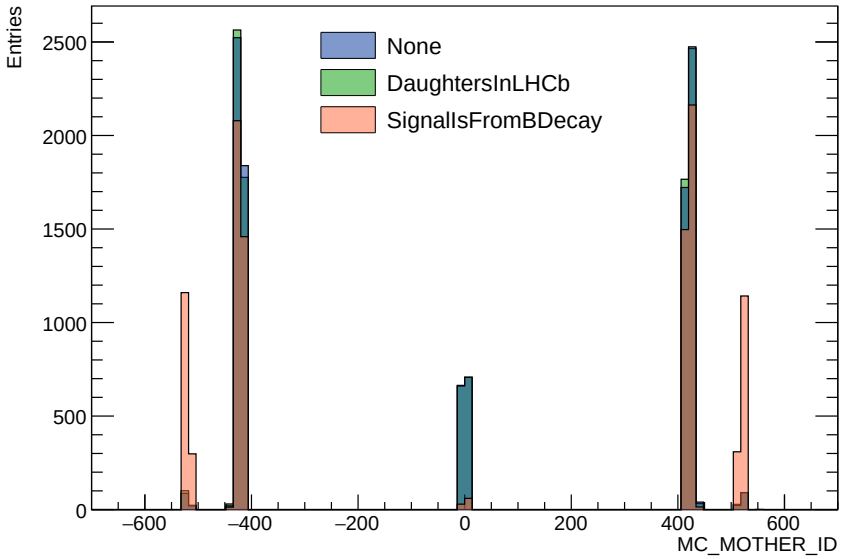

Figure 8: Mother ID distribution of the D^0 in a $D^0 \to K^-\pi^+$ signal.

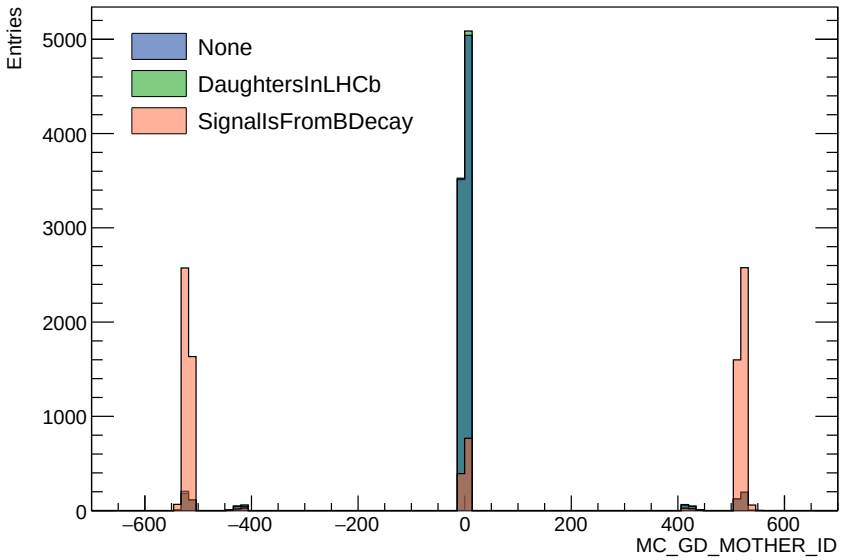

Figure 9: Grandmother ID distribution of the D^0 in a $D^0 \to K^-\pi^+$ signal.

HepMC3, with the most significant changes being made to the 'isFromB' method. As implied by its name, the purpose of this function is to check whether a given particle originated from a *b*-hadron at any point in its decay chain. It takes a pointer to a GenParticle as input, and returns a Boolean indicating the outcome. In Sim10, isFromB has a recursive structure; the decay tree is traversed by a loop until a *b*-hadron is identified, at which point the function returns 'true'. 'false' is returned when either (i) the entire decay tree did not contain any *b*-hadrons, or (ii) a recursion limit was reached. Statements to check for an invalid pointer, or whether the particle itself is a *b*-hadron, are also included in the function body.

Two different approaches to porting SignalIsFromBDecay to HepMC3 were tested in this work, the first of which is given below. This will be referred to 'Version A'.

```
/* Version A: */
bool SignalIsFromBDecay::isFromB( const HepMC3::
ConstGenParticlePtr part ) const
{
  // Check for invalid pointer:
  if ( part == nullptr ) return false;

  // Is this particle itself a b-hadron?
  if ( LHCb::ParticleID( part->pid() ).hasBottom() )
  return true;

  // Loop over parents:
  for ( HepMC3::GenParticlePtr ancr : HepMC3::Relatives
  ::ANCESTORS( part ) )
  {
    if ( LHCb::ParticleID( ancr->pid() ).hasBottom() )
    return true;
  }

  // If we get here, then not from a b-hadron:
  return false;
}
```

The key change made in this implementation of isFromB is that the recursive behaviour has been replaced with a 'for' loop over the Relatives::ANCESTORS of the GenParticlePtr[5], resulting in a cleaner and more compact syntax, while

[5]GenParticlePtr is practically identical to GenParticle*, but it is preferable to use the former.

also utilising the native functionality available in HepMC3. The second approach, which will be referred to as 'Version B', is given below.

```
/* Version B: */
bool SignalIsFromBDecay::isFromB( const HepMC3::
ConstGenParticlePtr part ) const
{
  // Check for invalid pointer:
  if ( part == nullptr ) return false;

  // Filter to check for b-hadron:
  HepMC3::Filter f = []( HepMC3::ConstGenParticlePtr p )
  { return LHCb::ParticleID( p->pid() ).hasBottom(); };

  // Is this particle itself a b-hadron?
  if ( f( part ) ) return true;

  // Are any of its ancestors b-hadrons?
  if ( !applyFilter( f, HepMC3::Relatives::ANCESTORS(
  part ) ).empty() ) return true;

  // If we get here, then not from a b-hadron:
  return false;
}
```

This approach is even more straightforward; instead of looping over the ancestors of the particle or traversing the decay tree, a HepMC3::Filter object is constructed from a lambda expression, and subsequently applied to both the particle and its ancestors. 'true' is returned when either (i) the particle or (ii) at least one of its ancestors pass the requirements of the filter, which are set using the 'hasBottom' method from the LHCb::ParticleID namespace, otherwise the function returns 'false'.

After adding both versions of the ported cut tool to Sim11, I proceeded by testing them, for which the process was relatively similar to that described in the 'Preliminary Test in Sim10' section. With Pythia8 as the generation tool, 1000-event samples were obtained for the same $D^0 \to K^-\pi^+$ signal as before, using (i) no cuts, (ii) DaughtersInLHCb, and (iii) the two versions of SignalIsFromBDecay. The results can be seen clearly in Fig. 10 and 11, where the distributions obtained from Version A and Version B overlap perfectly. Since both samples were run with the same random seed, this outcome is expected and demonstrates that the two approaches are equivalent. Furthermore, the same spike in the $\pm$500–550 range

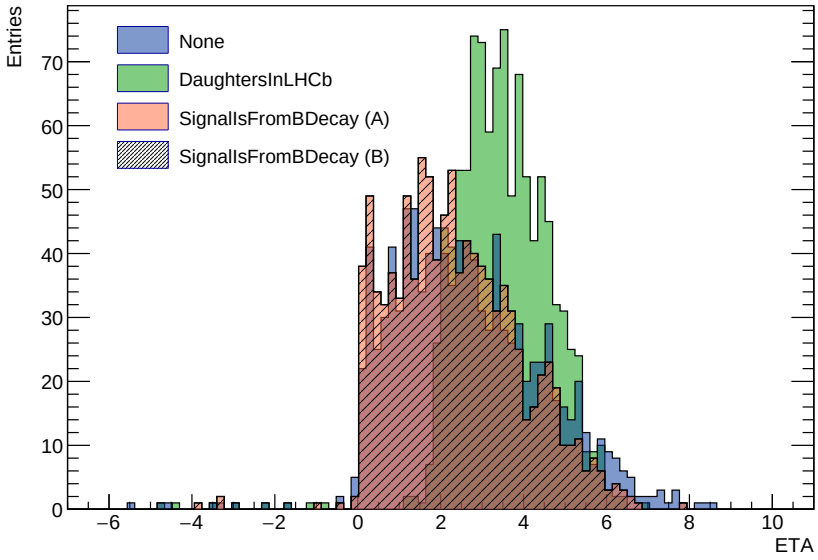

Figure 10: Pseudorapidity distribution of the D^0 in a $D^0 \to K^-\pi^+$ signal.

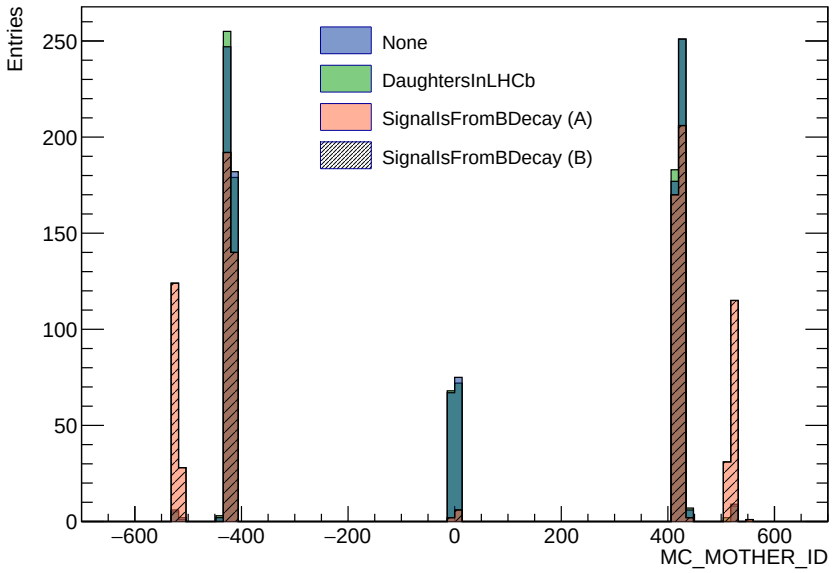

Figure 11: Mother ID distribution of the D^0 in a $D^0 \to K^-\pi^+$ signal.

of the MC_MOTHER_ID when SignalIsFromBDecay is applied can be observed, indicating that SignalIsFromBDecay works correctly in both Sim10 and Sim11.

5 Developing a New Generic Cut Tool

5.1 Extending HepMC Functors

As mentioned in the 'Software' section, the StandardSelector class in HepMC3 contains a small group of predefined Selectors; these are listed in Tab. 1 under the 'legacy' range. While the attributes included in StandardSelector are highly useful for physics analysis, there are many that appear to be missing. For example, one may want to construct a Filter to select only particles with a total momentum above some threshold, or whose polar angle falls within a specified range. This quickly becomes a problem, as LoKiGen has a significantly wider range of functors available.

Therefore, I began by identifying the most frequently used LoKiGen functors, and cross-checking with those already available in HepMC3. Functors found to be missing were implemented in a new 'UserSelector' class, designed to supplement StandardSelector, and are listed under the 'extended' range in Tab. 1. Once compiled successfully, both StandardSelector and UserSelector were re-implemented[6] as Features (i.e. Feature<int> or Feature<double>), and merged into a single 'StandardFeature' namespace that spans everything listed in Tab. 1. At the time of writing, StandardFeature covers a reasonable portion of the most essential LoKiGen functors, but it is possible that more will be added in future.

5.2 Parsing Cutstrings

Finally, this section of the report will focus on the development and testing of a new generic tool for specifying generator-level cuts, building on the concepts discussed up to this point. As mentioned before, the ability to correctly parse user-defined cutstrings is essential for a fully generic cut tool, which was handled using the Spirit.Qi library from Boost. At its core, Spirit.Qi is designed to facilitate the task of building a parser of any complexity — ranging from a comma-separated list of integers to arbitrary nested expressions — from an Extended Backus-Naur Form (EBNF) specification inlined in C++ [7]. Two structures from this library were especially useful: 'qi::**symbols**', a symbol table containing key-value pairs, and 'qi::**grammar**', a collection of rules that define the cutstring syntax. An example of a symbol table for physical units is shown below, where each unit is mapped onto its counterpart in Gaudi::Units. This ensures that, whenever a unit

[6]Certain problems later encountered in the parsing stage were resolved by switching from Selectors to Features.

Functor name	Data type	Explanation
	Legacy range	
STATUS	int	Status.
PDG_ID	int	PDG ID.
PT	double	Transverse momentum.
ENERGY	double	Energy.
RAPIDITY	double	Rapidity.
ETA	double	Pseudorapidity.
PHI	double	Azimuthal angle.
ET	double	Transverse energy.
MASS	double	Mass.
	Extended range	
P	double	Total momentum.
PX	double	Momentum x-component.
PY	double	Momentum y-component.
PZ	double	Momentum z-component.
THETA	double	Polar angle.
PVX	double	Production vertex x-position.
EVX	double	End vertex x-position.
PVY	double	Production vertex y-position.
EVY	double	End vertex y-position.
PVZ	double	Production vertex z-position.
EVZ	double	End vertex z-position.

Table 1: All functors available under the HepMC3::StandardFeature namespace.

is specified in the cutstring, the appropriate conversion of the numerical value is automatically carried out (e.g. "2 keV" $\rightarrow$ 2000 eV).

```
struct unit_table : qi::symbols<char, double> {
  unit_table() {
    using namespace Gaudi::Units;
    add
      // Energy:
      ( "MeV", MeV )( "eV", eV )( "keV", keV )
      ( "GeV", GeV )( "TeV", TeV )( "PeV", PeV )
      // Distance:
      ( "nm", nm )( "um", um )( "mm", mm )
      ( "cm", cm )( "m", m )( "km", km )
      // Time
      ( "ns", ns )( "ms", ms )( "s", s )
      // Angle:
      ( "rad", rad )( "mrad", mrad )
      ( "deg", deg )( "sr", sr );
  }
};
```

Initially, the objective was to write a simple parser that could read a "Feature Operator Value" cutstring (e.g. "PT > 500"). Once this had been accomplished, the above unit table was implemented, plus an additional rule in the grammar to handle the conversion. At this point, the parser could read a "Feature Operator Value Unit" cutstring (e.g. "PT > 500 MeV"). However, to parse a more complex cutstring, support for binary operators (i.e. '&&' and '||') and nested expressions was required, and the grammar was modified accordingly. Then, the generic cut tool was implemented as a new class named 'GenericCuts' in Gaussino. This class inherits publicly from both GaudiTool and IGenCutTool, with its own implementations of the 'applyCut' and 'initialize' methods, as well as one additional protected method named 'updateCuts' that calls the parser via 'parse_cutstring'. It also has a private 'm_filter' property to store an internal HepMC3::Filter, and a private 'm_cuts' property to store an internal Gaudi::Property<std::string>, which holds the cutstring itself. In summary, m_cuts is passed as the argument to parse_cutstring, which returns a HepMC3::Filter that is assigned to m_filter, and this is subsequently used by the applyCut method to invoke the desired generator-level cuts.

5.3 Testing the Generic Cut Tool

To ensure that the generic cut tool works as expected, I proceeded to test it in Sim11 with a $D^0 \to K^-\pi^+$ signal, passing the following cutstring: "(ETA > 2 && ETA < 5) && PT > 2 GeV". Note that this is very similar to the custom LoKiGen cuts used in the 'Preliminary Test in Sim10' section — the statement enclosed in brackets confines the signal particle (D^0) to the LHCb acceptance, while the latter part sets a minimum bound on its transverse momentum. For a reliable comparison, I ran a second sample with custom LoKiGen cuts, using an analogous expression but with slightly different syntax: "(GETA > 2) & (GETA < 5) & (GPT > 2 * GeV)". As before, Pythia8 was set as the generation tool, running with 1000 events. In Fig. 12 and 13, it can be seen that the distributions overlap perfectly; this proves that (i) the generic cut tool was configured successfully from a user-defined cutstring, and (ii) the result is identical to that obtained by using LoKiGen functors.

6 Conclusions

In summary, I have described the development and implementation of a new generic tool for specifying generator-level cuts in Gaussino. To validate that the tool works as intended, a sample cutstring with an arbitrary combination of cuts was passed; observing the distributions of the signal tracks, and comparing with those obtained by specifying the equivalent cuts using LoKiGen functors, verified that the tool produces a consistent output. However, at present, the tool can only be configured to apply cuts to the signal itself, so a future development could involve adding the ability to select specific particles within the decay descriptor, drawing inspiration from the existing LoKiGen syntax.

Furthermore, I have outlined the general process of porting an existing cut tool, SignalIsFromBDecay, from HepMC2 to HepMC3 format, making it compatible with the newer LHCb software for Run 3. This was accomplished by replacing the recursive behaviour of the cut tool with a HepMC3 native filter, resulting in a cleaner syntax and more compact implementation; a similar approach could be applicable to the remaining tools that have not yet been ported.

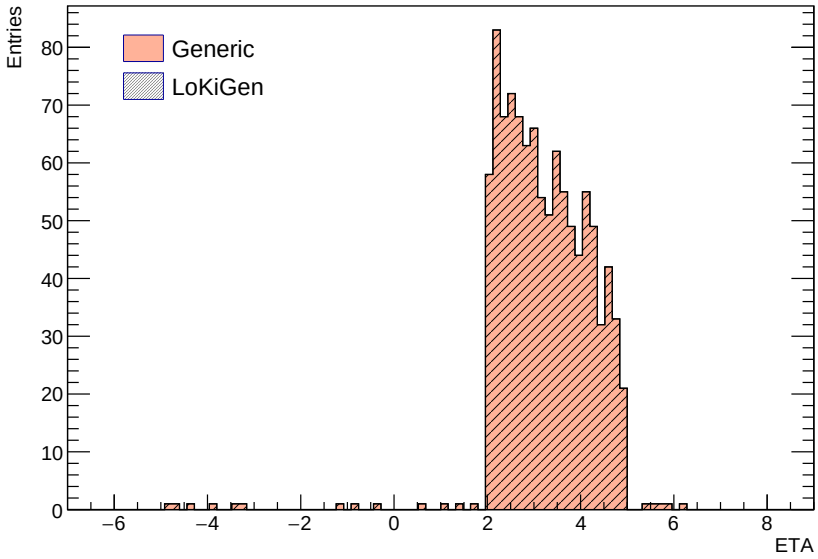

Figure 12: Pseudorapidity distribution of D^0 in a $D^0 \to K^-\pi^+$ signal.

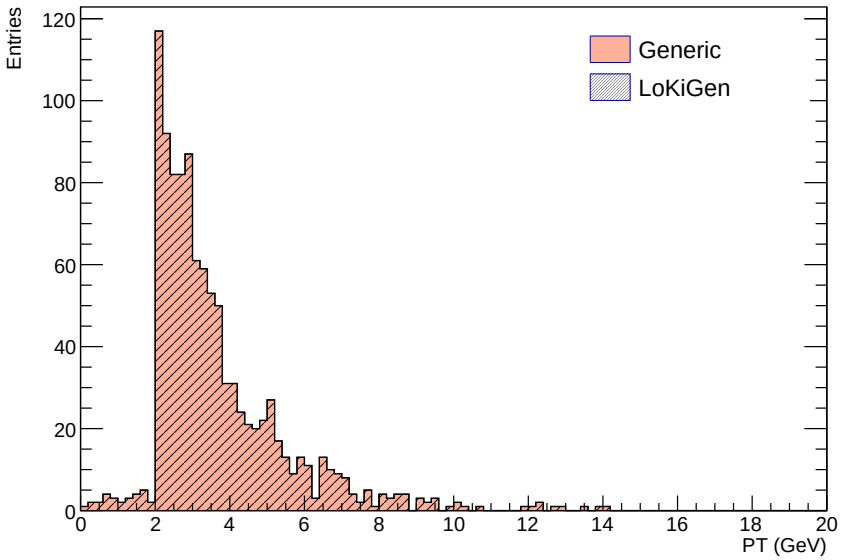

Figure 13: Transverse momentum distribution of D^0 in a $D^0 \to K^-\pi^+$ signal.

Acknowledgments

I would like to thank my supervisors, Gloria Corti and Adam Morris, for their invaluable support, guidance and supervision over the duration of this work, as well as my other colleagues at the LHCb Collaboration for welcoming me into this warm and friendly community. It has been a pleasure to contribute to the experiment, and I greatly look forward to the prospect of returning in the near future. I also wish to extend a word of gratitude to my family and friends, who have been incredibly supportive and encouraging throughout my time here, and to the Summer Student Programme Team for providing me with the opportunity to enjoy an exciting, formative and memorable three-month stay at CERN.

References

[1] The LHCb Collaboration *et al.* (2008) 'The LHCb Detector at the LHC', *Journal of Instrumentation*, 3(8), pp. S08005–S08005. doi:10.1088/1748-0221/3/08/S08005.

[2] Clemencic, M. *et al.* (2011) 'The LHCb Simulation Application, Gauss: Design, Evolution and Experience', *Journal of Physics: Conference Series*, 331(3). doi:10.1088/1742-6596/331/3/032023.

[3] Siddi, B. G., and Müller, D. (2019) 'Gaussino — a Gaudi-Based Core Simulation Framework', *2019 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*, Manchester, United Kingdom, 26 October – 2 November, pp. 1–4. doi:10.1109/NSS/MIC42101.2019.9060074.

[4] CERN, 'Gaussino Documentation'. Available at: https://gaussino.docs.cern.ch/. Date accessed: 20 September, 2024.

[5] Buckley, A. *et al.* (2010) 'The HepMC3 Event Record Library for Monte-Carlo Event Generators', *Computer Physics Communications*, 260(107310). doi:10.1016/j.cpc.2020.107310.

[6] CERN, 'DaVinci Documentation'. Available at: https://lhcb-davinci.docs.cern.ch/. Date accessed: 20 September, 2024.

[7] de Guzman, J. and Kaiser, H. (2011) 'Spirit 2.59 Documentation'. Available at: https://www.boost.org/doc/libs/1_86_0/libs/spirit/doc/html/index.html. Date accessed: 20 September, 2024.