

ANALYSIS SOFTWARE
AND
FULL EVENT INTERPRETATION
FOR THE
BELLE II EXPERIMENT

Zur Erlangung des akademischen Grades eines
DOKTORS DER NATURWISSENSCHAFTEN
von der Fakultät für Physik des
Karlsruher Instituts für Technologie (KIT)
genehmigte

DISSERTATION

von

Dipl.-Phys. Christian Pulvermacher
aus Remscheid

Tag der mündlichen Prüfung: 6. November 2015
Referent: Prof. Dr. M. Feindt
Korreferent: Prof. Dr. G. Quast

Contents

| | |
|--|-----------|
| 1. Introduction | 1 |
| 2. Experimental Setup | 5 |
| 2.1. KEKB and SuperKEKB | 5 |
| 2.2. The Belle Experiment | 7 |
| 2.3. The Belle II Experiment | 9 |
| 3. The Belle II Analysis Software Framework (BASF2) | 15 |
| 3.1. Python and Packages | 16 |
| 3.2. Modules and Data Exchange | 17 |
| 3.3. Path Control Flow | 19 |
| 3.4. Input/Output | 21 |
| 3.5. Parallel Processing | 22 |
| 3.6. Merging Objects | 28 |
| 3.7. Event Display | 31 |
| 3.8. Summary and Conclusions | 38 |
| 4. Analysis Tools | 39 |
| 4.1. Particle Candidates | 40 |
| 4.2. Vertex Fitting | 43 |
| 4.3. Monte Carlo Matching | 44 |
| 4.4. Multivariate Classification | 48 |
| 4.5. Skimming | 50 |
| 4.6. Best Candidate Selection | 51 |
| 4.7. Saving n -Tuples | 51 |
| 4.8. High-Level Reconstruction Tools | 52 |
| 4.9. Summary and Conclusions | 53 |
| 5. Tag-Side Reconstruction at Belle | 55 |
| 5.1. Control Variables | 56 |
| 5.2. Cut-based Full Reconstruction | 56 |
| 5.3. Neural-network-based Full Reconstruction | 57 |
| 5.4. Extensions of the Full Reconstruction | 61 |
| 5.5. Summary | 63 |
| 6. Full Event Interpretation | 65 |
| 6.1. Software Architecture | 65 |
| 6.2. Particle Selection and Combination | 71 |
| 6.3. Reducing Combinatorics | 75 |

Contents

| | |
|--|------------|
| 6.4. Classifier Trainings | 79 |
| 6.5. Automatic Reporting | 83 |
| 6.6. Training Modes | 89 |
| 6.7. Distributed FEI Trainings | 95 |
| 6.8. Conclusions | 97 |
| 7. Estimation of Physics Performance | 99 |
| 7.1. Monte Carlo Sample | 99 |
| 7.2. Training | 100 |
| 7.3. Results | 103 |
| 7.4. Comparison with Full Reconstruction | 112 |
| 7.5. Discussion of Signal Definition | 113 |
| 7.6. Summary and Conclusions | 117 |
| 8. FEI on Converted Belle Data | 119 |
| 8.1. Conversion of Belle mDST Data | 119 |
| 8.2. Monte Carlo Sample | 121 |
| 8.3. Training | 121 |
| 8.4. Results | 125 |
| 8.5. Summary and Conclusions | 132 |
| 9. Conclusions | 135 |
| Appendix | 137 |
| A. Example FEI Report for a Small Training | 139 |
| A.1. Summary | 139 |
| A.2. CPU time per channel | 141 |
| A.3. Particle configuration | 142 |
| A.4. Final state particles | 144 |
| A.5. Particle: D^+ | 149 |
| A.6. Particle: B^0 | 154 |
| Bibliography | 161 |

1. Introduction

In a theoretical framework in which most experimental facts of physics can be interpreted, discovery frequently involves not the entirely unknown, but rather those areas that lie just beyond the framework's sphere of influence. One such framework is the standard model of particle physics (SM), which is able to explain a host of phenomena and allows some of the most accurate predictions of experimental values to date [1]. With the confirmation of the Higgs boson in 2012, all particles predicted by the standard model have also been found [2, 3]. The model's explanatory power is, however, not all-encompassing.

For example, from observations of the rotational velocities of galaxies [4] and other gravitational effects, the presence of an invisible substance, dark matter, can be inferred, which cannot reasonably be described using standard model particles. Another somewhat central problem is the observed matter–antimatter asymmetry in the universe. While it is possible to explain the asymmetry using a combination of baryon number violation, C and CP violation, and thermal inequalities in the early universe [5], the size of CP violation in the SM is not sufficient to explain the baryon asymmetry visible around us [6, p. 8f.].

In the standard model, CP symmetry is broken through the Cabibbo–Kobayashi–Maskawa (CKM) quark mixing mechanism [7, 8], in which it is described by an irreducible phase in the associated mixing matrix. It was first detected in the form of *indirect CP violation* in the neutral kaon system in 1964 by Cronin and Fitch [9], where an asymmetry exists in K^0 – \overline{K}^0 oscillations. In the B meson system, the predicted indirect CP violation was small for B^0 – \overline{B}^0 mixing, but a large *direct* effect was expected in certain decays of B mesons [10]. The decay $B^0 \rightarrow J/\psi K_S^0$, with its charge-conjugate $\overline{B}^0 \rightarrow J/\psi K_S^0$, is known as a ‘golden mode’ due to the relative ease of measuring CP asymmetries between the two decays [11, p. 2]. As a $B^0\overline{B}^0$ pair produced through decay of the $\Upsilon(4S)$ resonance is in an entangled state until one of them decays, determining the flavour (i.e. B^0 or \overline{B}^0) of one B also allows inferring the flavour of the other meson. Using elaborate experimental techniques, the flavour of both B mesons can thus be determined with good accuracy using the decay products of one meson. The other meson can decay into a CP eigenstate like $J/\psi K_S^0$ in multiple ways, as shown in [Figure 1.1](#). If CP symmetry is violated, there is a difference in weak phase between these diagrams, which can cause constructive or destructive interference depending on the decay time. Since this interference effect is inverted for \overline{B}^0 when compared to B^0 , this results in a time-dependent difference of the decay rates of both B meson flavours. Detecting this asymmetry requires $B^0\overline{B}^0$ pairs and a way to measure their relative decay time.

To facilitate this type of measurement, two B factories, KEKB in Tsukuba, Japan, and PEP-II in Stanford, United States, (with the experiments Belle and BaBar, respectively) were proposed, and both started taking data in 1999. As electron–positron colliders operating on the $\Upsilon(4S)$ resonance (a $b\bar{b}$ state), they were designed to produce a large number of B mesons, with asymmetric beam energies introducing a boost to the center-of-mass system that enables measuring the decay length of B mesons, and hence their relative lifetime. Since

1. Introduction

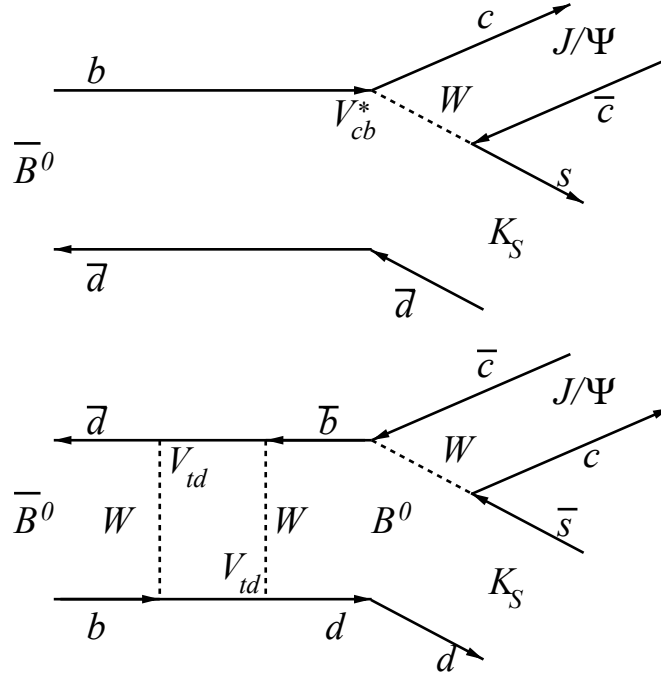


Figure 1.1.: Dominant Feynman diagrams for $\bar{B}^0 \rightarrow J/\psi K_S^0$ decays, with a direct decay (top) and a decay with \bar{B}^0 - B^0 mixing (bottom). Taken from [11].

the mass of the $\Upsilon(4S)$ resonance lies just above the threshold for the production of pairs of B mesons, no further particles are produced. Together with the relatively low energies available through B meson decays, which are not enough to produce jets of particles, the $\Upsilon(4S)$ events recorded at B factories are very clean.

The confirmation at Belle and BaBar of a large CP asymmetry, as shown in [Figure 1.2](#), which was consistent with the CKM quark mixing formalism was recognised in the awarding of the Nobel prize to Kobayashi and Maskawa in 2008 [13]. Besides CP violation, B factories were also able to measure other parameters of the CKM matrix with improved resolution, and could detect a number of very rare B decays [14]. Since certain models of physics beyond the standard model predict significantly different branching ratios for these decays, this makes them interesting for actually testing the *new physics* predictions against those of the standard model. Additionally, a variety of new states have been found, including charmonium($c\bar{c}$)- and bottomonium($b\bar{b}$)-like resonances, some of which may be tetraquark states ($qq\bar{q}\bar{q}$) or molecules [14, p. 85ff.].

Thus, B factories find themselves in a unique position at the frontier of the standard model to probe its predictions with high precision. To allow further discoveries and improved measurements using their unique environment, a successor to the Belle experiment and the KEKB accelerator was suggested in 2004, and the Belle II collaboration formed in 2008 [15, p. 1]. For the Belle II experiment, the accelerator will be upgraded to achieve an instantaneous luminosity 40 times higher than that of KEKB [15, p. 20]. The detector itself is also upgraded to deal with a number of changes in the environment (e.g. beam background) and to be able to provide competitive measurements. First $\Upsilon(4S)$ events are expected to be recorded in

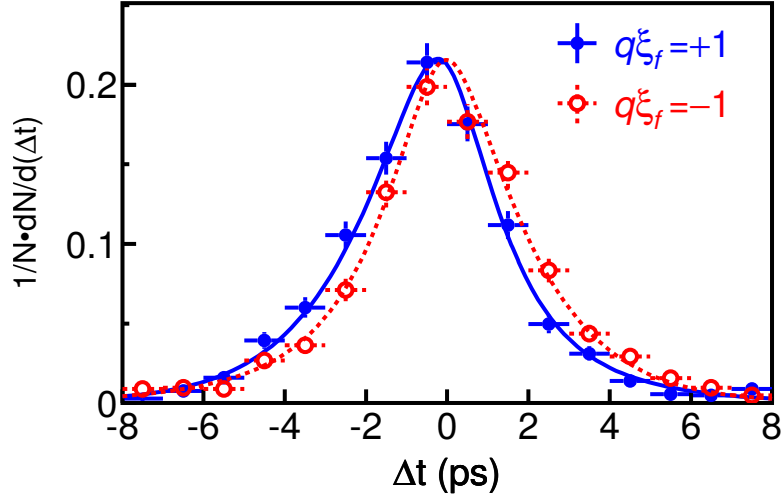


Figure 1.2.: Distribution of Δt for events with different $q\xi_f$ measured by Belle, where q is the flavour tagging output (+1 for B^0 , -1 for \overline{B}^0 -like), and ξ_f is the CP eigenvalue of the final state. The difference between the two distributions indicates that CP symmetry is violated. Taken from [12].

2018 [16]. Both the Belle and Belle II experiments are described in more detail in [Chapter 2](#).

Over the last century, analysis techniques in particle physics have evolved dramatically. While early discoveries in particle physics, like that of the positron in 1932 [17] and of the kaon in 1947 in cosmic rays [18], could be confirmed using individual cloud chamber images, more recent findings are usually based on larger numbers of events. For example, the 1973 discovery of weak neutral currents was based on an analysis of almost 300 000 bubble chamber images [19], while experiments at the Large Hadron Collider (LHC) have output data rates of many hundred events per second. Advances in information technology have shifted most experiments from manual methods to an entirely electronic readout of detectors and a chain of components that processes the data into something that is useful for analysts.

For Belle II, all these steps are implemented using a single software framework, which consequently has to deal with both large data rates and very dissimilar use-cases, including performing a fast online-reconstruction on the high-level trigger, online-monitoring, saving of detector data in a common file format, reconstruction tasks like track finding, and finally, physics analyses. As such, the framework is used by a large group of developers with different amounts of programming experience and benefits from a consistent and user-friendly interface. For this thesis, the Belle II Analysis Software framework (BASF2) was made more user-friendly and reliable, and was extended to make use of the multiple CPU cores present in modern processors. It was also integrated more tightly with the Python scripting language, allowing faster prototyping and tests. Both the framework in general and the improvements to it are discussed in [Chapter 3](#).

The components used for physics analyses are an especially important aspect of the software, as they define how analyses are performed, and by extension which part of analyses physicists spend time on. For the Belle II collaboration, a modular approach has been developed that provides common analysis tasks in an interoperable way, so they can be

1. Introduction

combined to support even the most complicated tasks. A crucial point is that all common analysis steps are already available and do not need to be implemented by each user. This also allows for significant gains in reliability, since algorithms can be tested thoroughly and in different circumstances. Many analysis tools were redesigned or improved within the scope of this thesis. They are described in detail in [Chapter 4](#).

The aforementioned cleanliness of $\Upsilon(4S)$ events at B factories also allows the development of specialised analysis techniques. One such technique uses the fact that such events contain exactly two B mesons by reconstructing¹ one B meson with high efficiency in a number of possible decays, and combining it with a B meson reconstructed in a user-defined signal decay channel. If both B mesons are reconstructed correctly, this is equivalent to reconstructing the $\Upsilon(4S)$, and thus the entire event. As a result, the analyst can greatly enhance the purity of their signal selection by requiring that no tracks remain in the event. Additionally, since the four-momentum of the $\Upsilon(4S)$ is known, the reconstructed four-momentum of one B meson also determines that of the other B . Within this thesis, a new, highly automated algorithm for Belle II is introduced that performs the necessary high-efficiency reconstruction of B mesons and is based on the analysis tools presented earlier. This algorithm, called *Full Event Interpretation*, offers high reconstruction efficiencies, flexibility, and can improve many analyses. After introducing a similar algorithm and its applications for the Belle experiment in [Chapter 5](#), the Full Event Interpretation will be discussed in [Chapter 6](#) and the following chapters.

¹ In the context of particle candidates, ‘reconstruction’ refers to the creation of the candidate through combinations of other particles and may include other steps to improve the selection.

2. Experimental Setup

As the analysis software of both experiments will be discussed in this thesis, the experimental setup of both Belle and Belle II, as well as the associated accelerators will be introduced in this chapter. Only a brief description of the detectors will be given, more detailed information can be found in references [11] and [20] for the Belle experiment and in reference [15] for Belle II.

2.1. KEKB and SuperKEKB

The KEKB accelerator providing electron and positron beams for the Belle experiment was operated from 1999 until 2010. It was installed in the tunnel previously used by the TRISTAN accelerator, with an improved linear injection accelerator to ensure sufficient positron production rates [11, p. 5]. A large instantaneous luminosity could be achieved by using a large crossing-angle combined with so-called crab-cavities, which reorient the beam bunches so they collide frontally; this ultimately resulted in a peak luminosity of $2.11 \times 10^{34} \text{cm}^{-2}\text{s}^{-1}$, twice as high as the design value [11, p. 5]. To create the boost necessary for measuring the relative decay times of B mesons, the beam energies of the two rings are asymmetric, with a low-energy positron ring (3.5 GeV) and a high-energy electron ring (8.0 GeV). Over its entire run time, KEKB was able to accumulate a record integrated luminosity of over $1\,000 \text{fb}^{-1}$, surpassing PEP-II by a factor of almost two, as can be seen in Figure 2.1.

Alongside the upgrade to the Belle II experiment, the KEKB collider is updated to SuperKEKB, with its main feature being a greatly increased instantaneous luminosity. A significant difference to the KEKB accelerator design is the ‘nano-beam scheme’, which involves squeezing the vertical beta functions β_y^* of both beams at the interaction point [15, p. 19]. As can be seen in Table 2.1, the design values for the vertical beta functions at SuperKEKB are smaller than those of KEKB by almost a factor twenty. The beam currents I were also increased to almost twice their previous values. The luminosity of a collider can be expressed as [15, p. 19]

$$L \sim \frac{\gamma_{\pm}}{2er_e} \left(\frac{I_{\pm} \xi_{y\pm}}{\beta_{y\pm}^*} \right),$$

where γ_{\pm} is the Lorentz factor and $\xi_{y\pm}$ the vertical beam–beam parameter for positrons (+) and electrons (–), e the elementary charge and r_e the classical electron radius. The proportionality constant here contains certain reduction factors, e.g. due to the crossing angle. The changes in β_y^* and I alone thus suggest great increases in luminosity. The design value for SuperKEKB is $80 \times 10^{34} \text{cm}^{-2}\text{s}^{-1}$, an increase of almost a factor forty compared to KEKB. The beam parameters and luminosity of both accelerators are compared in Table 2.1.

As these changes to the beam parameters also affect the rates for background processes occurring through interactions of the beams with themselves or remaining gas atoms, significant increases in the amount of background are expected for Belle II. Since the dominant

2. Experimental Setup

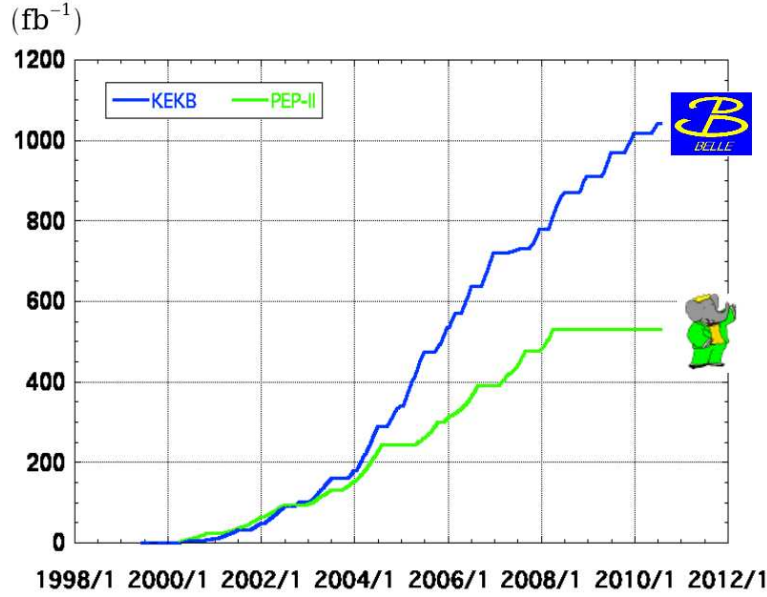


Figure 2.1.: Integrated luminosity over time at the B factories KEKB (Belle) and PEP-II (BaBar). Taken from [21], based on [22].

Table 2.1.: Beam parameters at KEKB and SuperKEKB. ξ_y denotes the vertical beam–beam parameter, β_y^* the vertical beta function at the IP, and I the beam current. Taken from [15, p. 20].

| | KEKB Achieved | SuperKEKB |
|---|---------------|-------------|
| Energy (GeV) (LER/HER) | 3.5/8.0 | 4.0/7.0 |
| ξ_y | 0.129/0.090 | 0.090/0.088 |
| β_y^* (mm) | 5.9/5.9 | 0.27/0.41 |
| I (A) | 1.64/1.19 | 3.60/2.62 |
| Luminosity ($10^{34}\text{cm}^{-2}\text{s}^{-1}$) | 2.11 | 80 |

background source, the Touschek effect, is proportional to E^{-3} (with beam energy E), for SuperKEKB the asymmetry was reduced using energies of 4.0 GeV for positrons and 7.0 GeV for electrons as mitigation [15, p. 22].

First beams are expected at SuperKEKB in 2016, with a commissioning detector being used instead of Belle II, followed by collisions at a non- $\Upsilon(4S)$ resonance energy in 2017 with part of the Belle II detector, and operation at the $\Upsilon(4S)$ energy with the full experiment in 2018 [16].

For the detectors, a standard coordinate system is used, with the x axis being horizontal (pointing away from the center of the accelerator), y being up, and z being in forward direction. The conventions for Belle and Belle II differ subtly: At Belle, the z axis is identical to the low-energy ring at the IP (with positrons flying towards $-z$), at Belle II it lies halfway between the low- and high-energy beams [23].

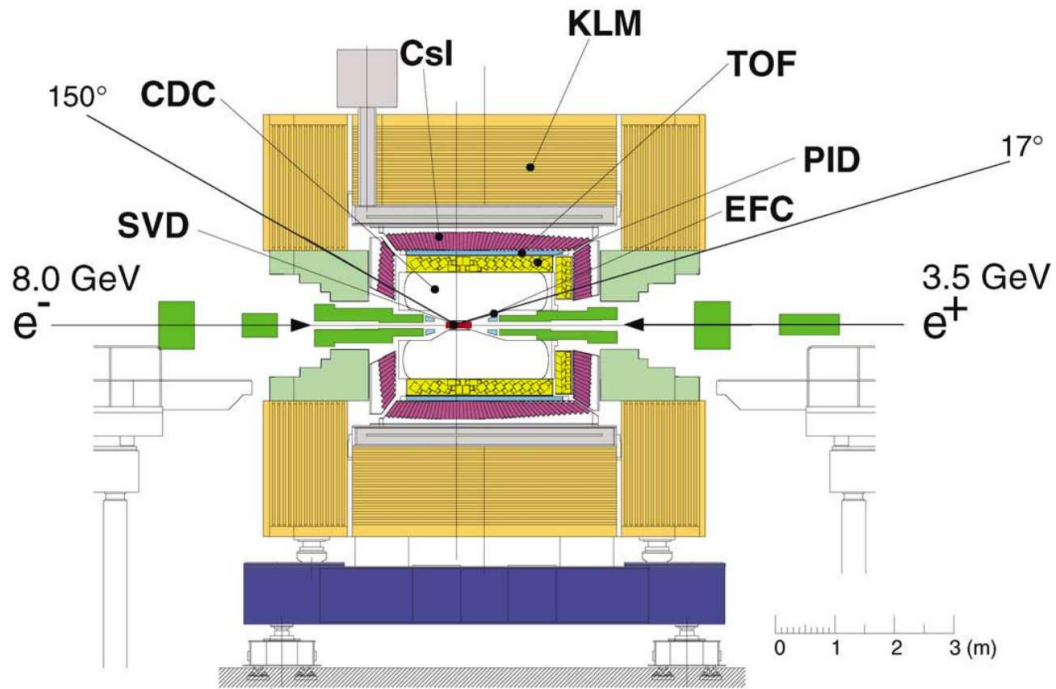


Figure 2.2.: Cross-section of the Belle detector. Adapted from [20].

2.2. The Belle Experiment

The Belle experiment is a 4π detector covering the interaction region of KEKB from 17° to 150° in the polar angle θ and in 360° of the azimuth angle ϕ [20]. Due to the boost provided by the accelerator, the detector is arranged somewhat asymmetrically, with more coverage in the front, i.e. in the direction of the high-energy beam. The inner detectors, consisting of tracking detectors, particle identification detectors, and the electromagnetic calorimeter, are contained in a superconducting solenoid producing a 1.5 T magnetic field. A cross-section of the Belle detector giving an overview of the arrangement of sub-detectors can be found in [Figure 2.2](#).

2.2.1. Tracking Detectors

The tracking detectors are used to record signals (hits) of the passage of charged particles and allow the determination of their momentum by measuring the track curvature in the magnetic field. The innermost detector, called the Silicon Vertex Detector (SVD), was placed in close proximity to the beam pipe, and thus to the interaction region. Its main goal was to provide the vertex resolution needed to measure the distance between two B decay vertices for CP violation measurements.

A first version of the detector, SVD1, was used at the beginning of data-taking operations in 1999, and already replaced in 2000 due to radiation damage to the readout electronics [11, p. 26, p. 44]. In 2003, SVD1 was retired entirely and replaced with the greatly improved SVD2, which was more resilient and allowed great increases to the luminosity of KEKB. This final

2. Experimental Setup

version consisted of four layers of double-sided silicon strip detectors (DSSD) installed at radii between 20 mm and 88 mm [11].

The Central Drift Chamber (CDC) was the larger of the two tracking detectors, and had inner and outer radii of 80 mm and 880 mm, respectively, though no inner wall was used to avoid multiple scattering [11, p. 27]. The CDC contained a total of 8 400 drift cells arranged in 50 layers [20]. With the installation of SVD2, the inner three layers were replaced to make room for the larger silicon detector [11]. In addition to being used for track reconstruction, energy depositions of particles in the CDC were also used for particle identification using the specific energy loss dE/dx .

2.2.2. Particle Identification

More powerful particle identification (PID) information came, however, from two detectors dedicated to this task and positioned outside of the CDC: the Aerogel Cherenkov Counter (ACC) and the Time-Of-Flight (TOF) detector. The ACC (labelled 'PID' in Figure 2.2) consisted of aerogel blocks of different refractive indices covering an acceptance of 13.6° to 127.9° in θ (i.e. in the barrel and forward regions). As a Cherenkov detector, the ACC could only detect Cherenkov radiation for particles over a certain velocity threshold. Heavier particles with the same momentum, having a lower velocity, would in many cases not be faster than the local speed of light (phase velocity) in the aerogel, and would not create a signal. The number of photons detected for a given track thus was a powerful PID variable.

The time-of-flight detector, on the other hand, was installed in the barrel region only, and thus had a far smaller acceptance of 33° to 121° in θ [11, p. 29]. With a design time resolution of 100 ps, the detector could provide accurate velocity measurements for tracks. When combined with a momentum measurement from the track reconstruction, the particle's estimated mass could be used to distinguish between different kinds of particles. Tracks with a transverse momentum below 280 MeV could not reach the TOF detector, and needed to be identified by the ACC (if in forward direction) or the CDC alone.

2.2.3. Electromagnetic Calorimeter

The electromagnetic calorimeter (ECL) encloses both the tracking and PID detectors, and is responsible for detecting photons as well as measuring the energy deposition associated with tracks to aid in identifying electrons. The calorimeter consists of 8 736 Tl-doped CsI scintillator crystals covering a range of $17^\circ < \theta < 150^\circ$ in the polar angle and weighs a total of 43 tons [20, p. 59]. At the end of each crystal, a photomultiplier (PMT) detects light emitted by the scintillator, which can be used to measure the amount of energy deposited. Clusters of activated crystals in the vicinity of a track are used to identify electrons, which deposit significantly more energy in the calorimeter. Clusters with no associated track can be interpreted as photons, though some backgrounds (e.g. from beam interactions) are also present at lower energies.

2.2.4. Extreme Forward Calorimeter

The Extreme Forward Calorimeter (EFC) extends the acceptance of the ECL in both the forward and – contrary to the name – in backward region for an improved sensitivity to

physics processes like $B \rightarrow \tau \nu$ [20, p. 11]. Its 2×160 $\text{Bi}_4\text{Ge}_3\text{O}_{12}$ crystals have an acceptance of $6.2^\circ < \theta < 11.6^\circ$ in the forward, and $163.1^\circ < \theta < 171.5^\circ$ in the backward region [24, p. 18]. The EFC was attached to KEKB solenoid magnets near the interaction point and also functioned as a beam mask to shield the central drift chamber from backgrounds [20].

2.2.5. K_L^0 and Muon Detection

The return yoke of the 1.5 T solenoid, covering both the barrel and the forward and backward endcap regions, consists of iron plates of 4.7 cm thickness, which sandwich resistive plate counters (RPCs) that can detect passing particles [20, p. 77]. Since this makes it useful for detecting neutral hadrons (especially K_L^0) and muon identification, it is called the K_L^0 and Muon Detector (KLM). As muons at Belle usually have momenta that make them minimum ionising particles, they can traverse even iron without great energy loss. Hadrons, on the other hand, interact strongly with the iron nuclei, losing energy and creating showers of secondary particles in the process. These differences allow the KLM to distinguish very well between charged pions and muons, which are not easy to separate for the other PID detectors because of their similar mass. Clusters in the KLM not matched with a charged track indicate detection of a neutral hadron, possibly a K_L^0 . Due to the low energies, only few secondary particles are produced in the hadronic showers, resulting in large statistical fluctuations. An energy measurement for neutral clusters thus is not possible.

2.3. The Belle II Experiment

With the upgrade to SuperKEKB, many of the previously described detectors were replaced. In part, this was done out of a desire to improve the performance, in particular of the particle identification systems, but the changed conditions with a much higher luminosity and a different setup of focussing magnets and the accelerator itself also drove certain changes.

The Belle and Belle II detectors are, however, fairly similar in overall structure, and components have been reused where appropriate. One example is the superconducting solenoid, which is inherited from Belle, so the same 1.5 T magnetic field will be used. The ECL and KLM components will also be reused, while Belle's SVD, CDC, ACC, TOF, and EFC are retired. A cross-section of the new Belle II detector and its different sub-detectors can be found in [Figure 2.3](#).

2.3.1. Tracking Detectors

As a result of the decreased boost at SuperKEKB (see [Section 2.1](#)), the tracking detectors at Belle II need to be improved significantly to maintain or exceed the vertex resolution achieved at Belle. For this reason, a pixel detector was added in close proximity to the interaction point as a third tracking detector. Additionally, the beam pipe itself decreased in size to a radius of about 10 mm [15, p. 76].

The PiXel Detector (PXD), consisting of two layers of sensors at radii of 14 and 22 mm, thus sits very close to the beam line [15, p. 78]. Each sensor contains a large number of DEpleted Field Effect Transistor (DEPFET) pixels, for a total of eight million pixels. This particular type of sensor can be made very thin, which reduces multiple scattering in the detector material,

2. Experimental Setup

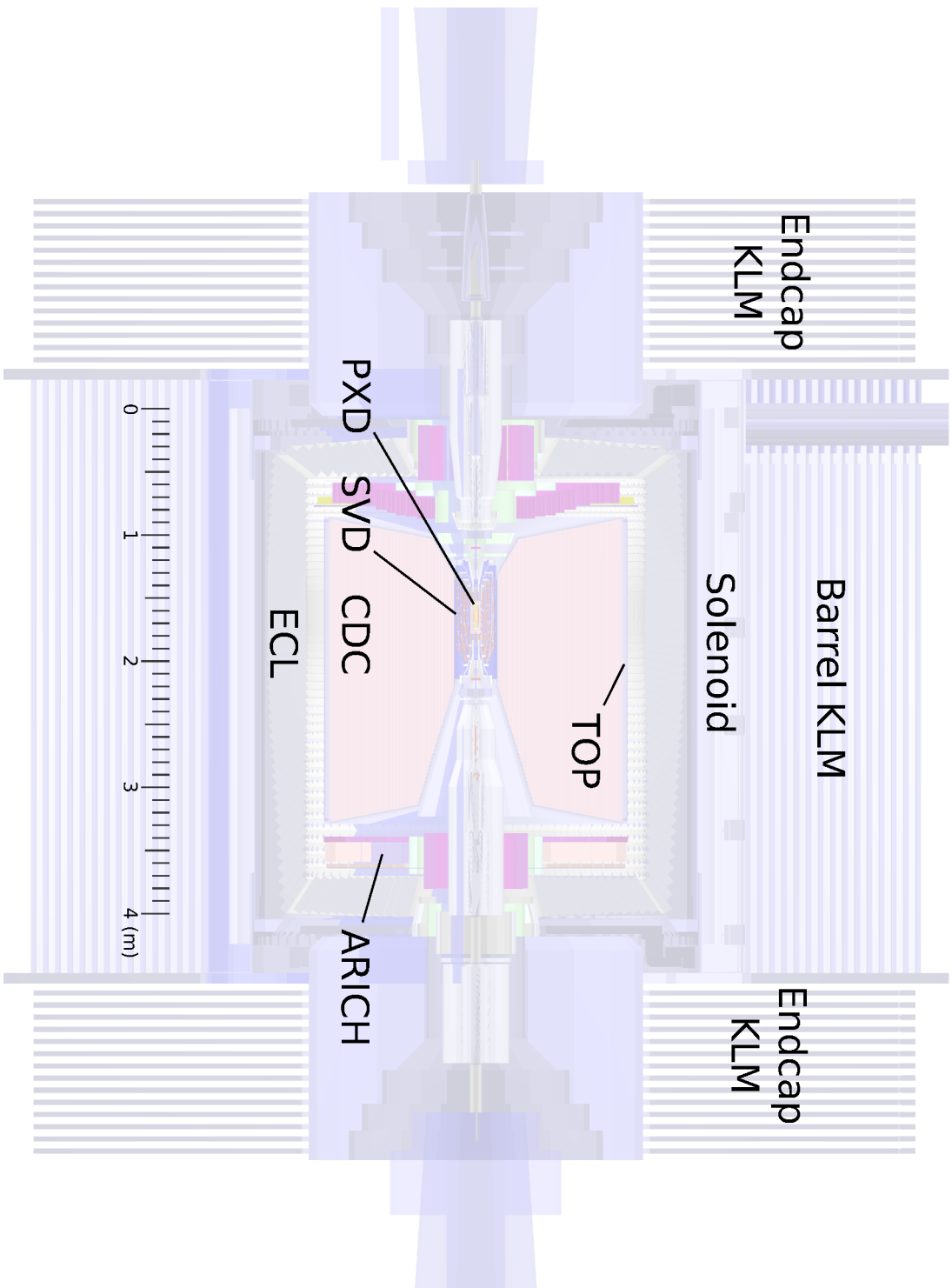


Figure 2.3.: Cross-section (side view) of the Belle II detector, using the geometry as implemented in BASF2 in May 2015.

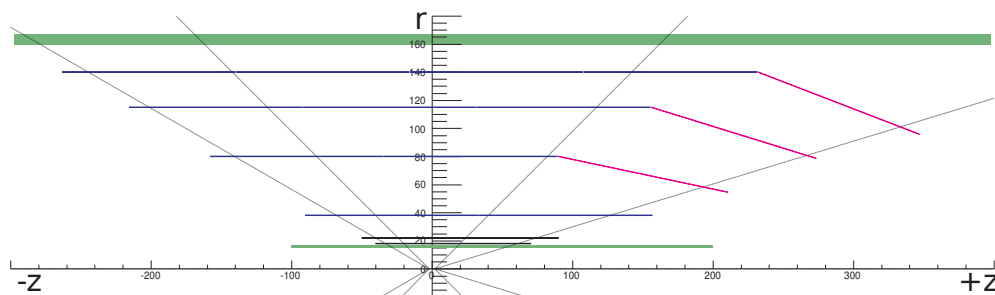


Figure 2.4.: Configuration (side view) of the four SVD layers, with the two PXD layers shown directly outside the beam pipe. All dimensions in mm. Adapted from [15, p. 142]

and does not need to be continually powered to hold the collected charge, reducing power consumption. As a result, the PXD can be cooled using CO_2 (melting point: -56.6°C [25]) flowing through channels integrated in the support structure and does not require, e.g., liquid nitrogen. Due to the large number of channels and its slow readout time of $20\ \mu\text{s}$, the volume of data generated by the PXD is very large and needs to be reduced before it can be saved (see also Section 2.3.5).

The PXD is surrounded by four additional layers of silicon strip sensors at radii of 38 to 140 mm, which make up the Silicon Vertex Detector (SVD) [15, p. 142]. The SVD consists of double-sided silicon strip detector sensors, where one side contains n-doped strips and the other side p-doped strips approximately perpendicular to the n -strips. This arrangement provides great position resolution for charged particles passing through it, while requiring far fewer readout channels than the PXD. Its faster readout time also makes the SVD much less vulnerable to beam-background hits, and allows performing a stand-alone track finding using only SVD data [26]. To decrease the amount of silicon needed to cover polar angles from 17° to 150° , slanted sensors are used in the forward region. The arrangement of the SVD layers around the PXD is illustrated in Figure 2.4.

The Central Drift Chamber of the Belle experiment is also replaced with an improved system. As Belle II's SVD is somewhat larger than at Belle, the inner radius of the CDC is increased to 160 mm. Since Belle II, however, does no longer contain the voluminous ACC system, the outer radius is also increased to 1130 mm [15, p. 202]. The new CDC contains 14 336 drift cells distributed in 56 layers, which are in turn grouped into superlayers. Superlayers alternate between axial and stereo wire configurations, meaning that wires in these layers are oriented either along the z axis, or at an angle to it. This can be used by tracking algorithms to also extract z information for tracks, which would not be possible using axial layers alone. With a higher luminosity, the amount of background originating from the beam is also expected to increase significantly. To compensate for this, the innermost layers of the CDC contain more cells, reducing the occupancy in the region most strongly affected by common backgrounds. The wire configuration in the CDC is illustrated in Figure 2.5, showing both the increased density for the inner layers, and alternating superlayers.

2. Experimental Setup

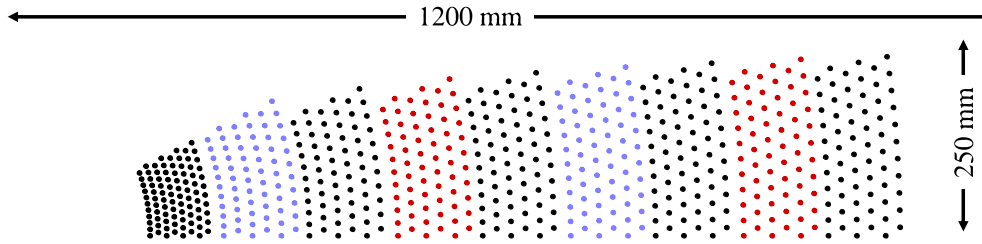


Figure 2.5.: Cross-section of the CDC wire configuration, with axial superlayers in black, and stereo superlayers in violet and red for positive and negative stereo angles, respectively. Taken from [21], based on [15, p. 204].

2.3.2. Particle Identification

The particle identification systems at Belle II saw a significant redesign which is expected to greatly improve their performance. Both TOF and ACC systems were replaced by entirely new systems: the Aerogel Ring-Imaging Cherenkov (ARICH) detector in forward direction, and the Time-Of-Propagation (TOP) counter in the barrel region.

While also using Cherenkov radiation like the ACC, the ARICH detector does not rely on threshold effects, but measures the Cherenkov angle for particles that pass it. Through $\cos\phi = 1/(n\beta)$ (with Cherenkov angle ϕ , index of refraction n , and β being the particle velocity relative to the speed of light in vacuum) this is equivalent to a velocity measurement and thus allows identification of particles when combined with a momentum measurement.

The time-of-propagation counter replaces Belle's TOF and consists of 16 quartz radiator bars with a thickness of 2 cm placed on the outer rim of the CDC. Its principle of operation is a combination of Cherenkov ring-imaging and time-of-flight measurement [15, p. 220]. A particle passing through one of the quartz bars with a velocity higher than the phase velocity of light in the medium produces Cherenkov radiation, which is internally reflected along the radiator bar until it reaches an array of photomultipliers at the end. The measured values are the arrival time of the Cherenkov photons (with a very precise time-resolution) and the spatial coordinates. Figure 2.6 shows an example distribution for two different types of particles with the same momentum. The structures seen are the result of the reflected Cherenkov cone photons being detected at a later time, with visible differences between the two distributions especially for higher values of t .

As was done at Belle, the energy depositions in the CDC are used for dE/dx particle identification, but the same procedure is also possible in the silicon detectors. Due to different physical effects for ionisation in silicon and gas, using energy depositions in the SVD for particle identification promises significant improvements for identifying low-momentum electrons [21]. As the θ acceptance of ARICH and TOP is limited, charged tracks leaving at the backward end of the CDC can only be identified using dE/dx (and, for electrons, the ECL).

2.3.3. Electromagnetic Calorimeter

The electromagnetic calorimeter is one of the components that are at least partly inherited from the Belle experiment. In this case, the crystals 8 736 scintillator crystals are kept, as the light output was not greatly reduced through radiation damage sustained during the run

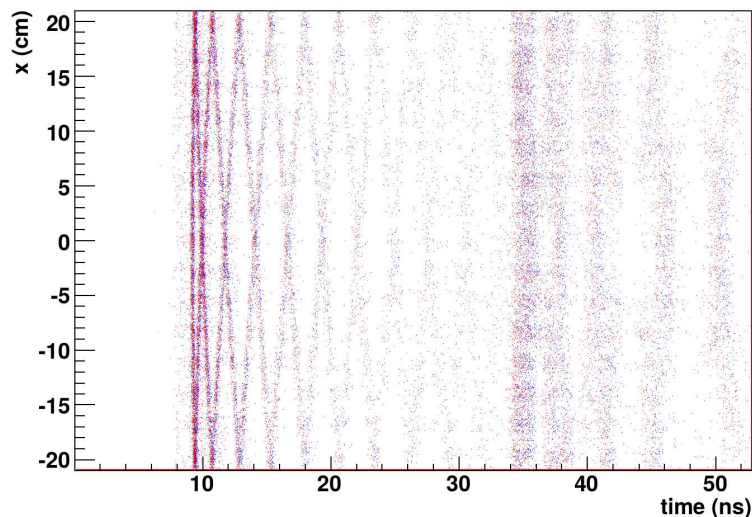


Figure 2.6.: Simulated example $x-t$ distributions for Cherenkov photons generated by a set of 500 pions (red) and kaons (blue) at 3 GeV and normal incidence, for a slightly different experimental setup with split quartz bars. A slight difference between both distributions is visible, in particular for times >35 ns. Taken from [15, p. 241].

time of KEKB. Even for the much higher luminosities and increased radiation dose expected with SuperKEKB, the crystals remain viable [15, p. 286f.]. The electronics, on the other hand, will be upgraded to provide a faster readout. For the end-cap, the possibility of switching to pure CsI crystals, which are much faster but have only 10 % of the light output of CsI(Tl) crystals, is being researched [27].

The ECL is also used to provide trigger signals for other detectors, and for the detection of K_L^0 mesons (in conjunction with the KLM).

2.3.4. K_L^0 and Muon Detection

Like the ECL, the KLM will only receive a partial upgrade. Here, the effects of background are more strongly felt due to the long dead time of the resistive plate chambers. To compensate for the increased background rates, the RPCs in high-rate arrays (in the entire end-caps and the inner two layers of the barrel region) are replaced with plastic scintillators [15, p. 313][28].

2.3.5. Data Acquisition

The data acquisition setup for Belle II makes accommodation for the high data rate of the PXD, which could not be stored without reduction. For this reason, pixel data is transferred to a buffer node, which decides which data should be kept through so-called regions of interest (ROIs) it receives. ROIs can be created by the high-level trigger (HLT), which processes data from SVD, CDC, and other detectors using the same software also used for offline reconstruction. After finding tracks using this data, they can then be extrapolated into the pixel detector, and regions of pixels likely to contain further track hits are sent as ROIs. The regions can also be generated by an FPGA-based track-finding algorithm running on the data

2. Experimental Setup

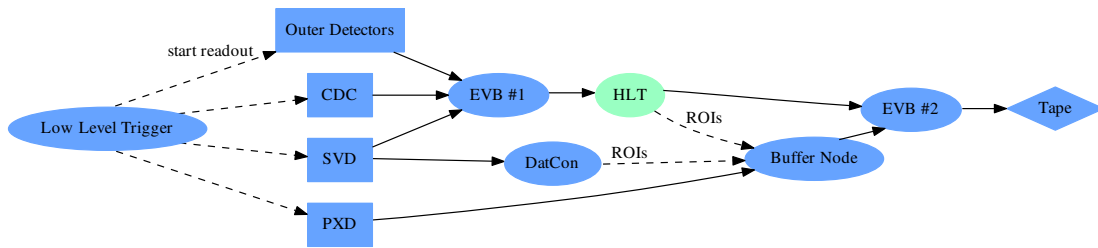


Figure 2.7.: Data acquisition and event-building procedure for the Belle II experiment. Adapted from [29].

concentrator (DatCon), which only uses SVD data [29]. After selecting regions of interest, the kept pixel data is sent to a second event-builder that combines PXD data and the output of the other detectors from the high-level trigger. The data flow of this two-step procedure is sketched in [Figure 2.7](#).

3. The Belle II Analysis Software Framework (BASF2)

The software framework for a particle physics experiment is responsible for supporting the development and execution of a variety of data-processing tasks, including event generation, detector simulation, tracking, physics analyses, monitoring and quality control. While these tasks might at first seem quite disjunct, they all face similar challenges in increased input/output (I/O) rates and data sizes with newer experiments. A common framework can help solve these problems, while also encouraging code reuse between different components. This chapter describes the software framework of the Belle II experiment, which was improved and extended significantly within the scope of this thesis. In particular, many of the elements discussed in the following sections were necessary for the development and implementation of the advanced analysis tools described in later chapters.

For the Belle experiment, the software was called Belle Analysis Framework (BASF) and provided a framework for analysis code written in C, C++ and Fortran [30]. It integrated central storage of data for communication between modules with a custom I/O format called Panther tables and also featured runtime configuration via steering files and dynamic linking of modules [31]. This modularity allowed its use for both offline and online applications: It managed both physics analyses and applications like data acquisition and high-level-trigger that receive data from the detector. The software however suffered from duplicate implementations, poor documentation, usability issues and limitations of the Panther format.

It was originally planned to retain full compatibility with BASF in the Belle II software [15, p. 438] and a first version of a framework called roobasf was developed, which was based upon BASF, but replaced Panther tables with an object-oriented ROOT I/O scheme [32]. Reasons for this incremental approach included a desire to profit from ‘a substantial accumulation of software experience’ and to ‘avoid reimplementing similar software’ [15, p. 438]. The volume of necessary modifications however led to the independent parallel development of a new software framework, BASF2, based on ideas from both its predecessor and other high-energy physics experiments [33]. After evaluation, it was chosen as the common software framework over its competitor roobasf. Given the aforementioned issues with the legacy code base (specifically those making maintenance more difficult), this intentional incompatibility allows for great improvements to software quality by enforcing higher standards for new code.

In Belle II, the same software framework is used for Monte Carlo simulation, data acquisition, online monitoring, high-level trigger, reconstruction, and physics analysis, which allows us to reuse the components that comprise the overlap of these areas. The software itself consists of independent modules that process an event in small steps, plus a core framework that is responsible for configuration, data exchange between modules and a number of features that allow using modules in more powerful ways. While this is a standard architecture for high-energy physics software frameworks, BASF2 also uses modules for central facilities

3. The Belle II Analysis Software Framework (BASF2)

like I/O. This provides additional flexibility since I/O modules do not necessarily need to be placed at the very beginning or end of an execution sequence. The system is written in C++11, a recent update to the C++ standard, which simplifies common programming tasks, specifically initialisation and resource management [34]. Besides the framework itself, the GNU Compiler Collection (GCC) and various libraries are also part of the software. These libraries provide many features that would otherwise need to be reimplemented, in particular, boost (a large set of libraries complementing the C++ standard template library) [35], ROOT (a data analysis framework with a focus on particle physics) [36], Geant4 (which simulates the interaction of particles through matter) [37], googletest (a unit-testing framework) [38] and SCons (an advanced build system) [39] are widely used throughout BASF2. This is in contrast to the Belle software, which in many cases relies on its own implementations of algorithms/facilities that are already available in common libraries.

3.1. Python and Packages

BASF2 is configured using *steering files* written in Python, in which the user can create modules and module paths, which are simple linear collections (chains) of modules that define their execution order. Modules can be configured via parameters identified by a unique name and supporting a wide range of types (including lists and tuples). This is demonstrated in the following listing:

```
from basf2 import *
main = create_path()

eventinfosetter = main.add_module('EventInfoSetter')
eventinfosetter.param('evtNumList', [15])

main.add_module('Progress')

process(main)
```

This example first loads the BASF2 interface, importing it into the global namespace. The steering file then creates a new path called `main` and adds the module `EventInfoSetter` to it, which is responsible for setting the event, run and experiment numbers that uniquely identify each event. (Run numbers are usually incremented when detector conditions change significantly; experiment numbers provide a more coarse-grained grouping.) By setting the parameter `'evtNumList'`, the module will generate a given number of events (in this case 15) in each run/experiment combination. The values for run and experiment numbers can be set via other parameters. By not setting them explicitly, they will be set from default values that can be defined for each parameter. This module is followed by the `Progress` module, which prints the number of events already processed to standard output. The last line, calling `process()` on the module path, starts the execution of modules in the path.

One can also use the entire spectrum of functionality provided by the Python language and associated libraries to integrate more complex programs with BASF2. This is for example

used to create tests (to execute modules and validate the result) and high-level reconstruction tools like flavour tagging and the Full Event Interpretation (see [Chapter 6](#)).

BASF2 is grouped into different organisational units called packages. Noteworthy examples include:

framework contains the core libraries used to create, configure, and arrange modules, the Python interface, as well as database and file I/O functionality. It is discussed in the following sections. The package also contains the `EventInfoSetter` and `Progress` modules from the previous example.

generators is comprised of Monte Carlo event generators, which form the starting point of any detector simulation. E.g. the `EvtGenInput` module is commonly used to create decays of $\Upsilon(4S)$ mesons with realistic branching ratios.

simulation contains modules to simulate the interaction of a given set of particles with the detector.

tracking uses simulated or real detector data from PXD, SVD and CDC to find and reconstruct tracks for charged particles.

mdst defines the data structures that contain necessary information for physics analyses. mDST is short for mini Data Summary Tables (see [Section 3.4](#)).

display provides a graphical user interface (GUI) to visualize detector geometry, simulated and real detector data and reconstructed objects. It is discussed in [Section 3.7](#).

analysis contains tools for physics analyses which are introduced in [Chapter 4](#).

3.2. Modules and Data Exchange

Modules are usually developed in C++ and are linked into shared libraries that are dynamically loaded at runtime if requested. In some cases, modules can also be written in Python (even defined in the steering file itself), which might be useful when ease of development is more important than runtime performance. In both cases, they inherit from the base class `Module`, which defines a common interface, and then implement certain methods that will get called during event processing. This includes `initialize()` and `terminate()`, which are called at the beginning and end, respectively, of execution and are responsible for allocating and freeing resources. Most modules perform their tasks inside the `event()` method, which is called for each event handed to the module and can access the current event's data. Finally, `beginRun()` and `endRun()` are called when the run number changes and allow the module to react to changes in detector conditions by, e.g. updating its internal state accordingly. [Figure 3.1](#) illustrates the order in which methods are called. The same sequence can also be interpreted as applying to a single module, where each block equals a single function call, or to an entire path, where a block represents calling the same method on each module in the order defined by the path.

As mentioned previously, modules can be configured using parameters. They are defined in the constructor of the `Module`-derived class and consist of a string identifier, a description

3. The Belle II Analysis Software Framework (BASF2)

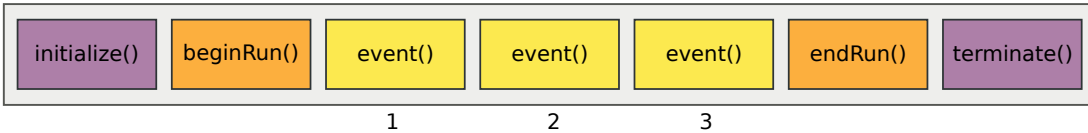


Figure 3.1.: Illustration of the execution sequence of module methods for a single run with three events, with time progressing from left to right.

explaining its function and possible values, a member variable that should be made available through this interface, and an optional default value. This is handled through a function that is templated on the member variable type and is able to convert the type to Python, e.g. `list(int)` for `std::vector<int>`. The actual type conversion is handled by Boost.Python, a library that provides interoperability between the C++ and Python languages. Type-safety, to some extent, is provided by throwing an exception if a parameter value cannot be converted to the underlying C++ type.

Communication between modules is handled via the *data store*, which allows storage and retrieval of almost arbitrary objects or arrays of objects. This concept defines a data-based interface between modules, encouraging an abstraction level focused on concrete objects like detector hits, tracks and particle identification likelihoods.

Many data are the direct product of some input data, and it can be useful to be able to trace these connections at some later point in time. To this end, any two objects in arrays in the data store can be linked via *relations*, which encode a directed relation with an optional floating-point weight. They are stored separately from the objects they connect, so they can be added without changes to the existing class structure. Relations are in most cases used to signify connections specific to the type of objects they connect, e.g. ‘created-a’ for a `MCParticle`→`SimHit` relation, or ‘reconstructed-from’ for `MCParticle`→`Track`. Classes stored in arrays inherit from an interface class that provides an easy way for users to retrieve objects (and associated weights) related with any given object or create a new relation between two objects. Accesses through this interface are done via a bi-directional index structure which provides fast lookups with a runtime of $\mathcal{O}(\log n)$, where n denotes the number of relations between two arrays. The bi-directional index also allows looking up relations in the reverse direction with no performance penalty, which would not easily be possible with pointers stored inside the object pointed from.

Any object or array in the data store is uniquely identified by the combination of a name and a durability. The durability also controls the data’s lifetime, with current options being one event only (e.g. Tracks), or persistent through the entire execution of BASF2 (e.g. histograms containing a certain variable distribution cumulated over all events). Users can access data via templated accessor classes that attach to it and behave similar to a smart pointer or container class, for objects and arrays, respectively. The following listing shows a brief example of their use.

```

StoreObjPtr<EventMetaData> eventmetadata;
if(eventmetadata)
    B2INFO("Currently in event: " << eventmetadata->getEvent());

StoreArray<CDCSimHit> cdcsimhits;
//loop over all CDC simhits
for(const CDCSimHit& hit : cdcsimhits) {
    // Use hit's data here...
}

```

Python implementations of these classes are also provided to make data available to modules written in Python; their usage is analogous to the C++ case. Actually using objects in the Data Store from Python (e.g. calling member functions) requires the class definition to be available to the Python interpreter in some form. In BASF2, this is provided by PyROOT, a ROOT interface to Python that relies on automatically generated information about classes. These auto-generated definitions are also used by the I/O system (see [Section 3.4](#)). All classes saved in the data store (and thus being I/O compatible) can in this way also be used from within Python for, e.g., prototyping or tests.

Modules can also declare their inputs and outputs. (This is optional for inputs, but required for outputs.) This information can be used by the framework to generate a graph of the dependencies. Specifying required inputs can also help users to notice problems within their steering file, since a required input that does not have a matching output in a previous module will cause the execution to stop with an appropriate message. For each module, visual representations of these graphs are automatically generated and integrated into the class documentation. [Figure 3.2](#) shows such a graph for the `MdstPID` module, responsible for storing particle identification information from different detectors in an array of `PIDLikelihood` objects. These objects are created for each charged track, and are saved into `mDST` files for later use by analysis modules. It distinguishes between `Tracks`, which are a required input produced by tracking modules, and optional inputs from different detectors (which may or may not be available). Similar graphs can also be produced for entire steering files.

3.3. Path Control Flow

3.3.1. Module Conditions

Besides simple linear execution, the framework also supports changing the control flow on a per-event basis. One possibility is to branch into a different path when a specified condition is met: setting a condition via, e.g., `module.if_value('<0', conditionPath)` will cause event processing to continue inside a given path (`conditionPath`) if the condition is true; it is checked against an integer-valued return code that can be set in the module. For convenience, aliases `if_true()` and `if_false()` are provided to check for values ≥ 1 and < 1 , respectively.

3. The Belle II Analysis Software Framework (BASF2)

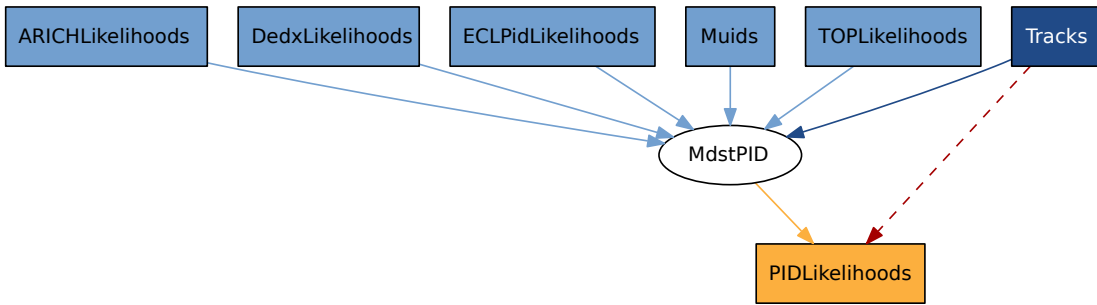


Figure 3.2.: Graph of inputs and outputs of the MdstPID module. Required inputs are shown in dark blue, optional inputs in light blue and outputs in orange. The dashed red line denotes an (output) relation between arrays.

After a condition branch is taken, execution usually stops after the condition path for the current event. Via an optional argument to `if_value()`, `if_true()`, etc., this can also be changed to continuing execution after the module that the condition was set on, i.e. modules following the condition module are executed regardless of whether the conditional path is executed or not. This feature is particularly useful for creating skims of larger data samples or selecting events for quality monitoring purposes. Skim refers to selecting events according to certain physics criteria and discarding all other events, which can easily be achieved by adding an output module (see [Section 3.4](#)) in the condition path.

3.3.2. Sub-event Iteration

Generally, a module's `event()` method is called once per event while any repeated processing is performed inside this call by e.g. looping over an array of input data. For more complex tasks, such as modifying the behaviour of a chain of modules according to each entry in an array, this is no longer sufficient, or would lead to overly convoluted implementations. To remedy this, the framework supports sub-dividing events for a certain path by using `outerPath.for_each(loopObjectName, arrayName, loopPath)` in the steering file. This has the effect of calling the `event()` method of modules in `loopPath` for each entry in the array identified by `arrayName`. For each run through `loopPath`, an object stored under the name `loopObjectName` will contain the array entry for this iteration. Modules can modify stored data to share it with other modules, but objects/arrays of event durability are reset after each iteration through the path and thus are not visible to following modules in `outerPath`.

Using the functionality introduced in this section, it becomes possible to perform reconstruction tasks repeatedly on different parts of the event. Mainly this is performed through `RestOfEvent` objects: after reconstructing a signal decay candidate, it is often useful to look at the event again while excluding all final state particles used for the signal candidate. This can be done by creating `RestOfEvent` objects for all signal candidates – these encode which final state particles have not been used and thus are still available – and then using `for_each()` to iterate over them and perform further reconstruction tasks on only these remaining tracks or clusters.

The following example shows the structure for one of the main use cases: building tag-side B^0 candidates for a set of existing signal candidates ('`B0:signal`').

```
# create signal B0 candidates... (omitted)

buildRestOfEvent('B0:signal', path=path)

roePath = create_path()
# add reconstruction of tag side to roePath... (omitted)

path.for_each('RestOfEvent', 'RestOfEvents', roePath)
```

Using this feature will in most cases also require some way of specifying the loop object to the modules and allowing them to act on this information. Inside `roePath`, the `isInRestOfEvent` variable can be used to select final state particles that are part of the current loop iteration's `RestOfEvent` object. (Variables and the high-level reconstruction tools are introduced in [Chapter 4](#).)

3.4. Input/Output

Data shared via the data store can also be written to file using ROOT's serialisation functionality, by adding input or output modules at the desired position. This allows separating the module chain between any two modules, to e.g. perform simulation and reconstruction in separate processes to test certain reconstruction steps on the same input data.

Certain standard sets of data have been defined and are frequently referred to — the terminology mostly follows Belle conventions:

DST Data Summary Table files contain the output of reconstruction, including detailed tracking information and detector hits (about 300 kB / Event)

mDST mini-DST files contain only a subset of objects in DST files and reduce them to the information necessary to perform most analyses: Tracks, particle identification likelihoods, calorimeter clusters, Monte Carlo particles and important meta data. (about 40 kB / Event)

μ DST micro-DST files are *skimmed* mDST files with analysis-level particles added. The name might appear misleading, given the larger number of objects in this file, but it reflects only the reduced file size from discarding irrelevant events. (Size depends strongly on the type of skim, but might be orders of magnitude below mDST size.) See also [Section 4.5](#).

In many cases, files in these formats will be produced by the responsible group and made available to collaboration members.

In most cases, functions implementing the serialisation and deserialisation of objects to or from a byte stream are generated automatically using ROOT's C/C++ interpreter (currently `rootcint`). This frees users from the potentially error-prone task of writing their own (de)serialisation functions and keeping them up-to-date with every change to the data members of a C++ class. The serialised objects are stored as separate branches in `TTree` objects [\[40\]](#)

3. The Belle II Analysis Software Framework (BASF2)

in a ROOT file, which provides optimized read performance and allows quick navigation between events or accessing only some parts of the event. It also allows for transparent file compression (to about 40 % of the uncompressed size) and backward compatibility even in case of complex changes to class layouts [41].

All ROOT files saved by the output module contain additional metadata in a persistent `FileMetaData` object, which provides information on the file's contents and the environment used to create it. Among other things, it contains the number of events, information on the run and experiment numbers contained in the file and the creation time, as well as the contents of the steering file, the host-name and the random seed used to produce it. The object also includes a unique identifier (ID) for the file, and a list of the unique IDs of the files used as input when creating it (called *parent files*). In conjunction with a database mapping these IDs to the physical position of a file (e.g. a file name)¹, the `RootInput` module can use this information to load objects from both an input file and its parent files by setting the `parentLevel` parameter to a non-zero value. If the parent files can be assumed to remain available, this enables saving disk space, e.g. by allowing smaller μ DST files which contain only particles and refer to the parent mDST files for other objects. A different use-case might be *index files*, which are files that are practically empty, but serve only to define a certain sub-set of the parent file contents. Events in the file are uniquely identified by their metadata, and can be matched efficiently with their counterpart in the parent file through the use of `TTreeIndex` [42].

Besides the standard ROOT I/O modules, there are also modules to send and receive data over the network (for the data acquisition system) or exchange data between processes, which also rely on ROOT's serialisation mechanisms. Applications of the inter-process communication are the topic of the following section.

3.5. Parallel Processing

For over a decade prophets have voiced the contention that the organisation of a single computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers in such a manner as to permit cooperative solution. – *Gene Amdahl, 1967* [43]

This statement makes it plain that the trend towards parallelisation is certainly not new, and that the reasons for it have not changed much in fifty years. However, limitations to the scalability of sequential code have become more obvious recently. While transistor numbers per integrated circuit continue to follow Moore's law [44] by doubling roughly every two years, this does no longer translate directly into more instructions per second. Instead, CPU clock rates have stagnated due to increased cooling requirements associated with higher clock rates and manufacturers are focusing on (among other things) improved pipelining and instruction sets, as well as increasing the number of CPU cores.

Since computational problems in particle physics are *embarrassingly parallel* [45], i.e. computation is done on independent collision events, we can make use of this feature by having each core work on a different event, regardless of the individual actions performed

¹Currently an implementation using plain files instead of a database is available.

on them.² While one can separate the input data manually and simply start independent processes, on a single computer this will not result in optimal resource use. In concrete terms, independent BASF2 processes would use significantly more memory and repeat common initialisations needlessly, compared to the parallelised implementation introduced in the following.

BASF2 provides an implementation of multi-core processing that takes care of the partitioning and collection of events and allows for some resource-sharing. It is inspired by a similar feature in the Belle software framework which also shares a number of implementation details. To avoid the difficulties associated with ensuring all modules are programmed in a thread-safe manner, the mechanism uses the POSIX `fork()` call to create independent sub-processes after the common initialisation has been performed. Modules responsible for creating/reading events or writing to files cannot easily be run in parallel. BASF2 thus separates the module path into three sections: an input path, containing modules at the beginning of the job that can only be run sequentially, followed by the parallel path with those modules that set a flag asserting they are compatible with parallel execution (e.g. avoid writing to files) and the output path with the remaining modules, starting with the first one that does not set the parallel flag. After initialisation, these three paths are separated, with one process each for input and output path, and a user-specified number of worker processes for the parallel section. Communication between processes is performed by serialising the data store contents of one event using special modules and storing them in a ring buffer (used as a FIFO queue) in shared memory using System V interprocess communication (IPC) mechanisms [46]. This is illustrated in [Figure 3.3](#). The data flow between processes is shown in [Figure 3.4](#). Each buffer keeps track of the number of processes sending data to it, so that during `terminate()`, the process also detaches from the ring buffer it sends events to (not shown for clarity). This allows the subsequent processes to detect when no further events are available and the event processing should be stopped.

In a simplified model, the performance improvements of a partially parallelized program when compared with strictly sequential execution (the speedup S) is given by Amdahl's law

$$S(N) = \frac{1}{(1 - P) + P/N},$$

where P is the fraction of parallel code and N is the number of worker processes [43]. It is apparent that the time spent in the sequential part (here: input and output paths) is a lower limit for the runtime of the parallel version. In practise, the speedup is also influenced by a number of other factors, including communication overhead between processes. Since input and output paths are executed in independent processes, a realistic calculation would also need to take this into account, however as long as the execution time is dominated by the parallel path, they will mostly be in a blocked state (waiting for buffers) and thus can be neglected. This assumption can be verified by measuring the speedup of parallel processing with a single worker process (plus one input and one output process), which should be about one in this case.

²Note that this approach might not be feasible in all cases. For some of the LHC experiments, for example, the working set per event may grow to hundreds of megabytes. For advanced parallel architectures with e.g. 50 cores on a PCIe card, this severely restricts scalability. In that case, parallelisation between and/or inside modules becomes necessary.

3. The Belle II Analysis Software Framework (BASF2)

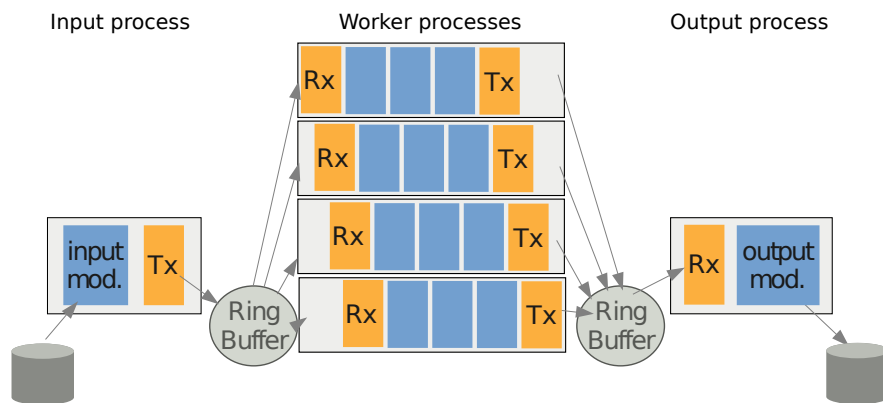


Figure 3.3.: Illustration of the parallel processing mechanism with serial sections in the input and output processes, and distribution of events to worker processes by splitting the original module path with the Rx and Tx modules. Based on [47].

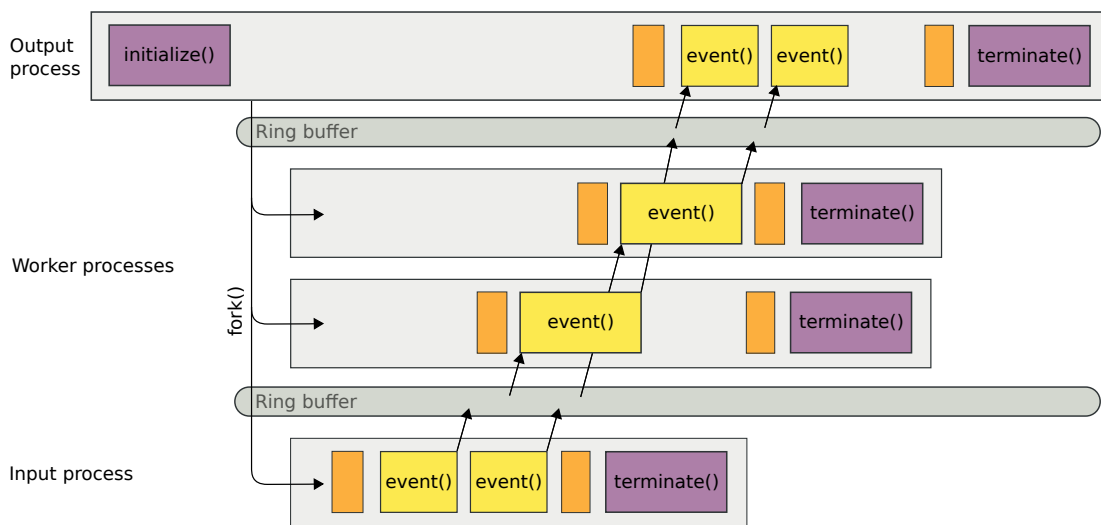


Figure 3.4.: Illustration of the execution sequence and data flow between processes when using parallel processing. Orange boxes denote `beginRun()` and `endRun()` calls. Cf. Figure 3.1.

Figure 3.5a shows the speedup with parallel processing by number of worker processes used on a machine with four physical cores; this increases to eight cores with HyperThreading, a technology that doubles the number of logical cores but not the number of execution engines or caches, avoiding complete stalls when one logical core is waiting for data. In these measurements, a larger number of events was generated, simulated and reconstructed using a standard set of modules.³ The fraction of parallel code P is estimated to be around 97.5%, which is used to produce an ideal curve according to Amdahl's law (dashed). Measured performance remains well below this ideal curve, only reaching a speedup of three for four worker processes, and saturating at around four when including the additional cores provided by HyperThreading. This might be caused by limited shared system resources like CPU caches, or by the additional system load due to the input and output processes. Using the additional logical cores provided by the CPU seems to be beneficial, and appears to agree with the 30% performance improvement through HyperThreading claimed by the manufacturer [48].

For comparison, **Figure 3.5b** graphs the speedup for the same task, but on an older system with 2×4 physical cores (16 with HyperThreading). Here, the measured performance follows the idealised estimate quite closely up to eight worker processes, even exceeding it in places. This might be an effect of the separate L3 caches for the two CPUs (8 MB each), compared to the single CPU from **Figure 3.5a** where the cache (of same size) is shared by all cores. But while scalability seems to be better, it should be noted that the absolute performance of the older processors is about 50% lower for the same number of processes.

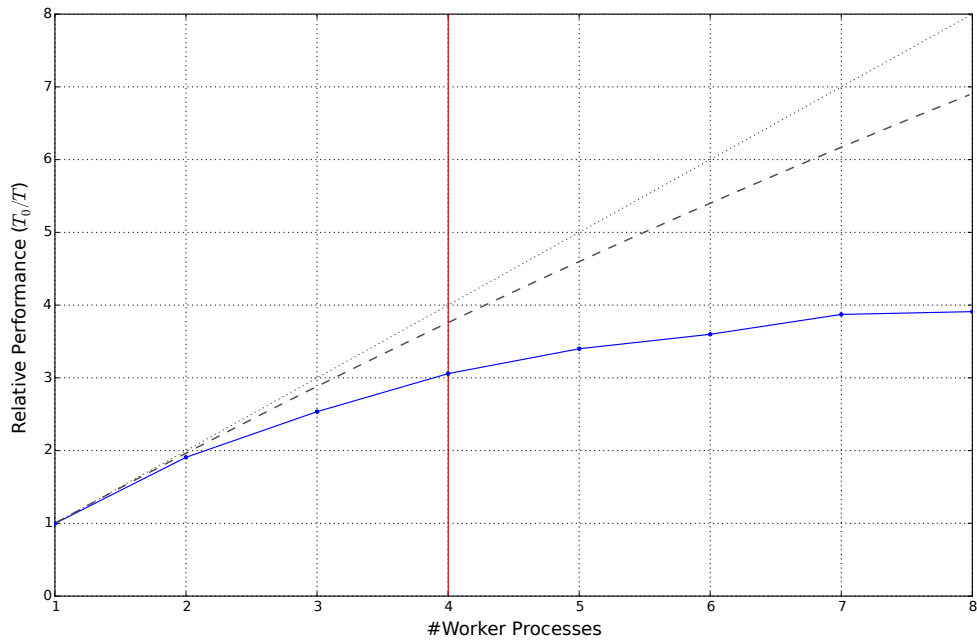
Processing a similar (if slightly different because of different random seeds) workload using separate BASF2 processes yields very similar speedups for a given number of BASF2 instances / worker processes. This suggests that the effects seen in **Figure 3.5** can for the most part be attributed to kernel scheduling algorithms and/or hardware, and that the communication overhead can be neglected for this particular workload. Due to the increased amount of work required from the user to manually split their workload into separate jobs and possibly merge output files at the end, the greater usability of the multi-core solution (which can be enabled by specifying the desired number of worker processes after the `-p` flag to `basf2`) makes it the preferred option.

One more important advantage of the parallel-processing over starting entirely separate processes is that it performs expensive initialisation only once. This is particularly important for those modules where the initialisation may include database accesses, and the geometry creation, which can take up a significant portion of the process' memory. Since on Linux `fork()` is implemented using copy-on-write pages [49], any memory allocated during initialisation will be shared between the processes until written to. Thus, for constant data like the detector geometry, only one copy needs to be kept in memory. Any memory written to after the `fork()` call will be copied into the process' own (un-shared) address space, so the processes cannot influence each other directly.

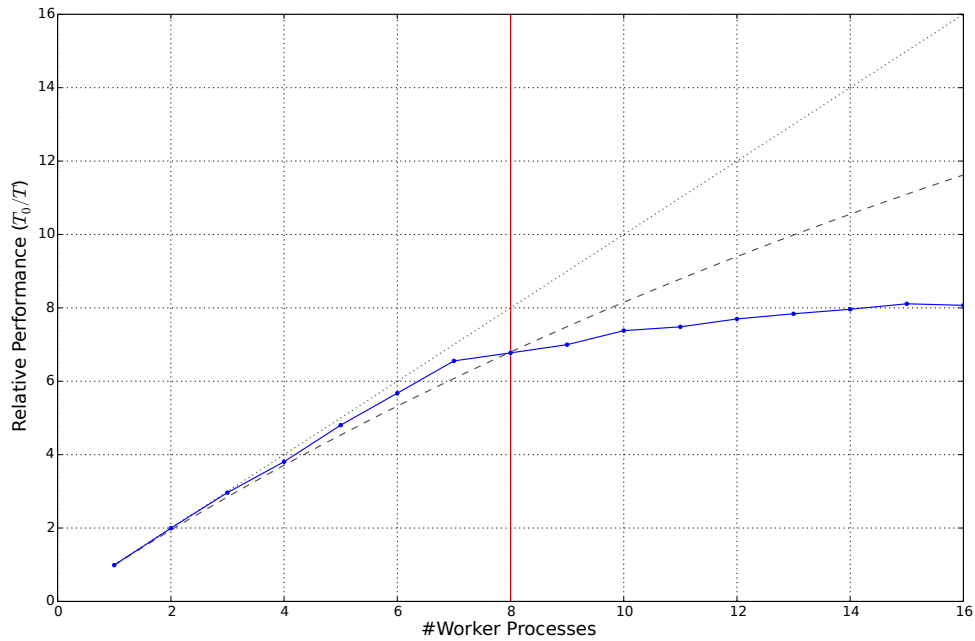
The resulting memory savings can be seen in **Figure 3.6a**, which shows memory usage over execution time when generating, simulating and reconstructing a fixed number of events, either via four independent processes started simultaneously or multi-core BASF2 with four worker processes. Besides using 2.5 GB less memory, the multi-core version is also faster: The initialisation is performed in a single process, which decreases load on memory and cache

³Workload was `reconstruction/examples/example.py`, with a fixed random seed to ensure reproducibility.

3. The Belle II Analysis Software Framework (BASF2)

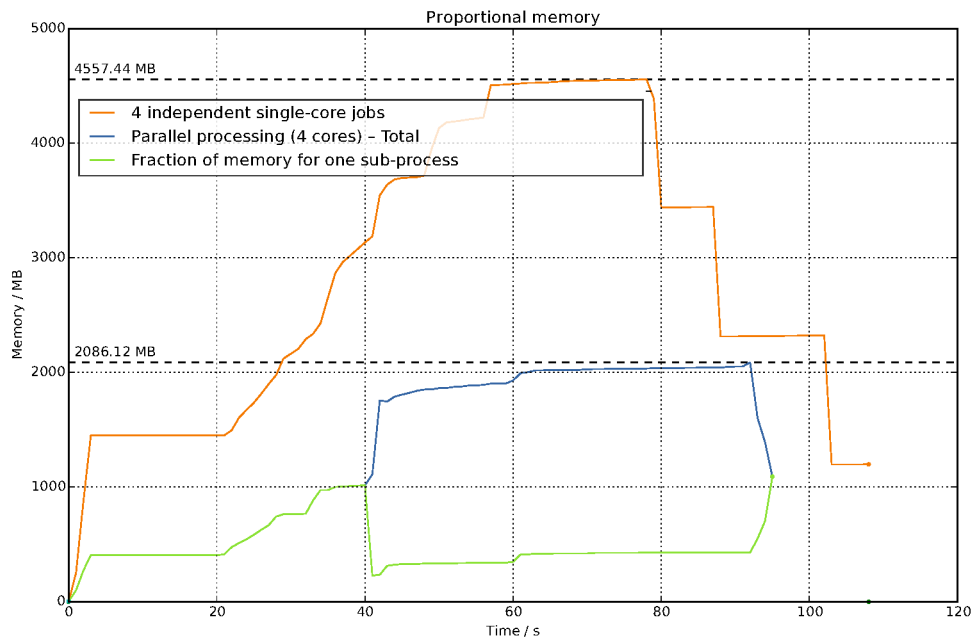


(a) Benchmark on a Core i7-4770 CPU with 4 cores (plus HyperThreading), using 5000 events.

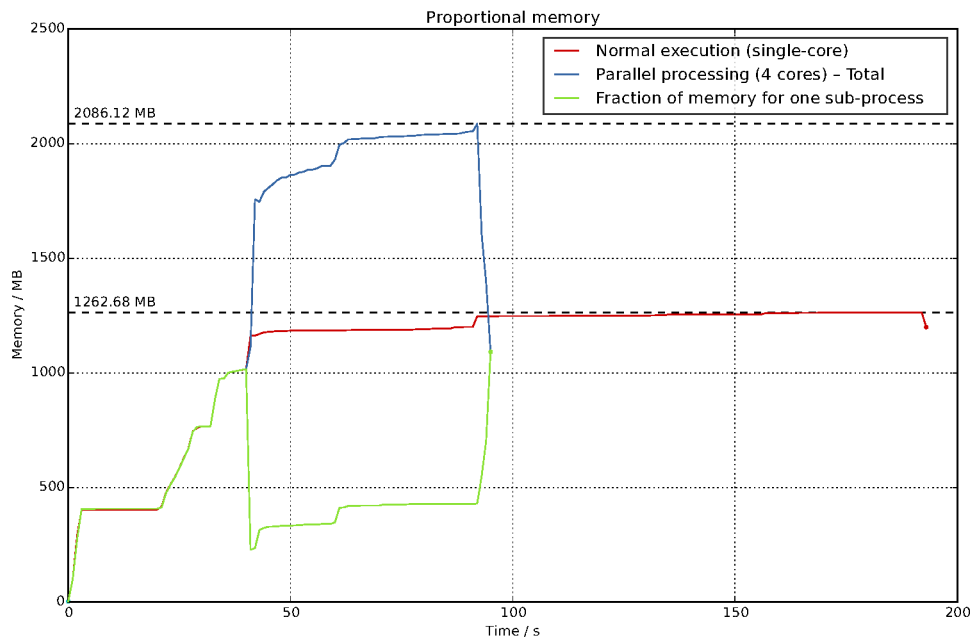


(b) Benchmark on 2 Xeon E5520 CPUs à 4 cores (plus HyperThreading), using 2500 events.

Figure 3.5.: Relative performance (speedup) when using parallel processing with different numbers of worker processes (blue), with ideal performance predicted by Amdahl's law for $P = 97.5\%$ (dashed) and $P = 100\%$ (dotted). The number of physical cores is denoted by the vertical red line. The runtime T_0 of a true single-core job is used as reference time. Events were generated using standard simulation+reconstruction, using SVN revision 16967.



(a) Comparison with four independent single-core jobs.



(b) Comparison with single process.

Figure 3.6.: Proportional memory (PSS) usage over time for BASF2 with parallel processing (blue, per process in green) vs. four independent single-core jobs ((a), orange) and a single process ((b), red). 100 events were generated using standard simulation+reconstruction, using SVN revision 16967.

and might explain the performance improvement compared to independent processes. This reduced memory footprint can be particularly useful for Grid computing, where the current 2 GB per core limit can easily become problematic.

It is important to note that the memory measurements show the sum of each process' proportional share of all associated memory segments (PSS), which differs from the resident memory that process monitoring programs like `top` usually show. The latter does not take shared pages into account and will in fact be slightly higher for multi-core jobs because of the added input and output processes. In particular, this can be a problem in job-scheduling software, which might forcibly abort multi-core jobs by acting upon strict limits on resident memory instead of the effective memory consumed. Support from the scheduling software is thus essential to make proper use of the parallel-processing mode.

Figure 3.6b also compares multi-core processing with an equivalent single-core job. During the initialisation phase (until $t = 40$ s) memory usage is almost identical, but rises afterwards by about 200 MB and 1 GB for single- and multi-core jobs, respectively. This indicates that some standard modules allocate additional memory immediately following initialisation, which suggests future opportunities for improvement.

Besides the expected dependence on a high fraction of compatible modules (though the compatibility requirements do not place strong constraints on them), the way a path is split into parallel and non-parallel sections can limit the performance gains. For example, a single non-parallel module in between parallel-compatible modules can force the second half into the (single-core) output process. Additionally, the serialisation necessary to transfer complex and possibly memory-allocating data structures between processes can become a serious bottleneck for some applications. This particularly affects physics analysis code, where tremendous numbers of candidate particles might be produced in each event, which would then need to be transferred to the output process unless further optimisation is implemented.

3.6. Merging Objects

Some objects whose durability extends beyond a single event contain information accumulated across multiple events. Examples include histograms of, e.g., the invariant mass of combined particle candidates over a data set and by-module CPU time and memory statistics (saved in `ProcessStatistics` objects). For a single process, this does not present a problem, since there is only a single histogram (or other object) where the appropriate data gets added.

However, when using parallel processing, multiple instances of the object exist. It might receive new data in any of the parallel and/or input and output processes, but needs to be in a consistent state again in the output process, where it can be written out. This is implemented via special handling for classes that implement the `Mergeable` interface and define how to merge two objects and how to remove all data. Each event passed to another process also contains a copy of these long-lived objects which is used to update the state of the process-local object: upon receiving a mergeable object, a process merges the data into its existing local object. After an object is transferred to another process, it is cleared to ensure that data added to the object only exists in one process at any point in time. Otherwise, the same contents (plus new ones) would be sent in the next event, and added once more. At the end of execution, the object residing in the output process has seen all data that was stored and can then be analysed or saved. The data flow is shown in more detail in **Figure 3.7**.

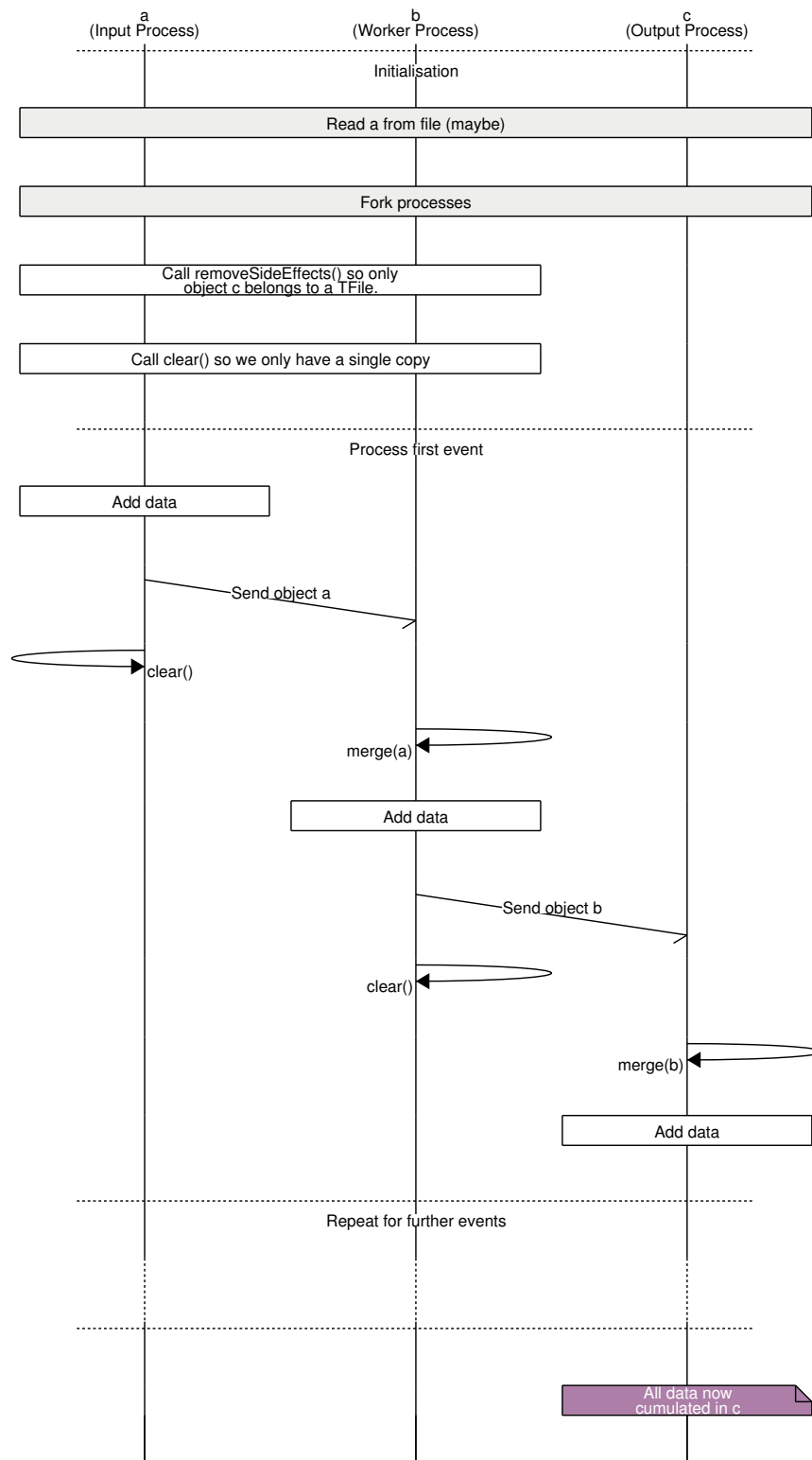


Figure 3.7.: Sequence chart showing the flow of Mergeable object data between BASF2 processes (in parallel processing mode).

3. The Belle II Analysis Software Framework (BASF2)

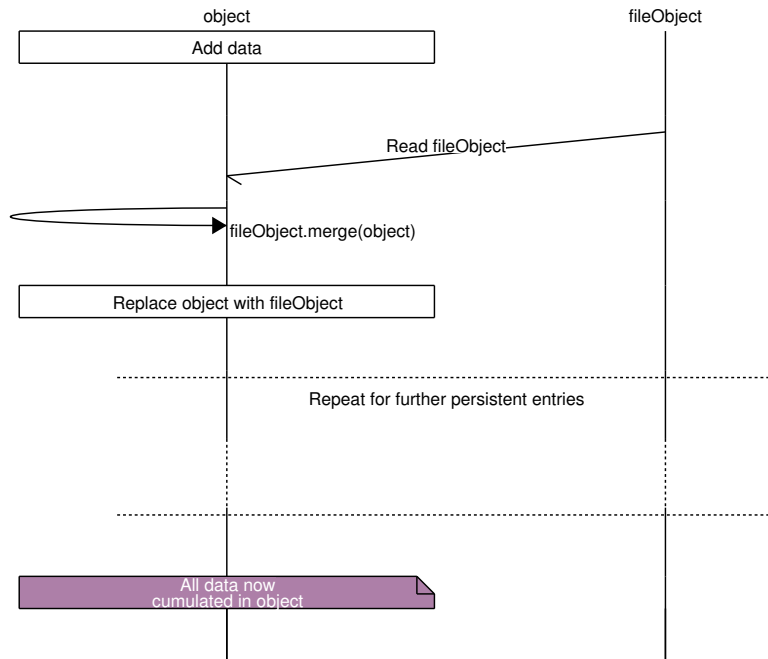


Figure 3.8.: Sequence chart showing the flow of `Mergeable` object data when reading from a file. The merging step is repeated for each entry of persistent durability in the input (typically when changing to another input file).

Since differential data is serialised and deserialised in each event, this might lead to problems for larger objects like multi-dimensional histograms. For `TTree` objects, however, incremental transfers might turn out to be an advantage, since the size of data added since the last event is likely to be significantly smaller than the cumulated size divided by the number of processes. Additionally, there is also a hard limit on the size of objects serialised via `ROOT` without the use of a `TTree`, which can be avoided this way.

The same system can also be used to merge objects read from files, so that the user can cumulate results produced in independent job executions, e.g. from processes generating Monte Carlo data on different computers, or processing different input files. In this case, the in-memory data is merged into the object read from file and replaced by it. Since the files were saved before starting the current job, this order ensures that the newer data is added to the older data (i.e. `oldObject.merge(newObject)`). This is important for some types of data, like per-module statistics, and analogous to the parallel-processing case. [Figure 3.8](#) illustrates how data read from file are merged with existing data. In the Full Event Interpretation (see [Chapter 6](#)) this functionality is used to collect aggregate statistics of CPU time spent in each channel and module.

A templated implementation is provided to allow merging of `ROOT` histograms and trees via `RootMergeable<T>`. `MapMergeable<T>` provides the same for `std::map` instances. The implementation deals with ownership-related issues that would otherwise arise, while allowing `TFile`-backed objects in the output process, so that e.g. accumulated data in a `TTree` does not need to remain in memory.

An alternative approach that was already available previously does not store objects in the

Table 3.1.: Performance comparison between Mergeable objects and the alternative file-based HistoManager approach when merging histograms of the specified size over 10 000 events. For object sizes IEC prefixes are used, with Ki = 1024 and Mi = 1024².

| | Histogram bins | Size | Run time (m:s) |
|--------------|----------------|---------|----------------|
| Mergeable | 1 000 | 3.9 KiB | 0:45 |
| | 10 000 | 39 KiB | 0:46 |
| | 100 000 | 390 KiB | 0:60 |
| | 1 000 000 | 3.8 MiB | 4:17 |
| HistoManager | 1 000 | 3.9 KiB | 0:43 |
| | 1 000 000 | 3.8 MiB | 0:44 |

data store, but provides a separate interface to add histograms and TTree objects. During execution the objects in different processes belong to different files, which are merged into a single file after the job finishes (similar to the hadd tool). This avoids the aforementioned bandwidth problems with large objects. The merged data are however not available to the job itself and are necessarily stored in a single file. It is also limited to predefined ROOT classes and requires a special HistoManager module to be added to the path.

Table 3.1 shows a comparison of run times when filling a histogram over 10 000 events using multiple processes and merging it using the two approaches. With increasing object sizes the effect of having to transfer Mergeable objects between processes in each event becomes more apparent. Performance decreases significantly once object size becomes comparable to the total event size (around a few 100 KiB in this case). The HistoManager approach on the other hand only touches the objects once at the end of the job, and incurs only a limited performance penalty with larger sizes. This makes it the better choice for histograms with very fine binning or high dimensions, but for histograms of commonly used sizes, TTrees (where only differences are transferred), or objects that require a user-defined merging procedure, Mergeable objects have clear advantages.

3.7. Event Display

In most disciplines, visualisation is one of the foremost tools in verifying that some component behaves as expected or, if it does not, in understanding how it misbehaves. In particle physics this often involves aggregated information, e.g. in the form of histograms showing the distribution of a value in a data set, but often a more fundamental view is also helpful. The event, i.e. one real or simulated collision, serves as a natural unit for visualisation. Event displays are also useful to provide illustrations for physics results, even if most results are statistical inferences over aggregated data rather than interpretations of a single event. Moreover, science outreach also benefits from having otherwise invisible collisions visualised in an aesthetically pleasing manner.

In BASF2, this visualisation is implemented inside a module, but can also be started using a stand-alone executable that opens a given file. For its functionality it mainly draws upon

3. The Belle II Analysis Software Framework (BASF2)

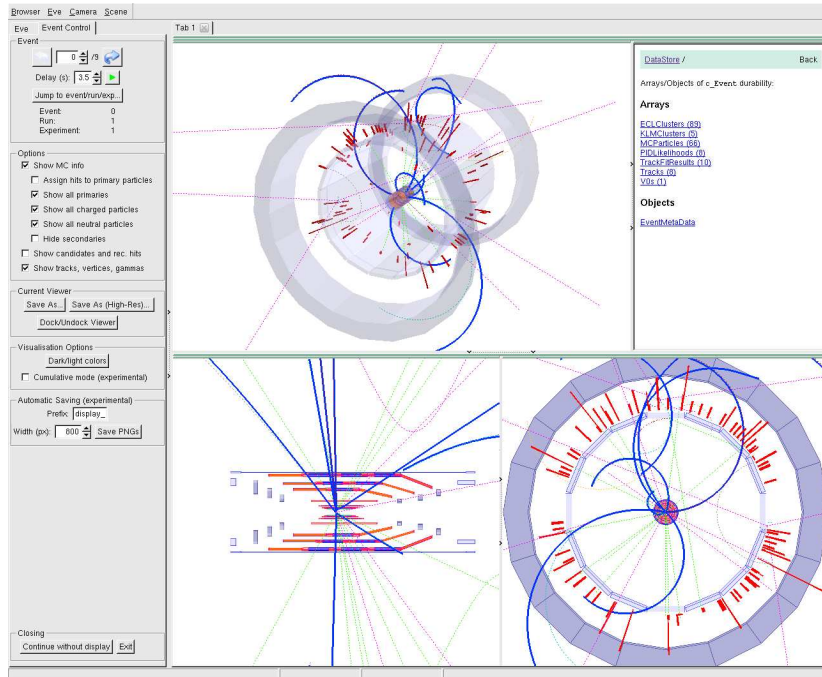


Figure 3.9.: Screenshot of the Belle II event display in default configuration (with light background), showing MC particles (thin lines), fitted tracks (thick lines), calorimeter deposits (red bars), and a simplified version of the Belle II detector geometry.

the Eve toolkit, an event visualisation framework that is part of ROOT, and provides OpenGL-based three-dimensional rendering of generic shapes as well as high-level interfaces for particle physics concepts like tracks and calorimeters [50]. It also directly supports geometry visualisation for TGeo-based geometry definitions, which can be converted from Geant4 geometry data (the native geometry format used in BASF2) automatically using the VGM (Virtual Geometry Model) library [51].

The application's main window consists of different elements. The visualisations are available in a three-dimensional view as well as two-dimensional projections into the $r-\phi$ and $\rho-z$ planes. Each element's relative size can be adjusted, and it is also possible to 'undock' elements into their own separate windows to e.g. show a full screen view. The left side of the window is taken up by event navigation and configuration, which can be used to jump to a different event in the file(s) currently opened (if any), and to configure which objects to visualise. By controlling the RootInput module, the display can navigate to the next or previous entry in the file, or jump directly to a specific event, run or experiment.

Since the Belle II geometry is quite detailed, visualising it in its entirety is rarely needed and can be quite taxing on graphics hardware. Because of this, a simplified subset of geometry components is shown by default that only includes the KLM boundaries, TOP, and the vertex detectors to provide some reference scale for event-level visualisations (see Figure 3.9). Users can however enable the full geometry and use its hierarchical organisation to limit the level of detail according to their requirements.

Event-level objects are organized into three groups:

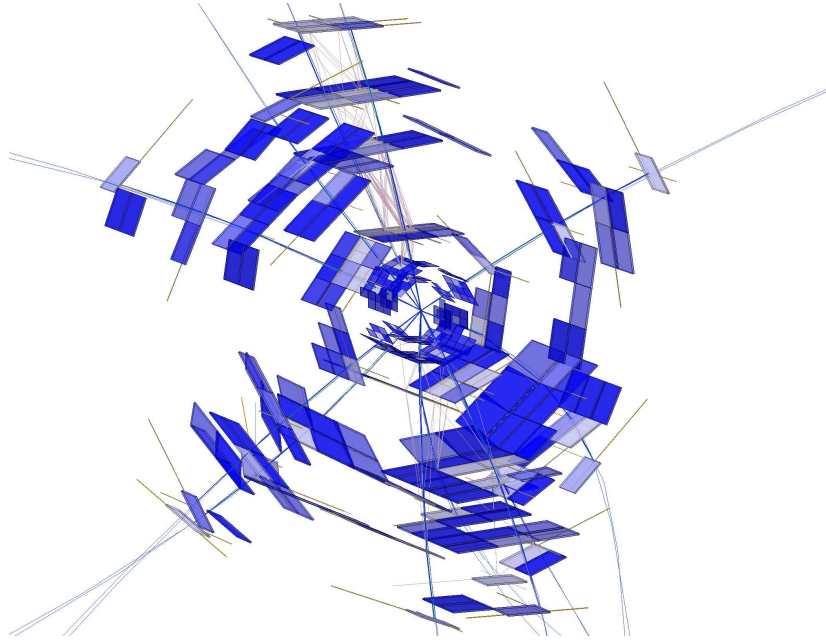


Figure 3.10.: Screenshot of the Belle II event display showing custom visualisations for the VXD track finder, showing sectors (shades of blue) considered for finding further hits and cells (lavender) used by a cellular automaton to separate track candidates (thin blue lines) with overlapping sectors. Image provided courtesy of Jakob Lettenbichler.

Monte Carlo information i.e. generated Monte Carlo particles and simulated detector hits,

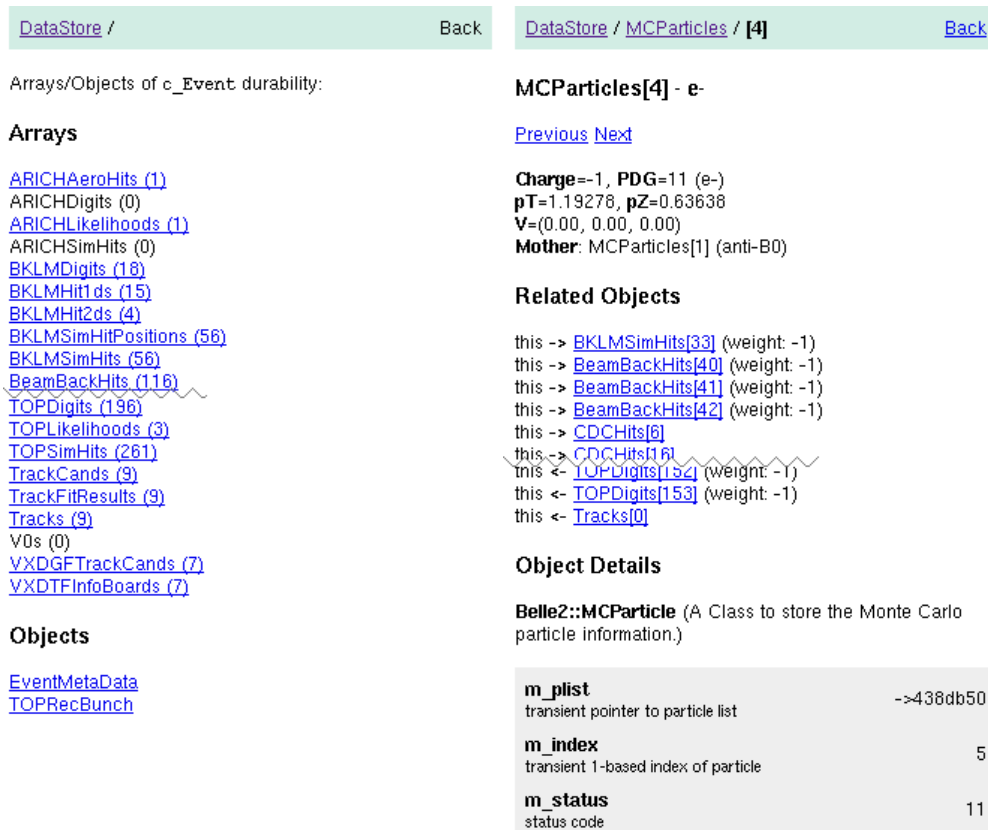
raw data and intermediate objects including track candidates and hits in the tracking detectors (which are not used directly in most analyses), and

reconstructed objects i.e. objects that would also be available on data, including fitted tracks, vertices and calorimeter clusters.

This grouping allows users to easily overlay the true generated information with that reconstructed to evaluate, e.g. clustering or tracking algorithms. For some use-cases, custom visualisations are provided for certain objects – an example for this is shown in [Figure 3.10](#), which shows such visualisations as used by the stand-alone track finder for the vertex detectors [26]. Users can also manually add some simple objects to the visualised scene, e.g. sets of points, arrows or text labels.

Special consideration has been given to also visualize the connection between objects to some degree: clicking an object will also highlight all (visualised) objects associated with it via relations (see [Section 3.2](#)). For a given reconstructed track, a user might for example check whether its trajectory matches that of the Monte Carlo particle which produced it (associated to the track via a relation). A pop-up shown when hovering the mouse pointer over an object also shows important physical properties, e.g. the p value and number of hits for track.

3. The Belle II Analysis Software Framework (BASF2)



(a) Main view of the data store browser. (b) Detailed view of electron generated by Monte Carlo in the data store browser, with custom information, list of related, and list of member variables.

Figure 3.11.: Screenshots of different data store browser views. Due to the length of the output, some information is omitted.

3.7.1. Data Store Browser

While the display itself also provides some information on objects that goes beyond pure visualisation, it also integrates a generic object information panel that can provide details on any object in the data store. When the display is started, it occupies a portion of the right-hand side of the main window, and shows a list of all stored arrays and objects in the current event (see Figure 3.11a). Clicking on an object – top-level or inside an array – opens a view similar to Figure 3.11b. It consists of the object's identifier (MCParticles[4]) – denoting its (unique) position in the data store – a short name (e^-), additional information (e.g. on charge, momentum and production vertex position), as well as a list of all related objects. Finally, it contains a list of all class member variables, associated documentation if available, and their values in the current object. It is generated automatically using ROOT's introspection mechanism and handles both simple types and nested objects.

Both the short name and the additional information shown for each class can be custom-

```

Particles[6] - K+
Previous Next

collection=Particles
PDGCode=321 Charge=1 PDGMass=0.4937
flavorType=1 particleType=1
mdstIndex=6 arrayIndex=6 daughterIndices: (none)
mass=0.4937
momentum=(0.19, 0.19, -0.39) p=0.4742
position=(-0.03, 0.04, -0.50)
p-value of fit (if done): 0.2617
error matrix (px, py, pz, E, x, y, z):

|          |         |          |          |          |          |          |
|----------|---------|----------|----------|----------|----------|----------|
| 0.00092  | 0.00087 | -0.0018  | 0.0015   | 3.3e-05  | -3.5e-05 | 6.8e-05  |
| 0.00087  | 0.00084 | -0.0018  | 0.0015   | -4e-05   | 4.1e-05  | 6.9e-05  |
| -0.0018  | -0.0018 | 0.0037   | -0.0031  | 7.2e-06  | -7.5e-06 | -0.00036 |
| 0.0015   | 0.0015  | -0.0031  | 0.0026   | -5.5e-06 | 5.8e-06  | 0.00024  |
| 3.3e-05  | -4e-05  | 7.2e-06  | -5.5e-06 | 0.00041  | -0.00043 | -4.2e-06 |
| -3.5e-05 | 4.1e-05 | -7.5e-06 | 5.8e-06  | -0.00043 | 0.00045  | 4.4e-06  |
| 6.8e-05  | 6.9e-05 | -0.00036 | 0.00024  | -4.2e-06 | 4.4e-06  | 0.0029   |

extra info=( SignalProbability=0.9038 mcErrors=0 )

```

Figure 3.12.: Example output for a Particle object, showing customized content for both name and information. The colour-coding used for the covariance matrix immediately conveys its rough structure by showing the largest positive and negative deviations from zero (in red and cyan, respectively).

ized by its authors by implementing a specified interface in the class. Developers can use a subset of HTML (HyperText Markup Language) to format text, add colours or structure contents in tables. Some utility functions are provided to translate objects representing e.g. vectors or matrices (see Figure 3.12) and to translate HTML into plain text for terminal output.

Internally, an array or object is addressed via a Uniform Resource Identifier (URI) [52] which describes the hierarchical structure of its position in the data store. E.g. the URI `event:Particles/6` denotes the object at index 6 of the array named ‘Particles’ of event durability. This scheme can easily be extended to support additional durabilities, or objects located elsewhere (e.g. geometry or data base contents). While the data store browser is intended to complement the display by working within it, it can also be started separately and can be controlled using Python.

3.7.2. Online Event Display

Considering that the event display itself is also a BASF2 module, input data does not need to come from ROOT files; it can also be generated in the same process or be received over the network, simply by replacing the input module. This latter feature is used to provide an on-line visualisation for detector data, since both the data acquisition (DAQ) and high-level trigger (HLT) also use the same software framework and can easily make a subset of events available via network links. As receiving events over the network might incur longer wait times, an asynchronous version (`AsyncDisplay`) is available that runs in a separate process while the main BASF2 process containing all other modules is receiving

3. The Belle II Analysis Software Framework (BASF2)

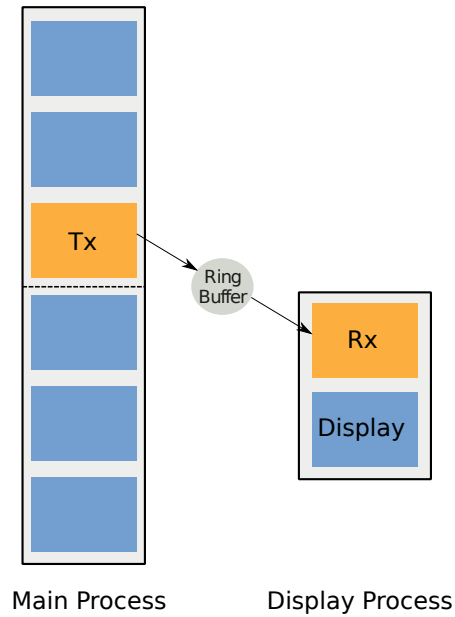


Figure 3.13.: Illustration of decoupling when using AsyncDisplay.

and processing data from the network. This is illustrated in [Figure 3.13](#). The implementation builds on some components used by the parallel processing feature (see [Section 3.5](#)), such as creating a new process only after initialisation and the shared memory communication that in this case provides a buffer for received events. This functionality is available using the `AsyncWrapper` class that can wrap any given module to decouple it from the main process. The current fill status of the shared memory buffer can be queried through the interface, which is used in the display's user interface to enable UI elements for advancing to the next event only when this will succeed immediately. To permit continuous monitoring of data, the user interface can also be used to advance to the next available event after a configurable time.

This online display was first used successfully during a combined beam test of PXD and SVD sensors at DESY in early 2014 [53]. Besides testing of the sensors themselves, the readout chain of both sensors and their combination was also checked for the first time, where strip data from the four SVD sensors was used to create and fit track candidates, which then could be extrapolated to the position of the two PXD sensors to select groups of active pixels (clusters) to save. Visualisation was provided for the geometry, the sensor data (pixels and strips), track candidates and tracks, as well as the Regions Of Interest (ROI) that define the likely position of interesting clusters on the pixel sensor planes (see [Figure 3.14](#)).

[Figure 3.15](#) shows an event recorded during the test beam run, with a reconstructed track passing through the six sensor layers under investigation, as well as its position at an equal number of telescope sensors, which can be used to validate the reconstruction.

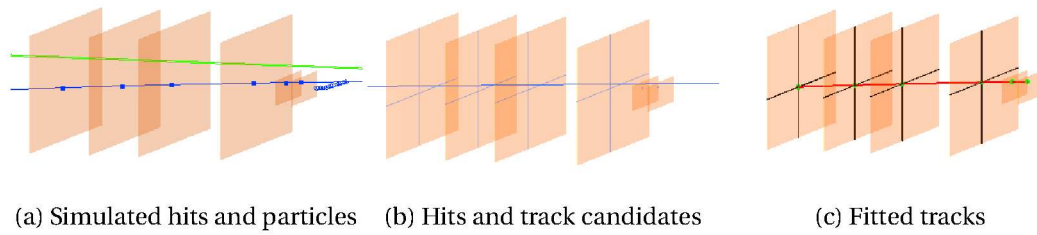


Figure 3.14.: Overview of different levels of visualisation using a simplified version of the DESY test beam geometry with 4 SVD and two 2 PXD sensors. The Monte Carlo information shown in (a) contains multiple **photons** and **electrons**. The hits reconstructed along the primary electron track are shown in (b), plus a track candidate with an approximate direction. Part (c) then shows the corresponding fitted **track** with accurate momentum information.

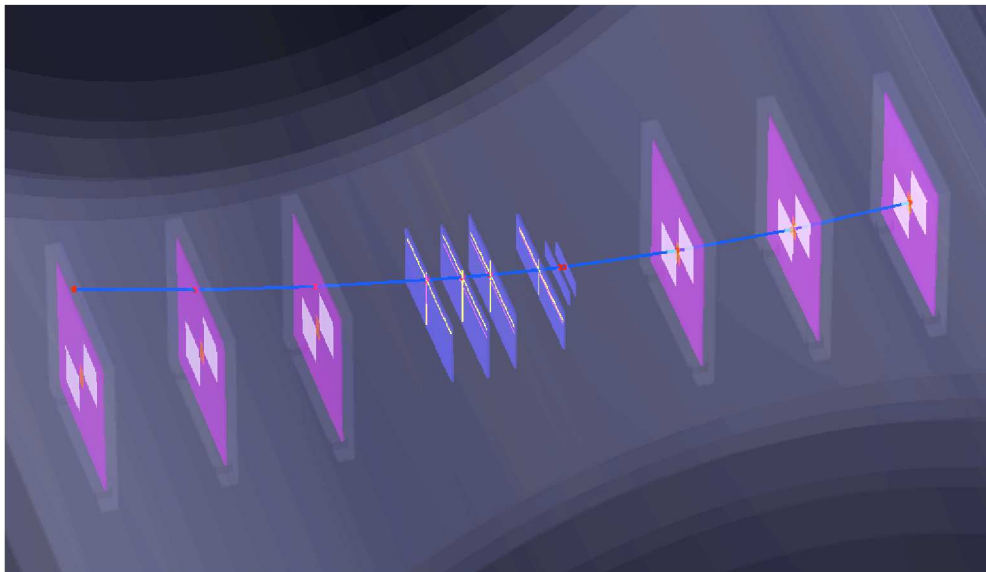


Figure 3.15.: Test beam data event with hits in all PXD and SVD sensors and a reconstructed track extrapolated to the 6 EUDET telescope planes. Adapted from [53].

3.8. Summary and Conclusions

As described in this chapter, the Belle II Analysis Software Framework was significantly improved and extended within the scope of this thesis to provide the capabilities necessary for data taking, Monte Carlo generation and analysis. This included additions to the interface to allow prototyping of modules in Python and an overhaul of the parallel processing feature so that it can now be enabled easily to increase resource-utilisation with any steering file. The I/O modules were also developed further and enhanced with a generic system for merging objects from different files or processes. Through the creation of the event display, users are supported in debugging, outreach and other tasks that benefit from visualisation. The software quality of the framework was increased as well, with unit tests now covering most of the core features and interfaces that are hard to misuse.

In conclusion, BASF2 provides a stable framework for further development of the software, and provides features that make it easy to create and use advanced analysis tools for the Belle II experiment.

4. Analysis Tools

Gathering data with a particle physics experiment is not an end in itself, but is done to enable physics analyses, which measure certain physical parameters like branching fractions that can be compared to theory predictions. To this end, each analysis needs to have a way of transforming the reconstructed detector response (like tracks or ECL depositions) into something that has a physical meaning, like decays or particles, so that measurements can be extracted. Typically, this involves a number of steps to inclusively or exclusively reconstruct certain decay channels. This process is based on and supported by the analysis tools (located in a package of the same name), which will be discussed in this chapter.

The analysis tools take full advantage of BASF2's modularity by encapsulating each of these reconstruction steps in a module acting on a common set of objects, and using Python to define a convenient and intuitive interface around them. The following listing demonstrates a simple analysis selection chain that is used to reconstruct $D^0 \rightarrow K^- \pi^+ \pi^0$ decays with π^0 going to two photons (with charge-conjugate decays being implied):

```
from basf2 import *
from modularAnalysis import *

main = create_path()
# load mDST file using RootInput module (see previous chapter)
inputMdst('myfile.root', path=main)

fillParticleList('K-', 'Kid > 0.1', path=main)
fillParticleList('pi+', 'piid > 0.1', path=main)
fillParticleList('gamma', '', path=main)

reconstructDecay('pi0 -> gamma gamma', '0.11 < M < 0.15', path=main)
fitVertex('pi0', conf_level=0.0, path=main)

reconstructDecay('D0 -> K- pi+ pi0', '1.8 < M < 1.9', path=main)
fitVertex('D0', conf_level=0.0, path=main)
matchMCTruth('D0', path=main)
# Use D0 candidates here...

process(main)
```

Even with no knowledge of BASF2 or the analysis tools, the Python code in the example is quite readable and should be mostly self-explanatory to the target audience, and involves (among other things) selecting final-state particles with cuts on particle identification outputs,

4. Analysis Tools

combinations of particles with invariant-mass cuts and vertex fits performed on the created candidates. The different functions and features shown, as well as their implementation, will be discussed in detail in the following sections.

4.1. Particle Candidates

Instances of the `Particle` class are the pivotal objects acted upon by the different analysis modules and represent candidate particles of a particular type (e.g. D^+ , D^- or γ). Besides this type information, a `Particle` object carries references to its daughters (if any), its three-momentum and mass, and its position, which is either the point-of-closest-approach (to the interaction point) for final-state particle candidates or the decay vertex for combined particle candidates. Additionally, a 7×7 error-matrix describing the error on four-momentum and position values, and in the case of a vertex or track fit, the p -value extracted from the χ^2 fit is saved.

This fixed information content aside, `Particle` objects can also store arbitrary floating-point data called *extra-info* with an identifying string. Particle candidates are likely to contain only a certain subset of keys and associated data, so sets of key strings are saved in a central object. This allows identifying extra-info data based on its position in an `std::vector<float>` alone, by mapping keys to positions in each key set. The ID of the used key set is also stored in the `Particle`. Storing n floating-point variables in a `Particle` thus only increases its size by $(n+1) \times 4$ Bytes compared to no variables being stored. An unoptimised implementation, with both strings and values stored in each `Particle`, would instead consume $\sum_i^n (4 + l_i)$ Bytes, where l_i denotes the length of the string identifier. For example, candidates with three extra-info fields with an average key length of $l = 19$ Bytes would require an additional $(69 - 16) = 53$ Bytes of space per candidate compared to the space-optimised case.

4.1.1. Lists of Particle Candidates

Since single particle candidates cannot usually be used to extract physical results, but rather are aggregated with lots of individual candidates of the same type, the `ParticleList` class was created to capture this usage. Instances of this class are saved in the data store under a name of the form 'particleType:label' and contain references to `Particle` objects stored in a central array (in the form of indices in this array). While the type of the stored candidates, e.g. 'D0' for D^0 or 'anti-D0' for $\overline{D^0}$, is required, the ':label' suffix is optional and can be used to distinguish different lists of candidate particles of the same type. Since in most cases, users want to automatically include charge-conjugated particles and decays in their workflow instead of duplicating all steps for both charges, particle lists are automatically created in pairs (e.g. 'D0:clean' and 'anti-D0:clean') which reference each other. Iterating over all particle candidates in a `ParticleList` thus loops over both particles and anti-particles, unless specified otherwise.

For some particle candidates, e.g. those combined in the decay $D^0 \rightarrow K^- K^+$, no information is available that determines their flavour quantum numbers (i.e. the set of quark-type-associated quantum numbers like strangeness, charm, etc.) of the produced candidates, so it not defined whether it should go in the particle or anti-particle list. This is solved by saving this third class of candidates in a separate structure that is duplicated over both

ParticleList objects, which is important when combining particle candidates (see [Section 4.1.4](#)). An iteration over all contents of a particle list will only include them once.

4.1.2. Variables

A common attribute of many reconstruction steps is the need to calculate certain quantities on Particle objects for a varied set of reasons, including applying cuts or saving n -tuples with physical quantities. In BASF2 this is handled through a central repository of *variables*, which are functions taking Particle objects and returning floating-point values. A key feature is that these functions are easy to create and register in the repository, as can be seen from the following listing:

```
double particleElectronId(const Particle* part)
{
    const PIDLikelihood* pid = part->getRelatedTo<PIDLikelihood>();
    if (!pid) return 0.5;

    return pid->getProbability(Const::electron, Const::pion);
}

VARIABLE_GROUP("PID");
REGISTER_VARIABLE("eid", particleElectronId, "electron (vs. pion) ←
    identification probability");
```

In this example a function to return a particle identification probability is defined and then registered with the name "eid" in the central repository using the provided REGISTER_VARIABLE preprocessor macro. A large number of these variables (e.g. for kinematics or PID information) are defined in a few central C++ source files. These are compiled into the analysis library, so they can then be used by any module that links to this library (see the following section for an example).

For certain tasks generalised variables are used, which carry additional parameters in their name that are used to modify their behaviour. One of the most commonly used variables of this form is "extraInfo(key)", which returns the floating-point value saved in a Particle's extra-info field (see [Section 4.1](#)), using the provided key string. Similarly, there are also variables that can perform relatively complex calculations which are guided by the argument, e.g. "daughterInvariantMass(i, j, ...)", which calculates the invariant mass of a subset of a candidate's daughters.

Even though all variables accept a const Particle* argument, some also accept a null pointer and are meant to return floating-point values that apply to the entire event. This is used, e.g. for determining the type of event (Monte Carlo or data), the total number of tracks in an event, or to get flags from the trigger system. As these variables simply ignore the Particle argument passed to them, they can also be used in conjunction with other non-event-based variables. For example, a user can save the kinematic information for a list of particle candidates, while also including the current event number ('evtNum') or the total number of tracks in the event ('nTracks').

4. Analysis Tools

Variables are also available in Python, which allows for simplified prototyping, and also provides an automatic listing of all available variables with the description string documenting their purpose. Using the `VARIABLE_GROUP` macro similar variables are put into visually separated groups, which simplifies finding a particular variable.

4.1.3. Creating Final-State Particle Candidates

Final-state particle candidates are created directly from outputs of the reconstruction (e.g. track finding and fitting, or the reconstruction of ECL and KLM outputs) using the `fillParticleList()` function (with the associated `ParticleLoader` module): Candidates for charged final-state particles like π^\pm , e^\pm , K^\pm , p^\pm , μ^\pm and d^\pm (deuterons) are produced from `Track` objects, which represent fitted tracks found in the tracking detectors (see [Chapter 2](#)). Similarly, clusters detected in the ECL and KLM are reconstructed as photons or K_L^0 , respectively, if no charged track has been associated with the cluster. Finally, two tracks forming a 'V' shape at some point away from the interaction region, which are reconstructed into a `V0` object using a special finder, are stored as a K_S^0 candidate. This assumes a decay of the K_S^0 into $\pi^+\pi^-$, which due to the typical flight length of the K_S^0 of a few centimeters tends to happen within or just outside of the silicon detectors.

Since charged tracks themselves offer no clear-cut way of distinguishing between the different hypotheses, lists of candidate pions, kaons, etc. created this way would all be created from the same track objects. Using the variables introduced in [Section 4.1.2](#), users can however improve the selection by applying cuts on the particle candidates created. For tracks, this typically involves a criterion based on the particle identification detectors, e.g. using "`Kid > 0.1`" to preferentially select true kaons using a combined likelihood ratio of dE/dx , TOP and ARICH (see [Chapter 2](#)). This criterion, or multiple criteria connected using boolean operators (and, or), can be added as an argument to `fillParticleList()` as shown in the listing at the beginning of this chapter.

With the exception of `fillParticleListFromMC()`, which can construct `Particle` objects from Monte Carlo particles instead of tracks or clusters, candidates for non-final-state particles are not usually created directly, but are built from other already existing particle candidates. The details of making these combinations will be discussed in the following section.

4.1.4. Combining Particle Candidates

Candidates for non-final state particles, which decay quickly and cannot be detected directly, are created by combining other particle candidates with each other. Typically this involves creating all possible combinations for a given set of potential daughters, and applying some criterion to determine which of the combined mother particle candidates to keep. This functionality is provided by the `reconstructDecay()` function (the associated module is called `ParticleCombiner`), which, as its first argument, takes a decay string defining both the particle to be combined and the previously created daughter particle lists that should be included in the combination. Decay strings start with the particle list name of the mother particle, followed by a "`->`" and a whitespace-separated list of daughter particle lists. A mother particle list (and associated anti-particle list) of the specified name is then automatically created by the module and used to store the combined particle candidates. E.g. for

combining D^0 candidates in the decay $D^0 \rightarrow K^- \pi^+ \pi^0$, one could specify "D0:mydecay \rightarrow $K^- \pi^+ \pi^0$:loose", in this case creating particle lists for D^0 and $\overline{D^0}$ with the label 'mydecay'. Daughters are specified using their particle list names, which makes it possible to e.g. use a specific list with a different selection, as done here for the π^0 list. As with other uses of ParticleList, the charge-conjugated decay is implied and also combined.

As making these combinations is a task that can potentially occur in other places, this functionality is internally provided by the ParticleCombiner library, which provides a generator that creates unique particle combinations as defined by a given decay string, and allows the user of the library to decide whether and where to store them. The generator takes care of both creating all possible combinations and keeping the combinations unique, by checking whether a combination was already created (e.g. with two π^+ daughters switched around), or whether a candidate uses two daughters that originate from the same reconstruction object (e.g. a π^+ and K^+ created from the same Track). It also takes care of assigning the produced candidates to the correct particle list (particle, anti-particle or unknown flavour) by setting the type information of the Particle correctly.

As with fillParticleList(), cuts using the variables can be given to reconstruct \leftarrow Decay(), which helps reduce the otherwise very large numbers of candidates. Commonly, a cut on the invariant mass might be added, e.g. "1.8 < M < 1.9" for a D^0 decay channel to limit its mass to values between 1.8 and 1.9 GeV (GeV is the default unit in basf2 for energies and momenta). It is also possible to add a decay-mode identifier to the reconstructDecay() call, which is an integer that will be saved in each created Particle object in an extra-info field with the key 'decayModeID'. When reconstructing a certain type of particle in many decay modes, this can be useful to quickly identify the decay mode used for each particle candidate, e.g. to evaluate differences in the kinematics for candidates reconstructed in different channels.

4.2. Vertex Fitting

Performing a vertex fit on a particle candidate refers to extrapolating the flight path of its daughters back to a common origin and, if successful, yields information on where the mother decayed. This can be used for lifetime-measurements, e.g. for CP -violation measurements of $B^0/\overline{B^0}$ mesons, or for rejecting incorrectly combined candidates by adding a cut on the vertex fit quality. Fits can be performed either as plain vertex fits, which take into account only the momentum and position information of the daughters, or as *kinematic fits* with additional constraints. A widely-used instance of constrained fits is the mass-constrained fit, which performs a vertex fit but adjusts the daughter kinematics in such a way that the invariant mass of the fitted particle candidates is constrained to the nominal mass.

In BASF2, fitVertex() performs a vertex fit of a specified type on all candidates in the given particle list (the default, fitType='vertex', performs a vertex fit without mass-constraint). As the quality of the vertex fit can be judged by the p -value calculated from the fit χ^2 value and the number of degrees of freedom, and since it can be used as an indicator for whether or not the assumed vertex actually exists, candidates with a p -value below a given threshold can also be removed. By default, only candidates where the fit did not converge are discarded. Besides performing a mass-constrained fit, one can also add a Gaussian constraint that enforces a vertex position near the interaction region (IP-tube constraint). This requires

4. Analysis Tools

supplementary information on where the beams actually collide, which may vary during the operation of the experiment. The nano-beam scheme introduced in [Section 2.1](#) with its (in the x - y plane) very narrow interaction region, however, makes this a very powerful constraint.

Two competing vertex-fitting algorithms are available: The KFit kinematic vertex fitter ported from Belle [54, p. 136] and the RAVE (Reconstruction in an Abstract, Versatile Environment) toolkit [55]. By default, the more powerful RAVE is used; in particular as it has improved support for constraints other than mass-constraints.

4.3. Monte Carlo Matching

When preparing an analysis, Monte Carlo simulations are a practically indispensable ingredient in optimising the selection and making sure the procedure developed can also be applied on data. This might involve taking the output of the analysis procedure and checking the amount of signal decays remaining after the selection. Similarly, the Monte Carlo information might be used to analyse the contribution of certain background decays and their behaviour. Mostly, simulated data contains the same objects as reconstructed data recorded by the detector. Monte Carlo samples, however, also contain `MCParticle` objects, which are created by either the event generator (e.g. EvtGen [56] for simulating B decays) or in interactions with the detector (through Geant4). These are often referred to as *primary* and *secondary* Monte Carlo particles, respectively. `MCParticle` objects are also connected to reconstruction outputs like `Track` or `ECLCluster` objects via relations, which are created by, e.g. tracking algorithms, to convey that a certain Monte Carlo particle's detector signals were used in the reconstruction of the track or cluster. For developing an analysis, this alone is not sufficient, as the `Particle` objects acted upon by the analysis modules in many cases represent unstable particles, i.e. those for which no relation to a reconstructed detector outputs exists (which would be the case with final-state particle candidates).

The objective of Monte Carlo matching is to provide these missing links by setting relations between `Particle` and `MCParticle` objects, as well as providing an indication of whether the combination is correct. For final-state particle candidates, creating the relation only involves copying it from the reconstruction object; the association between candidate and Monte Carlo particle is correct when their types match (e.g. a π^+ `Particle` is correct if it is related to an `MCParticle` that is also a π^+). For combined particles the process is more complicated and will be discussed in the following.

The key concept for setting relations for combined particle candidates is that of gathering all Monte Carlo particles that are associated with the final-state particle candidates used in the combination, and searching for the lowest common mother of these particles. In case of a correct reconstruction, the associated Monte Carlo particle will have the same type and decay chain as the reconstructed particle candidate. An example of this is shown in [Figure 4.1](#). This algorithm is implemented in the `MCMatching` module, which can be used with the corresponding `matchMCTruth()` function.

It should be noted, however, that the type will also be identical in case some final-state particle candidates are switched around, misidentified or not reconstructed at all. In fact, there are a number of ways a combination can be incorrect that is not directly captured by only looking at the Monte Carlo particle of a reconstructed candidate. To deal with

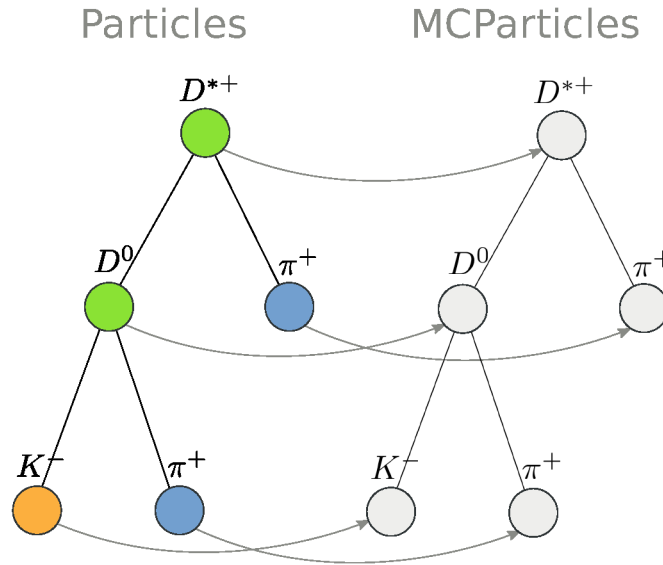


Figure 4.1.: Particle \rightarrow Monte Carlo particle relations for a correctly reconstructed $D^{*+} \rightarrow D^0(\rightarrow K^- \pi^+) \pi^+$ decay after applying `matchMCTruth()`: The tree of particle candidates is identical to a sub-tree of Monte Carlo particles, and each relation points to a particle object of the same type.

this problem, the Monte Carlo matching algorithm provides error bit flags through the `'mcErrors'` variable, with flags being cached in an extra-info field in the Particle. As each analysis user can accept different sets of flags, this allows for a custom definition of correctness. Table 4.1 lists all flags that can be set for a particle candidate.

An important feature of the flags is that they can be propagated from daughters to mothers, which ensures that errors in the combination of a candidate are visible in the flags of all its mothers and grandmothers, and allows the algorithm to avoid expensive reiterations over their daughters. The flags indicating a missing particle (i.e. a particle daughter present in Monte Carlo but not used in the combination) are not propagated; instead these flags always take into account the entire decay chain.

Figure 4.2 exemplifies a possible misreconstruction of a D^{*+} decay, where this fact is already quite clear by looking at the type of the associated Monte Carlo particle (in this case, a B^0). All candidates on the left-hand side of the tree of reconstructed particles will get one or more error flags: The π^+ has the `c_MisID` flag, since it was created from a track produced by a kaon; this flag is also propagated to the D^0 candidate, which additionally receives `c_AddedWrongParticle` (since the associated Monte Carlo particle has a different flavour and the *reconstructed* decay allows its determination) plus `c_MissingResonance` and `c_MissGamma` due to the missing π^0 daughter of the D^0 . It should be noted that the `c_AddedWrongParticle` flag is set when a non-final-state particle candidate is related to a wrong Monte Carlo Particle; this can indicate that there is indeed an additional wrong daughter, but the flag is also set for other misreconstructions. The D^{*+} , finally, has all the flags of the D^0 , plus `c_MissMassiveParticle` due to the simulated π^- grand-daughter of the B^0 not being included in the D^{*+} candidate's tree of daughters.

4. Analysis Tools

Table 4.1.: List of Monte Carlo matching flags (MCMatching::MCErrorsFlags). A Particle generally has either `c_Correct` (=0) or the bitwise OR of other flags set.

| Flag | Description |
|------------------------------------|---|
| <code>c_Correct</code> | This Particle and all its daughters are perfectly reconstructed. |
| <code>c_MissFSR</code> | A Final-State Radiation (FSR) photon is not reconstructed. Currently the distinction from <code>c_MissGamma</code> is based only on the number of other daughters and may not be accurate. (In the future, information that identifies FSR photons in the Monte Carlo can be used instead.) |
| <code>c_MissingResonance</code> | The associated MCParticle decay contained additional non-final-state particles (e.g. a ρ) that were not reconstructed. This is probably acceptable in most cases. |
| <code>c_DecayInFlight</code> | A Particle was reconstructed from the secondary decay product of the actual particle. For example, a pion producing a secondary electron might have been missed, and the electron (with a possibly very different momentum) used to create a candidate. |
| <code>c_MissNeutrino</code> | A neutrino is missing (not reconstructed). |
| <code>c_MissGamma</code> | A photon (not FSR) is missing (not reconstructed). |
| <code>c_MissMassiveParticle</code> | A generated massive FSP is missing (not reconstructed). |
| <code>c_MissKlong</code> | A K_L^0 is missing (not reconstructed). This flag only occurs in combination with <code>c_MissMassiveParticle</code> . |
| <code>c_MisID</code> | One of the charged final-state particles is misidentified. |
| <code>c_AddedWrongParticle</code> | A non-FSP Particle does not match its associated MCParticle. If the reconstruction is almost correct, this means that one of the daughters (or their daughters) belongs to another Particle; it is however also likely to just be a random combination of tracks. |
| <code>c_InternalError</code> | There was an error in MC matching. Not a valid match. Might indicate fake/background track or cluster. |

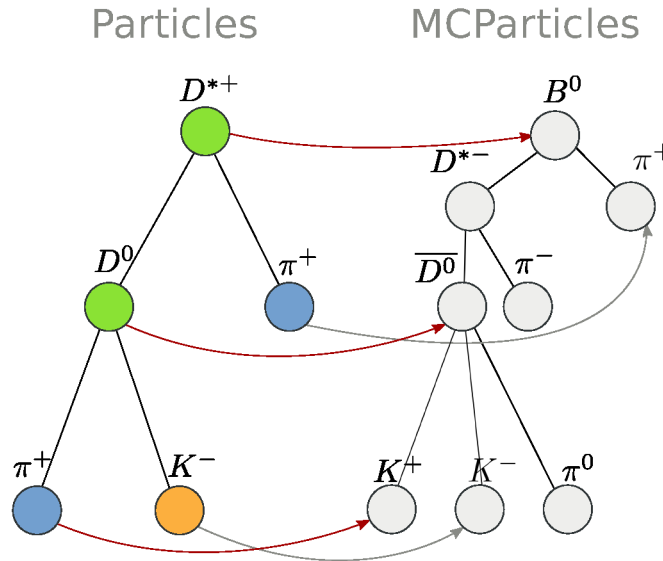


Figure 4.2.: Particle \rightarrow Monte Carlo particle relations for an incorrectly reconstructed $D^{*+} \rightarrow D^0(\rightarrow K^- \pi^+) \pi^+$ decay after applying `matchMCTruth()`, with relations to a particle candidate of different type marked in red.

Particle candidates very frequently receive the flags for missing final-state radiation photons, which are additional low-energetic photons radiated from decays, or missing resonance, where a non-final-state particle like ρ^0 , a_1 or even K^0 (a flavour-eigenstate precursor to $K_{S/L}^0$ that is added by EvtGen) is missing. Since this only rarely constitutes a bad particle candidate, a variable 'isSignal' was added that deems a candidate correctly reconstructed if it has no errors, but which ignores the `c_MissFSR` and `c_MissingResonance` flags. When reconstructing semileptonic decays (where the neutrino cannot be reconstructed), the variable 'isSignalAcceptMissingNeutrino' provides the same functionality while also ignoring missing neutrinos.

This implementation of Monte Carlo matching is a significant improvement over the previous state. The predecessor was also used at Belle to some extent, as one of multiple independently developed alternatives (see reference [57, p. 50] for a similar approach). The previous implementation also used relations to the lowest common ancestor, but had a less physically meaningful set of flags; flags also could not be combined or propagated. The new implementation contains unit tests for certain important corner cases that previously might have produced a false positive or negative. Some examples of cases that were not correctly handled previously (and are now covered by unit tests) include:

- Reconstructing a B^+ decay including $\pi^0 \rightarrow \gamma\gamma$, where one of the π^0 photons was missed and replaced with a photon originating directly from B^+ was labelled 'correct'.
- Decays in which all final-state particle candidates were correct and present, but which decayed over, e.g., a D^0 instead of a D^+ were labelled 'correct'. In a worst-case scenario, the same class of problem would accept an $\Upsilon(4S)$ as correct, even if candidates for final-state particles from both B mesons were switched around.

4. Analysis Tools

- Particles decaying in-flight, e.g. $\pi^+ \rightarrow \mu^+ \nu_\mu$, and reconstructed from the decay daughter (e.g. reconstructing π^+ from the μ^+ track) were not distinguished from a correct reconstruction. Similarly, candidates could be reconstructed from a track created by a secondary (which is likely to have very different kinematics) with no warning of this happening.

A disadvantage of this approach is that it still lacks granularity for some misreconstructions. For example, in many cases, reconstructing a decay with a low-energetic wrong photon does not greatly affect the kinematics of the candidate. In particular, the invariant mass of the candidate is usually not adversely affected by the assignment of a low-energy photon. Since the resulting candidates are not generally distinguishable from signal in the mass-spectrum, marking them as incorrect could prove problematic. As this also happens quite frequently, a modified Monte Carlo matching procedure that ignores photons in the assignment of a lowest common ancestor is available. The 'energyFromWrongPhotons' variable can be used to fine-adjust the total energy of these wrongly-assigned photons that the user is willing to accept for their signal-definition. In the future, an improved matching procedure that relates particle candidates to the most similar overlapping decay chain in Monte Carlo could provide a generalised solution.

4.4. Multivariate Classification

For practically all analyses, optimising the selection of signal entities in a list of candidates is a central task. In the simplest case, this can be done by applying cuts on one or maybe a set of variables, e.g. on the invariant mass of a particle. The effect of performing such a cut is commonly broken down into its *purity*, which is defined as the relative amount of signal entities in the sample after the selection, and its *efficiency*, which is the number of signal entities after the cut divided by the number of signal entities before the cut. For large amounts of background, however, the separation gained through rectangular cuts like these may not yield a purity and efficiency that is high enough to produce a competitive measurement. Multivariate classifiers, on the other hand, which can consider more than a single variable at a time, can account for correlations between two or more variables and thus can provide very powerful separation. This topic is closely related to machine learning, and in fact in almost all cases, classifiers are trained automatically using mechanisms that guide the classifier toward its optimal state for the given circumstances.

The multivariate classifiers in BASF2 are provided through its `TMVAInterface` library. As hinted at by the name, the classifiers themselves are implemented in TMVA [58], a ROOT-based toolkit for multivariate data analysis that is widely used in high-energy physics. It provides a rich set of different multivariate techniques, including artificial neural networks, Fisher discriminants, and support vector machines, as well as a set of common evaluation tools to judge the quality of a classifier training. Besides these built-in methods, a plug-in interface also allows the use of specialised classifiers, as long as they implement a simple interface.

The user interface based on this library acts upon `Particle` objects to train or apply a multivariate classifier on them. Through this integration with the analysis software, performing a training becomes much easier compared to users writing their own interfaces to the clas-

sification software. The main interface is provided by two functions, `trainTMVAMethod()` and `applyTMVAMethod()`. The most important configuration options for the training are the particle list used to gather candidates, the list of variables used as input to the classifier, the target and the classifier configuration itself (consisting of the classifier type and a configuration string passed to TMVA, with sane defaults being provided for both). Since inputs are filled directly using variables (see Section 4.1.2), users do not usually need to calculate inputs by themselves, as many important physical quantities are already implemented. If no particle list is given as input, all inputs are assumed to refer to event-level variables and all classification is done on the event itself. Through accumulating data in a `RootMergeable<TTree>` object (see Section 3.6), the training module can also be used in conjunction with BASF2's parallel-processing functionality. The training itself, however, requires repeated accesses to all data, and is started in the module's `terminate()` function. After the training is complete, the `showTMVAResults` command can be used on the training output files to show the control plots provided by TMVA's evaluation framework. As the training module also allows multiple methods to be specified, the plots can also be used to directly compare the separation power of different types of classifiers, or those with a different configuration. Details on the TMVA interface and the control plots in particular can be found in reference [59, p. 30].

The real usefulness of the classifier of course stems from applying it, which is performed by `applyTMVAMethod()`. It again requires a particle list as an argument, as well as the configuration generated by the training and the classification method to use (as the training might include more than one). After feeding the input variables (which variables are used is gleaned from the configuration) into the classifier, the output is saved in the current candidate as an extra-info field. For event-based trainings, a similar function is filled by the `EventExtraInfo` object. Whereas for the raw output of the classifier the interpretation of a certain value is not strictly defined (except that higher values should be more signal-like), the output can also be transformed into a probability before saving it in the particle candidate. This has the advantage of being easier to understand as well as simplifying the mixing of output from different classifiers. Details of this transformation, which is enabled by default, are explained in reference [59, p. 33].

Often it is beneficial to separate data accumulation and training from each other, e.g. when working with large amounts of data. To accommodate this use-case, the TMVA interface is also able to save its input data in a root file, which, in the case of multiple files created from different data sets, can be easily merged together. The stand-alone `externTeacher` executable then starts the actual training. Since this can be done multiple times with different method configurations or subsets of input variables, this feature is also useful for studying and optimising a given classifier training.

Due to performance concerns with the different classification methods shipped with TMVA, an optimized boosted decision tree (BDT) implementation called `FastBDT`, geared specifically to classification problems (as opposed to regression), is included in BASF2 as a plug-in for TMVA. It follows the description of boosted decision trees in reference [60] closely, but with CPU caches and pre-fetching in mind. As a result, it is up to fifty times faster than TMVA's BDT implementation during training, and up to ten times faster when applying the training [59, p. 34]. The separation achieved also tends to be better than that of the built-in methods and is comparable to that of the commercial `NeuroBayes` neural network classifier [61], which is also available as a TMVA plug-in. `FastBDT` is used as the default, if

4. Analysis Tools

the classification method is not explicitly changed when using `trainTMVAMethod()`.

While in most cases multivariate classifiers are trained using simulated data, where the classification target (e.g. whether a candidate is signal or background) is provided by the Monte Carlo simulation, there are also machine-learning techniques that can train multivariate methods on data. This can be very useful when no simulation is available or it is known to be insufficiently accurate, but usually needs additional physical insights in how to classify the data, as well as more care when selecting input variables. One such method is called *sPlot* [62], which replaces the binary target available on Monte Carlo with a probabilistic view on how likely it is that a given candidate is signal. This information is extracted from a fit to the data in a variable that both can be modelled well and separates between signal and background. An example might be the invariant mass of a reconstructed candidate, which can often be modelled by a flat or polynomial background shape plus a peaking signal component; fitting this model to the data then yields numbers of signal and background, and their distribution over the variable. For BASF2, this technique has been implemented in the TMVA interface [63] – trainings independent from Monte Carlo can thus be performed by specifying discriminating variables and appropriate fit models.

4.5. Skimming

As already hinted at in [Section 3.4](#), the μ DST data format contains the high-level reconstruction objects (e.g. `Track` and `PIDLikelihood` objects) present in the mDST format plus particle candidates reconstructed in certain decays; most of the size reduction implied by the name stems from discarding events that cannot be used to reconstruct the desired decays. Given that most of the time these decays are reconstructed in a multi-stage procedure (e.g. when creating D candidates that are later combined into B mesons), there may however be a large number of intermediate particle candidates. These are all stored in the central array of `Particle` objects in the data store and consume additional memory or disk space. Since the size of `ParticleList` is insignificant compared to the actual `Particle` objects, simply excluding the particle lists referencing these intermediate objects thus does not significantly reduce the size of the event.

To remove these additional candidates, a special module to clean up `Particle` objects that are not referenced by a given set of particle lists, called `RemoveParticlesNotInLists` (with a similarly-named Python function), was created. Due to `Particle` objects also referencing each other, having relations (see [Section 3.2](#)) from or to other objects and being referenced by particle lists, this module needs to be aware of these specifics when modifying the `Particle` array: First, all particle candidates in the lists to be kept, plus their daughters, grand-daughters, etc. are collected. The remaining candidates are then removed from the global `Particle` array, as well as any relations referencing them. Also, as the indices of `Particle` objects are changed due to rearranging of the array, all relations or daughter references with the kept candidates, as well as all particle lists including them, are corrected. The module thus leaves a coherent state for output modules or any further reconstruction steps.

[Table 4.2](#) compares the sizes of μ DST files with and without this reduction step applied, and also lists the size of the original mDST file for reference. For this example, particle candidates were created in various B^0 decay channels and, as a first step, all events discarded in which

Table 4.2.: Comparison of an mDST file with skimmed μ DST files containing a set of reconstructed B^0 decays, showing file sizes and total numbers of events and Particle objects in each file. For the μ DST files, the options of saving all candidates and saving only those involved in the B^0 decays of interest are compared. For object sizes IEC prefixes are used, with $K_i = 1024$ and $M_i = 1024^2$.

| | Events | Particles | File size (MiB) |
|----------------------------|--------|-----------|-----------------|
| mDST | 1 000 | 0 | 6.9 |
| μ DST (all candidates) | 191 | 69 521 | 2.8 |
| μ DST (only B0:tag) | 191 | 7 378 | 1.5 |

no B^0 candidate passing certain cuts could be found. This reduces the number of events to 19 % of the original number, but because of the large number of Candidates added on top of the mDST objects the file size is still about 40 %. Adding `removeParticlesNotInLists()` before saving the output discards most of these stray particle candidates and reduces the file size to 22 % of the mDST size, making the μ DST a truly lightweight format.

4.6. Best Candidate Selection

The size of the output can be reduced further by also limiting the number of candidates saved, which is commonly done by ranking them by a quality criterion and keeping only the best candidates. The reasons for this reduction not only include the size of the output, but also that often using multiple candidates per event would mean that e.g. the same track would be used in multiple candidates. As this overlap causes a correlation between the systematic uncertainties of multiple candidates (which a proper estimation of measurement systematics would need to include), analysis users may opt to select only one candidate.

The analysis tools provide two functions, `rankByHighest()` and `rankByLowest()` (with the associated module `BestCandidateSelection`), that define an ordering (highest-to-lowest and lowest-to-highest, respectively) based on a given variable. The rank for each particle candidate is saved in an extra-info field unique to the variable; optionally only a certain number of candidates of a low rank are kept. These functions can be used multiple times with different variables and the different rankings are saved with each candidate, which allows users to evaluate different selection methods and choose the optimal one.

4.7. Saving n -Tuples

After the analyst has used the tools presented in the previous sections to create lists of candidates that they are interested in, saving information on the candidates for further evaluation with other tools is a likely next step. The `variablesToNTuple()` function can be used for this purpose, and allows to create an n -tuple (in the form of a `TNTuple` object in a ROOT file) by saving a given list of variables for a given particle list. As with the trees used by the TMVA interface in [Section 4.4](#), the module is compatible with parallel execution and does

4. Analysis Tools

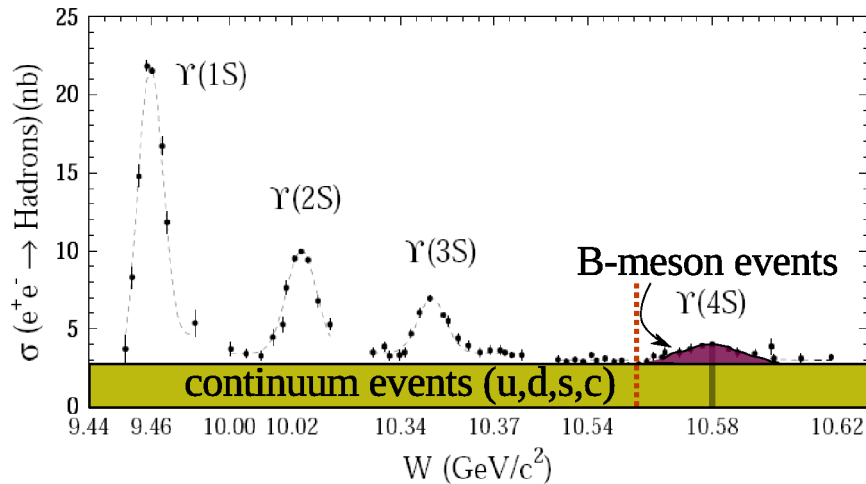


Figure 4.3.: Measured cross section for hadron production in e^+e^- collisions over the center-of-mass energy, showing the amount of continuum and resonance events produced. The dashed line denotes the threshold beyond which B meson pairs can be produced. Note that the energy range skips regions without resonances. Taken from [64].

not necessarily become a bottleneck when calculating the variables consumes a significant amount of CPU power.

As an alternative, the ‘Ntuple tools’ can save n -tuples which contain certain groups of branches that can be selected by the user (e.g. kinematics, Monte Carlo truth, or vertex information), and it is also possible to easily save information for certain selected daughter candidates in the decay chain. These tools are, however, not currently capable of running in parallel.

4.8. High-Level Reconstruction Tools

The previously described analysis tools provide a set of fundamental functions in a way that makes them useful as individual steps in an analysis reconstruction. There are however also tools that perform certain more complex, higher-level tasks for analyses; these will be discussed in this section. Additionally, the tag-side reconstruction algorithms introduced in the next chapter could also be included here. The implementation of such an algorithm in the Belle II software, will be discussed in detail in [Chapter 6](#).

4.8.1. Continuum Suppression

Even when running at energies required for creating pairs of B mesons, e.g. at the $Y(4S)$ resonance, B factories also generate a lot of non- $B\bar{B}$ events, which consist of $u\bar{u}$, $d\bar{d}$, $s\bar{s}$, and $c\bar{c}$ states and the resulting fragmentation products. These so-called *continuum* events are produced at a very wide range of center-of-mass energies and constitute a majority of events even at the $Y(4S)$ resonance, which can be seen from [Figure 4.3](#). As, at least for studying the

physics of B decays, these events only constitute a background, significant effort is invested to remove them.

As the $\Upsilon(4S)$ resonance lies just above the threshold for $B\bar{B}$ creation, the produced B mesons have only low amounts of kinetic energy. In the center-of-mass system, most directionality of decay products thus comes from the further *decay* of the B mesons and their daughters, resulting in a somewhat uniform angular distribution (or ‘spherical’ events). For continuum events, on the other hand, the energy difference between the $\Upsilon(4S)$ resonance and the produced $q\bar{q}$ state is available as kinetic energy. This results in events with a topology of two jets produced back-to-back (again in CMS system). These event shapes are commonly quantified using Fox–Wolfram moments [65].

In BASF2, an implementation of these moments based on the description in reference [11, p. 114] is available; individual moments are provided to analysts in the form of variables.

4.8.2. Flavour Tagging

For a number of analyses studying B^0/\bar{B}^0 mixing, e.g. for measuring mixing-induced CP violation in $B^0 \rightarrow J/\psi K_S^0$, it is important to know the flavour of both B^0 mesons, i.e. whether it was a B^0 or \bar{B}^0 . To distinguish between these two states, without exclusively reconstructing the B meson, is called *flavour tagging*. The flavour tagging algorithm implemented in BASF2, which makes use of many of the analysis tools described in this chapter, is described in reference [66].

4.9. Summary and Conclusions

This chapter described the analysis capabilities of the Belle II software, and the data structures and tools that provide them. Like other parts of BASF2, it is highly modular, and consists of a number of modules that act on `Particle` and `ParticleList` objects in the analysis package. These provide generic and tested implementations of all central steps of an analysis, including combining particle candidates, applying cuts, Monte Carlo matching, or training of multivariate classifiers. For all central modules of the analysis package, Python interface functions exist that provide a consistent user experience.

As a result, analysis users do not need to implement and test these error-prone steps themselves, and can focus on the physics instead. This is in stark contrast to the usage of analysis software at Belle, where some central utilities were provided, but which still required users to, e.g., iterate over candidate particles and combine them themselves. In many cases, users or groups of users ended up developing their own interfaces to be able to implement features not available in the Belle analysis software. One example of this is the *Full Reconstruction* software that is the topic of the next chapter, which substituted its own implementation for the Belle `Particle` class [67, p. 100].

Within the scope of this thesis, many of the tools described in this chapter were rewritten, fixed or extended. In particular, the `Particle` and `ParticleList` classes were improved (see Section 4.1), a generalised framework for variables was created (Section 4.1.2), the Monte Carlo matching was replaced (Section 4.3), as well as tools for reducing the output size (Sections 4.5 and 4.6) and for saving tuples suitable for further analysis (Section 4.7)

4. *Analysis Tools*

added. The requirements that motivated these enhancements, as well as the methods used to uncover problems with the existing implementations, will be discussed in [Chapter 6](#).

5. Tag-Side Reconstruction at Belle

The central feature of B factories – as discussed in [Chapter 2](#) – is the production of large numbers of $\Upsilon(4S)$ mesons, which decay with a branching ratio of almost 100 % into $B^+ B^-$ and $B^0 \bar{B}^0$ pairs. Analyses typically focus on measuring the properties of certain decays only, e.g. measuring the branching fraction of a B decay like $B^- \rightarrow \tau^- \bar{\nu}_\tau$, where the remaining tracks and electromagnetic clusters in the event (i.e. those not used in the measurement) are not of direct interest. Since the initial state, i.e. the four-momentum of the $\Upsilon(4S)$, is known, one can gain information and additional constraints by also considering the other B meson.

Through exclusively reconstructing one B meson with the highest possible efficiency and combining it with the selection of a signal decay of interest, one arrives at a more complete description of the event. In other words, the user creates B_{sig} meson candidates reconstructed in the desired signal decay, and combines them with a set of B_{tag} provided by a separate tool, yielding – if both B candidates are correct – a completely reconstructed $\Upsilon(4S)$ event. The additional information can be used to supplement the information available from the signal selection, and greatly increase its purity. This technique is sometimes called *tag-side reconstruction*. There are also complementary approaches that only reconstruct partial information, e.g. the four-momentum, vertex position, or the flavour (B^0 or \bar{B}^0) of the tag-side. These inclusive techniques tend to have much higher efficiency, but the reduced information content limits their usefulness to specific types of analyses.

Tag-side reconstruction of B mesons has been used extensively at Belle, with many analyses employing it to enhance the purity of their signal selection.¹ This works by requiring that there are exactly two B mesons in the event: one on the signal-side and one tag-side B meson, the latter being the output of the tag-side reconstruction. In the simplest case, one can require that after allocating tracks for both B mesons (i.e. reconstructing $\Upsilon(4S)$), no further charged tracks remain in the event. Compared to charged tracks, photons are harder to distinguish from beam background or detector noise, but it is possible to recognize additional photons in the event by looking at the energy E_{ECL} in the electromagnetic calorimeter that was not used in the reconstruction of the B mesons. Since E_{ECL} is concerned with the additional energy in the event, it is often referred to as *extra energy*. This quantity is used in many analyses as a powerful discriminator between correctly and incorrectly interpreted events, e.g. in $B \rightarrow \tau \nu$, or $B \rightarrow h^{(*)} \nu \bar{\nu}$ in references [\[70, 71\]](#) and [\[72\]](#), respectively.

If the signal decay contains a single neutrino and the tag-side is entirely hadronic, the method described also gives full kinematic information for the missing particle (neutrino). For correctly interpreted events, the invariant mass m_{miss} (called the *missing mass*) calculated from the missing four-momentum should then be equal to the neutrino mass, i.e. zero for Standard Model neutrinos with the given detector resolution. The missing mass is used in a number of analyses, e.g. a search for $B \rightarrow l \nu_l \gamma$, where the m_{miss}^2 distribution is fitted to

¹A B_{tag} reconstruction technique with similar intent and performance but employing a different algorithm was used by the BaBar experiment to enhance measurements [\[68, 69\]](#)

5. Tag-Side Reconstruction at Belle

extract the contribution of signal decays (around $m_{\text{miss}}^2 = 0$) [73]. This variable can also be used for analyses where the signal channel includes multiple neutrinos, but single-neutrino decays are an important background component. An example is the measurement of the relative branching fractions of $B \rightarrow D^{(*)} \tau \nu_\tau$ and $B \rightarrow D^{(*)} l \nu_l$ (where $l = e, \mu$), where m_{miss} is used to distinguish between decays with a τ lepton (three neutrinos, resulting in a broad peak in m_{miss}), a light lepton (one neutrino, i.e. $m_{\text{miss}} \approx 0$), and other backgrounds [74].

5.1. Control Variables

To evaluate the efficiency and purity of the tag-side B meson candidates, one frequently resorts to a set of two variables, m_{bc} and ΔE , that take advantage of the special kinematics at B factories [11, p. 85]. The beam-constrained mass

$$m_{\text{bc}} = \sqrt{E_{\text{beam, CMS}}^2 - \vec{p}_{\text{CMS}}^2}$$

is used in place of the more traditional invariant mass. It replaces the reconstructed energy of the B candidate with the beam energy $E_{\text{beam, CMS}}$, which is half of the total energy available in the center-of-mass system (CMS). \vec{p}_{CMS} is the three-momentum of the B candidate in the center-of-mass system. Since the candidate energy is replaced with a beam-parameter-derived quantity, the beam-constrained mass no longer depends on the mass hypotheses of particles used for reconstructing the B meson. For correctly reconstructed B mesons (and those where e.g. only a kaon track was misidentified as a pion) m_{bc} will take values around 5.28 GeV, which corresponds to the mass of B^0/B^\pm mesons. Misreconstructed candidates will usually have a more broad distribution in m_{bc} . The second variable, the energy difference ΔE , is defined as

$$\Delta E = E_{B, \text{CMS}} - E_{\text{beam, CMS}}.$$

Thus, since $Y(4S) \rightarrow BB$ is a two-body decay with two daughters of the same mass, correctly reconstructed B mesons should have an energy equal to half the total energy in the center-of-mass system, i.e. ΔE should be around zero. For misreconstructed candidates, ΔE will deviate from zero, e.g. to negative values if a photon is missed in the reconstruction.

Since both variables use different components of the B meson four-momentum, they are only weakly correlated.

5.2. Cut-based Full Reconstruction

At Belle, two different algorithms for tag-side B reconstruction (called *Full Reconstruction*) were used. The first, cut-based approach, reconstructed B mesons in a number of hadronic decay channels. The algorithm used eight different decay modes for each of B^+ and B^0 , two modes for \bar{D}^{*0} and D^{*-} each, seven modes for \bar{D}^0 , six modes for D^+ , two modes for D_s^+ , and one mode for D_s^0 . The number of candidates after making combinations was reduced by applying a selection based on the invariant mass or mass difference for $D_{(s)}^*$ of the candidate. The applied cuts were fairly wide, e.g. four to five standard deviations for D candidates, but the exact criteria for these cuts are unknown. In some cases, channels with high combinatorics were excluded for performance reasons, e.g. if many neutrals (π^0 or K_S^0) were used in a decay

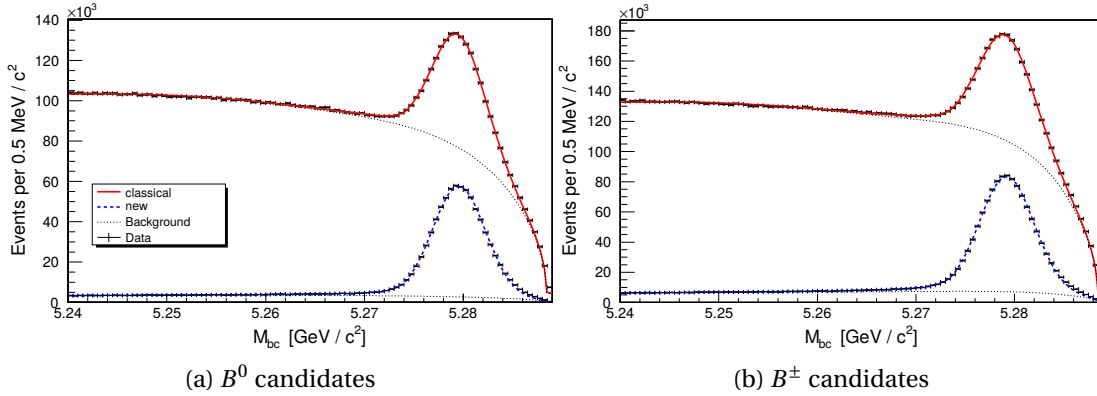


Figure 5.1.: Fitted beam-constrained mass (m_{bc}) distributions for B^0/B^\pm candidates on Belle data. The fit of candidates from the cut-based approach is shown as a solid red line, or a solid blue line for candidates produced by the neural-network-based approach. For each coloured line, the corresponding points with error bars are the fitted data, and the dotted black line is the fitted combinatorial and continuum background component. Cuts on the network output were chosen to yield a signal efficiency roughly equal to that of the cut-based algorithm. Adapted from [75].

mode, it was only combined with other low-multiplicity channels to form B candidates. Tag-side B mesons were selected using cuts on m_{bc} and ΔE . A best-candidate selection based on ΔE , and D invariant masses (or mass differences) was performed to ensure that at most one candidate per event is returned. The efficiencies for correctly reconstructing B^0 and B^+ mesons were 0.10 % and 0.14 %, respectively [11, p. 93].

5.3. Neural-network-based Full Reconstruction

The second algorithm replaced the rectangular cuts of its predecessor with neural networks and included additional decay channels. This increased the number of decay modes to fifteen for B^+ and thirteen for B^0 , and similarly greatly increased the covered branching fraction for D mesons. The addition of a multivariate classifier in the form of the NeuroBayes neural network package [61] allows one to use more information from each particle, and combines them into a single variable o_{NB} that is more powerful for separating between correctly and incorrectly reconstructed candidates. Through these changes, this approach yields maximum efficiencies of 0.18 % and 0.28 % for B^0 and B^+ , respectively (with a purity of about 10 %), which is roughly twice that of its cut-based predecessor. Since it also provides a single scalar output o_{NB} that can – given certain assumptions – be interpreted as a signal probability, users can make their own selections, increasing the B_{tag} sample purity at the cost of efficiency.

Figure 5.1 shows distributions of the beam-constrained mass for the cut- and neural-network-based algorithms for roughly equal signal efficiency, showcasing greatly increased purity of the second algorithm. Since it served as a model for the implementation of a similar algorithm in Belle II, details of the neural-network-based Full Reconstruction algorithm are described in the next section.

5. Tag-Side Reconstruction at Belle

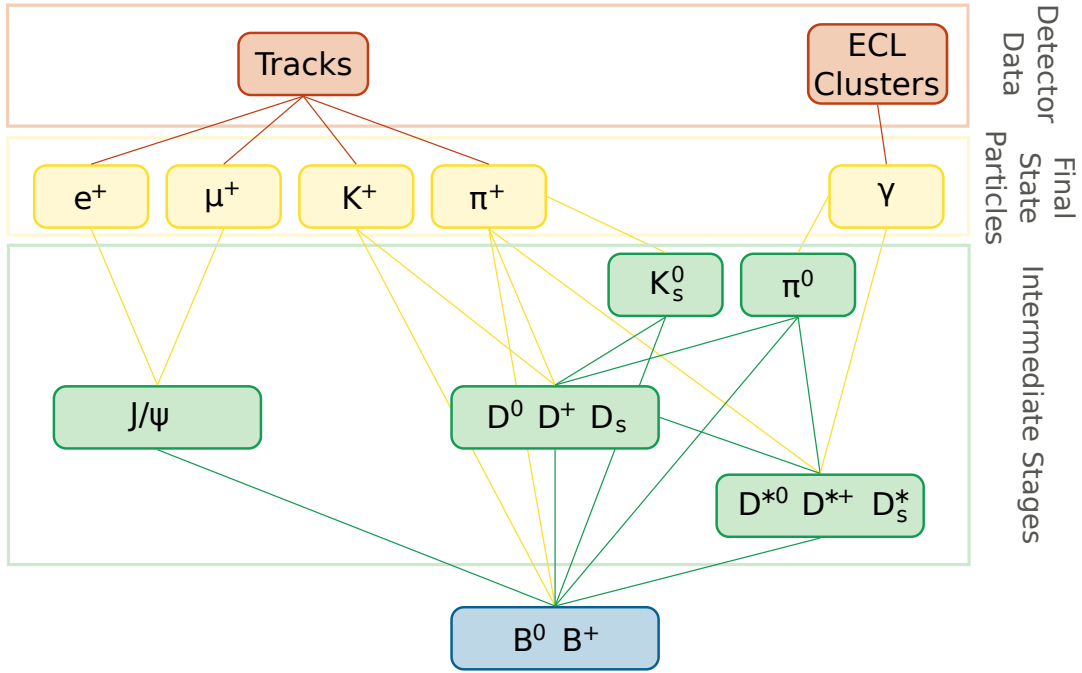


Figure 5.2.: Illustration of the hierarchical reconstruction from final-state particles to B mesons. Adapted from [76], based on [75].

For any tag-side reconstruction algorithm that explicitly reconstructs specific decays, increasing the number of channels also increases the covered B branching fraction and thus the total selection efficiency. Since particles produced via different decay channels only differ in their kinematic properties, the decay chain from B meson to final-state particles (tracks, photons) can be reconstructed in multiple steps, an approach that is shared by both algorithms. In the case of the neural-network-based algorithm, this allows covering 1104 exclusive B decays using 71 decay channels in multiple stages. Compared with the 25 B and D decay modes included in the cut-based Full Reconstruction algorithm, a significantly greater branching ratio is covered by this second approach. The hierarchical reconstruction in stages is illustrated in Figure 5.2.

Each of these 71 channels has a neural network associated with it that combines information from the invariant mass calculated for the candidate, angular information for and between daughters, (for D^* mesons) the mass difference between D^* and D meson, and the neural network output for each of the daughter particles. The trained neural networks are able to take existing correlations between these inputs into account and produce a scalar classification variable that tends to zero for background-like, and towards one for signal-like candidates. If the distributions (in particular the relative amount) of signal and background candidates is the same during training and application, NeuroBayes ensures that the network output can be interpreted as a Bayesian probability. Neural networks are also used to improve the selection of final-state particles by using information from dE/dx in the drift chamber, time of flight (TOF) and Cherenkov counter (ACC) detectors as input. For photons, different variables describing the shape of the shower detected in the electromagnetic calorimeter are

used.

A major problem is the combinatorics of decay channels that have a relatively high multiplicity (i.e. large number of daughter particles). Consider for example an event with twelve charged tracks, six with negative charge and six with positive charge. Assuming no cuts, reconstructing $D^0 \rightarrow K^- \pi^+$ (branching ratio $\approx 3.9\%$) produces

$$\binom{6}{1}^2 = 36$$

D^0 and \bar{D}^0 candidates each. For $D^0 \rightarrow K^- \pi^+ \pi^- \pi^+$ (branching ratio $\approx 8.1\%$) this increases to

$$\binom{6}{2} \cdot \frac{6!}{(6-2)!} = 450$$

candidates for each of the two charge hypotheses.² This already hints at two main factors that increase the combinatorics of a channel: the number of daughters that cannot be exchanged with another daughter without changing the meaning of the reconstruction (as with K^- and π^- in the example) and the total number of available candidates in each of these classes of daughters.

Every one of the hundreds of D^0 candidates – if not discarded – would then require further CPU-intensive processing, e.g. vertex fitting or calculating the inputs of the neural network, while only a few true D^0 could actually exist in one event. For channels that include neutral particles like π^0 , the situation deteriorates further due to the high number of electromagnetic clusters per event (with around 11 true photons on average). In the following D^* and B stages that already receive large numbers of candidates from the previous stages, creating tens of thousands of candidate particles is trivial and would make reconstructing high-multiplicity channels prohibitively costly. Avoiding this necessitates removing combined particle candidates before any expensive operations are performed on them. Through a cut on the product of daughter outputs

$$o_{\text{NB, prod}} = \prod_i o_{\text{NB}}^i,$$

where i runs over all daughter particles, candidates could be rejected quickly (requiring only the calculation of a product and fetching the network output o_{NB}^i for each daughter). $o_{\text{NB, prod}}$ also uses a large fraction of the available information, including particle identification, momentum or invariant mass information of the daughters. Implicitly this construction makes use of the NeuroBayes classifier's feature that its output can to some degree be interpreted as probability, as mentioned above.

Only soft cuts were made on the $o_{\text{NB, prod}}$ variable to ensure a high efficiency. For a given particle type, it is useful to optimize the allocation of CPU resources to different decay modes simultaneously, so that channels resulting in a large number of background candidates do not monopolize resources that might be more useful in cleaner channels. For the Full

²The calculation here uses two steps: choosing two π^+ from the list of six positive tracks (*combination*, yielding the binomial $\binom{6}{2}$), and choosing a K^- and a π^- from an equal number of negative tracks (not neglecting order, i.e. a *permutation*, yielding the larger term $6!/(6-2)!)$.

5. Tag-Side Reconstruction at Belle

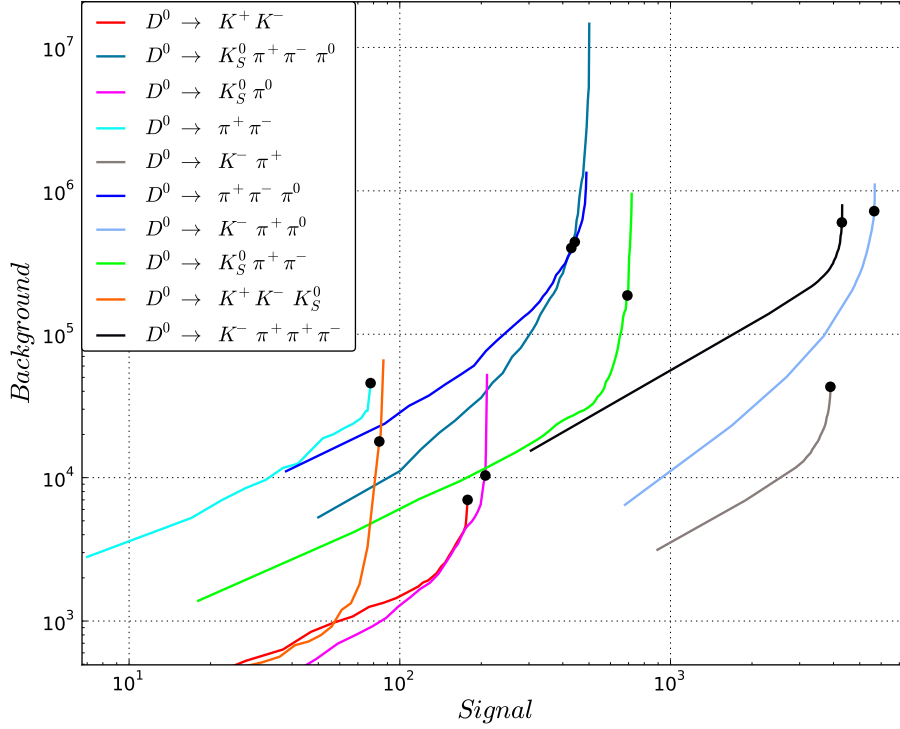


Figure 5.3.: Number of background candidates over number of signal candidates for different D^0 decay modes. Each coloured line corresponds to the set of cuts on $\sigma_{\text{NB, prod}}$ for a decay mode, with a possible choice of common cut values marked by black dots. The logarithmic scale of both axes accentuates the differences between modes, but also distorts the (equal) slope of the curves at the chosen cut values (see text). Adapted from [75].

Reconstruction, the cuts were determined by requiring that a variation of this cut to increase the number of signal candidates by a fixed amount yields an identical increase in the number of background candidates for each decay mode. When graphing the number of background candidates over the number of signal candidates selected by a cut, this criterion corresponds to an equal slope at the cut point for each decay mode. The common slope is a free parameter that determines the hardness of the cuts and was determined heuristically with the additional restriction of having an average processing time below 0.1 s per event [77, p. 73]. For D^0 mesons an example is shown in Figure 5.3. This optimization has the effect of preferably choosing looser cuts for channels that have a relatively high purity, whereas less pure channels get a harder cut to limit their influence on the processing time. A tacit assumption in this scheme is that the required CPU resources per channel depend only on the number of candidates, which – while not strictly true – takes into account most of the differences in run-time performance between channels.

At the final stage, B^\pm and B^0 candidates are produced and ranked by their network output σ_{NB} to allow later selection of the best candidate. The B -level network trainings do not include variables with a high correlation to m_{bc} to avoid influencing this control variable. As ΔE is only weakly correlated to m_{bc} and its separation power is high, it is also included in

these final trainings. The user of the Full Reconstruction can then choose a working point suitable for their analysis. In some cases, event shape information can also be included in the network as a form of continuum suppression (see [Section 4.8.1](#)) but this is not done by default to avoid biases this would produce in some analyses. Two different possibilities for input variables were used: one included Fox–Wolfram moments that characterise the shape of the event [65], allowing to distinguish between (in center-of-mass system) more spherical $B\bar{B}$ events and jet-like $q\bar{q}$ continuum background; the other included modified (or super-) Fox–Wolfram moments that treat the tag-side B meson candidate and the remaining event separately [78]. Purity–efficiency plots for the Full Reconstruction with and without continuum suppression variables are shown in [Figure 5.4](#).

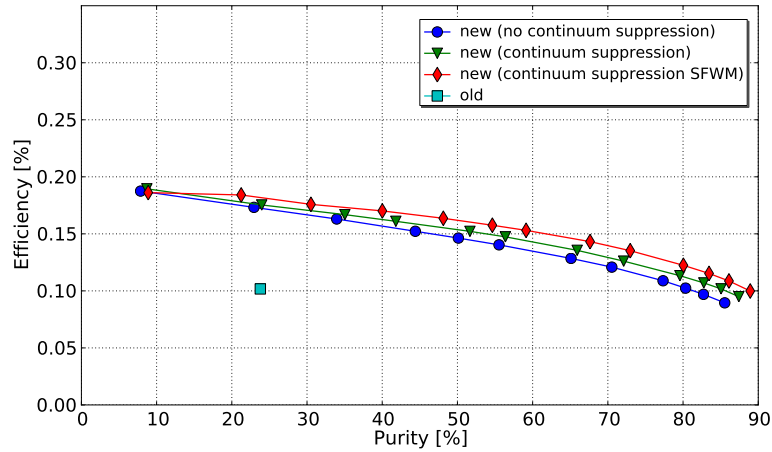
The tag-side reconstruction efficiency depends strongly on the branching fractions of the hadronic decays involved in reconstructing B_{tag} candidates. In many cases, no accurate measurement of these branching ratios exists and the Monte Carlo distributions are known to deviate significantly from data. Since the number of correctly reconstructed B_{tag} and thus the reconstruction efficiency is an important input to many analyses, the observed bias can have a significant effect on physics results. This effect can be corrected by measuring the actual reconstruction efficiency to produce correction factors for each hadronic decay mode: hadronic B_{tag} candidates can be combined with a B meson reconstructed in a channel with well-known branching fraction (e.g. $\bar{B} \rightarrow X_c l^- \bar{\nu}_l$), which allows one to compare the predicted number of events with that found on data. Averaging over all decay modes, a correction factor $\epsilon_{\text{Data}}/\epsilon_{\text{MC}} \approx 0.75$ is obtained, i.e. in most modes, the estimated reconstruction efficiency is too high [79].

5.4. Extensions of the Full Reconstruction

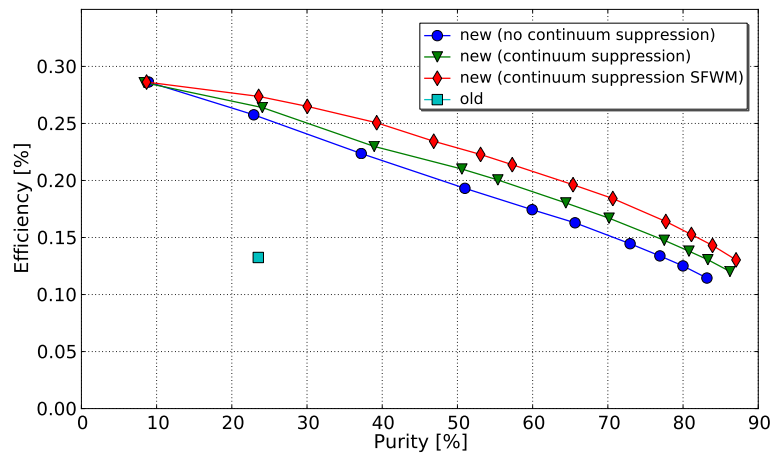
In addition to the hadronic tag-side reconstruction, the implementation was also adapted to allow reconstruction of semileptonic B meson decays in the modes $B^0 \rightarrow D^{(*)-} l^+ \nu_l$ and $B^+ \rightarrow \bar{D}^{(*)0} l^+ \nu_l$ (with $l = e, \mu$). These eight decay channels alone have a large combined branching fraction of around 16 % and can greatly increase the total tag-side efficiency. While the lepton is usually easy to select using particle identification information, the escaping (not measured) neutrino reduces the amount of kinematic constraints possible on the B candidates [80, 81]. The semileptonic Full Reconstruction was successfully used to measure the branching fraction of $B^+ \rightarrow \tau^+ \nu_\tau$ decays [71].

An extension to allow the use of the Full Reconstruction at the heavier $\Upsilon(5S)$ resonance was also developed and used to analyse the Z_B resonance [82]. Due to its higher mass, $\Upsilon(5S)$ can decay in more channels, which besides B^\pm and B^0 include B_s mesons and orbitally excited $B_{(s)}^*$ mesons. These additional B mesons are also included in the tag-side reconstruction since they provide information about the decay mode of the $\Upsilon(5S)$. This, in addition to the different kinematics, make the event reconstruction more complex. [Figure 5.5](#) shows the beam-constrained mass for B meson candidates reconstructed on the $\Upsilon(5S)$ resonance. The left-most peak corresponds to the lighter $B\bar{B}$ mesons also produced at the $\Upsilon(4S)$ resonance, while the peaks at higher masses result from different combinations of the heavier mesons. Besides the two-body decays into two mesons, where both mesons have the same fixed momentum, decays with more particles are also possible and produce broader structures in m_{bc} .

5. Tag-Side Reconstruction at Belle



(a) B^0 candidates



(b) B^\pm candidates

Figure 5.4.: Efficiency over purity (Receiver operating characteristic) for different cuts on o_{NB} for B^0 and B^\pm candidates. **Blue** markers and lines show the behaviour of the network-based Full Reconstruction without continuum suppression, **red** and **green** add continuum suppression with and without Super-Fox-Wolfram Moments (SFWM), respectively. For comparison, the **cyan** square shows the working point of the cut-based algorithm. Taken from [75].

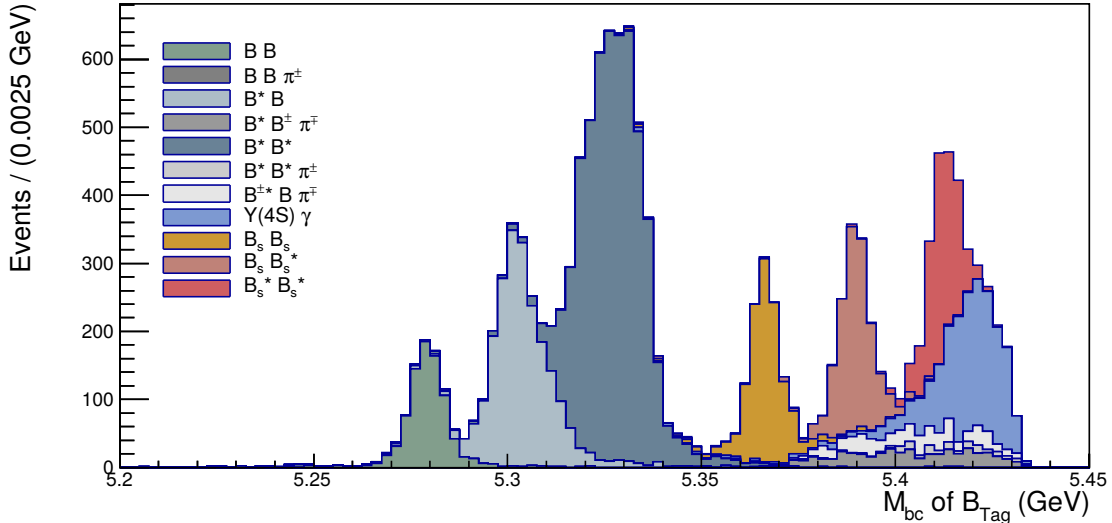


Figure 5.5.: m_{bc} distribution of correctly reconstructed B_{tag} candidates on simulated data, with colours denoting different $\Upsilon(5S)$ decay modes (and/or different B mesons). Note that for increased visibility distributions are not scaled according to their physical branching ratios. Taken from [82].

5.5. Summary

Though the neural-network-based Full Reconstruction was successfully used in many analyses, creating a training or modifying the tool was quite difficult. A major factor was the complexity of the tool, which consisted of a total of 18 221 lines of C++ code and a few hundred lines in bash or Python scripts.³ Also, even though the variants with reconstruction of semileptonic B decays or those optimised for the $\Upsilon(5S)$ resonance in large parts perform the same steps, they are not available in a central library and much of the implementation is copied instead. Decay modes were defined in the source code and manually assigned to one of the four reconstruction stages; cuts and lists of input variables are also part of the C++ source file [67, 64]. Performing a training began with the `ekpfullrecon` module writing a ROOT file containing the input variables for all decay channels in the current stage. The neural networks were then trained using executables (created using generated C++ code); the finished trainings could later be applied by adding generated code to the file containing the channel definitions and rebuilding it. Cuts on $\Pi \Pi_{\text{NB}}$ were evaluated using an external tool; the resulting cut values needed to be added to the channel definitions by hand afterwards. Trainings were started manually for each reconstruction stage, which also contained hard-coded lists of decay channels. A few weeks (real time, not CPU time) were required to complete the training on the KEK computing system.

This meant that even though the Full Reconstruction was a useful tool used by many analyses, starting a training or making any changes to the configuration required lots of manual steps and a somewhat arcane knowledge of the implementation details.

³Lines of code (ignoring comments and blanks) in the repository were measured using `ohcount` [83]. Generated code relating to the neural network experts was excluded.

6. Full Event Interpretation

For Belle II, we want to improve upon the previous reconstruction efficiency, while at the same time avoiding some of the technical issues that were present in Belle’s Full Reconstruction algorithm. This includes being able to configure decay channels, input variables used for the classifiers, and the classifier’s hyperparameters¹ in a user-friendly way. Besides configuration, the main interaction of users with the system consists of performing a training with a given configuration and applying the trained algorithm afterwards. To eliminate difficulties with this process, the training should be automatic to the extent that no user-intervention is required besides the (initial) configuration and that dependencies between reconstruction tasks can be determined and resolved automatically. To increase the usefulness of the automatic training procedure, it should also generate plots and statistics that describe the different steps performed in the training. These plots should be useful for users wanting to run their own training by providing information on the achieved efficiency/purity and the status of each decay channel as well as physics control plots (e.g. of the beam-constrained mass), without requiring deep knowledge about the implementation details.

These improvements also translate into possible efficiency improvements by making it inexpensive to change the channel configuration and retrain with optimized settings. In particular, increasing the covered branching fraction of B decays only requires specifying additional channels. The algorithm can then decide whether the channel actually meets the given efficiency and purity criteria, i.e. whether using it will provide a gain that is commensurate with the associated additional costs in CPU time. As the CPU time required by the training is a very important factor in how easily optimised settings can be studied, enhancement in the training speed also have a large impact on physics analyses relying on these trainings. There are other, more far-reaching changes made possible by this simplified and improved training process, the details of which will be discussed in [Section 6.6](#).

To capture this widened scope – particularly including additional constraints from the entire $\Upsilon(4S)$ event – the tag-side reconstruction algorithm for Belle II is called *Full Event Interpretation* (FEI). Details of its implementation and how it achieves the aforementioned goals are discussed in the following sections.

6.1. Software Architecture

For analyses, a major improvement of BASF2 over the Belle software is the introduction of the modular analysis tools presented in [Chapter 4](#): common analysis tasks are implemented as BASF2 modules that work on high-level data structures like `Particle` or `ParticleList` and a set of Python functions defines an interface that maps quite well to typical analysis reconstruction steps. When implementing a tag-side reconstruction algorithm (which shares

¹ Hyperparameter refers to a parameter of the classification method itself, e.g. the number of trees for boosted decision trees or the number of neurons for an artificial neural network.

6. Full Event Interpretation

many steps with a plain analysis but contains many more decay channels), these existing tools save us from having to write our own tools for e.g. reconstructing decay channels. This both simplifies the implementation and makes it more robust, since problems with the shared tools are likely to be discovered early. Using BASF2's steering file interface and the possibility of accessing data using PyROOT, it also becomes possible to profit from the abstraction provided by Python in the entire tag-side reconstruction framework.

Decay channels and associated multivariate classifiers can then be configured using Python functions/classes either inside the steering file, or inside a separate Python module that can be loaded using `import module_name` when needed. When starting the Full Event Interpretation, the channels defined in the configuration are converted into reconstruction tasks, e.g. selection or combination of particles, fitting of a decay vertex, or training of a multivariate classifier. Each of these tasks has a list of inputs and outputs, which taken together completely describe the dependencies between tasks. These dependencies are resolved automatically and tasks executed in a linearised manner. Since some tasks need access to the entire data set (e.g. a multivariate classifier needs to be trained on a suitably large data set before it can provide output), not all dependencies can be resolved in a single iteration. This is similar to the concept of *stages* in the Full Reconstruction algorithm at Belle, but is a dynamic consequence of the dependency resolution. Starting the Full Event Interpretation again will finish the previously blocked tasks (e.g. perform a classifier training) and in that way advance the algorithm to the next training stage. A complete training (from final-state particles to B mesons) thus only requires starting the algorithm the requisite number of times until all dependencies are met. [Figure 6.1](#) illustrates this architecture from the user interface (configuration) to the compiled analysis modules written in C++ that execute the internally generated tasks on large data samples.

Details on the implementation of these features and certain core steps are provided in the following sections.

6.1.1. Channel Configuration

The channel configuration is the main interface for setting down the physical meaning of the actions performed by the Full Event Interpretation algorithm, by defining the particles (B^\pm, B^0, D^{*0} etc.) to consider, the channels to reconstruct them in, and how each multivariate classifier is to be created. The algorithm itself thus should not need to know any physical details on the channels to reconstruct, and more importantly, the interface can be powerful enough to handle both hadronic and semileptonic B decays, as well as tag-side reconstruction at the $\Upsilon(5S)$ resonance via changes to the configuration only.

Configuration of the FEI is based on defining a list of particles to reconstruct. A minimal example including both final-state and combined particles can be found in the following listing:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from fei import *
```

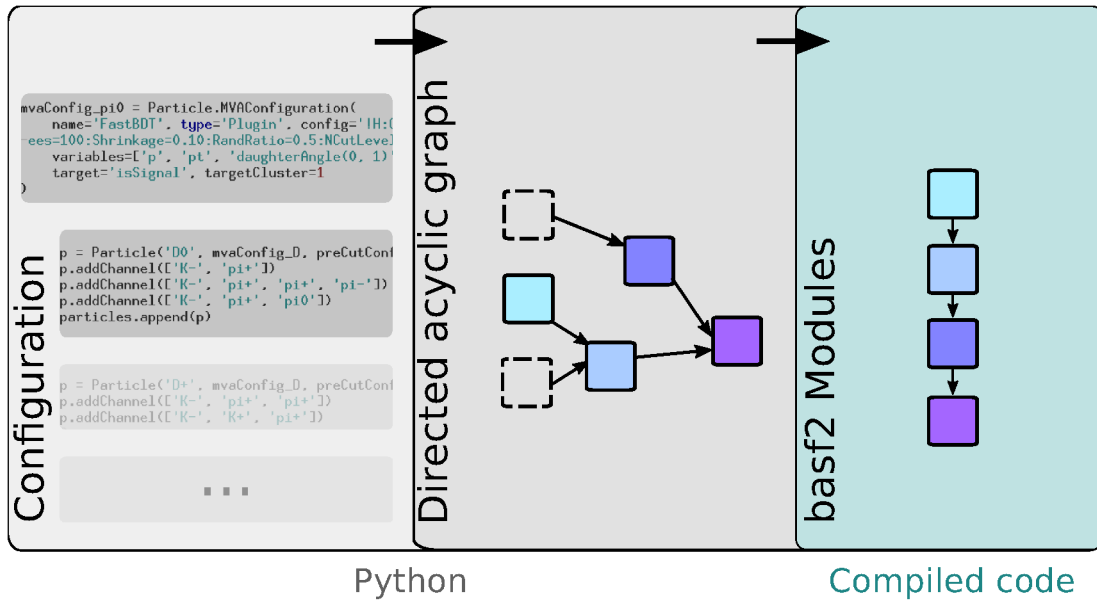


Figure 6.1.: Architecture of the Full Event Interpretation, showing high-level configuration of decay channels, automatic dependency resolution between reconstruction tasks created from the configuration, and the final data-intensive processing being handled by BASF2 modules. Taken from [76].

```
# example set of variables, many useful ones not included
trackVars = ['Kid_dEdx', 'Kid_TOP', 'Kid_ARICH', 'p', 'pt', 'chiProb' ←
]
mvaConfig_chargedFSP = MVAConfiguration(variables=trackVars, target=' ←
isSignal')

particles = []
particles.append(Particle('pi+', mvaConfig_chargedFSP))
particles.append(Particle('K+', mvaConfig_chargedFSP))

# definition of mvaConfig_D, preCutConfig_D omitted

p = Particle('D0', mvaConfig_D, preCutConfig_D)
p.addChannel(['K-', 'pi+'])
p.addChannel(['K-', 'pi+', 'pi-', 'pi+'])
particles.append(p)

# pass particles to FEI...
```

The example starts by defining a set `trackVars` of variables (introduced in [Section 4.1.2](#)) that help distinguish different charged track hypotheses from each other. They will be used by a multivariate classifier configured using the `MVAConfiguration` object. Besides the input variables, this requires specifying a target variable, e.g. `'isSignal'`, which in this case accesses

6. Full Event Interpretation

Monte Carlo truth information to determine whether a track actually originates from a pion or kaon. Details of the multivariate classifier, including its type, hyperparameters and other options to pass to TMVA can also be set, but have reasonable defaults that in most cases should not need to be modified. The classifier configuration is then used to create two types of final-state particles, 'pi+' and 'K+'. These names correspond to the `ParticleList` names used to identify particle types (as introduced in [Section 4.1.1](#)). When creating combined Particles like 'D0', one similarly supplies a classifier configuration, but one also includes details on how intermediate cuts are to be determined (see [Section 6.3.2](#)) and adds decay channels via the `addChannel()` method. In this example, the two decay modes $D^0 \rightarrow K^- \pi^+$ and $D^0 \rightarrow K^- \pi^+ \pi^- \pi^+$ are defined for the D^0 . Charge-conjugate channels and particles are included implicitly, so the 'K-' particle referred to is the charge conjugate of the 'K+' defined earlier, which is consistent with normal `ParticleList` behaviour. After all particles and decay channels are defined, the list is passed to the Full Event Interpretation algorithm, where it defines the tasks the algorithm should perform.

The Full Event Interpretation algorithm then uses this configuration to create a set of tasks that, taken as a whole, reconstruct the given list of particles. For the above example configuration, the algorithm would generate tasks to create the appropriate `ParticleList` objects, train or apply multivariate classifiers using the given parameters, create combined particles and apply cuts. Depending on its use case, a task may either make use of the modular analysis functions (see [Chapter 4](#)), which add modules to a BASF2 path that is executed later or immediately perform actions on available information or files, e.g. calculate quantities from histograms.

6.1.2. Dependency Resolution

An important feature of these tasks is that they are not independent, but have inputs and outputs that define their relation to one another: any input matches either the output of another task, or part of the configuration itself. Because of this, the Full Event Interpretation can determine which tasks can currently be executed, and in which order. Likewise, tasks whose output is not required for another task will not be executed unless they are explicitly marked as needed. (This is applied recursively, i.e. tasks only required by unnecessary tasks are also skipped.)

Going back to the previous example, the tasks responsible for creating kaon and pion particles (using `fillParticleList()`) only depend on the configuration itself, and can be executed immediately. For both lists, this is followed by tasks that deal with the associated multivariate classifiers. When first starting the algorithm, the classifiers have neither been trained, nor is the necessary data for starting a training available; thus the first step is to save an n -tuple of the input and target variables. Since the tuple is only complete after the current iteration, the task will not provide an output. This blocks execution of all dependent tasks.

In the next iteration (or stage in the terminology of the Belle algorithm), the previously executed tasks are run again. As the tuple files now exist, the input requirements for the classifiers can now be met and the training is started using the `externTeacher` tool. Then the `TMVAExpert` modules accessing the finished training are added to the path, completing reconstruction of the final-state particles. The tasks created for the reconstruction of D^0 particles are similar, and include combining the daughter particles previously created (using

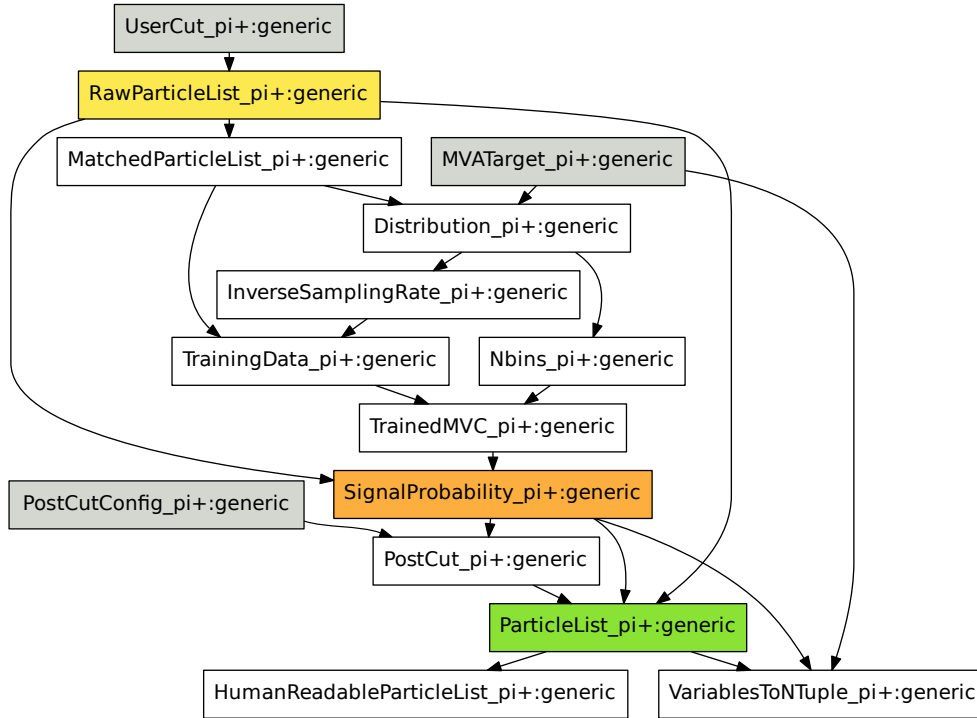


Figure 6.2.: Part of the directed acyclic graph responsible for π^+ reconstruction, with important resources highlighted: the raw particle list with only a user-cut applied (yellow), the signal-probability resource signifying that a trained classifier is available (orange), and the output particle list which is used in combined particles (green). Resources with no inward-directed graph edges (zero indegree) are shown in gray and correspond to the configuration inputs.

reconstructDecay()) while applying cuts, as well as another set of multivariate classifier trainings. Depending on the input variables, additional tasks to perform vertex fits, and/or dependencies on the classifier output of daughter particles are added. These actions are performed independently for each decay mode, afterwards a merged D^0 ParticleList is created.

The dependency structure defined by the inputs and outputs can be conceptualised as a *directed acyclic graph* (DAG). In the case of the Full Event Interpretation, this concept is encapsulated in the DAG and Resource Python classes. Resources represent individual nodes (tasks) in the graph, and consist of a unique identifier, a list of required inputs and a way to provide output when executed. In the simplest case, the output of the resource is taken directly from the configuration and no action is performed. Most tasks contain a function that produces the desired output using the input data. For this to work, every resource that was already executed saves its output for later use. The input data are passed as function arguments, where each argument is filled according to its associated resource identifier. Resources are managed by the DAG class, which is responsible for actually executing resources, and doing so in the order that their dependencies are met. It can also cache the output of resources, so that expensive tasks are not executed unnecessarily. The graph is filled

6. Full Event Interpretation

after reading the configuration by creating appropriate `Resource` objects for each particle and decay channel. An illustrative example in the form of the sub-graph for reconstructing π^+ can be found in [Figure 6.2](#). It begins with resources that are provided directly by the configuration and a first selection of particles, continues with a number of tasks related to the classifier training, and finally provides a particle list that will be used in later stages. Details of the different tasks shown will be discussed in the following sections.

To ensure changes to the configuration cause re-execution of all dependent tasks, the SHA-1 [84] hash of all requirements (i.e. the input data) is calculated for each resource.² If, for a cached resource, the calculated hash of inputs differs from the cached value, the cached output is discarded and the resource is re-run. Depending on whether the output of resources changes, re-execution propagates to all dependent tasks. For data stored in files, like trained multivariate classifiers, or lists of particles stored in the output of BASF2 modules, dealing with configuration changes poses a separate problem: how does one detect that a classifier training or particle list matches the current configuration, or if it does not? This is solved by also including hashes of input data in all file names and `ParticleList` names. As a result, each individual set of input variables to a multivariate classifier will result in a unique file name of the associated training file name, and thus trigger retraining if the file is missing. Likewise, lists of particles will change when, e.g., the cuts for selection or a classifier changes. Due to these mechanisms, the Full Event Interpretation is capable of automatically dealing with configuration changes.

After all tasks of the current iteration have been executed (i.e. no further tasks have requirements that can be satisfied), BASF2 modules corresponding to each task (if any) can be arranged in a path and used to process data. To make this more efficient on a single computer, one can use the parallel processing functionality introduced in [Section 3.5](#) to make use of more than one CPU core. As mentioned before, this is limited to parallel execution of consecutive compatible modules, e.g. if a chain of modules that can be run in parallel is interrupted by a single module that can not, only the first half of the chain will be parallelized. To optimize the data-intensive part of the processing, the Full Event Interpretation reorders the linearised tasks (and the associated BASF2 modules) such, that all non-parallel modules are moved to the end of the path, as far as permitted by their dependencies. This maximises the length of the parallel section and thus the possible speedup from multi-core processing. For larger input data sets, executing the Full Event Interpretation on a single computer becomes infeasible and a different approach is required. As this does not require deep integration within the dependency resolution framework, the details of trainings using distributed computing will be presented in [Section 6.7](#). Using multiple processes on each node however may still have advantages in some cases, as this might entail a significant decrease in the total number of jobs required, leading to reduction of job scheduling and file operation overhead.

So far, the description of tasks has been simplified in favour of discussing their interaction. In the following sections, the different reconstruction tasks will be explained in detail.

²While SHA-1 is no longer deemed safe for cryptographic use, no hash collisions (identical hash values for different inputs) have been found, and there are no disadvantages to using it for the generation of unique identifiers. There are also no barriers to using a different hashing algorithm that has a low likelihood of collisions.

6.2. Particle Selection and Combination

As a first step in the reconstruction of each particle, candidates need to be created according to the criteria specified in the configuration. Final-state particles ($\pi^\pm, K^\pm, \mu^\pm, e^\pm, \gamma$) are created directly from `Track` and `ECLCluster` objects using the `ParticleLoader` module (see [Section 4.1.3](#)), and afterwards can be fed into associated multivariate classifiers. For combined particles, more complex actions are required for each decay mode. There, candidates are first created using the `ParticleCombiner` module (see [Section 4.1.4](#)) with an appropriate cut to limit the combinatorics (see next section). Before they can be used for either creating or applying a classifier training, there may however be need for additional reconstruction tasks to be able to calculate the variables used as input. For this reason, a vertex reconstruction task is added to the DAG that, when executed, performs a vertex fit using the `KFit` fitter available through the vertex fit module introduced in [Section 4.2](#). Whether this task is actually executed depends on the list of classifier input variables: Only if they contain variables requiring a vertex fit is a dependency of the classifier actually added. Without this dependency, the vertex fit task is deemed unnecessary by the dependency resolution, and is skipped. Something similar happens for the multivariate classifiers themselves: When a higher-level training includes a variable like `daughter(i, extraInfo(SignalProbability))` (and only then), a dependency on the tasks providing a classifier output for the daughter particles is added. Thus, classifier trainings are – with the exception of *B*-level trainings – only performed when they are a prerequisite of higher-level trainings. To be able to use candidates independent of their specific decay mode in later stages, candidates from all modes are combined into a single list for each particle in the configuration.

All of these tasks are dependent on the list of particles and decay modes in the configuration, each of which can contribute to the total reconstruction efficiency. For each decay mode, different parameters determine the size of its impact on this quantity, like the purity and efficiency of the associated classifier or the reconstruction efficiency of the daughters. One of the most important parameters, however, is the branching fraction of each decay

Table 6.1.: Decay modes included in the FEI default configuration, with branching ratios extracted from `EvtGen`. Values marked by ‘*’ were missing or differed significantly from PDG averages, and have been fixed to PDG values [85]. (*Table continued on the following pages.*)

| (a) Decay modes for π^0 . | | (b) Decay modes for K_S^0 . | |
|-----------------------------------|-----------------|---------------------------------|-----------------|
| Decay channel | Branching ratio | Decay channel | Branching ratio |
| $\pi^0 \rightarrow \gamma \gamma$ | 98.82 % | $K_S^0 \rightarrow \pi^+ \pi^-$ | 69.13 % |

| (c) Decay modes for J/ψ . | |
|----------------------------------|-----------------|
| Decay channel | Branching ratio |
| $J/\psi \rightarrow e^+ e^-$ | 5.94 % |
| $J/\psi \rightarrow \mu^+ \mu^-$ | 5.93 % |
| Total | 11.87 % |

6. Full Event Interpretation

(d) Decay modes for D^0 .

| Decay channel | Branching ratio |
|---|-----------------|
| $D^0 \rightarrow K^- \pi^+ \pi^0 \pi^0$ | 15–20 %* |
| $D^0 \rightarrow K^- \pi^+ \pi^0$ | 13.90 % |
| $D^0 \rightarrow K^- \pi^+ \pi^+ \pi^-$ | 8.08 %* |
| $D^0 \rightarrow K_S^0 \pi^+ \pi^- \pi^0$ | 5.40 % |
| $D^0 \rightarrow K^- \pi^+ \pi^+ \pi^- \pi^0$ | 4.20 %* |
| $D^0 \rightarrow K^- \pi^+$ | 3.89 % |
| $D^0 \rightarrow K_S^0 \pi^+ \pi^-$ | 2.94 % |
| $D^0 \rightarrow \pi^- \pi^+ \pi^0$ | 1.43 %* |
| $D^0 \rightarrow \pi^- \pi^+ \pi^0 \pi^0$ | 0.94 % |
| $D^0 \rightarrow \pi^- \pi^+ \pi^+ \pi^-$ | 0.74 %* |
| $D^0 \rightarrow K^- K^+ K_S^0$ | 0.46 % |
| $D^0 \rightarrow K^- K^+$ | 0.39 % |
| $D^0 \rightarrow K^- K^+ \pi^0$ | 0.33 %* |
| $D^0 \rightarrow \pi^- \pi^+$ | 0.14 % |
| Total | 57.85 % |

(e) Decay modes for D^+ .

| Decay channel | Branching ratio |
|---|-----------------|
| $D^+ \rightarrow K^- \pi^+ \pi^+$ | 9.40 % |
| $D^+ \rightarrow K_S^0 \pi^+ \pi^0$ | 6.90 % |
| $D^+ \rightarrow K^- \pi^+ \pi^+ \pi^0$ | 5.99 %* |
| $D^+ \rightarrow K_S^0 \pi^+ \pi^+ \pi^-$ | 3.12 %* |
| $D^+ \rightarrow K_S^0 \pi^+$ | 1.49 % |
| $D^+ \rightarrow \pi^+ \pi^+ \pi^- \pi^0$ | 1.16 % |
| $D^+ \rightarrow K^- K^+ \pi^+ \pi^0$ | 1.05 % |
| $D^+ \rightarrow K^- K^+ \pi^+$ | 0.95 %* |
| $D^+ \rightarrow K^+ K_S^0 K_S^0$ | 0.46 % |
| $D^+ \rightarrow \pi^+ \pi^+ \pi^-$ | 0.32 %* |
| Total | 30.85 % |

(f) Decay modes for D^{+*} .

| Decay channel | Branching ratio |
|--------------------------------|-----------------|
| $D^{+*} \rightarrow D^0 \pi^+$ | 67.70 % |

(g) Decay modes for D^{0*} .

| Decay channel | Branching ratio |
|---------------------------------|-----------------|
| $D^{0*} \rightarrow D^0 \pi^0$ | 61.90 % |
| $D^{0*} \rightarrow D^0 \gamma$ | 38.10 % |
| Total | 100.00 % |

(h) Decay modes for D_s^+ .

| Decay channel | Branching ratio |
|---|-----------------|
| $D_s^+ \rightarrow K^+ K^- \pi^+ \pi^0$ | 6.30 %* |
| $D_s^+ \rightarrow K^+ K^- \pi^+$ | 5.39 %* |
| $D_s^+ \rightarrow K^- K_S^0 \pi^+ \pi^+$ | 1.64 % |
| $D_s^+ \rightarrow K^+ K_S^0$ | 1.49 % |
| $D_s^+ \rightarrow \pi^+ \pi^+ \pi^-$ | 1.08 % |
| $D_s^+ \rightarrow K^+ K_S^0 \pi^+ \pi^-$ | 0.96 % |
| $D_s^+ \rightarrow K^+ K^- \pi^+ \pi^+ \pi^-$ | 0.86 %* |
| $D_s^+ \rightarrow K^+ \pi^+ \pi^-$ | 0.65 %* |
| $D_s^+ \rightarrow K_S^0 \pi^+ \pi^0$ | 0.50 % |
| $D_s^+ \rightarrow K_S^0 \pi^+$ | 0.12 % |
| Total | 18.99 % |

(i) Decay modes for D_s^{+*} .

| Decay channel | Branching ratio |
|-------------------------------------|-----------------|
| $D_s^{+*} \rightarrow D_s^+ \gamma$ | 94.20 % |
| $D_s^{+*} \rightarrow D_s^+ \pi^0$ | 5.80 % |
| Total | 100.00 % |

6.2. Particle Selection and Combination

 (j) Decay modes for hadronic B^+ .

| Decay channel | Branching ratio |
|---|-----------------------|
| $B^+ \rightarrow \overline{D^0} \pi^+ \pi^+ \pi^- \pi^0$ | 2.03 %* |
| $B^+ \rightarrow \overline{D^{0*}} \pi^+ \pi^+ \pi^- \pi^0$ | 1.80 % |
| $B^+ \rightarrow \overline{D^0} \pi^+ \pi^0$ | 1.34 %* |
| $B^+ \rightarrow \overline{D^{0*}} D^{0*} K^+$ | 1.12 %* |
| $B^+ \rightarrow \overline{D^{0*}} \pi^+ \pi^+ \pi^-$ | 1.03 % |
| $B^+ \rightarrow \overline{D^{0*}} \pi^+ \pi^0$ | 0.98 %* |
| $B^+ \rightarrow D_s^+ \overline{D^{0*}}$ | 0.82 % |
| $B^+ \rightarrow D_s^{+*} \overline{D^0}$ | 0.76 % |
| $B^+ \rightarrow D_s^+ \overline{D^0}$ | 0.76 %* |
| $B^+ \rightarrow \overline{D^0} \pi^+ \pi^+ \pi^-$ | 0.68 % |
| $B^+ \rightarrow \overline{D^0} D^{0*} K^+$ | 0.63 %* |
| $B^+ \rightarrow \overline{D^0} \pi^+ \pi^0 \pi^0$ | 0.61 %* |
| $B^+ \rightarrow \overline{D^{0*}} \pi^+$ | 0.52 % |
| $B^+ \rightarrow \overline{D^0} \pi^+$ | 0.48 % |
| $B^+ \rightarrow \overline{D^{0*}} D^0 K^+$ | 0.23 %* |
| $B^+ \rightarrow \overline{D^0} D^0 K^+$ | 0.21 % |
| $B^+ \rightarrow D^- \pi^+ \pi^+ \pi^0$ | 0.20 % |
| $B^+ \rightarrow J/\psi K^+ \pi^+ \pi^-$ | 0.11 % |
| $B^+ \rightarrow D^- \pi^+ \pi^+$ | 0.11 % |
| $B^+ \rightarrow J/\psi K_S^0 \pi^+$ | 9.40×10^{-4} |
| $B^+ \rightarrow \overline{D^{0*}} \pi^+ \pi^0 \pi^0$ | 5.00×10^{-4} |
| $B^+ \rightarrow \overline{D^0} D^+$ | 3.80×10^{-4} |
| $B^+ \rightarrow \overline{D^0} K^+$ | 3.68×10^{-4} |
| $B^+ \rightarrow J/\psi K^+ \pi^0$ | 1.00×10^{-4} |
| Total | 14.64 % |

 (l) Decay modes for semileptonic B^+ .

| Decay channel | Branching ratio |
|---|-----------------|
| $B^+ \rightarrow \overline{D^{0*}} \mu^+$ | 5.68 % |
| $B^+ \rightarrow \overline{D^{0*}} e^+$ | 5.68 % |
| $B^+ \rightarrow \overline{D^0} \mu^+$ | 2.23 % |
| $B^+ \rightarrow \overline{D^0} e^+$ | 2.23 % |
| $B^+ \rightarrow D^{*-} \pi^+ \mu^+$ | 0.61 % |
| $B^+ \rightarrow D^{*-} \pi^+ e^+$ | 0.61 % |
| $B^+ \rightarrow D^- \pi^+ \mu^+$ | 0.21 %* |
| $B^+ \rightarrow D^- \pi^+ e^+$ | 0.21 %* |
| Total | 17.46 % |

 (k) Decay modes for hadronic B^0 .

| Decay channel | Branching ratio |
|--|-------------------------|
| $B^0 \rightarrow D_s^{+*} D^{-*}$ | 1.77 % |
| $B^0 \rightarrow D^{*-} \pi^+ \pi^+ \pi^- \pi^0$ | 1.76 % |
| $B^0 \rightarrow D^{*-} \pi^+ \pi^0$ | 1.50 %* |
| $B^0 \rightarrow D^{*-} D^{0*} K^+$ | 1.18 % |
| $B^0 \rightarrow D_s^+ D^{-*}$ | 0.80 % |
| $B^0 \rightarrow D_s^{+*} D^-$ | 0.74 % |
| $B^0 \rightarrow D_s^+ D^-$ | 0.72 % |
| $B^0 \rightarrow D^{*-} \pi^+ \pi^+ \pi^-$ | 0.70 %* |
| $B^0 \rightarrow D^- \pi^+ \pi^+ \pi^-$ | 0.64 %* |
| $B^0 \rightarrow D^- D^{0*} K^+$ | 0.46 % |
| $B^0 \rightarrow D^{*-} D^0 K^+$ | 0.31 % |
| $B^0 \rightarrow D^{*-} \pi^+$ | 0.28 % |
| $B^0 \rightarrow D^- \pi^+$ | 0.27 % |
| $B^0 \rightarrow D^- \pi^+ \pi^+ \pi^- \pi^0$ | 0.25 %* |
| $B^0 \rightarrow D^- D^0 K^+$ | 0.17 % |
| $B^0 \rightarrow J/\psi K^+ \pi^-$ | 0.12 %* |
| $B^0 \rightarrow J/\psi K_S^0 \pi^+ \pi^-$ | 0.10 % |
| $B^0 \rightarrow D^{*-} \pi^+ \pi^0 \pi^0$ | 0.10 % |
| $B^0 \rightarrow D^- \pi^+ \pi^0 \pi^0$ | 0.10 % |
| $B^0 \rightarrow \overline{D^0} \pi^+ \pi^-$ | 8.40×10^{-4} |
| $B^0 \rightarrow D^- \pi^+ \pi^0$ | 7.80×10^{-4} * |
| $B^0 \rightarrow J/\psi K_S^0$ | 4.36×10^{-4} |
| Total | 12.17 % |

 (m) Decay modes for semileptonic B^0 .

| Decay channel | Branching ratio |
|---|-----------------|
| $B^0 \rightarrow D^{*-} \mu^+$ | 5.01 % |
| $B^0 \rightarrow D^{*-} e^+$ | 5.01 % |
| $B^0 \rightarrow D^- \mu^+$ | 2.17 % |
| $B^0 \rightarrow D^- e^+$ | 2.17 % |
| $B^0 \rightarrow \overline{D^{0*}} \pi^- \mu^+$ | 0.49 % |
| $B^0 \rightarrow \overline{D^{0*}} \pi^- e^+$ | 0.49 % |
| $B^0 \rightarrow \overline{D^0} \pi^- \mu^+$ | 0.21 %* |
| $B^0 \rightarrow \overline{D^0} \pi^- e^+$ | 0.21 %* |
| Total | 15.77 % |

6. Full Event Interpretation

Table 6.3.: Comparison of the number of decay modes and covered branching ratios for each particle in the Full Reconstruction and Full Event Interpretation. Data are from references [77, p. 66] for hadronic, and [80, p. 24] and [81, p. 56] for the semileptonic Full Reconstruction.

| Particle | Full Reconstruction | | Full Event Interpretation | |
|----------------------|-----------------------|--------------|---------------------------|--------------|
| | N_{channels} | Total BR (%) | N_{channels} | Total BR (%) |
| J/ψ | 2 | 11.9 | 2 | 11.87 |
| D^0 | 10 | 37.9 | 14 | 57.85 |
| D^+ | 7 | 29.4 | 10 | 30.85 |
| D^{+*} | 2 | 98.4 | 1 | 67.70 |
| D^{0*} | 2 | 100.0 | 2 | 100.00 |
| D_s^+ | 8 | 17.9 | 10 | 18.99 |
| D_s^{+*} | 1 | 94.2 | 2 | 100.00 |
| B^+ (hadronic) | 17 | 12.0 | 24 | 14.64 |
| B^0 (hadronic) | 15 | 10.4 | 22 | 12.17 |
| B^+ (semileptonic) | 4 | 15.92 | 8 | 17.46 |
| B^0 (semileptonic) | 4 | 14.26 | 8 | 15.77 |

mode. Ideally, the list of decay modes for each particle should cover a large fraction of all decays. Table 6.1 lists the 105 decay modes used by the Full Event Interpretation to reconstruct each particle (in the default configuration), final-state particles are not listed. When all possible decay modes are considered, this corresponds to 2 854 exclusive hadronic B^+ decay channels, 1 868 for hadronic B^0 decays, and 132 channels each for semileptonic decays of B^+ or B^0 . Note that channels that do not provide sufficient statistics to actually train a multivariate classifier are disabled automatically during the training (see Section 6.3.2). Even low-branching-fraction decay modes can however be useful if the created candidates are sufficiently pure.

The number of decay modes for most particles has been increased significantly from the Full Reconstruction introduced in Section 5.3. This can be seen in Table 6.3, which compares the number of modes for each particle and the total branching ratio covered between both algorithms. It should be noted that the branching ratios for some channels present in both algorithms were listed with different branching ratios in the early publications. Some high-branching-ratio modes are not currently available in the FEI, like $D^{+*} \rightarrow D^+\pi^0$ (30.7 % BR), which was removed for technical reasons. They will be added at the nearest opportunity. Overall, only moderate increases in the covered branching fraction are visible for most particles, with the exception of lower coverage for D^{+*} (due to the removed decay mode) and higher coverage for D^0 , where most of the difference originates from the addition of $D^0 \rightarrow K^-\pi^+\pi^0\pi^0$, which has a very high, but badly measured branching ratio.

Significant improvements in the reconstruction efficiency are thus not likely to come about by increasing the covered branching ratio, but rather by adding cleaner channels or improving the selection procedure that candidates are required to pass. These selection steps, and the problems that necessitated their introduction, are discussed in the following section.

6.3. Reducing Combinatorics

As already hinted at in [Section 5.3](#), it is very easy to create tens of thousands of candidates when combining particles, especially in high-multiplicity channels. Without countermeasures, this would consume a great amount of CPU resources for the combinatorial background, to the extent that it would make the Full Event Interpretation unsuitable for processing the billions of collisions (to be) recorded by the experiment. Thus, a central problem of a tag-side reconstruction algorithm is deciding how to reduce the combinatorics to a reasonable level while keeping the efficiency high, i.e. how many particle candidates to cut away and in what way.

Finding appropriate cuts is an optimisation problem, and requires weighing the efficiency of the selection against the additional CPU time required by including the selected particles. Due to the complexity of the problem, cuts are applied in three steps to discard candidates as early as possible: Manual *user-cuts* are applied first, they are followed by automated *pre-classifier cuts* to reduce the number of candidate particles before any expensive actions are performed on them (e.g. vertex fits). Finally, *post-classifier cuts* make use of the additional separation power of the multivariate classifiers to discard additional candidates with high signal efficiency, thus allowing relaxation of the other types of cuts.

As a large fraction of the total CPU time is spent in relatively few events that have very large numbers of candidates, a cut limiting the total number of candidates when making combinations is available as a fourth option. This procedure is special in that it is the only cut used in the Full Event Interpretation that cannot distinguish between signal and background, and is a purely technical measure to reduce the CPU time. However, as the overwhelming majority of candidates in the high-combinatorics events in question are incorrect combinations, the cut will end up reducing the background, and prevent these events from dominating the multivariate classification trainings. With a reasonable threshold, most events should not be affected by this limit at all. In the following, this fourth type of cut is not yet applied; conceptually it should be placed immediately after the pre-classifier cuts.

6.3.1. User-Cuts

As the only manual, free-form cut applied to particle candidates, user-cuts give the opportunity to inject physics knowledge into their selection by adding an explicit cut on an almost arbitrary set of variables to a particle or decay channel. The cut is then applied when first selecting particles for the decay channels in question, before any other cut. In the default configuration for hadronic decays of B mesons, only a user-cut of $m_{bc} > 5.2 \text{ GeV}$ and $|\Delta E| < 0.5 \text{ GeV}$ is applied on the final list of B candidates to reduce the number of candidates very unlikely to be correct. Other types of particles (e.g. final-state or intermediate) do not currently receive a user-cut and rely on the other types of cuts that are the topic of the following sections. This type of cut might also be used to explicitly exclude particles in a region where there is a known difference between data and Monte Carlo.

User-cuts can be configured separately for each decay channel, whereas the pre- and post-classifier cuts are set on a per-particle basis.

6.3.2. Pre-Classifier Cuts

Pre-classifier cuts are the primary method to reduce combinatoric background in the Full Event Interpretation algorithm. Since manually choosing appropriate selection criteria for hundreds of decay channels would neither be very transparent nor help with automating the training process, the concrete cuts are determined according to decay-channel-independent criteria, namely the efficiency and purity specified in the configuration.

These requirements alone do not determine the cuts without ambiguity, so an additional procedure is needed that ideally should take purity differences between decay modes into account. To this end, the Full Event Interpretation requires the ratio s/b of signal to background candidates *at* the cut points to be equal for all decay channels. Note that this does not refer to the signal-to-background ratio of candidates passing the cut, but rather the differential s/b ratio, or, in the case of histograms, the s/b ratio in the particular bin corresponding to the cut value. To some extent this criterion compensates for the wildly different efficiencies and purities in some channels.

This method for choosing a specific cut is identical to the same-slope criterion used for the intermediate cuts in the Full Reconstruction at Belle. This can be seen by the following: Let the constant

$$c = \frac{s}{b} = \frac{s(x)}{b(x)}$$

be the common cut on different decay channels. After rearranging and integrating both sides of the equation with respect to the cut variable x , we arrive at

$$\int dx s(x) = c \int dx b(x).$$

This integration over x can be thought of as collecting all candidates which pass a given one- or two-sided cut on x , but the specific range integrated over is irrelevant for this argument. We can define the numbers of signal and background events passing the selection (i.e. with the same integration range) as

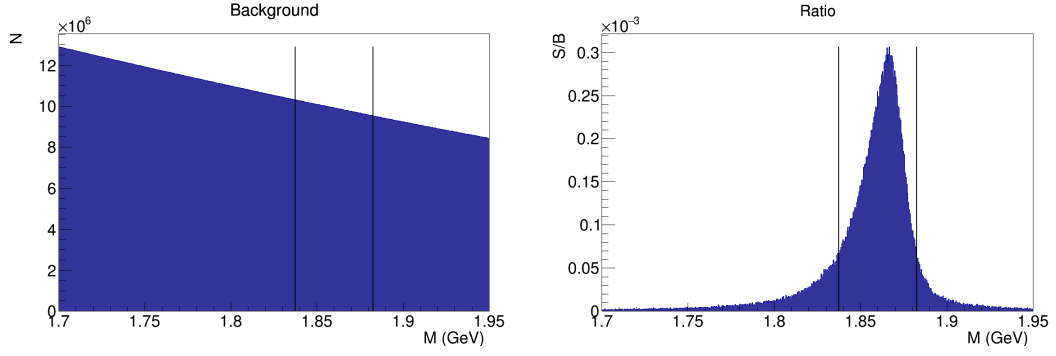
$$S(x) = \int dx' s(x'), \quad B(x) = \int dx' b(x').$$

Substituting this definition in the previous equation yields $S = cB$, which allows us to calculate the derivative with respect to the total number of background candidates B :

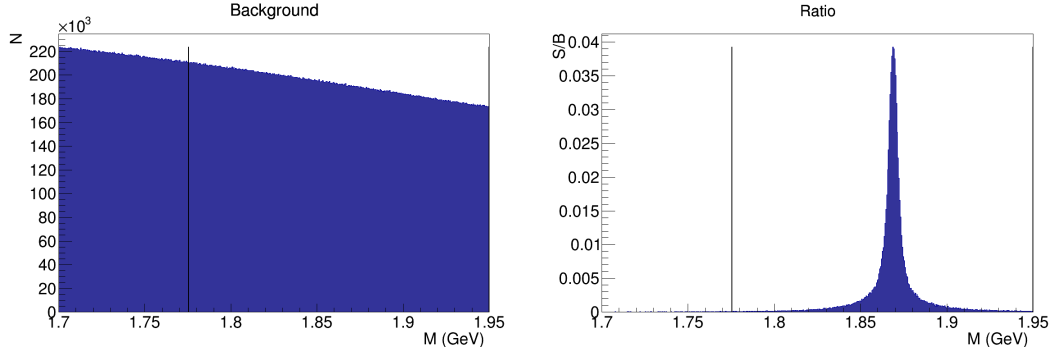
$$\frac{dS}{dB} = c.$$

This shows that the common cut value c is equal to the slope in a plot of S over B , or the inverse of the slope in [Figure 5.3](#), which plots B over S . The simpler reformulation of this principle makes it much easier to implement and interpret. It should be noted that choosing cuts at a fixed s/b value also easily accommodates two-sided cuts in case the distribution of s/b in the cut variable is not monotonous. The implementation does not take into account the possibility of variable distributions with multiple peaks, and would most likely not give reasonable results. For the pre-cut variables used in the following (invariant mass, released energy, and product of daughter classifier outputs) this is not the case.

6.3. Reducing Combinatorics



(a) $D^+ \rightarrow \pi^+ \pi^+ \pi^- \pi^0$ (efficiency: 72.9 %, purity: 0.018 %)



(b) $D^+ \rightarrow K^- K^+ K^+$ (efficiency: 95.9 %, purity: 0.24 %)

Figure 6.3.: Distribution of the number of background candidates (left) and s/b (right) over the invariant mass M for two D^+ decay channels with different purity. The chosen two-sided pre-classifier cuts are shown with vertical lines.

The concrete value for c is then determined simultaneously over all decay channels using the efficiency and purity values in the pre-cut configuration: First, an s/b cut value that results in a signal efficiency equal to the required efficiency is determined; if the purity requirement is not met, the cut is tightened until it is (i.e. configured purity overrides efficiency). If, after applying the cut, not enough signal candidates remain to perform a reasonable classifier training, the channel is ignored and the cut-optimisation procedure repeated without the channel. Currently, at least 1 000 candidates are required for both signal and background. Finally, the s/b cut is converted into (possibly multi-sided) cuts in the separation variable. Since more impure channels have lower overall s/b levels, this produces broader cuts in pure channels, and narrower cuts in impure channels. Very impure channels, where the purity requirement will result in an extremely narrow cut around the maximum of s/b , are usually discarded.

Instead of the product of multivariate classifier outputs, we prefer to use the invariant mass as a separation variable where applicable. Like the product of classifier outputs, the invariant mass can be calculated quickly while each candidate is being considered, but it also has the advantage of having an intrinsic physical meaning. This also makes it useful as a crosscheck

6. Full Event Interpretation

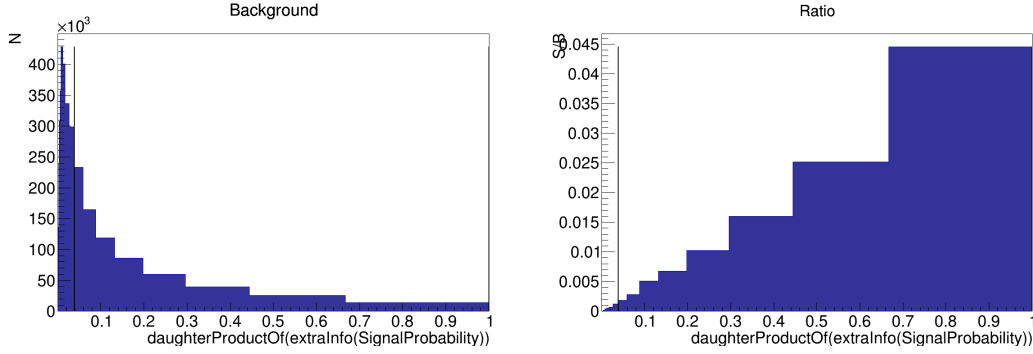


Figure 6.4.: Distribution of the number of background candidates (left) and s/b (right) over the product of daughter classifier outputs for $B^0 \rightarrow \overline{D^0}\pi^- e^+$. The chosen one-sided pre-classifier cut is shown as a vertical line. (efficiency: 80.1 %, purity: 0.6 %)

to guard against, e.g., systematic shifts in mass, or problems in matching with Monte Carlo truth. However, the variable used for this type of cut is part of the configuration and can easily be replaced with something else, e.g. the released energy Q , which has more separation power than the invariant mass for $D^* \rightarrow D\pi$ decays. For B mesons, using the invariant mass would also be sub-optimal, since it is highly correlated with the beam-constrained mass m_{bc} , which we intend to use as a control variable. To avoid these correlations, the product of the classifier outputs of each daughter of the B meson candidate is used as a separation variable (this is equal to the $o_{NB, \text{prod}}$ variable used for all particles at Belle).

An example of this algorithm for two (out of ten) D^+ decay modes is shown in [Figure 6.3](#). It can be seen that choosing a common s/b value as a cut criterion takes into consideration the different shape of the distribution of signal candidates and, e.g., produces an asymmetric cut for the broader s/b distribution in [Figure 6.3a](#). Additionally, the purer channel in [Figure 6.3b](#) with a much higher overall s/b ratio receives a broader cut to include more candidates. [Figure 6.4](#) shows the monotonous s/b distribution when using the product of daughter classifier outputs (or $o_{NB, \text{prod}}$) for a B meson decay channel. Here it produces one-sided cuts, and will again discard more background candidates if the separation power of the variable is higher. One can also see that a custom binning can be used, which helps avoid effects of almost empty bins with very unequal distributions like shown here, where only very few candidates have values close to 1.

To apply the cut-determination algorithm described above, the distribution of s/b in the designated cut variable has to be obtained first; this involves making all combinations passing the user-cut, calculating the value of the variable for each candidate, determining whether a candidate is true or false using Monte Carlo information and adding the calculated value in the appropriate histogram. This is done independently for each decay mode. Since running the Monte Carlo matching algorithm (see [Section 4.3](#)) for each candidate would be very expensive, this is optimised using the additional knowledge of which decay mode was reconstructed. First, the Monte Carlo particles are searched for a particle decaying in the specified decay. For each matching MC particle found, it is checked if all of its daugh-

ters are already reconstructed, and if they exist, they are combined to a signal candidate. Finally, the Monte Carlo matching is run on this handful of signal candidates and used with a user-specified target variable (e.g. `isSignal` or `isSignalAcceptMissingNeutrino`) to ensure the signal definition is identical to a normal, slower reconstruction. Thus, running the relatively expensive Monte Carlo matching is limited to the (rare) case of the decay existing in Monte Carlo and the candidate already being mostly correct. The entire process is encapsulated in the `PreCutHistMaker` module, which produces separate histograms for signal and signal+background given a decay string and variable.

6.3.3. Post-Classifier Cuts

In contrast to the pre-classifier cuts, which reduce the combinatorics before the training of the multivariate classifier but are rather simple (rectangular) in nature, post-classifier cuts can make use of the classifier output to improve the selection: After the multivariate classifiers have been applied, a user-specified cut can be applied on the output probability. This reduces combinatorics in further stages and makes use of all the classifier input variables and their correlations. Achieving a similar reduction with pre-classifier cuts alone would be possible, but would also require significantly harder cuts because less information is available (i.e. it only involves a single variable).

In the default channel configuration cuts at low classifier output values are included, which discards mostly background candidates without affecting correctly reconstructed particle candidates. To the extent that the output can be interpreted as a probability, the selection is based on the amount of signal and background at that point. Even the manual specification of a cut value is in this case fairly generic and independent of the specific decay channel. Post-classifier cuts are thus applied on a per-particle basis, which differs from the decay-channel specific user- and pre-classifier cuts.

6.4. Classifier Trainings

Multivariate classifiers lie at the heart of the Full Event Interpretation algorithm and serve to distinguish correct from incorrect candidate particles in a highly automated fashion. This is useful both for combined and final-state particles, and the output can be used both to immediately discard some candidates, and as input in later stages. The classifiers used by the FEI are created through the TMVA interface introduced in [Section 4.4](#), which takes care of passing the data from BASF2 to the classification method.

Two significant problems when automatically training classifiers as part of the Full Event Interpretation are the very large number of training events and wildly differing purity between different channels, e.g. many channels will create hundreds of millions of candidates, with most candidates being correct for a π^\pm training, but only ten thousand candidates might be correct for a B -level decay channel. Most of these background candidates will not add more information about the distribution of the input variables, but each candidate increases the total sample size and thus the total disk and memory requirements for the training. Since both of these resources are limited, the FEI limits the number of candidates used as input data and reweighs the inputs appropriately. Before the training is started, the total number of candidates in each class (i.e. signal and background) is determined either

6. Full Event Interpretation

from the pre-classifier cut histograms or (for final-state particles) in an additional run over the data. The total number of candidates is then limited to below ten million by skipping additional candidates (skipped candidates are distributed uniformly across the entire data) and reweighting the output of the expert using the known signal-to-background ratio. This process limits the training input file size to below 3 GB per channel, each of which is trained in a separate process using the `externTeacher` stand-alone training utility (which also allows starting multiple trainings in parallel).

Due to its availability and superior performance the FastBDT method is used by default, but this can be replaced by other algorithms from the configuration if deemed appropriate. Given an appropriate fit model, it is also possible to use the `sPlot` formalism to create a data-driven training. Due to the relatively large effort required to create an optimised fit model for a single decay channel, this will most likely only be used for channels where there is a known difference between Monte Carlo and data. Besides the classification method, users can configure the hyperparameters of the method (with sane defaults being provided) and the variables used as training input. Finally, the target variable is used to distinguish between signal and background in the classifier training. For most channels, the `isSignal` variable is used, for semileptonic B decays this changes to `isSignalAcceptMissingNeutrino`. For final-state particles (charged tracks and photons), the signal definition explicitly excludes secondary particles, which are particles created through interactions with the detector material. Since this means that their production is independent of the decay channels reconstructed by the Full Event Interpretation, reconstructing them is not beneficial; marking them as background thus increases the purity of certain channels (especially J/ψ and semileptonic B decays).

The input variables, as the most important factor for the separation power of the classifiers (for a given classification method), are listed and discussed in the following sections. While users can choose their own sets of variables, doing so can be tricky and it is advisable to use the variables of the default configuration described here.

6.4.1. Inputs for Charged Final-State Particles

For charged final-state particles (i.e. π^\pm , K^\pm , e^\pm , and μ^\pm), the multivariate classifiers serve as a high-level replacement of and improvement upon the particle identification (PID) facilities provided by the analysis software. Thus, the input variables consist mostly of PID variables, plus variables that describe the charged track itself.

PID variables Most of the information originates from the particle identification variables, where the probability of the track being of one particle hypothesis versus another are used. In this case, the probabilities for kaon vs. pion, electron vs. pion, muon vs. pion and proton vs. pion are used; the data from dE/dx , ARICH, TOP, and (for electrons) ECL are kept separate to give the classifier more information, as the combination of all detectors using combined likelihoods is not yet performing as expected. The direct combination is however also used as an input to ensure a certain minimum of separation.

Track momenta (p , p_t , p_z) Since the performance of the PID detectors depends strongly on the momentum of the tracks, the total momentum is included as an input variable;

the transverse and z component of the momentum are also included to describe acceptance effects.

Track quality The quality of the track might also influence the performance of the PID, e.g. if the momentum measurement or extrapolation to outer detectors is wildly inaccurate. To pass information about the quality to the classifier, the χ^2 probability of the track fit is also passed as an input. In the future, this might be improved by adding other related variables, e.g. hit patterns, which describe where a track was detected in the silicon detectors and the drift chamber.

Impact parameters (d_ρ, d_z) A primary way to distinguish primary from secondary particles is through the track impact parameters; in this case the radial and z distances (d_ρ, d_z) to the track point closest to the beam line (perigee) are used as input variables. Secondary particles are usually produced some distance away from the interaction point and thus should have larger values.

6.4.2. Inputs for Photons

Cluster shape (E_9/E_{25}) The shape of clusters is mainly described by the E_9/E_{25} variable, which is the ratio of total energies in the inner 3×3 crystals to that in the outer 5×5 crystals. Higher values are expected for true photons originating from the interaction point, compared to beam background or other photons with a different incident angle.

Photon energy and direction Both energy and direction (in the form of p_t and p_z) are included in the input.

Cluster quality The variable `goodGamma` contains optimised cuts that separate between true photons and beam background and noise, this quantity is added in two versions for calibrated and uncalibrated photon energy.

Cluster timing The photomultipliers of the ECL have a fairly good time resolution, so the detection time of a cluster (`clusterTiming` variable) can help identify energy depositions created from beam background.

Detector region Because of varying background levels and different crystal geometries in the barrel and end-cap regions of the ECL, the `clusterReg` variable is used to include the position of the current cluster in the three categories forward, backward and barrel.

6.4.3. Inputs for π^0

Invariant mass The invariant mass is the most important variable for this classifier training, and tends to be within 10 MeV of the π^0 mass (135 MeV) for signal.

Daughter classifier output Since correct candidates cannot usually be created from incorrectly reconstructed daughter particles, the classifier output of each daughter is used as an input.

6. Full Event Interpretation

Angle between photons Given that both photons are (in the absence of other information) assumed to originate from the interaction point, the angle between them captures most of the kinematic information.

Energy and direction Similar to photons or charged final-state particle candidates, the energy and direction (in the form of E , p_t , p_z) is included for neutral pions.

6.4.4. Inputs for $D_{(s)}^{(*)}$ and J/ψ Mesons

Released energy The released energy Q is defined as the mass of the D candidate minus the masses of all daughter particles.

Daughter classifier output Since correct candidates cannot usually be created from incorrectly reconstructed daughter particles, the classifier output of each daughter is used as an input. The product of the output of all daughters is also included (equivalent to $\mathcal{O}_{\text{NB, prod}}$).

Invariant masses of daughters The invariant masses of pairs, triples, etc. of daughters are also used as an input, which allows the classifier to learn about possible intermediate states that are not explicitly reconstructed. For example, encountering values close to the invariant mass of a ρ or a^0 meson indicates that the involved daughters are more likely to share the same mother, thus increasing separation.

It should be noted that the usefulness of the invariant masses depends on the quality of the Monte Carlo for each of the involved decay channels and resonances. Larger differences between Monte Carlo and data in these variables might result in undesirable MC–data differences in the reconstruction efficiency; further tests are needed to estimate their size.

Distance to daughter vertices For correctly reconstructed candidates this corresponds to the flight length of the D^0 for $D^* \rightarrow D^0 X$, or for charged final-state particle daughters compares the fitted decay vertex to the impact parameters of the daughter track. These variables are given in the rest frame of the candidate.

Daughter momenta The momentum of each daughter in the candidate rest frame is used as an input, and can capture some of the kinematics of the decay.

Daughter (vertex) quality Vertex fit quality can vary greatly between different candidates and is usually described by the χ^2 probability. Low values indicate a bad fit, while negative values are used for cases where the fit did not converge – both cases suggest that the fitted particles do not originate from the same vertex. For final-state particle daughters, the meaning of the `chiProb` variable changes to the χ^2 probability of the track fit.

Vertex quality Like the vertex fit quality of the daughters, the fit quality of the combined particle itself is also included.

Direction of daughters Using the `decayAngle(i)` variable, the angle between the momentum vectors of the candidate and its i th daughter is also included.

6.4.5. Inputs for K_S^0 Mesons

The input variables for K_S^0 mesons are mostly identical to those for $D_{(s)}^{(*)}$ or J/ψ mesons, with the addition of variables for the vertex position – K_S^0 have a typical flight length of a few centimeters – and the candidate energy, which helps relate flight time and flight length.

6.4.6. Inputs for B Mesons

The main difference between B -level trainings and those for $D_{(s)}^{(*)}$ mesons is their exceptional position as final output of the Full Event Interpretation, so most of the variables from the previous section are retained. The invariant mass variables are removed to avoid a correlation with the beam-constrained mass m_{bc} that is frequently used as a control variable. In addition to the variables also used at the D -level, the following inputs are used:

Vertex position In contrast to the D -level classifiers, the decay vertex position is also used directly here by adding the distance to the interaction point in three dimensions, plus the total distance divided by one standard deviation assuming a Gaussian distribution of the error on the vertex (via the `significanceOfDistance` variable). As the interaction point is presently not well simulated (B mesons originate from $(0, 0, 0)$ in official Monte Carlo samples), this may be replaced with variables that do not rely on the global position of the candidate. One possibility may be to use the flight length of the D daughters by comparing B and D vertices.

Energy difference As mentioned in [Chapter 5](#), the energy difference ΔE can separate correctly reconstructed B meson candidates from those with missing or misidentified particles and thus can significantly improve the classification. It is also only weakly correlated to m_{bc} , and most strongly so for background-like candidates that are removed using the user-cut introduced in [Section 6.3.1](#).

6.5. Automatic Reporting

Given that many central aspects of the Full Event Interpretation algorithm are automated, like the determination of pre-classifier cuts and by extension the channel selection (since impure channels can be discarded), it is vital for users to be able to check the results of this process. For this reason, the FEI also automatically produces a report that contains statistics and plots for each included particle (typically hundreds of pages) as well as summary information on the training as a whole.

The following items are included in the report:

Efficiency summary One of the most important sections in the report are the tables of the efficiencies and purities for each particle included in the channel configuration. Both efficiencies and purities are listed separately for each type of cut: Starting with the raw reconstruction efficiency (just combining daughters, with no cuts), it shows the effect of a user-cut, if present, followed by the pre- and post-classifier cuts. For final-state particles, the tables are limited to the detector and post-cut efficiencies and the corresponding purities.

6. Full Event Interpretation

Table 6.4.: Example efficiency and purity summary for final-state particles, showing values before and after the applied post-cuts.

| Final-state particle | Efficiency in % | | Purity in % | |
|----------------------|-----------------|----------|-------------|----------|
| | recon. | post-cut | recon. | post-cut |
| π^+ | 78.68 | 77.94 | 66.898 | 85.250 |
| e^+ | 70.04 | 63.45 | 4.866 | 71.223 |
| μ^+ | 86.22 | 52.50 | 4.489 | 52.289 |
| K^+ | 79.50 | 76.68 | 12.230 | 74.818 |
| γ | 89.02 | 88.20 | 53.761 | 56.645 |

Table 6.5.: Example per-particle efficiency and purity summary, showing values before and after the applied user-, pre-classifier- and post-classifier-cuts.

| | Efficiency in % | | | | Purity in % | | | |
|--------------------|-----------------|----------|---------|----------|-------------|----------|---------|----------|
| | recon. | user-cut | pre-cut | post-cut | recon. | user-cut | pre-cut | post-cut |
| π^0 | 87.51 | | 67.49 | 44.79 | 2.713 | | 6.046 | 37.817 |
| K_S^0 | 40.09 | | 38.13 | 34.32 | 1.911 | | 5.959 | 79.285 |
| D^0 | 17.01 | | 15.94 | 9.47 | 0.018 | | 0.334 | 18.099 |
| D^+ | 10.81 | | 10.15 | 6.91 | 0.018 | | 0.337 | 21.044 |
| D^{+*} | 3.15 | | 3.00 | 2.98 | 1.285 | | 57.333 | 63.893 |
| D^{0*} | 5.37 | | 5.36 | 2.44 | 0.103 | | 0.127 | 14.274 |
| D_s^+ | 6.85 | | 6.27 | 3.30 | 0.009 | | 0.065 | 15.777 |
| D_s^{+*} | 2.92 | | 2.87 | 1.64 | 0.353 | | 0.533 | 14.554 |
| J/ψ | 9.58 | | 7.95 | 7.95 | 0.428 | | 41.315 | 41.315 |
| B_{had}^0 | 0.45 | 0.45 | 0.34 | 0.30 | 0.009 | 0.781 | 1.280 | 1.324 |
| B_{sl}^0 | 1.01 | | 0.92 | 0.92 | 0.707 | | 1.767 | 1.767 |

Examples can be found in Tables 6.4 and 6.5 for final-state and combined particles, respectively. In both tables it can be seen that both pre- and post-classifier cuts usually result in great increases in purity, with only a moderate effect on the efficiency. Only the aforementioned user-cut of $m_{bc} > 5.2 \text{ GeV}$ and $|\Delta E| < 0.5 \text{ GeV}$ was applied to hadronic B meson candidates, gaining a factor > 80 improvement in purity with no significant change of the efficiency. If final state-particles produced through interactions with the detector material (i.e. secondary particles, see Section 6.4) were accepted as correct, the efficiencies in Table 6.4 might rise over a hundred per cent. The training associated with these two tables was performed on only one million $B^0\bar{B}^0$ events, so the efficiencies should not be taken to be indicative of the final performance of the Full Event Interpretation.

As it is possible that a certain decay in the Monte Carlo is reconstructed multiple times, using the number of correctly reconstructed candidates in the efficiency is not entirely accurate. An example is the decay $B^+ \rightarrow \bar{D}^{*0} D_s^+ (\rightarrow K^- K^+ \pi^+)$, which can have the same final-state particles as $B^+ \rightarrow \bar{D}^{*0} D^0 K^+$ with $D^0 \rightarrow K^- \pi^+$. To avoid including the same decay more than once in the efficiency, only one correct reconstruction is accepted per Monte Carlo Particle for B mesons. The same procedure is also applied to the control plots discussed in the following paragraphs.

Control plots for B mesons Since the FEI aims at producing high-quality B_{tag} candidates for combination with the signal-side selection, a number of special control plots are provided for B mesons. This includes the receiver operating characteristic for the B_{tag} candidates (which is a mixture of candidates from different decay channels with their own, separate multivariate classifier) and plots of an appropriate control variable (see Section 5.1) for the given particle. The control plots for hadronic and semileptonic channels are presented in Figures 6.5a and 6.5b for different cuts on the classifier output and show both the purity-enhancing effect of the cut and how the shape changes under their influence (if at all). For hadronic B candidates the beam-constrained mass m_{bc} is used as a control variable, which shows a clear peak at around 5.28 GeV with an increasingly suppressed background for harder cuts on the classifier output. For B candidates reconstructed in semileptonic decay channels, not enough kinematic information is available to yield a reasonable mass resolution, so instead of m_{bc} the cosine of the angle between the estimated B direction and the $D^{(*)}l(\pi)$ system assuming a single missing neutrino is used as a control variable. Correctly reconstructed candidates should satisfy $|\cos\theta_{B,D^{(*)}l}| \leq 1$, whereas background has a broader distribution [11, p. 106]. The shape is however slightly asymmetric and contains background processes that also peak in the same region.

One important feature is that the classifier output is not strongly correlated to m_{bc} or $\cos\theta_{B,D^{(*)}l}$, otherwise cuts would artificially produce a signal-like shape in the background-only distribution.

Pre-classifier cut histograms For each decay channel, the distributions of the variable used for the pre-classifier cuts are shown, as in Figures 6.3 and 6.4. This also includes the exact values used in the cut, as well as its purity and efficiency (which may differ significantly from both the per-particle values and the requirements in the configuration).

6. Full Event Interpretation

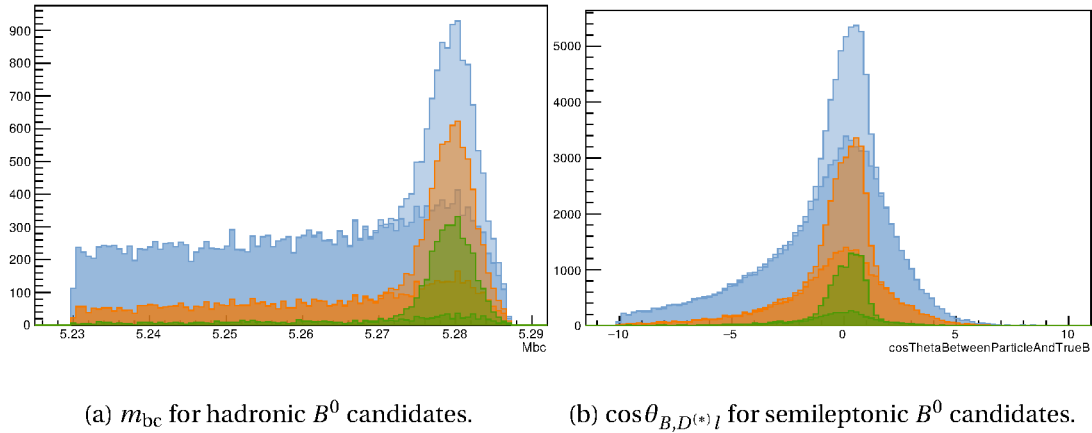


Figure 6.5.: Control plots for hadronic (a) and semileptonic (b) B_{tag} candidates, showing the effect that different cuts on the classifier output o_{MVA} have on the distribution. From top to bottom: $o_{\text{MVA}} > 0.01$, $o_{\text{MVA}} > 0.1$, and $o_{\text{MVA}} > 0.5$. Lighter shades denote signal, darker shades are used for background.

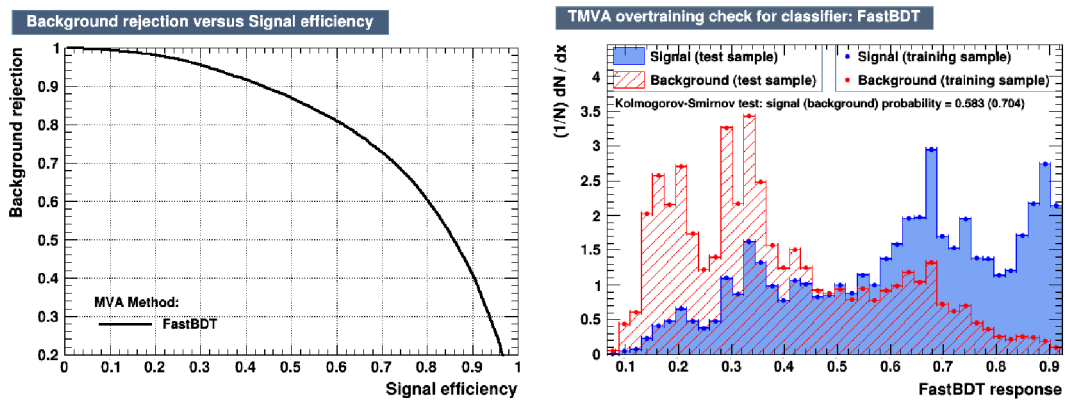


Figure 6.6.: Example MVA control plots for photons, with the receiver operating characteristic showing moderate separation (left), which is also reflected in the shape of the overtraining check plot (right). Distributions of test and training samples are compatible, which can be confirmed visually or by checking the output of the Kolmogorov–Smirnov test. (Plots contrast-enhanced for better readability.)

Control plots for multivariate classifiers The pre-classifier cut histograms are followed by detailed information and control plots for the multivariate classifier trained for each decay channel or final-state particle, consisting of the receiver operating characteristic (ROC), an overtraining check and a plot of the purity per bin of the classifier output. Examples of these plots are shown in [Figure 6.6](#). The ROC curve plots the purity over the signal efficiency for different cuts on the classifier output and provides a good impression of the overall separation power of the multivariate classifier. The overtraining check plot shows the distribution of the classifier output for signal and background candidates for two samples: The training sample used as input to the classifier training, and the test sample, which is an independent sample not used for the training. For a good training, the distributions of these two samples should be identical, while differences indicate overtraining of the classifier, which is equivalent to worse classifier performance on any sample not used in the training. To quantify the compatibility of the distributions, a Kolmogorov–Smirnov test is performed on both the signal and background distributions. The resulting two values lie in the interval $[0, 1]$, with differing distributions (i.e. overtraining) producing low values, and histograms generated from the same distribution creating values that are uniformly distributed in $[0, 1]$.

A third plot, graphing purity in bins of the classifier output, shows whether the output for this decay channel can be interpreted as a probability. Depending on the TMVA method used for classification, this may not be guaranteed; with FastBDT this is only true to some degree, while for NeuroBayes it usually holds. TMVA can however also perform a corrective transformation that makes the output more probability-like. This ‘diagonal plot’ is shown in two places: for each multivariate classifier, where the uncorrected output (and without applying prior information) is shown, and for each particle, where the plot shows corrected outputs for all decay channels of the particle combined. For good trainings, the points in the second plot should lie on the diagonal (purity = σ_{MVA}), while the per-classifier plot should simply show a smooth distribution without excessive jumps. Examples of both plots are shown in [Figure 6.7](#)

Besides the control plots, the input variables used in the classifier are also listed and ranked by their importance in the training (the meaning of this rank might change depending on the type of multivariate classifier used). Variables that are deemed unimportant during the training are also distinguished.

For each particle (i.e. the union of all its decay modes), a plot of the purity in each bin of the classifier output is shown. This allows evaluation of the probability interpretation after mixing together particles reconstructed in different decay channels and tends to be much better than the same plot for individual channels. Strong fluctuations might hint at problems like insufficient statistics.

CPU usage statistics Also tabulated is the CPU time needed for the reconstruction tasks of each channel, as shown in [Table 6.6](#). The table contains both absolute and relative CPU time spent on each channel, the time necessary to reconstruct a single candidate (distinguishing between any candidate at all and true candidates, specifically) and a stacked bar chart showing the relative time spent in each module. Since the time needed for the training or cut determination is not included, the table reflects the CPU

6. Full Event Interpretation

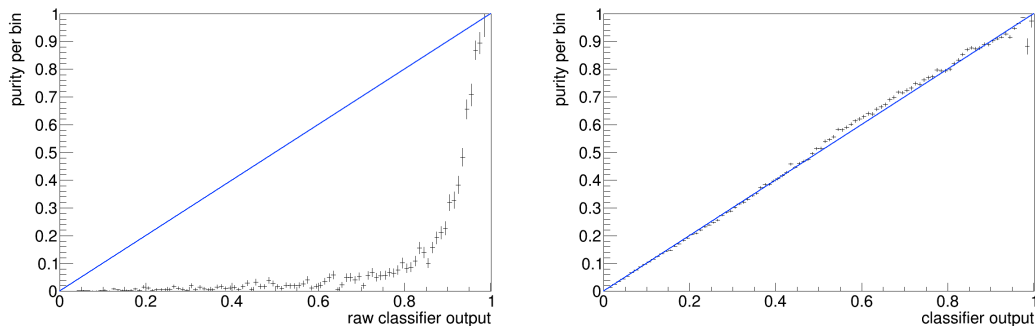


Figure 6.7.: Purity per bin plotted over raw classifier output for a B^+ decay channel (left), and purity per bin plotted over the corrected classifier output over the combined set of all B^+ decay channels. For the raw classifier output, the data points are far from the diagonal, but show a smooth distribution, when errors are considered. The corrected combined output follows the diagonal quite closely.





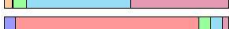




time needed to apply a Full Event Interpretation training. This chart is primarily useful for developers of the analysis software, e.g. it plainly shows that vertex fitting is the most expensive task in most decay channels, which makes it a target for improvement. The significantly larger amount of CPU time consumed by `ParticleLoader` for K^+ compared to other charged final-state particles is an artefact of it being the first particle created, which results in a number of caches being filled here for the first time.

Since cuts on candidates are chosen in a way (see [Section 6.3.2](#)) that stresses the interchangeability between channels in regard to variations of the cut, the CPU time spent per channel is not optimised directly. In most cases, the purity set in the channel configuration should ensure that the CPU time required stays manageable, since it is proportional to the number of candidates. The proportionality constant, i.e. the time per candidate, is not known a priori and depends on the channel's multiplicity, the types of daughter particles, and details of the vertex fitting algorithm. In extreme cases, a single channel with high combinatorics might even take up a significant fraction of the total time. A future optimisation opportunity might be to use the detailed per-channel information shown in this table in the cut determination, e.g. to optimise cuts for the time required to create a true candidate.

The tabulated information is available in the `ProcessStatistics` object, which tracks the CPU time spent in each module. Since modules are assigned unique names by the FEI according to the associated particle or decay channel, the CPU time can be easily extracted and matched with other information like the number of candidates. As `ProcessStatistics` objects are mergeable (see [Section 3.6](#)), this information is also available when using BASF2 with multiple processes, and can also be merged when the execution of the Full Event Interpretation is spread over a computer cluster.

Configuration summary The entire content of the decay channel configuration, including the detailed settings for cuts, input variables and MVA hyperparameters is also repro-

Table 6.6.: Excerpt of the CPU usage statistics, showing total CPU time spent in event() calls for each channel. Bars show ParticleLoader, ParticleCombiner, ParticleVertexFitter, MCMatching, TMVAExpert, and others, in this order.

| Decay | CPU time | by module | per (true) candidate | Relative time |
|-----------------------------------|----------|---|---------------------------------------|---------------|
| π^+ | 15m46s |  | $72\mu\text{s}$ ($127\mu\text{s}$) | 0.68% |
| e^+ | 13m13s |  | $118\mu\text{s}$ (1ms) | 0.57% |
| μ^+ | 12m52s |  | $114\mu\text{s}$ (1ms) | 0.55% |
| K^+ | 22m55s |  | $189\mu\text{s}$ ($907\mu\text{s}$) | 0.98% |
| γ | 8m46s |  | $29\mu\text{s}$ ($62\mu\text{s}$) | 0.38% |
| $\pi^0 \rightarrow \gamma \gamma$ | 1h31m |  | $530\mu\text{s}$ (2ms) | 3.94% |
| $K_S^0 \rightarrow \pi^+ \pi^-$ | 39m54s |  | $317\mu\text{s}$ (6ms) | 1.71% |
| $D^0 \rightarrow K^- \pi^+$ | 2m11s |  | $741\mu\text{s}$ (5ms) | 0.09% |
| $D^0 \rightarrow K^- \pi^+ \pi^0$ | 42m36s |  | $627\mu\text{s}$ (73ms) | 1.82% |

duced in a more human-readable form. This enables direct comparisons between trainings using only the corresponding FEI reports and ensures the results can be reproduced easily. The summary also contains the decay-mode identifier (see Section 4.1.4), which can be used to quickly identify in which decay mode each particle candidate produced by the FEI was created.

Taken together, the plots and statistics contained in the Full Event Interpretation Report give a comprehensive view of the entire reconstruction chain. This allows users to monitor the training performance and pinpoint regressions, to optimise certain parts of the configuration (e.g. the selection of variables or channels), and for cross-checks of both the analysis software and the input data. These cross-checks in particular have been found to be very useful, and have been used to discover problems in the Monte Carlo matching, higher-than-expected CPU usage of individual modules, and systematic shifts in the photon energy.

An example FEI report for a minimal configuration with only three final-state particles, one D^+ , and one B^0 decay channel is shown in Appendix A.

6.6. Training Modes

The Full Reconstruction introduced in Section 5.3 was only trained in a single, fixed way: using generic Monte Carlo, where events with correctly reconstructed B_{tag} mesons always also contained another B meson decaying generically. This is referred to as *generic* training in the following. This section discusses the disadvantages of this approach for many applications and which alternatives can be pursued to avoid them.

6.6.1. Generic Training

A strong advantage of the generic training was that a single centrally produced B_{tag} sample could be provided to all users, with manageable costs in CPU time and disk space. When used in analysis, however, a signal selection would be applied to the remainder of the event after

6. Full Event Interpretation

subtracting the B_{tag} , likely discarding a majority of events. The resulting set of events and the B_{tag} candidates within it may then have properties quite different from the sample used during the original training. As an illustrative example, consider a signal selection for studying $B^- \rightarrow l^- \bar{\nu}_l$: As exactly one lepton track can be found on the signal side, a cut discarding events with additional tracks (besides the lepton and those used in B_{tag}) greatly increases the purity of the selection. For the tag-side, on the other hand, this can be problematic. Since the prior distributions of classifier inputs are changed significantly between the training sample and the analysis sample that includes the signal selection, this introduces a bias in the classifier output and violates the assumptions allowing its interpretation as a probability. In particular, as the signal selection will likely have a different effect on different tag-side decay modes (e.g. $B^+ \rightarrow \bar{D}^0 \pi^+$ vs. $B^+ \rightarrow \bar{D}^0 \pi^+ \pi^+ \pi^- \pi^0$), one can no longer assume that the classifier output for candidates can be interpreted in the same way. Additionally, the overall performance of the tag-side reconstruction is reduced (when compared to no signal selection), since much effort is wasted on events that are later discarded.

6.6.2. Analysis-Specific Training

The highly automated training process made possible by the implementation of the FEI, however, permits an entirely different mode of training: To avoid the performance reduction of multivariate classifiers and biases the generic training may introduce, the user can create trainings that are specific to their analysis (and signal channel); the training is thus referred to as *specific*. These trainings are optimised for the specific prior distributions in the analysis and profit from any reduction of combinatorics that the signal-side cuts may introduce. This also has the potential to permit the use of more channels or softer cuts. As here, finally, the classifier output can be interpreted as a probability, the particular decay channel used for a tag-side candidate should not matter too much for an analysis using it. In particular, the probability also makes hadronic and semileptonic Full Event Interpretation outputs more comparable, and may allow for an easier combination of both in a single analysis. An additional benefit of the specific training is that it focuses on events relevant to the analysis. For example, with a signal selection that includes only few tracks it does not make sense to look at events that have more tracks than the tag side (plus signal side) can maximally reconstruct. Instead, events with many tracks (which consume the majority of the CPU time for any FEI training) can be discarded early, allowing the training to concentrate on and optimise for low-multiplicity events.

Technically, the chain of analysis modules for an analysis-specific FEI training differs somewhat from the case of a generic training: As a first step, the user needs to define a BASF2 path containing their signal reconstruction, i.e. create final-state particles, combine them into B_{sig} candidates and select those most likely to conform to their signal channel. To benefit from the effects of discarding inviable events, the selection should also include a cut that limits the total number of tracks in the event. This path is handed over to the Full Event Interpretation together with the name of the particle list containing signal candidates. For each candidate, a `RestOfEvent` object is created that contains references to all remaining track or cluster objects not used in the candidate. The object carries a relation back to the candidate; each signal candidate thus separates the event into objects belonging to the signal or tag side (i.e. the rest). Using the sub-event iteration functionality introduced in

Section 3.3.2, all actions of the FEI are then performed inside a loop over the RestOfEvent objects defining possible tag-sides. All final-state particles used in the training are selected with the condition `'isInRestOfEvent == 1'`, which will reject all detector signals not in the tag side of the current loop iteration. As the iteration using `for_each()` is entirely transparent to the modules involved, the remainder of the FEI training continues as described in previous sections.

The effect of performing an analysis-specific Full Event Interpretation training was already studied extensively in reference [59], so only an overview will be given in this section. For the study, a generic training was compared with a specific training with a signal-side selection of $B^+ \rightarrow \tau^+ (\rightarrow \mu^+ \bar{\nu}_\mu \nu_\tau) \bar{\nu}_\tau$ decays, which is arguably a selection where the difference between both approaches is most pronounced.

The signal selection procedure for the specific training uses the standard analysis tools and is shown in the following listing (from [59, p. 69], updated to reflect software changes):

```
fillParticleList('mu+', 'muid > 0.6 and nTracks <= 12')
reconstructDecay('tau+ -> mu+')
reconstructDecay('B+:sig -> tau+')
matchMCTruth('B+:sig')
```

Besides the cut on $n_{\text{Tracks}} \leq 12$ (reflecting the maximum multiplicity of exclusive channels that could be reconstructed by the FEI), a user-cut (i.e. a manual cut applied before any other cut, see **Section 6.3.1**) was added for B_{tag} candidates that required them to be able to produce a correct $\Upsilon(4S)$ when combined with the signal selection (i.e. no tracks should remain after the combination). The Full Event Interpretation was then trained on 40 million events of Monte Carlo (with generically decaying $B^+ B^- / B^0 \bar{B}^0$) plus 20 million events of signal Monte Carlo ($B^+ B^-$ with one B decaying in the signal channel, the other decaying generically). Since the applied user-cut required a correct signal-side candidate, the signal Monte Carlo was necessary to ensure reasonable statistics for the B -level classifier trainings; the earlier stages for D mesons are not affected and can be adequately trained using generic Monte Carlo.

Compared to a generic training on 100 million events (i.e. almost twice as many events), the specific training required only 7.2 % of the total CPU time for processing particle, and half the time for classifier trainings [59, p. 69]. This is mostly caused by the aforementioned cuts, which significantly reduce the combinatorics and thus the effort expended on inviable events. Thus, even for larger input samples, the reduced CPU requirements of the analysis-specific mode allow the completion of the training using a moderately powerful computing system.³ As a result of the smaller data sample available for the B -level trainings, only 9 out of 21 hadronic B^+ decay channels were used in the training (compared to 19 out of 21 for a generic training); the semileptonic channels were unaffected [59, p. 69].

Using an independent test sample consisting of one million events each of $B^+ B^-$, $B^0 \bar{B}^0$, $u\bar{u}$, $d\bar{d}$, $s\bar{s}$ and $c\bar{c}$, as well as one million $B^+ \rightarrow \tau^+ \bar{\nu}_\tau$ signal Monte Carlo events [59, p. 66], the different characteristics of the generic and specific training on the candidates available to the analyst were evaluated.⁴

³As the total CPU time in this example was 585 hours, even a single 8-core machine would complete the analysis-specific training in three days (assuming perfect scalability and sufficient disk space).

⁴Compared to what is expected from detector data, this underestimates the amount of continuum background

6. Full Event Interpretation

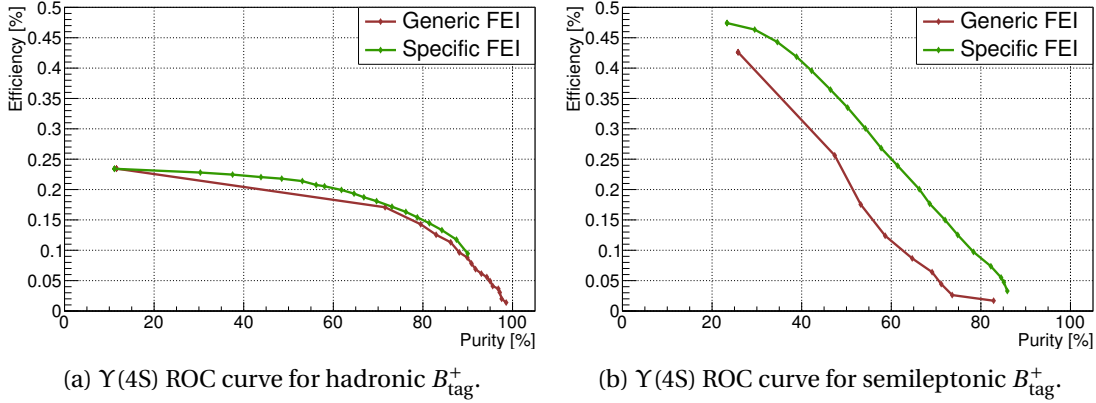


Figure 6.8.: Efficiency and purity of $\Upsilon(4S)$ mesons combined from the signal side and hadronic (a) or semileptonic (b) B_{tag}^+ candidates for different cuts on the B_{tag} classifier output. Results for the generic (red) and the specific FEI training (green) are compared and show better performance for the analysis-specific training. Note that for the specific training's hadronic B_{tag}^+ decay channels, only nine B^+ decay modes were used due to smaller training inputs (compared to 19 for generic), resulting in non-ideal performance. Adapted from [59, p. 74].

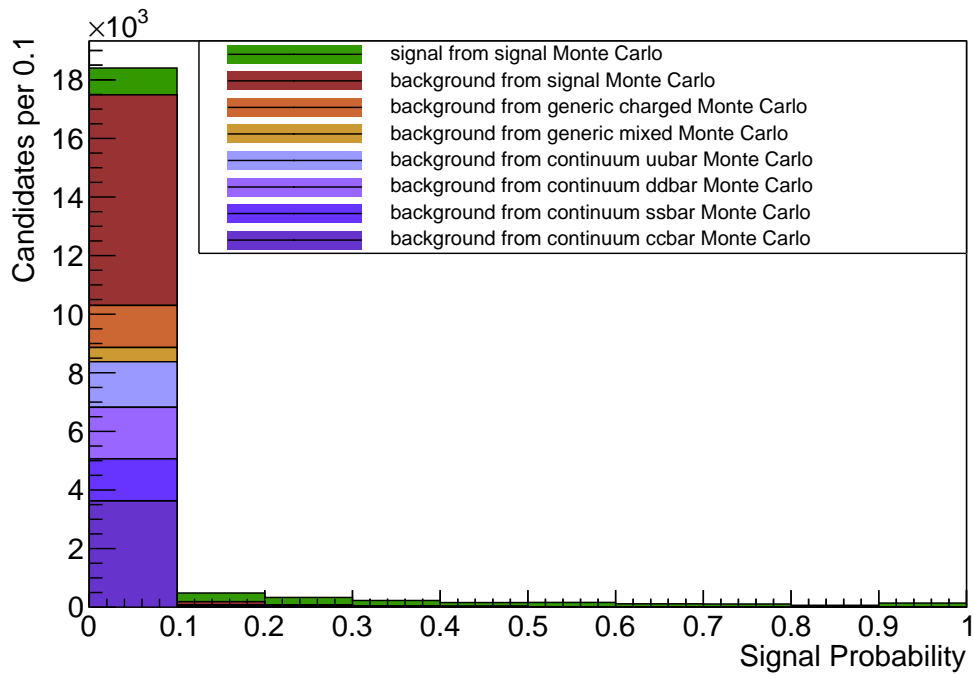
For both training modes, this involved looking at the efficiency and purity of the $\Upsilon(4S)$ mesons obtained when combining the signal selection with the produced B_{tag} candidates. As a result, the efficiencies presented in the following are not pure tag-side values, but include the selection efficiency of the signal channel of 65 % and the efficiency of the cut removing events with additional tracks of 87 % and 90 % for hadronic and semileptonic B decay modes, respectively [59, p. 73].

Figure 6.8 shows ROC curves for the resulting $\Upsilon(4S)$ mesons produced using both hadronic and semileptonic B_{tag} decay modes, and compares the selection efficiency and purity between the generic and specific trainings. In both plots, the analysis-specific FEI training performs significantly better than the generic training; for the hadronic decay modes the improvement is not as large, as a majority of decay modes was not included. A training with a larger signal Monte Carlo data sample is liable to produce even better results.

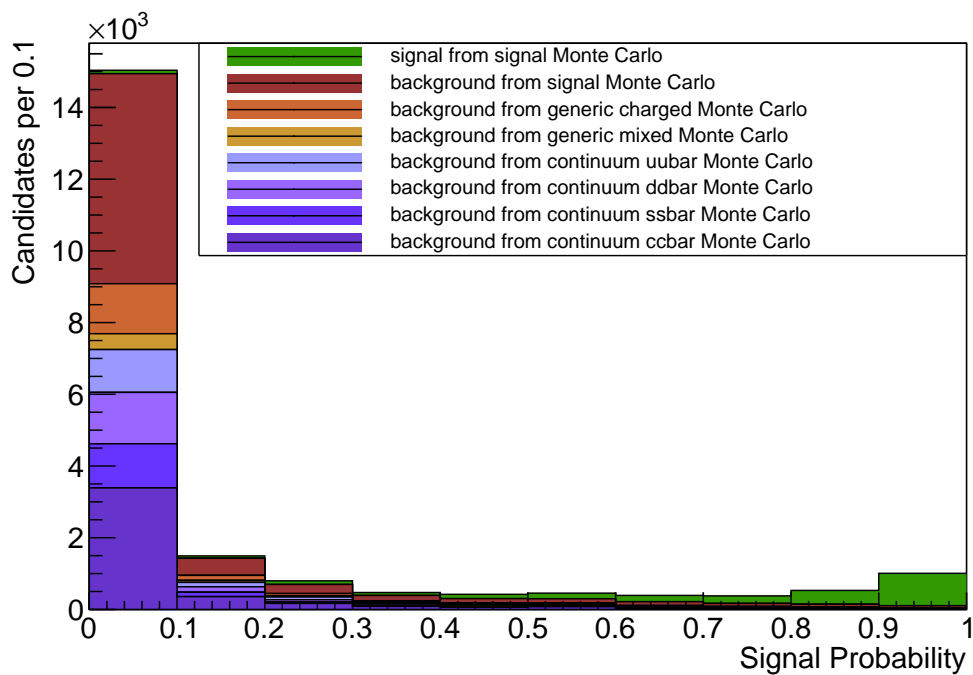
Perhaps even more interesting than this improvement in efficiency is the difference in the distribution of the classifier output for B_{tag} candidates. Figures 6.9 (for hadronic) and 6.10 (for semileptonic channels) clearly show the effect of training the B -level classifiers on a sample that only includes candidates that would leave no remaining tracks: For the analysis-specific training (bottom), the separation between true signal from the signal Monte Carlo and the remaining background components is not only more pronounced, but corresponds to what one would expect for a probability-like distribution.

Thus, analysis-specific trainings are not only made possible by the greater automation of the training process, but are actually an order of magnitude faster than a comparable generic training, while providing significantly better efficiency and purity in conjunction with a more natural classifier output for the signal mode in question. The addition of the analysis-

(especially $c\bar{c}$). As the results of the two trainings are, however, meant to be compared to each other, this does not influence the following results.



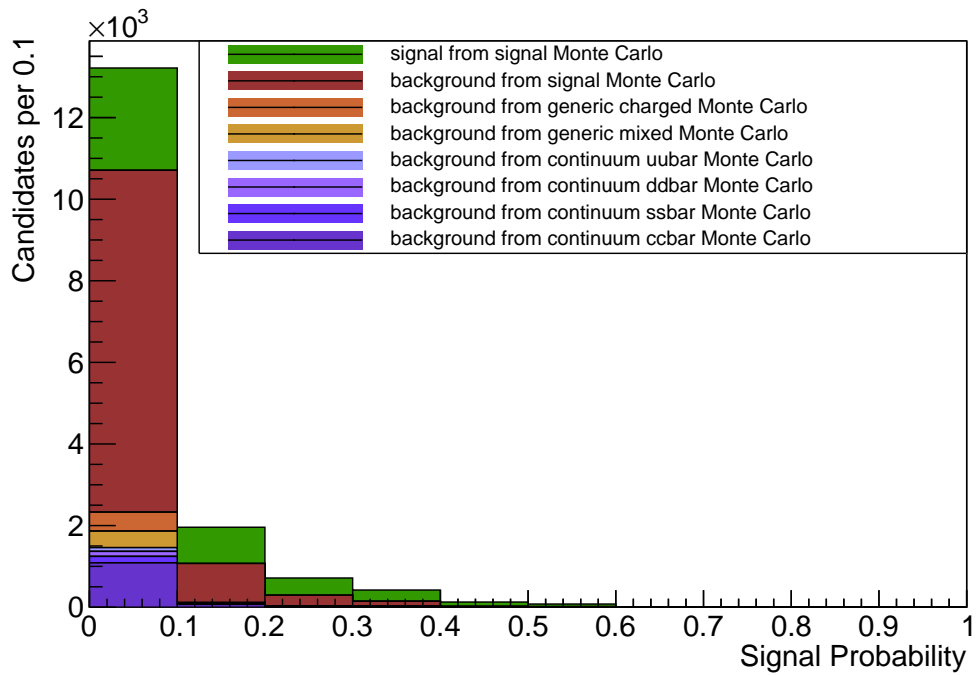
(a) Classifier output for generic training.



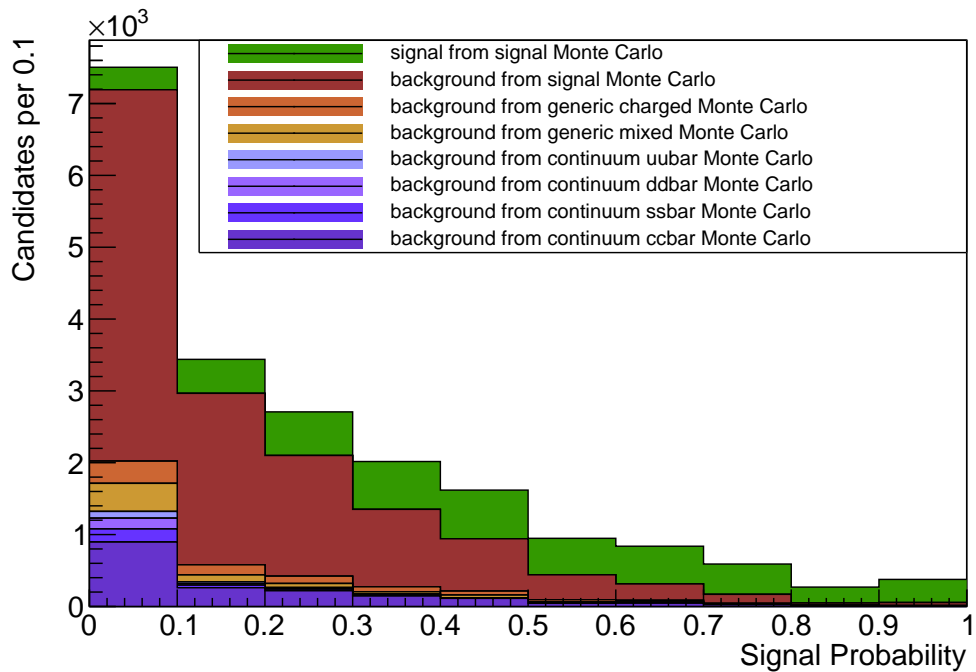
(b) Classifier output for specific training.

Figure 6.9.: Distribution of the B_{tag} classifier output for hadronic decay modes for specific and generic FEI trainings. Taken from [59, p. 75].

6. Full Event Interpretation



(a) Classifier output for generic training.



(b) Classifier output for specific training.

Figure 6.10.: Distribution of the B_{tag} classifier output for semileptonic decay modes for specific and generic FEI trainings. Taken from [59, p. 76].

specific mode is also one of the main reasons behind the name ‘Full Event Interpretation’, as it indeed produces an interpretation of the entire event and makes decisions grounded in the cleanliness of $\Upsilon(4S)$ decays. Other possible selections that consider the entire event also contribute to the name, e.g. limits on the total number of tracks per event, including continuum suppression or using a variable indicating the quality of events.

A disadvantage of the analysis-specific training is that it cannot be used for analyses measuring inclusive decays, since there is no longer a single signal-side decay and the purity of the tag-side candidates is used to define the set of remaining tracks and ECL clusters used for the signal-side. Inclusive analyses thus should be combined with a generic FEI training. For exclusive analyses that reconstruct multiple signal decays, it may be possible to use a more inclusive signal selection instead.

6.6.3. Mono-Generic Training

As an approximation of the specific training for signal selections with a low number of tracks, and with no remaining tracks after combining tag- and signal-side, a *mono-generic* training can also be used. This type of FEI training uses a special Monte Carlo sample containing one generically decaying B meson, and one invisible B . When used as a training input, this is very similar to the sample obtained after a successful signal selection, i.e. one where all final-state particles of the signal-side (and only those) are removed. A main difference between a mono-generic and a specific FEI training thus is the complete absence of background in which final-state particles from both B mesons in the event are combined. While correctly reconstructed B_{tag} candidates are not affected, this would still result in different prior distributions for training and application, and — since high-multiplicity channels are more likely to receive a wrong track — would also have decay-mode-dependent effects. The mono-generic training however has the advantage of being easy to produce centrally, while introducing less bias for signal-selections with few tracks than a comparable generic training. The differences between mono-generic and specific trainings have not been studied in detail, and may be an opportunity for future work.

This type of training might prove useful to support other high-level reconstruction tools with its output. These applications might include Flavour Tagging, tag-side vertex reconstruction, or determining the decay topology of an event.

6.7. Distributed FEI Trainings

Since the Full Event Interpretation needs to be trained in multiple iterations on very large sets of data, running this training on a single machine is not usually a realistic option, and typically distributed computing systems with many hundreds of nodes are used instead. To ensure users can perform their own training, facilities for a distributed training are included in the FEI; these take care of splitting the input sample, job scheduling and other common tasks for the KEK computing center (KEKCC). This training approach follows a MapReduce[86] model: First, the input data (the size of which may be on the order of terabytes) is split into subsets for the desired number of jobs; then the jobs are run in parallel using KEKCC’s batch submission system (`map()` step). After all jobs have been executed, the produced n -tuples and histograms are merged on a central node for starting the classifier trainings and creating

6. Full Event Interpretation

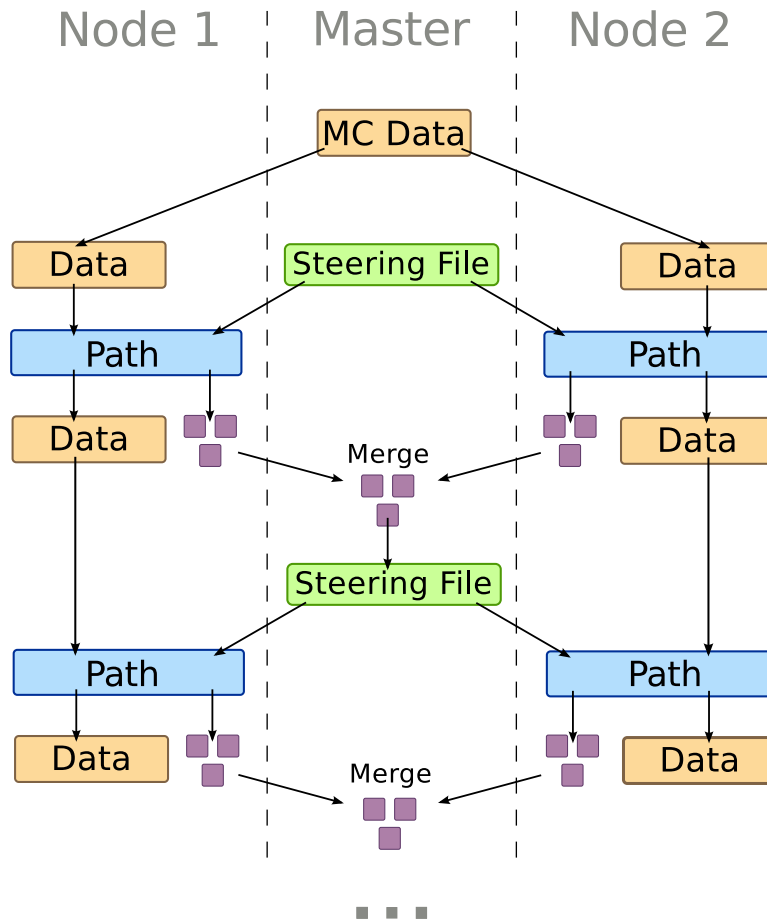


Figure 6.11.: Illustration of the MapReduce model used for training the Full Event Interpretation on a distributed computing system with two nodes and one master. Note that, after initialisation, only relatively few data are transferred between master and nodes; the largest share is taken up by n -tuples used for classifier trainings (limited to a few gigabytes per training). Adapted from [59].

aggregate statistics (reduce() step). The map() and reduce() steps are then repeated until all particles are trained. An illustration of this method is shown in Figure 6.11.

An important factor in this model is that only modules are executed on the computing nodes, while all high-level code (Python) is run on the master node. This is achieved by serialising the path of modules on the master node and executing the path instead of the normal steering file on computing nodes. This mirrors the separation shown in Figure 6.1, where the Python part of the Full Event Interpretation deals with the dependency resolution and tasks that act on extracted information (e.g. to determine cuts), while the event-processing is performed in C++ modules.

A more in-depth description of the distributed training approach used by the FEI can be found in [59, p. 40]. The same model can in principle also be applied to large-scale distributed computing systems like the Belle II grid, though some additional work is necessary to ensure that cached data is kept locally on nodes and are not copied around needlessly. The scalability

for larger amounts of data is currently limited by the time needed to merge tuples and histograms from all nodes and for interacting with the job submission system. The bulk of the processing time for the `reduce()` step is however taken up by the classifier trainings, which due to the implemented sampling (see [Section 6.4](#)) take almost constant time.

6.8. Conclusions

This chapter introduced the Full Event Interpretation, an advanced tag-side reconstruction tool for the Belle II experiment that was developed within the scope of this thesis, and discussed the details of its implementation, as well as some improvements made possible by it. While the hierarchical reconstruction approach of the FEI is similar to that employed by the Full Reconstruction at Belle, the algorithm's implementation differs greatly, with practically all physically relevant attributes of the reconstruction being contained in the configuration, the use of the directed acyclic graph framework for dependency resolution, and the high degree of automation throughout the entire training process.

Another striking difference is the size of the implementation of both algorithms themselves: For the Full Reconstruction, the bulk of the implementation consisted of 18 221 lines of C++ code, with a few helper and training scripts. The Full Event Interpretation, on the other hand, is built using only 2 632 lines of Python code, over one thousand lines of which are used for the automatic reporting functionality (not present in the Full Reconstruction). Mostly this is achieved through the development and extension of modular tools that perform discrete analysis steps. Thus, most of the modules described in [Chapter 4](#) were either made more powerful to work with the FEI, or written from scratch to serve a specific purpose (like the TMVA interface). They are, however, written in such a way that they are generic and work easily with other available modules. In particular, other high-level reconstruction tools like continuum suppression or Flavour Tagging now also make use of the same modules [\[66\]](#). An additional few hundred lines of C++ code is taken up by the `PreCutHistMaker` module and variables that are only used by the FEI currently. It should also be noted that a significant number of problems in the analysis tools were found during the development of the FEI, since it covers a large part of the steps present in an analysis, but in many more decay channels. The automatic reporting functionality was very helpful in this regard, and many issues were visible in the plots and tables generated.

As a direct consequence of these technical improvements, concerning in particular the dependency resolution and training procedure, the usability of the Full Event Interpretation is greatly increased: The high degree of automation, e.g. when determining necessary cuts, allows the FEI to be trained without requiring adjustments, and the training can thus be performed by any user without requiring a large amount of FEI-specific knowledge (given a reasonable default configuration). This is tremendously useful for the optimisations introduced in [Section 6.6](#), which require information about the signal channel (i.e. the decay channel(s) of the signal-side B), and thus need to be trained separately for each analysis.

As already shown, the analysis-specific training mode greatly improves the performance of the Full Event Interpretation, through increasing the reconstruction efficiency and ensuring the output is probability-like, by optimising the training for the signal selection chosen by the user. Other factors also lead to possible improvements of the physics capabilities of analyses, mostly through the generalisation of the reconstruct steps and the ease with which

6. Full Event Interpretation

new decay modes can be added. The generalisation of the FEI – with configurable signal definitions, input variables, and control plots – makes it possible to add particles decaying through semileptonic modes in the default configuration. Especially when combined with the analysis-specific training mode, this allows for analyses that combine both hadronic and semileptonic tag-side reconstruction, where previously these were performed in separate analyses (e.g. for $B^+ \rightarrow \tau^+ \nu_\tau$ decays). Since the reconstruction of e.g. D mesons is not affected by the exact decay mode of the B mesons, this also allows for savings in CPU time and disk space. Additionally, since it is easy to replace the configuration, it will also be possible to reconstruct the heavier $\Upsilon(5S)$ resonance simply by loading an $\Upsilon(5S)$ -specific configuration file.

Adding new decay modes has become much easier with the Full Event Interpretation, since only the newly added channels and those dependent on them need to be retrained, and the necessary adjustments of cuts are performed automatically. Compared to the 71 neural networks used by Belle’s Full Reconstruction algorithm to reconstruct 1 104 exclusive hadronic B decay channels [75], the FEI (with default configuration) trains 110 boosted decision trees, which are used to reconstruct 4 986 exclusive B decays (264 of those in semileptonic channels).

The effect these different factors have on the total performance of the Full Event Interpretation, and what analysts can expect in regard to the reconstruction efficiencies and purities will be discussed in the following chapters.

7. Estimation of Physics Performance

After describing the implementation details of the Full Event Interpretation and the – primarily software-related – improvements compared to Belle’s Full Reconstruction in the previous chapter, this chapter will discuss concrete applications of the FEI. To this end the results of a large-scale training for Belle II will be presented. Particular focus will be given to the quantities that primarily effect the performance of analyses relying on the tag-side reconstruction, i.e. the total reconstruction efficiency of the tag-side reconstruction, and the purity associated with it.

In the following, only generic trainings of the Full Event Interpretation (as defined in [Section 6.6.1](#)) are evaluated. This makes it easier to compare the results with those of the Full Reconstruction and thus assess the effect that using the FEI can have on typical analyses. As by construction an analysis-specific training is optimised for a certain signal decay, the performance of analyses using this mode can be assumed to be even better.

7.1. Monte Carlo Sample

This study was performed using a Monte Carlo sample generated in a large-scale Monte Carlo campaign in 2014 (MC 3.5). As the beam background in the electromagnetic calorimeter (ECL) as simulated in this Monte Carlo generation campaign is significantly overestimated and cannot be easily suppressed, training of channels including photons would have been impossible due to exorbitant CPU time requirements. The following study thus uses a sample without beam background, which results in both better tracking performance and much fewer wrong hits in the ECL compared to what is expected for real data.

Input for this FEI training were 40 so-called *streams* of Monte Carlo, consisting of one half mixed ($B^0\bar{B}^0$) and one half charged (B^+B^-) events, with generically decaying B mesons.¹ This corresponds to 80 million events in total, with the mDST data taking up about 543 GB of disk space. Continuum background was not used in the training, as correctly reconstructed particle candidates in the continuum sample (which includes the same types of particles as the $B\bar{B}$ samples, with the exception of B mesons) have very different kinematics. Consequently, it is not clear whether to regard them as signal or background: A particle reconstructed in one of the continuum Monte Carlo components would, even if it matched perfectly with the Monte Carlo truth, likely have a very different distribution in kinematic variables and could never be used to create a correct B meson candidate. Regarding them as background, however, might also have negative effects on the multivariate classifier, since at least in some cases, their kinematics might be similar to correct candidates in $B\bar{B}$ Monte

¹ At Belle, one stream of Monte Carlo is meant to correspond to the quantity of collisions in real detector data, with multiple streams per data-taking period being available to provide reasonable statistics for most analyses. The size convention is not very meaningful for Belle II just yet.

7. Estimation of Physics Performance

Table 7.1.: List of decay channels ignored in this training, with associated branching fractions extracted from EvtGen. Values marked by ‘*’ were missing or differed significantly from PDG averages, and have been fixed to PDG values [85]. (Cf. Table 6.1)

| Decay channel | Branching ratio |
|---|-----------------------|
| $D^0 \rightarrow K^- \pi^+ \pi^0 \pi^0$ | 15–20 %* |
| $D^0 \rightarrow \pi^- \pi^+ \pi^0 \pi^0$ | 0.94 % |
| $B^+ \rightarrow \overline{D^0} \pi^+ \pi^0 \pi^0$ | 0.61 %* |
| $B^+ \rightarrow \overline{D^{0*}} \pi^+ \pi^0 \pi^0$ | 5.00×10^{-4} |
| $B^+ \rightarrow \overline{D^0} D^+$ | 3.80×10^{-4} |
| $B^0 \rightarrow D^{-*} \pi^+ \pi^0 \pi^0$ | 0.10 % |
| $B^0 \rightarrow D^- \pi^+ \pi^0 \pi^0$ | 0.10 % |

Carlo. As a separate tool for continuum suppression has been developed by others (see Section 4.8.1), it can be applied independently of the Full Event Interpretation.

7.2. Training

Using this Monte Carlo sample, a generic FEI training (see Section 6.6.1) was performed using the default configuration. The trainings were started on the KEK Central Computing System (KEKCC) with the MapReduce approach introduced in Section 6.7 using 2 000 jobs for the map() step. The jobs were processed by the IBM LSF [87] system, with (depending on system utilisation) up to three hundred jobs running in parallel.

The training in total took about three days to complete and consumed 2 970 hours of CPU time on reconstruction tasks, excluding time for training and I/O. To avoid unnecessary repetition of reconstruction steps, the output of the map() step, including created particles, was saved to disk as cache. Summed over all jobs, this cache consumed 3.2 TB at the B level, with peak usage being slightly higher. Due to otherwise limited disk space, both the cache and all ROOT files created during the training were stored on a tape-based file system (HSM) with many hundreds of terabytes of free space. While the throughput of the HSM system is quite high and does not slow down job execution, the latency when accessing a file or performing other file operations is quite high (many seconds). Using a faster disk-based file system thus might reduce the training time even further.

In this training, the decay channels listed in Table 7.1 were discarded automatically, including all channels with multiple π^0 daughters. This occurs regardless of the channel’s branching ratio; this could be traced back to a software problem related to the vertex fitting. This problem should be fixed in the near future, likely resulting in improvements of the reconstruction efficiency. The $B^+ \rightarrow \overline{D^0} D^+$ channel was discarded because only very few candidates were produced.

The training’s success can be confirmed by looking at the summary section of the automatically generated FEI report, which contains the tables of the efficiency and purity of the final-state particles, and those of combined particles. The final-state particles have very

Table 7.2.: Per-particle efficiency and purity before and after the applied user-, pre-classifier- and post-classifier-cuts.

| | Efficiency in % | | | | Purity in % | | | |
|--------------------|-----------------|----------|---------|----------|-------------|----------|---------|----------|
| | recon. | user-cut | pre-cut | post-cut | recon. | user-cut | pre-cut | post-cut |
| π^0 | 88.90 | | 68.56 | 49.96 | 2.632 | | 5.773 | 30.639 |
| K_S^0 | 39.76 | | 37.81 | 34.82 | 1.858 | | 6.280 | 72.572 |
| D^0 | 18.54 | | 17.62 | 13.52 | 0.016 | | 0.271 | 6.907 |
| D^+ | 11.28 | | 10.72 | 8.31 | 0.009 | | 0.182 | 6.465 |
| D^{+*} | 4.30 | | 4.08 | 4.07 | 0.203 | | 22.514 | 27.362 |
| D^{0*} | 8.35 | | 7.61 | 5.81 | 0.069 | | 0.266 | 5.222 |
| D_s^+ | 7.06 | | 6.71 | 4.85 | 0.008 | | 0.124 | 5.227 |
| D_s^{+*} | 4.50 | | 4.44 | 2.98 | 0.108 | | 0.226 | 4.021 |
| J/ψ | 9.84 | | 8.29 | 8.29 | 1.215 | | 42.980 | 42.980 |
| B_{had}^+ | 1.11 | 1.10 | 1.04 | 0.93 | 0.001 | 0.069 | 0.178 | 0.189 |
| B_{sl}^+ | 1.32 | | 1.25 | 1.25 | 0.333 | | 1.174 | 1.174 |
| B_{had}^0 | 0.63 | 0.62 | 0.59 | 0.51 | 0.002 | 0.167 | 0.584 | 0.633 |
| B_{sl}^0 | 1.45 | | 1.37 | 1.37 | 0.172 | | 0.573 | 0.573 |

similar efficiencies to those listed in Table 6.4, which were obtained from a training with almost identical configuration, but on a much smaller Monte Carlo sample. Efficiencies and purities for combined particles are reproduced in Table 7.2. The most important values in this table are the post-cut efficiencies for the four categories of B mesons: In hadronic B decays, efficiencies of 0.93 % (B^+) and 0.51 % (B^0) were achieved, for the semileptonic channels, somewhat higher values of 1.25 % (B^+) and 1.37 % (B^0) are reached. These are already much higher than the corresponding B_{tag} efficiencies for the neural-network based Full Reconstruction at Belle, which were 0.28 % for hadronic B^+ and 0.18 % for hadronic B^0 candidates (see Section 5.3). The values in this table, however, were computed on the same Monte Carlo sample used for the training, and do not include cuts that would discard, e.g., candidates with background-like kinematics. As a result, these values are maximum efficiencies, and similarly the purities shown are their minimal values. An analysis user will usually add at least a cut on the classifier output, increasing the purity of the candidates, while reducing the efficiency below the maximum values given in the table.

When comparing the table with that created on a smaller Monte Carlo sample in Table 6.5, significant differences in the efficiencies are visible. As for many particles the efficiencies of the large-scale sample are more than 50 % higher, this suggests that the difference is mostly caused by the decay modes that were excluded from the smaller training because of insufficient statistics.

The automatic reporting also provides information on the quality of the multivariate classifiers created during the training. Figure 7.1 shows the TMVA control plots that were generated for the $B^+ \rightarrow \bar{D}^0 \pi^+$ decay mode, with an excellent separation between correct and incorrect candidates being visible. The Kolmogorov–Smirnov test reveals a possible slight overtraining ($p = 0.047$) in the signal class, which is however only barely visible in the second-highest bin of the distribution. As it is, the result of the test should be no cause for

7. Estimation of Physics Performance

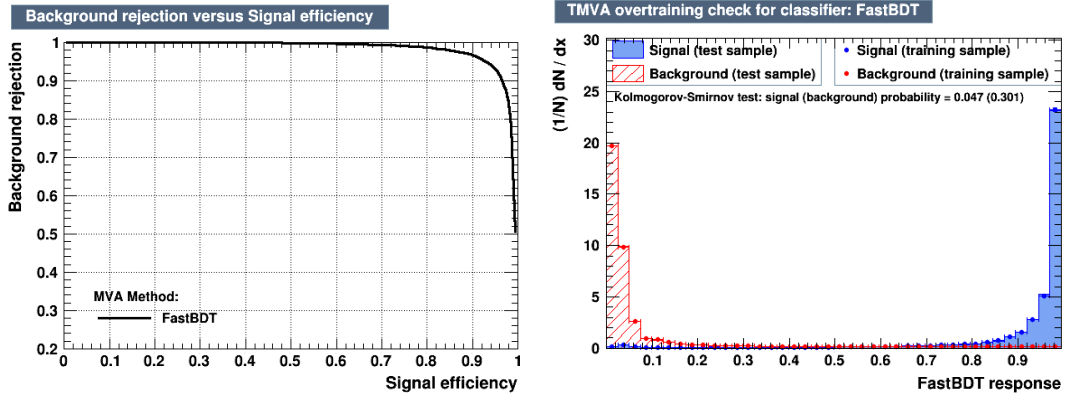


Figure 7.1.: TMVA control plots for $B^+ \rightarrow \overline{D^0}\pi^+$, showing purity over efficiency for the classifier (left) and the distribution of signal and background (for training and test sample) in the classifier output (right).

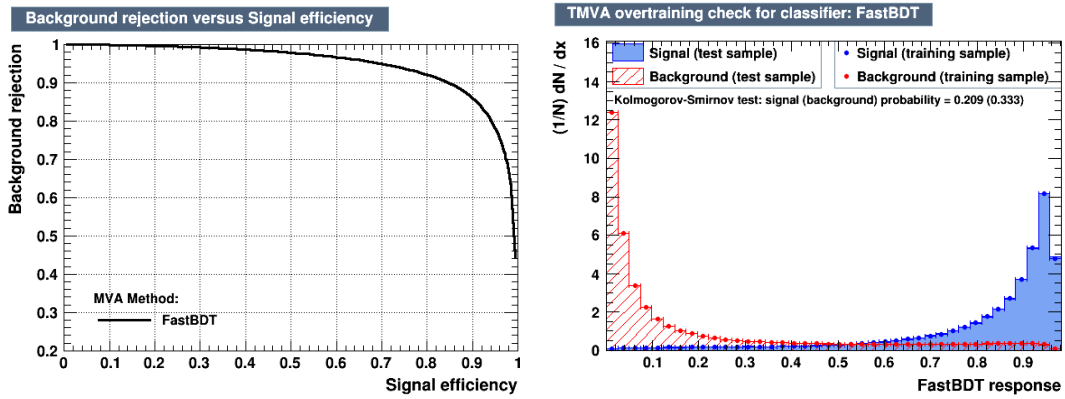


Figure 7.2.: TMVA control plots for $B^+ \rightarrow \overline{D^0}e^+(\nu_e)$, showing purity over efficiency for the classifier (left) and the distribution of signal and background (for training and test sample) in the classifier output (right).

great concern. For comparison, [Figure 7.2](#) shows control plots for the semileptonic decay $B^+ \rightarrow \overline{D^0} e^+ (\nu_e)$. Here, the purity–efficiency curve covers a much smaller area, i.e. the classifier has a worse separation, as in this channel an electron neutrino is not reconstructed, which decreases the amount of kinematic information available to the classifier. No overtraining is visible in the overtraining check on the right-hand side.

To illustrate the output of the Full Event Interpretation, two visualisations of events with a correct B_{tag} candidate produced using the BASF2 event display (see [Section 3.7](#)) are shown in [Figure 7.3](#). Objects corresponding to final-state particles used on the tag-side are highlighted in green, showcasing the differences between the complex decay chain (seven tracks) in [Figure 7.3a](#) and the somewhat simpler chain (three tracks and two photons) with a semileptonic B decay in [Figure 7.3b](#). The remaining tracks and clusters after subtracting the highlighted tag-side objects then belong to either the signal side, or secondary processes of the tag side, or are caused by background processes or detector noise. ECL clusters that are in the vicinity of a track are not used for photon candidates but provide particle identification information, as exemplified by the large energy deposition for the electron track in the lower left of [Figure 7.3b](#).

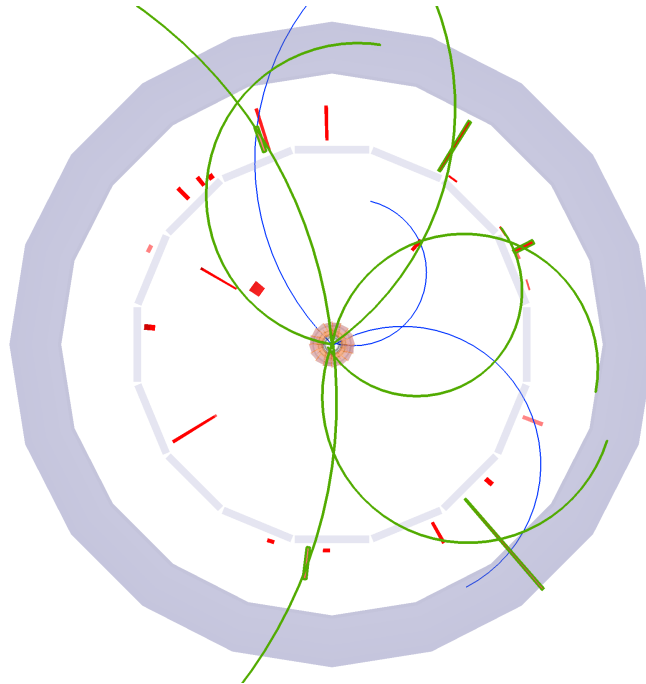
7.3. Results

The FEI training was further evaluated using an independent test sample of ten streams of Monte Carlo, consisting of ten million $B^+ B^-$ and $B^0 \overline{B^0}$ events each, plus an amount of continuum background equivalent to that expected from detector data. For each of the four final lists of B_{tag} candidates (B^+ and B^0 in both semileptonic and hadronic variants) an n -tuple was produced. For the 82 million events in the sample, applying the Full Event Interpretation took a total of 1 621 hours of CPU time, which corresponds to 0.07 seconds per event. Because of the different sample composition, the processing time estimated by the automatic reporting, in this case 0.13 seconds per event, differs from this value. This suggests candidates created from continuum events are discarded more quickly. The independent sample is used for all further results in this section.

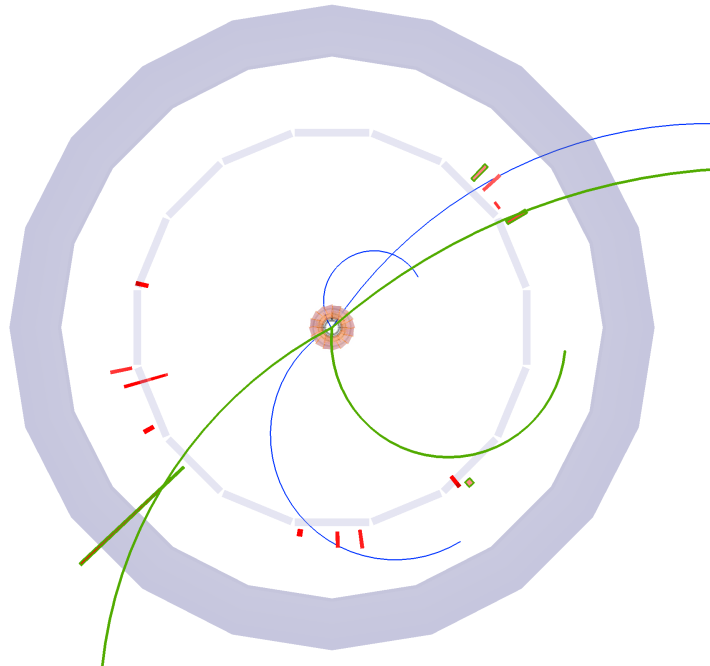
As the Full Event Interpretation can produce many B_{tag} candidates for each collision event, including all of the candidate particles might produce unwanted effects such as an overestimation of the reconstruction efficiency. To avoid these effects, a best candidate selection (as introduced in [Section 4.6](#)) was applied to the B_{tag} particle lists. In each event, only the candidate with the highest classifier output in each list was retained.

The output of the multivariate classifier for each hadronic B candidate is presented in [Figure 7.4](#), showing the signal and the different classes of background candidates as distinguished by the type of Monte Carlo they were produced from. Since the amount of events in the continuum background Monte Carlo is over three times that of the $B^0 \overline{B^0}$ and $B^+ B^-$ samples, it also very clearly dominates in the number of candidates. Even though no information about continuum background is included in FEI training, the candidates produced on continuum Monte Carlo are nonetheless shifted to very low values in the classifier output. Correctly reconstructed candidates, on the other hand, are clustered at high values and can thus be separated well by cutting on the output. The remaining background classes are made up of candidates that were created on events containing $\Upsilon(4S) \rightarrow B\overline{B}$, which are more signal-like and thus harder to separate. It is apparent that for B^+ candidates, the back-

7. Estimation of Physics Performance



(a) A correctly reconstructed tag-side $B^+ \rightarrow D^0 \pi^+ \pi^- \pi^-$ decay (with $D^0 \rightarrow K_S^0 \pi^+ \pi^-$).



(b) A correctly reconstructed tag-side $B^- \rightarrow D^{*0} e^- \bar{\nu}_e$ decay (with $D^{*0} \rightarrow D^0 \pi^0$ and $D^0 \rightarrow K^- \pi^+$). A low-energetic final-state-radiation photon from the D^0 decay was missed.

Figure 7.3.: Visualisation of two events with B_{tag} mesons correctly interpreted by the Full Event Interpretation. All tracks and clusters used on the tag-side are highlighted in green; the remaining tracks and calorimeter clusters in blue and red, respectively.

ground component from B^+B^- events is enlarged, and similarly the $B^0\overline{B}^0$ component for B^0 candidates.

For semileptonic B decay modes, the output is shown in [Figure 7.5](#). As a result of the larger efficiency, the plots show a significantly larger number of candidates than in the hadronic modes. The separation between the background classes and signal is less clear, as the multivariate classifiers for semileptonic B decays receive less information than the hadronic ones (e.g. the separation power of ΔE is much less because of the missing neutrino).

The fine binning in these four graphs uncovers some interesting structure in the output of the multivariate classifier, like the two peaks around $o_{\text{MVA}} = 0.7$ in [Figure 7.4](#) (top figure). These structures are a side-effect of the transformation applied by the TMVA expert to transform the output to a probability; even when the classification method itself, in this case FastBDT, produces an almost probability-like output, this transformation can be beneficial. However, as it works by shifting candidates in the classifier output, e.g. from a bin at $o_{\text{MVA}} = 0.5$ with a purity of 55 % into the bin at $o_{\text{MVA}} = 0.55$, it can introduce peaks in arbitrary bins. Nonetheless, the probability interpretation should be more accurate, and cuts on the classifier output should not produce sudden jumps in the efficiency or purity. It should also be noted that the distributions cannot be compared to the plots of the classifier output shown in [Section 6.6.2](#), where both tag- and signal-side B mesons needed to be correctly reconstructed to count as signal.

The effect of different cuts on the classifier output can be seen in [Figures 7.6 and 7.7](#), which show the distribution of the control variables m_{bc} and $\cos\theta_{B,D^{(*)}l}$ and their change for increasingly harder cuts. In all cases, the background components are reduced more strongly than the signal candidates, thus greatly enhancing the signal peak for harder cuts. The enhanced background from B^+B^- events in B^+ and in $B^0\overline{B}^0$ events for B^0 candidates that was already observed in the classifier output is more pronounced here. Again, this background component is very signal-like, i.e. it is concentrated around the B mass in m_{bc} and around 0 for $\cos\theta_{B,D^{(*)}l}$, however, real background components would likely appear in both of the $B\overline{B}$ Monte Carlo samples, e.g. also in B^+B^- events for B^0 candidates. The difference is caused by candidates that are almost correctly reconstructed, but e.g. contain a wrong photon and so do not pass the signal definition used here. In the beam-constrained mass, the peaking background distribution appears to be slightly broader than the true signal component due to slightly misreconstructed kinematics.

For the hadronic channels in [Figure 7.6](#), the beam-constrained mass m_{bc} exhibits a strong peak at the B^\pm or B^0 mass of 5.279 or 5.280 GeV, respectively. Below 5.27 GeV, practically no correctly reconstructed candidates remain. For the semileptonic B decay modes, the four components are much more similar and have peaks in the same central region. In contrast to the other components, however, the signal component is mostly constrained to the physical region of $|\cos\theta_{B,D^{(*)}l}| \leq 1$. A minority of correctly reconstructed B candidates lie outside the allowed range as a result of resolution effects for the measured four-momenta. Practically all signal, however, is contained within $|\cos\theta_{B,D^{(*)}l}| < 2$. These signal regions, namely $m_{bc} > 5.27$ GeV for hadronic and $|\cos\theta_{B,D^{(*)}l}| < 2$ for semileptonic channels, will be used for further evaluation in the following.² The efficiencies for the m_{bc} cut are 98.6 % for B^+ and 98.2 % for B^0 ; for the $\cos\theta_{B,D^{(*)}l}$ cut on semileptonic decay modes somewhat lower

²The $|\Delta E| < 0.5$ GeV user-cut was also reapplied, as the vertex fitting caused a handful of candidates to be shifted outside this region.

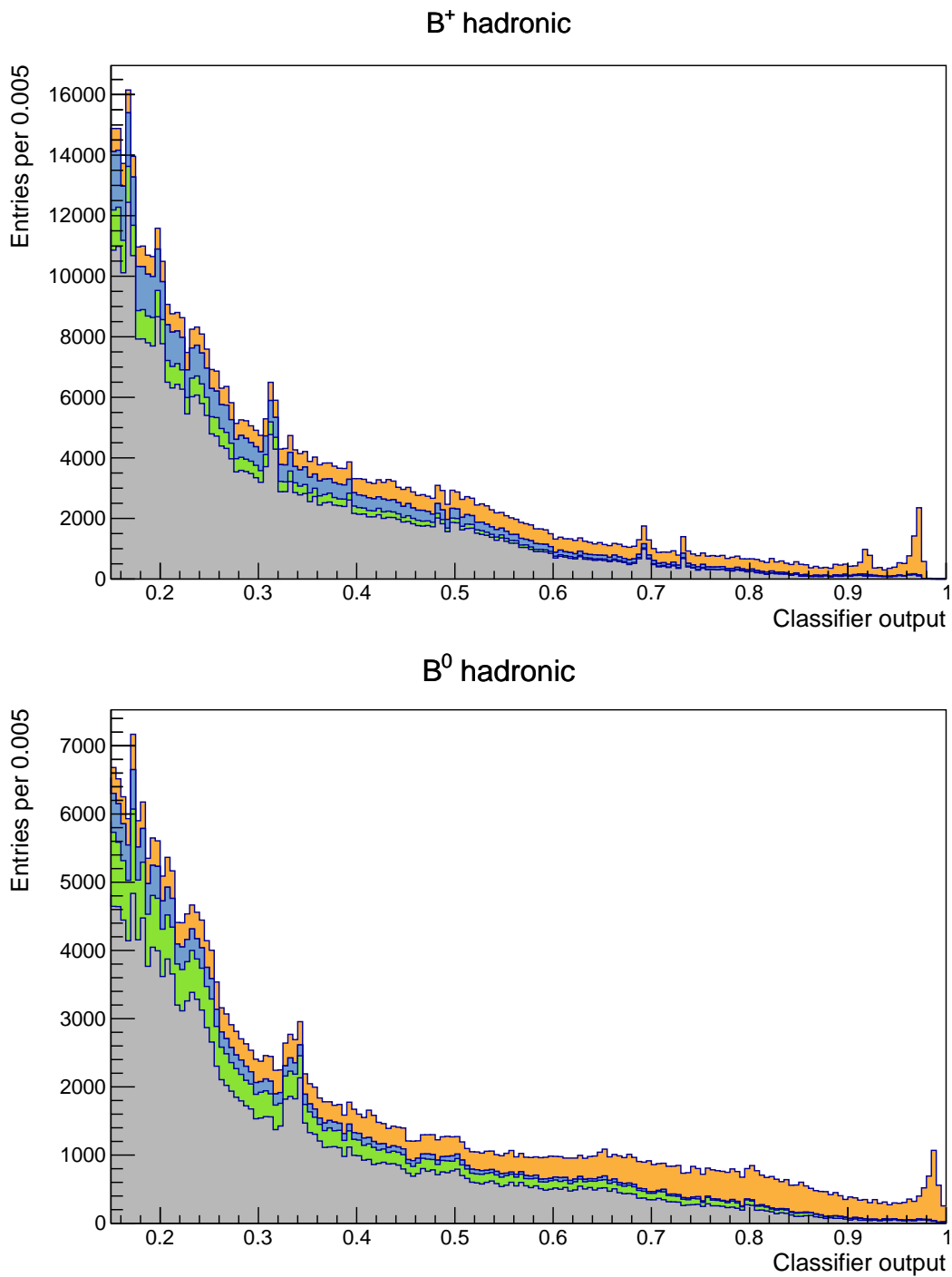


Figure 7.4.: Distributions of the multivariate classifier output for hadronic B decay channels. Correctly reconstructed candidates are shown in orange, background from $B^+ B^-$ events in blue, from $B^0 \bar{B}^0$ events in green, and continuum background in gray. Large numbers of background candidates are found at values below 0.15; for clarity, these are omitted.

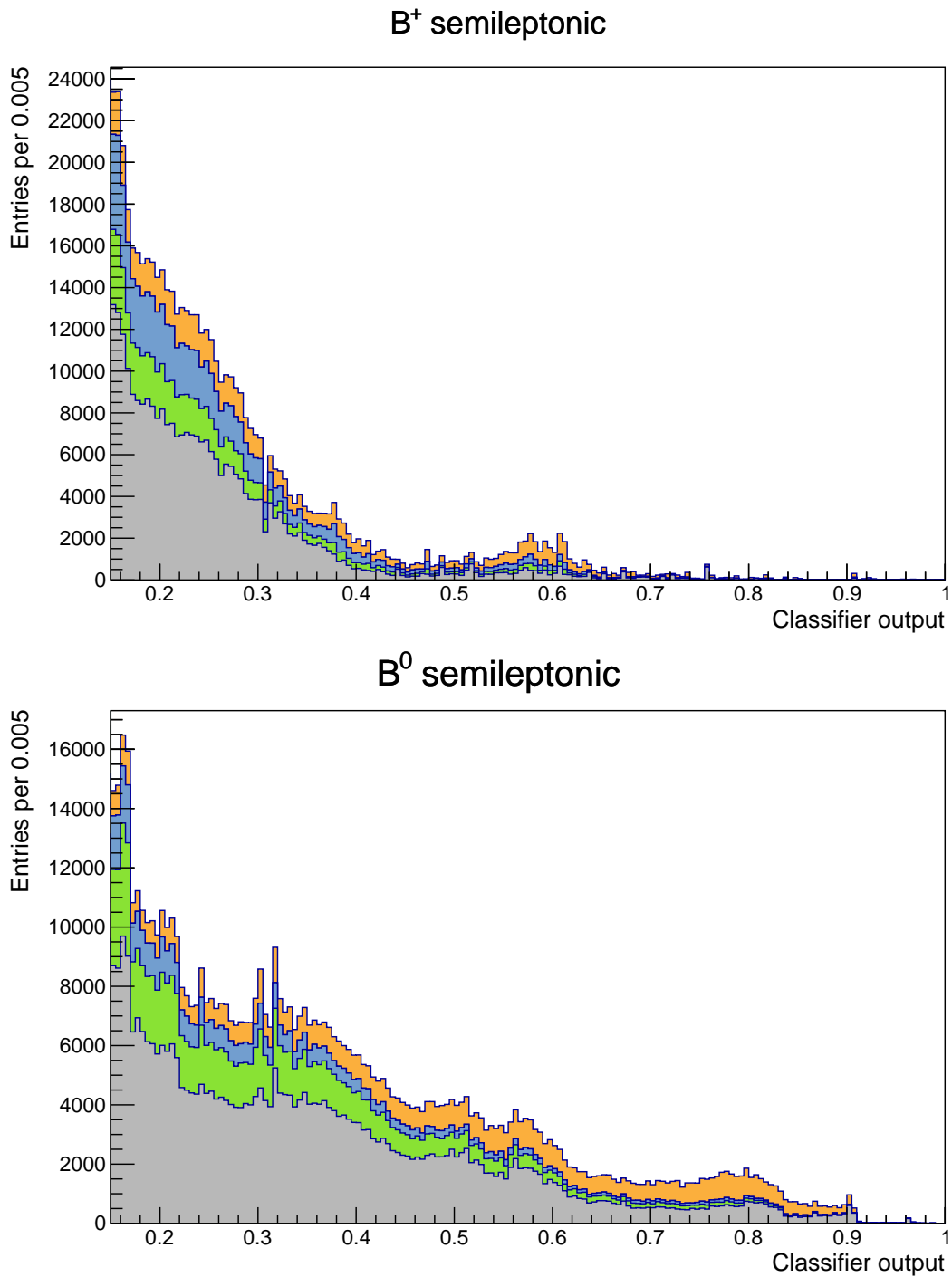


Figure 7.5.: Distributions of the multivariate classifier output for semileptonic B decay channels. Correctly reconstructed candidates are shown in orange, background from $B^+ B^-$ events in blue, from $B^0 \bar{B}^0$ events in green, and continuum background in gray. Large numbers of background candidates are found at values below 0.15; for clarity, these are omitted.

7. Estimation of Physics Performance

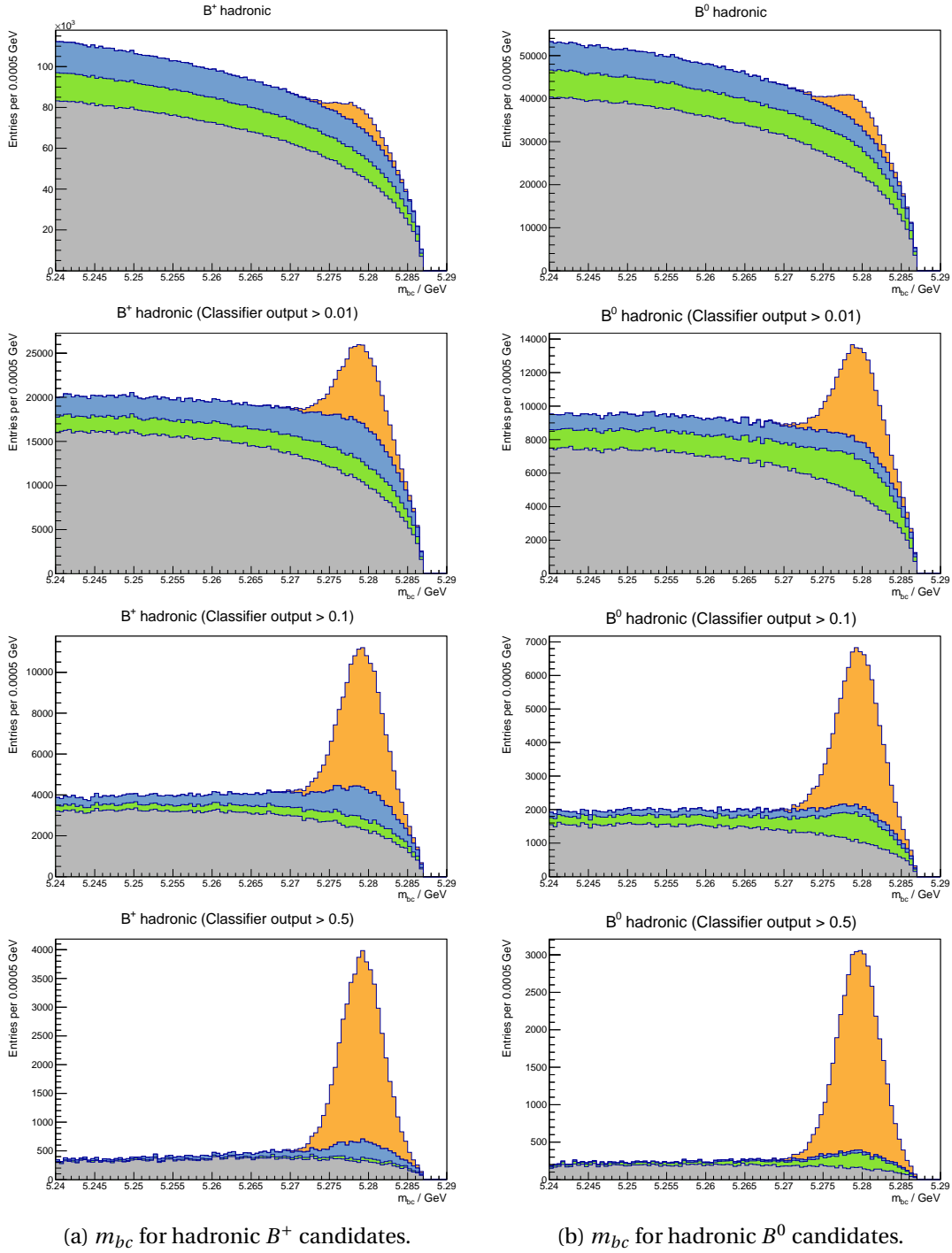


Figure 7.6.: Control plots for hadronic B^+ (left) and B^0 candidates (right), for (from top to bottom) no cut, $\sigma_{MVA} > 0.01$, 0.1 and 0.5 . Correctly reconstructed candidates are shown in orange, background from B^+B^- events in blue, from $B^0\bar{B}^0$ events in green, and continuum background in gray.

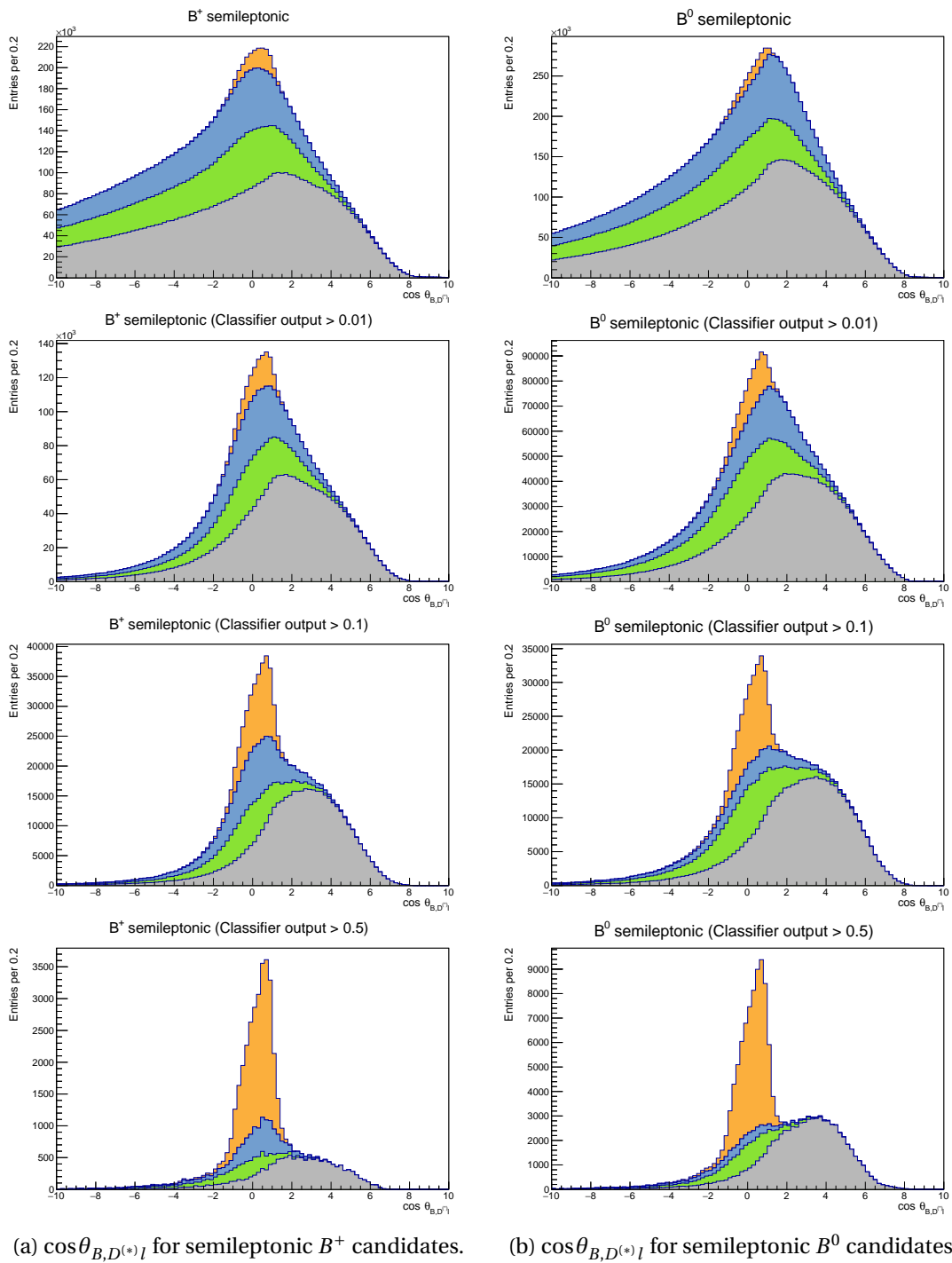


Figure 7.7.: Control plots for semileptonic B^+ (left) and B^0 candidates (right), for (from top to bottom) no cut, $\sigma_{\text{MVA}} > 0.01$, 0.1 and 0.5. Correctly reconstructed candidates are shown in orange, background from $B^+ B^-$ events in blue, from $B^0 \bar{B}^0$ events in green, and continuum background in gray.

7. Estimation of Physics Performance

Table 7.3.: The left-hand side shows maximum B_{tag} efficiencies as listed in the automatic report (AR) and the efficiencies of the signal region (SR) and best-candidate selection (BCS) cuts applied on the candidates. The right-hand side shows the maximum B_{tag} efficiencies predicted from the values on the left, and as determined on the independent test sample. All values are given in per cent (%).

| | B_{tag} efficiency from AR | BCS cut | SR cut | AR + cuts | B_{tag} efficiency on indep. sample |
|--------------------|--|------------|-----------|-----------|---|
| hadronic B^+ | 0.93 | 73 | 98.6 | 0.67 | 0.65 |
| hadronic B^0 | 0.51 | 82 | 98.2 | 0.41 | 0.40 |
| semileptonic B^+ | 1.25 | 81 | 94.0 | 0.95 | 0.91 |
| semileptonic B^0 | 1.37 | 61 | 93.3 | 0.78 | 0.78 |

efficiencies of 94.0 and 93.3 % for B^+ and B^0 , respectively, are determined.

For a physics analysis, the effect of using B_{tag} candidates created by the Full Event Interpretation can largely be quantified by the reconstruction efficiency, which can be combined with the signal-side efficiency to estimate the total number of signal entities, and the purity, which influences the amount of background that the analyst will have to deal with. Efficiency–purity curves for different cuts on the classifier are shown in [Figure 7.8](#) for hadronic B decay channels, and in [Figure 7.9](#) for semileptonic channels. The values for purity and efficiency are determined only on candidates within the previously defined signal region. It can be seen that the purity can – at the cost of efficiency – be increased to around 95 % for hadronic and around 70 % for semileptonic channels, with a smooth slope in between. The highest efficiency reached corresponds to the post-cut efficiency in the automatic reporting and the total efficiencies given in [Chapter 5](#). It should be noted that the presence of (unsuppressed) continuum background does not change the efficiency, but rather affects the shape of the curve and reduces the overall purity of the sample.

When compared with the maximum efficiencies calculated by the automatic reporting, the highest efficiency that can be reached in the efficiency–purity curves shown is noticeably lower. Theoretically, this might be caused by classifier overtraining, but from the mostly successful overtraining checks included in the automatic reporting one can conjecture that the additional cuts performed on the candidates have a larger effect. This is confirmed by [Table 7.3](#), which shows that the reported ideal efficiency multiplied with the cut efficiencies of the best-candidate selection and the signal region cut predicts the final efficiency (on the independent sample) very well. Only for B^+ candidates reconstructed in semileptonic decay modes is there a slightly larger relative difference of around 4 %, which does not suggest a strong overtraining. It is apparent that the best-candidate selection has the largest impact on the final tag-side efficiency, and also differs greatly between lists, going from 82 % for hadronic B^0 to only 61 % for semileptonic B^0 . Even for semileptonic B^0 candidates, however, 94 % of correct candidates occupy the first or second position when ranked by classifier output, and over 99% within the first five positions. Consequently, these cuts represent a great optimisation opportunity and – depending on the use-case – might allow higher efficiencies. Performing a best-candidate selection after the signal region cut is already likely to be a much better choice for physics analyses.

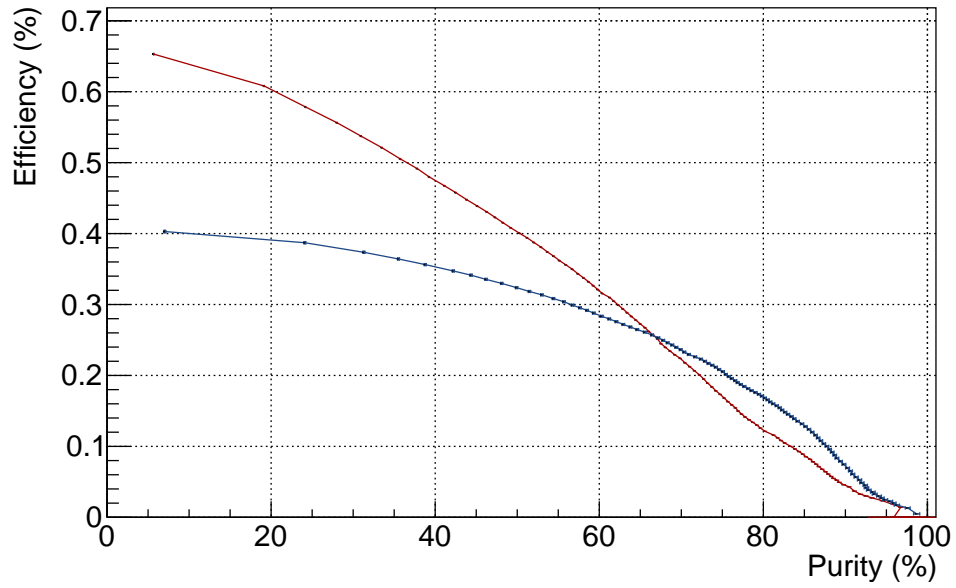


Figure 7.8.: Efficiency–purity curves for B_{tag} candidates reconstructed in hadronic channels. Values for purity and efficiency were determined in the signal region $m_{bc} > 5.27 \text{ GeV}$. B^+ candidates are shown in red, B^0 candidates in blue.

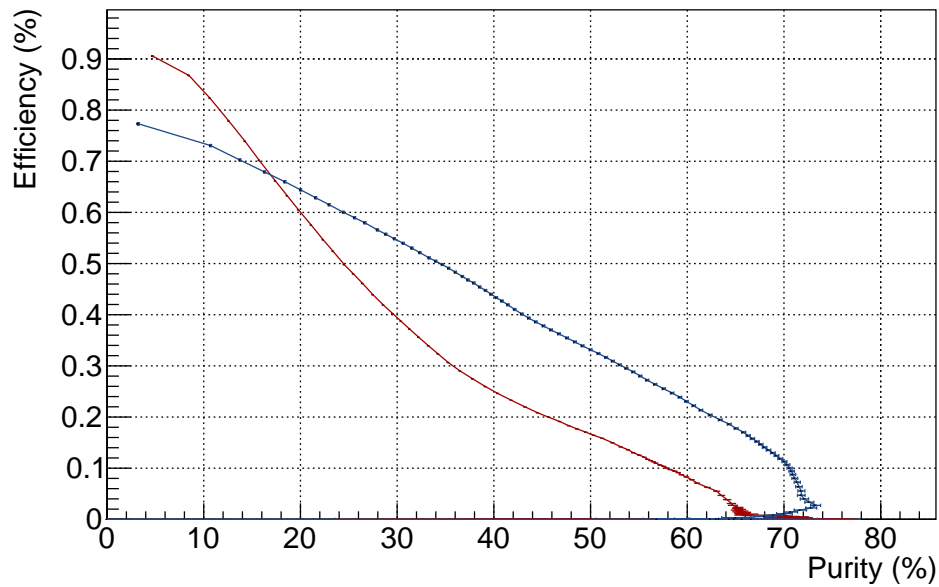


Figure 7.9.: Efficiency–purity curves for B_{tag} candidates reconstructed in semileptonic channels. Values for purity and efficiency were determined in the signal region $|\cos\theta_{B,D^{(*)}l}| < 2$. B^+ candidates are shown in red, B^0 candidates in blue.

7. Estimation of Physics Performance

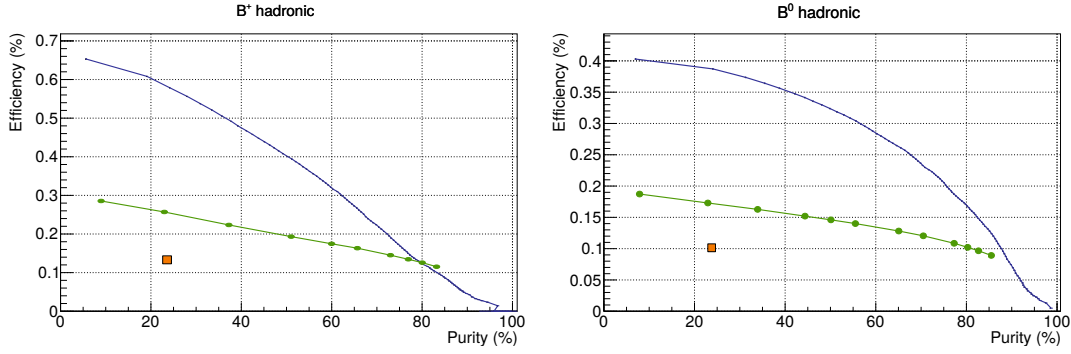


Figure 7.10.: Comparison of hadronic B_{tag} reconstruction efficiency and purity for the Full Event Interpretation on Belle II Monte Carlo (blue), the neural-network-based Full Reconstruction without continuum suppression (green) and the cut-based Full Reconstruction (orange) on Belle data. Data for Full Reconstruction extracted from [75, Fig. 7–8].

7.4. Comparison with Full Reconstruction

The tagging efficiencies and purities can also be compared to those of the Full Reconstruction algorithm used at Belle. For the semileptonic Full Reconstruction only the maximum efficiency of 0.65 %, averaged over both B mesons (B^+ and B^0), is available, which is noticeably lower than the efficiencies of 0.91 % (B^+) and 0.78 % (B^0) reached by the FEI [81, p. 70].³ For hadronic modes, detailed information on efficiencies and purities for different cuts on the network output is available. Figure 7.10 compares efficiency–purity curves for both the cut-based and neural-network-based Full Reconstruction (FR) with those for the Full Event Interpretation. To ensure similar circumstances for the comparison, the signal region for hadrons is the same ($m_{bc} > 5.27 \text{ GeV}$) as that used for the FR curves [75, p. 16].⁴

As the FEI does not include any information on continuum background, it is compared to the Full Reconstruction variant without continuum suppression. Two FR variants including continuum suppression exist; if added to the graphs, their efficiency would be increased slightly for higher purities (Cf. Figure 5.4). They would, however, have no effect on the maximal reconstruction efficiency (the left-most point). For the FR, the efficiencies were extracted on data using a fit to m_{bc} , using a Crystal Ball function for the signal, and an ARGUS function for the background [67, p. 74]. The background shape is largely determined by its distribution below $m_{bc} < 5.27 \text{ GeV}$ and does not describe a narrow, peaking structure on top of its usual shape well. Since for the FEI, the efficiencies are studied on Monte Carlo, the determined values will differ from what would be extracted from a fit: In particular, the strict signal definition used mis-categorises some signal events as background, and thus underestimates the efficiency. On the one hand, reconstruction efficiencies estimated on

³ For the Full Reconstruction efficiency, a narrower signal region of $|\cos\theta_{B,D^{(*)}}| < 1.075$ was used. With this tighter cut, the efficiencies for the FEI decrease to about 0.82 % (B^+) and 0.70 % (B^0).

⁴ In reference [67, p. 75] a pre-selection of $|\Delta E| < 50 \text{ MeV}$ was also applied to the candidates in the same type of plot, which is much harder than the user-cut of $|\Delta E| < 500 \text{ MeV}$ applied by the FEI. It is unclear whether this cut was used for the efficiency–purity curves shown in reference [75]. Using this tighter cut in the FEI selection, the maximum efficiencies are reduced to about 0.55 % for B^+ and 0.34 % for B^0 .

Monte Carlo may be higher than on real data (as was the case at Belle [79]), but due to the restrictiveness of the signal definition this comparison is still quite conservative.

The efficiency here is defined as the number of correct candidates (allowing at most one in each category per event) divided by the total number of Monte Carlo Particles of the same type. For example, for hadronic B^\pm , 132 432 correct candidates were reconstructed in 10 million $B^+ B^-$ Monte Carlo events. The maximum efficiency is then

$$\epsilon_{B^\pm} = \frac{N(B_{\text{correct}}^\pm)}{2N(B^+ B^-)} = \frac{N(B_{\text{correct}}^\pm)}{N(B^+ B^-) + N(B^0 \bar{B}^0)} \approx 0.66\%.$$

This definition is identical to that used for the Full Reconstruction [75, p. 15].⁵

It is evident from these graphs that the total reconstruction efficiency of the Full Event Interpretation at Belle II is much higher than that of its predecessor, reaching a factor two for the maximum efficiency. The performance is similar at the maximum of around 85 % purity reached by the FR, but significantly higher efficiencies are visible over a large range of purities.

7.5. Discussion of Signal Definition

As mentioned in the previous sections, the signal definition used to determine which candidates were correctly reconstructed has a significant effect on the output of the Full Event Interpretation. When using the somewhat strict ‘isSignal’ definition (or ‘isSignalAcceptMissingNeutrino’ for semileptonic channels), a lot of candidates in the m_{bc} signal region were classified as wrong, resulting in a peaking background component in only one of the Monte Carlo types.

A modified Monte Carlo (MC) matching procedure was introduced in Section 4.3, which is able to ignore photons in the assignment of Monte Carlo particles. This allows the creation of a signal definition that includes candidates with a mis-assigned photon. Whether this is acceptable depends strongly on the type of analysis to be performed. Inclusive analyses in particular, or those that rely on the energy E_{ECL} of unassigned clusters in the calorimeter for background rejection, might indeed be adversely effected by including these wrong photons. For other analyses however, which reconstruct a certain decay channel only, low-energy photons added by mistake are not likely to adversely affect the quality of the reconstructed B mesons. To avoid accepting candidates with mis-assigned high-energy photons the variable ‘energyFromWrongPhotons’ can be used to set a limit on the sum of energies of wrongly assigned photons.

Figure 7.11 shows the distribution of the beam-constrained mass m_{bc} for different background components when using the strict ‘isSignal’ signal definition. The continuum and $B^0 \bar{B}^0$ components are equivalent to those in Figure 7.6, but the remaining background from $B^+ B^-$ is broken down into its components. The largest signal-like component is ‘other signal’, which are those candidates which would be associated with a B^\pm if photons are ignored in the algorithm and the summed energy of photons wrongly assigned to it is below 400 MeV (while the standard MC matching associates them with an $\Upsilon(4S)$ MC particle). It can be seen that treating ‘other signal’ as signal instead of background would result in a

⁵ Measurements of $\Gamma(\Upsilon(4S) \rightarrow B^+ B^-) / \Gamma(\Upsilon(4S) \rightarrow B^0 \bar{B}^0)$ by Belle and BaBar are compatible with 1.

7. Estimation of Physics Performance

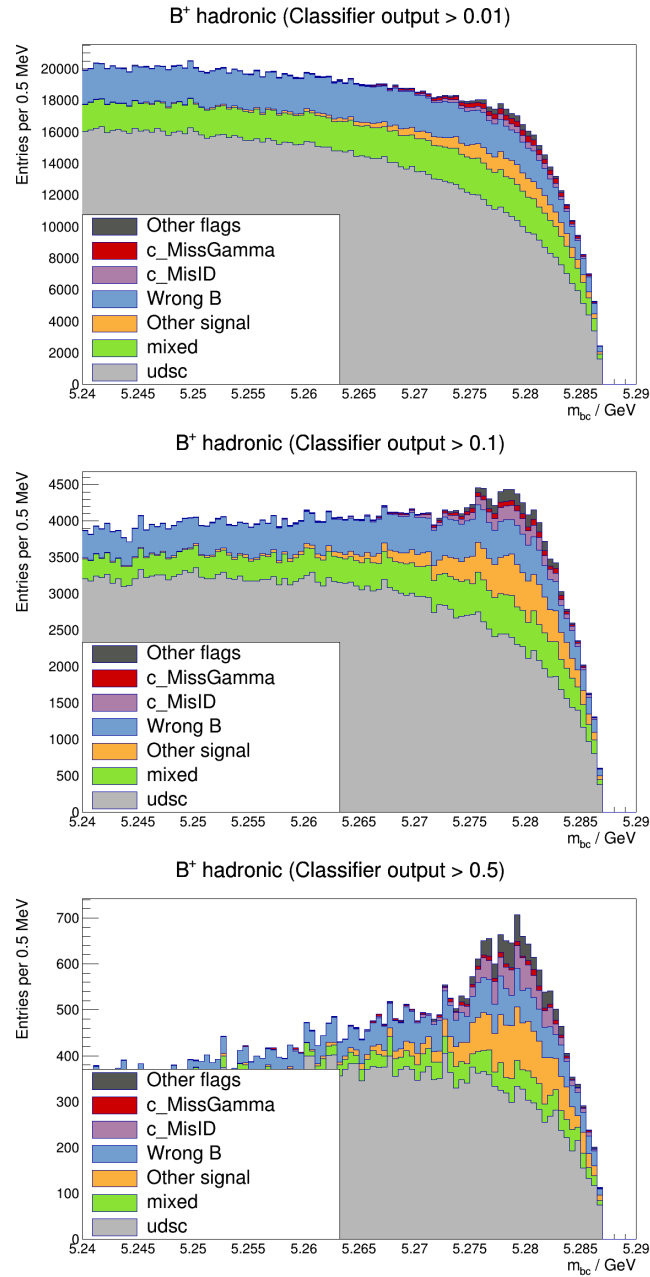


Figure 7.11.: m_{bc} plots for hadronic B^+ candidates, for (from top to bottom) $o_{MVA} > 0.01$, 0.1 and 0.5. Here, only candidates deemed incorrect by 'isSignal' are shown, broken down into those from other Monte Carlo components like continuum (udsc) or $B^0\bar{B}^0$ (mixed), those that would pass the alternative signal definition (other signal), and those in the B^+B^- sample that would not. This last component is further divided into those associated with $\Upsilon(4S)$ (Wrong B), and those associated to a B^\pm Monte Carlo particle but discarded because of certain flags.

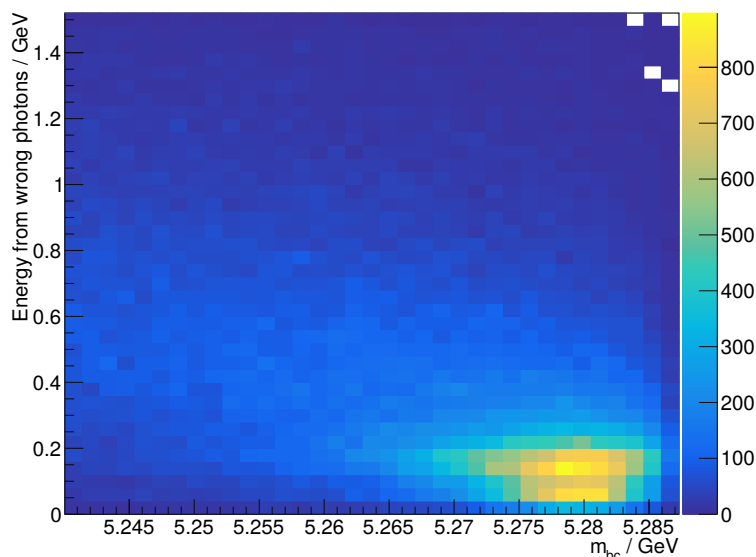


Figure 7.12.: Energy from photons originating from (daughters of) the other B meson plotted over the beam-constrained mass m_{bc} for hadronic B^+ candidates. Only candidates which are associated to $Y(4S)$ and thus marked as incorrect by the standard MC matching procedure are shown.

more flat, non-peaking background shape. For candidates which are discarded because of certain Monte Carlo matching flags (see Table 4.1), the most common flags are `c_MisID` and `c_MissGamma`, with the fraction of mis-identified candidates increasing significantly for harder cuts on the classifier output. The fraction of candidates with missing photons, on the other hand, is reduced with higher cuts, as this type of misreconstruction produces shifts in ΔE (see Section 5.1), which is an input to the multivariate classifier.

The m_{bc} distribution of candidates that would be accepted in addition to those from ‘isSignal’ for a certain sum of wrong photon energies E_{wrong} is shown in Figure 7.12. Since no fully correct candidates are included, energies start at values slightly above zero. Most candidates with an E_{wrong} less than 400 MeV in fact have beam-constrained masses close to the B mass, and only a relatively small tail toward lower values. For more conservative use-cases, smaller cut values can be used. For example, limiting the wrong energy to values below 100 MeV results in candidates with significantly less deviation from the B mass while still retaining about half the candidates a looser cut might include. It is likely possible to further refine the criteria, e.g. by using the maximum energy of mis-assigned photons instead of the sum and/or the number of mis-assigned photons. Even with the relatively loose cut $E_{\text{wrong}} < 400$ MeV, most candidates in the signal region cannot be distinguished from entirely correct candidates in either m_{bc} or ΔE , as can be seen in Figure 7.13.

The effect of the alternative signal definition on the efficiency and purity of the produced candidates can be seen in Figure 7.14. Given that these additional candidates cannot easily be separated in either ΔE or m_{bc} from entirely correct candidates, efficiencies relevant for analyses (or obtained from fits to the data) may be significantly higher than shown Figures 7.8

7. Estimation of Physics Performance

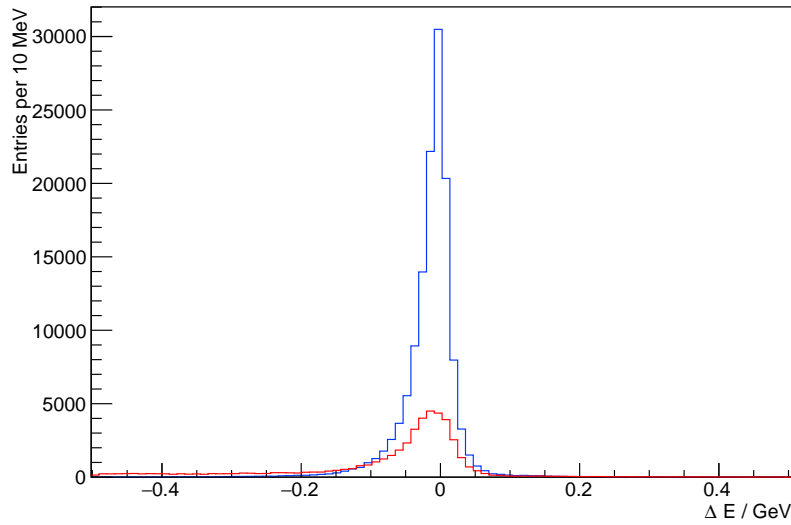


Figure 7.13.: ΔE distribution for hadronic B^+ candidates passing ‘isSignal’ (blue) and for additional candidates that would be included when using the alternative signal definition with $E_{\text{wrong}} < 400$ MeV (red). The additional candidates follow a slightly broader distribution, but most lie near $\Delta E = 0$ and are not easily identifiable. Only candidates fulfilling $|\Delta E| < 0.5$ and $m_{bc} > 5.27$ GeV are shown.

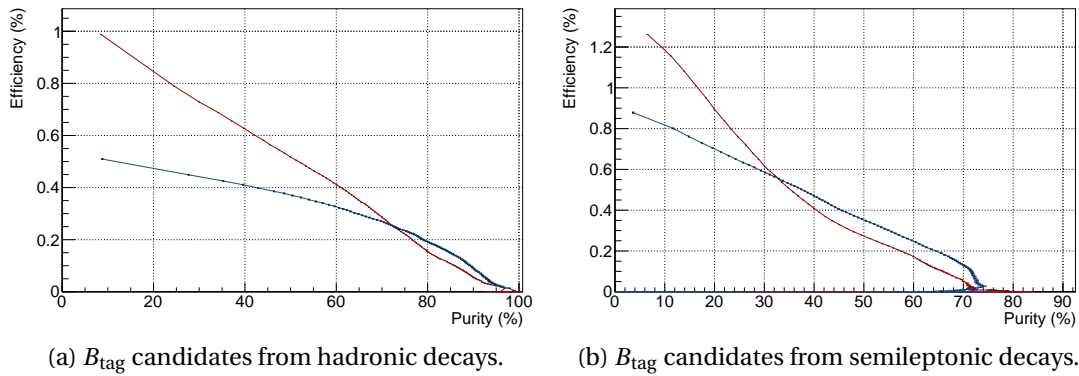


Figure 7.14.: Efficiency–purity curves for B_{tag} candidates using the alternative signal definition (see text). B^+ candidates are shown in red, B^0 candidates in blue. Cf. Figures 7.8 and 7.9.

and 7.9, and much closer to those using the alternative signal definition. In practise, the amount of acceptable candidates will also depend on the signal-side selection and should be evaluated for each analysis.

7.6. Summary and Conclusions

In this chapter, a large-scale training of the Full Event Interpretation was studied on Monte Carlo, and performance indicators relevant to its effect on physics analysis compiled in the form of efficiency–purity curves. The resulting maximal efficiencies greatly surpass those of the Full Reconstruction algorithm used at Belle, both in hadronic and semileptonic B decay modes, while also providing a larger range of purities. Since the signal definition used for these efficiencies is quite conservative, real world applications may see even greater numbers of candidates.

An analysis relying on the Full Event Interpretation (in generic training mode) to produce hadronic tag-side B meson candidates thus receives more than twice the number of candidates, compared to an estimate using the reconstruction efficiencies of the predecessor algorithm at Belle. While efficiencies for semileptonic B mesons are only about 30 % higher, it becomes much easier to use them, as semileptonic B modes are part of the standard configuration and created alongside the hadronic modes. When creating an analysis-specific training of the FEI, further gains in efficiency are expected.

A number of opportunities for further improvement have been identified. These include adding missing high-branching-ratio decay modes to the channel configuration, further optimisation of the multivariate classifier trainings, and providing a more efficient best-candidate-selection procedure to users.

8. FEI on Converted Belle Data

The previous chapter estimated the effect that using the Full Event Interpretation at Belle II would have on a physics analysis, by primarily looking at the efficiency and purity of the reconstruction. Compared to the Full Reconstruction at Belle, a very significant increase in reconstruction efficiency could be observed, but very different factors may have contributed to it. Besides algorithmic improvements, these include bias from using Monte Carlo, the absence of beam background in the simulation, or improvements in the tracking or particle identification detector or software components. This last aspect, in particular, would be very difficult to study using Belle II Monte Carlo alone.

However, the recent creation of modules to convert Belle mDST data into the Belle II format also enables tests of the same algorithm on Belle Monte Carlo and data, which will be discussed in the following sections.

8.1. Conversion of Belle mDST Data

In an effort to make use of the improved Belle II software to analyse data recorded by the Belle experiment, a task-force was created to implement an appropriate conversion procedure. As the necessary input for physics analyses, only the mDST file format was targeted. DST files, which contain raw data including detector hits, are deeply linked with geometry and calibration information stored elsewhere and would be much harder to convert than their processed output. Thus, reprocessing of raw Belle data to, e.g., profit from improved track reconstruction will not be possible.

As already briefly described in [Chapter 3](#), the Belle software used its own I/O format called Panther, which differs greatly from the ROOT-based file format used by Belle II. For the conversion of both the file format and the stored contents (i.e. translating Belle objects to Belle II objects), multiple options were considered:

- Saving Belle II mDST files from within the Belle software. This would entail integrating data classes and ROOT I/O into the Belle software framework and keeping it up-to-date. The conversion also would only be possible for people familiar with the old software.
- Reading Belle mDST files from within BASF2. For this option, the parts responsible for I/O as well as the Panther table definitions would need to be extracted and made available to BASF2. This option would allow saving disk space in many cases since Belle mDST files could be used as input for the Belle II analysis tools, without any separate conversion step. The possibility of reading Belle data in the new software also addresses the need for data preservation, since it avoids the need for maintaining the entirety of the legacy software and ensuring compatibility with modern computing systems [[11](#), p. 58].

8. FEI on Converted Belle Data

- A stand-alone conversion tool that reads Belle mDST and saves their content in Belle II mDST format.

Given its various advantages, the second option was deemed the most appropriate, and a `b2bii` package was created in BASF2 to house the necessary conversion software.

A central module to the conversion software is the `B2BIIMdstInput` module, which reads events from Panther files and makes the stored tables available through a set of global manager objects. As an intermediate step, the `B2BIIFixMdst` module is used on Monte Carlo simulations to apply similar filters as are present on recorded detector data. It also applies a large number of experiment- and run-dependent corrections and calibrations to various reconstructed objects. The filtering is implemented using a module condition (see [Section 3.3.1](#)).

The `B2BIIConvertMdst` module then accesses the data and converts them into BASF2 objects, which are added to the data store (see [Section 3.2](#)). The conversion itself differs for each affected class, since the data formats differ significantly between both experiments. For the following entities a conversion exists:

Tracks Charged tracks are converted from the `Mdst_charged` table into the corresponding Belle II objects, `Track` and `TrackFitResult`.

Particle identification (partial) For each charged track, the associated particle identification information is extracted for the CDC, ACC, and TOF detectors (see [Section 2.2.2](#)), as well as the KLM for muon identification, and stored as separate log-likelihoods in a `PIDLikelihood` object. A simple mapping of ACC to ARICH and TOF to TOP is used to store the log-likelihoods for detectors not present at Belle II in the same object.

ECL clusters Clusters in the electromagnetic calorimeter are converted from `Mdst_ecl` table entries into `ECLCluster` objects, which can be used to reconstruct photons or be associated with tracks.

Monte Carlo particles Information on the true Monte Carlo decays used to generate the event are available on simulated data in the `Gen_hepevt` table. They are converted to `MCParticle` objects.

Photons and π^0 At Belle, preselected lists of photons and π^0 particles are part of the mDST in the form of `Mdst_gamma` and `Mdst_pi0` tables. They are converted to `Particle` objects and appropriate particle lists created to allow usage in further combinations.

'V'-shaped tracks Two charged tracks forming a 'V' some distance away from the interaction region are found in the `Mdst_vee` table and converted to `V0` objects. They can be used to improve the reconstruction efficiency for decays like $K_S^0 \rightarrow \pi^+ \pi^-$ or photons converting into $e^+ e^-$ pairs before reaching the calorimeter.

Beam parameters The measured beam energies for the low- and high-energy ring, as well as the angle between both beams are retrieved from a database server and stored in a `BeamParameters` objects in the data store. As the z axis definition has been changed for Belle II (see [Section 2.1](#)), this information is also saved. In the analysis tools, these energies and angles are automatically used for transformations (e.g. into the center-of-mass system) and dependent variables like m_{bc} and ΔE .

Relations (see [Section 3.2](#)) between the created objects (e.g. from tracks to Monte Carlo particles) are created from indices used for that purpose at Belle.

For some tables present in the Belle mDST format, no conversion has been implemented yet. The most important of these missing items is arguably the electron identification using the ECL, which regrettably is not provided as an independent set of likelihoods, as is the case for Belle II. Currently, identification of electrons on Belle mDST thus relies on the other PID detectors and allows some separation. Clusters in the KLM detector are not currently converted, but might be useful for forming K_L^0 candidates. Data from the EFC (see [Section 2.2.4](#)) is also not converted and would likely require a dedicated class for storage, as there is no similar subsystem in Belle II. Information on the size and position of the interaction region, which would be useful for performing constrained vertex-fits, is also not available yet. A side effect of the conversion is that the produced ROOT files are on average 55 % smaller than the original Panther mDST files, likely as a result of both improved compression in TTree and missing or skipped conversion of some objects.

For the Full Event Interpretation, the converted objects currently available should be enough to create a reasonable training, albeit with a somewhat reduced performance due to incomplete particle identification information for electrons. In the following sections, a generic FEI training (see [Section 6.6.1](#)) will be performed on converted Belle Monte Carlo.

8.2. Monte Carlo Sample

The study was performed on a Monte Carlo sample of 1.7 million $B^0\bar{B}^0$ and 1.7 million B^+B^- events of Belle's experiment 61. In contrast to that used for the Belle II study in the previous chapter, this sample contains beam background, which for Belle was created by adding the detector output for randomly triggered events on real data after the detector simulation [11, p. 50]. As in the previous chapter, no continuum background was included in the training Monte Carlo sample. The provided lists of photon and π^0 candidates, as well as V0 objects for K_S^0 , are not used in the training, and the particles combined manually instead.

8.3. Training

The training itself was similar to the one performed for Belle II, and used the default configuration with practically no adjustments. The PID performance of Belle, however, is significantly worse than expected for Belle II, which motivates some differences. Performance degradation is obvious when looking at [Figure 8.1](#), where the separation from other hypotheses for both electrons and kaons is inferior to that of Belle II. Since for kaons all available likelihoods are included in the conversion, the differences there can be attributed to the improved particle identification systems at Belle II. For electrons, the difference is not as large, and is expected to decrease when information from the electromagnetic calorimeter is added. As this decreased separation between particle hypotheses would produce significantly more final-state particle candidates for each hypothesis and subsequently much greater numbers of candidates for combined particles, the post-classifier cuts (see [Section 6.3.3](#)) are tightened to 0.2 for final-state particles (from 0.1).

8. FEI on Converted Belle Data

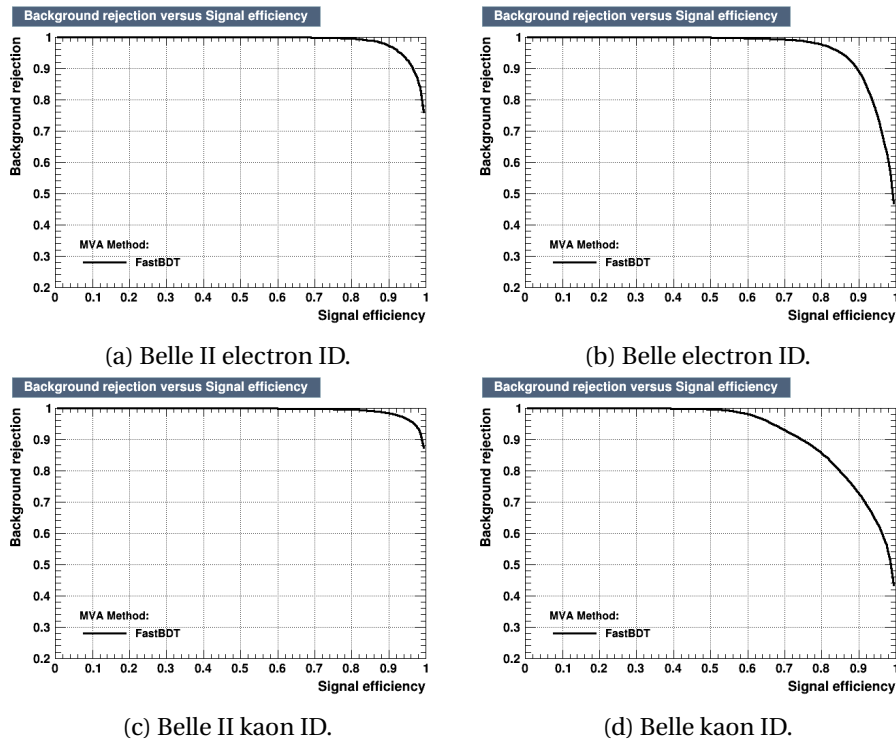


Figure 8.1.: Purity–efficiency plots for electron (top) and kaon (bottom) classification trainings, comparing performance on Belle II (left) and Belle Monte Carlo (right). For electron identification at Belle, no information from the ECL was used.

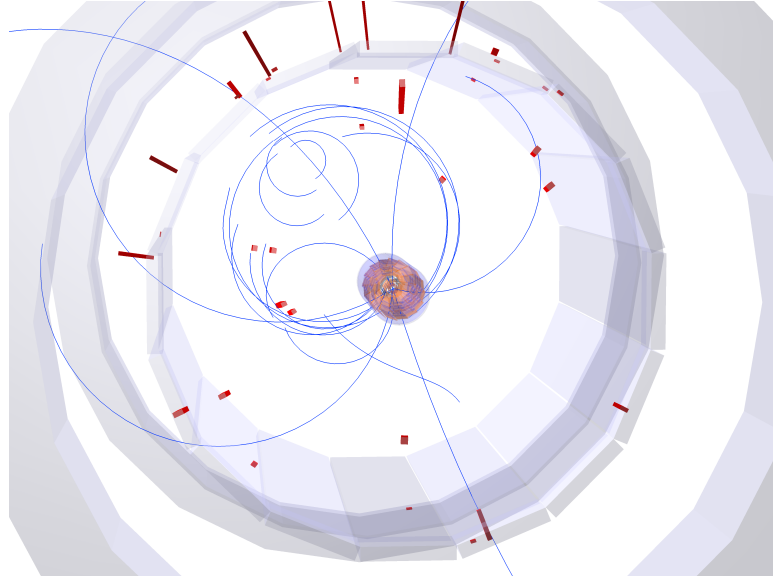


Figure 8.2.: Event display of a Belle Monte Carlo event, showing multiple semi-circles in the upper left for a curling track. N.B.: The Belle II geometry is shown here; in Belle the SVD is smaller and no PXD exists.

Another difference between the two data samples is the tracking performance: For Belle, charged particles curling inside the central drift chamber – i.e. with insufficient transverse momentum to reach the outer detectors – were usually not reconstructed once, but as multiple semi-circles, which can be seen in [Figure 8.2](#). This can artificially increase the number of candidates for charged tracks in some events in a way that is not expected at Belle II. For Belle analyses, no established method of merging these tracks exists. Instead, these tracks were commonly discarded by adding cuts on the impact parameters of tracks. Also, the lack of a standalone track-finding algorithm for the silicon detectors meant lower reconstruction efficiencies for low-momentum tracks.

This – with 40 GB total input file size relatively small – training was performed on a single Core i7-4770 CPU with 4 cores (plus HyperThreading) using the parallel processing feature introduced in [Section 3.5](#) and completed in less than a week. Since no on-disk cache of intermediate particles was used, the final B level stage consumed 206 hours of CPU time in total for reconstruction tasks alone, ignoring the time taken up by trainings, I/O, or the overhead of the interprocess communication.

The efficiencies reported for final-state particles after completion of the training are shown in [Table 8.1](#). Efficiencies greater than one can be seen for pions and kaons, these are an artefact of the efficiency employed by the FEI, which includes the duplicated tracks already seen in [Figure 8.2](#). Since real tracking efficiencies (for primary particles) on Monte Carlo are around 82 % for electrons, and between 90 and 94 % for pions, kaons and muons, the table suggests an overestimation of the efficiency of up to 30 %. It is unclear what, e.g., the strong efficiency reduction for charged kaons through the post-cut means for the actual efficiency. This increases the number of candidates (and thus the combinatorics) significantly for some events, and it might be beneficial to combine or reject tracks to reduce this effect. The

8. FEI on Converted Belle Data

Table 8.1.: Final-state particle efficiency and purity, showing values before and after the applied post-cut. Efficiencies greater than 100 % are due to duplicated tracks (see text).

| Final-state particle | Efficiency in % | | Purity in % | |
|-------------------------|-----------------|----------|-------------|----------|
| | recon. | post-cut | recon. | post-cut |
| π^+ | 105.41 | 102.33 | 49.701 | 60.145 |
| e^+ | 89.21 | 66.68 | 3.283 | 66.758 |
| μ^+ | 97.72 | 46.55 | 2.853 | 77.037 |
| K^+ | 120.15 | 79.42 | 12.547 | 67.361 |
| γ | 95.31 | 83.90 | 29.125 | 52.837 |

Table 8.2.: Per-particle efficiency and purity before and after the applied user-, pre-classifier- and post-classifier-cuts.

| | Efficiency in % | | | | Purity in % | | | |
|--------------------|-----------------|----------|---------|----------|-------------|----------|---------|----------|
| | recon. | user-cut | pre-cut | post-cut | recon. | user-cut | pre-cut | post-cut |
| π^0 | 76.87 | | 63.64 | 31.91 | 2.636 | | 5.606 | 41.744 |
| K_S^0 | 67.33 | | 56.52 | 37.91 | 1.133 | | 2.656 | 75.449 |
| D^0 | 19.94 | | 15.34 | 7.89 | 0.008 | | 0.101 | 19.955 |
| D^+ | 14.53 | | 11.15 | 5.57 | 0.004 | | 0.097 | 20.810 |
| D^{+*} | 4.07 | | 3.83 | 3.21 | 0.720 | | 1.158 | 37.726 |
| D^{0*} | 3.13 | | 3.10 | 1.45 | 0.191 | | 0.338 | 14.558 |
| D_s^+ | 10.05 | | 6.90 | 3.01 | 0.002 | | 0.100 | 17.819 |
| D_s^{+*} | 2.45 | | 2.41 | 1.34 | 0.444 | | 0.795 | 14.406 |
| J/ψ | 9.07 | | 7.46 | 7.46 | 1.943 | | 41.737 | 41.737 |
| B_{had}^+ | 0.47 | 0.42 | 0.34 | 0.32 | 0.001 | 0.084 | 0.208 | 0.209 |
| B_{sl}^+ | 0.64 | | 0.61 | 0.61 | 1.142 | | 2.044 | 2.044 |
| B_{had}^0 | 0.37 | 0.33 | 0.25 | 0.25 | 0.002 | 0.185 | 0.529 | 0.529 |
| B_{sl}^0 | 1.07 | | 0.99 | 0.99 | 0.900 | | 1.948 | 1.948 |

efficiencies and purities can be compared with those in [Table 6.4](#) on page 84, which shows results for a similar small training on Belle II Monte Carlo (1 million $B^0\overline{B}^0$ events, with a post-classifier cut of 0.1). For charged tracks, some differences in efficiencies and purities can be seen, but are likely influenced by the aforementioned duplicated tracks; given the decreased separation power of the associated classifiers, real efficiencies are likely to be lower than at Belle II. For photons, the post-classifier cut efficiency and purity values do not differ greatly between both trainings, while the purity at reconstruction level (29 % for Belle vs. 54 % for Belle II Monte Carlo) shows the effect of added beam background in the Belle sample. The reasonable purity and efficiency after applying a cut on the classifier output thus indicate that the background can be suppressed easily and without additional effort.

Similarly, the efficiencies and purities of combined particles can be found in [Table 8.2](#). Purity values are somewhat similar to those in [Table 6.5](#), but for most particles, efficiencies are reduced. Since the amount of excluded decay channels in both samples is quite similar, this is likely caused by a reduced (real) efficiency of final-state particles. As for particles other than B mesons no correction is performed to exclude candidates reconstructed correctly multiple times (e.g. through duplicated tracks), the actual values are likely to be lower. The post-classifier cut efficiencies for B meson candidates reconstructed in hadronic decay channels are 0.32 % for B^+ and 0.25 % for B^0 , which suggests a performance similar to the Full Reconstruction algorithm, and also similar to that of a FEI training with only a small set of Belle II Monte Carlo.

Many channels, however, have been excluded due to statistics insufficient for training a multivariate classifier. An overview of the number of included decay modes is given in [Table 8.3](#), and shows that hadronic modes of B mesons are most strongly affected. In particular, all channels including $D_s^{+(*)}$ or J/ψ have been removed, which are quite useful due to their high purity. Most semileptonic B decay modes, on the other hand, are included. It should be noted that even for included channels, the remaining number of correct candidates might be barely above the threshold for inclusion, which at 1 000 candidates might produce suboptimal classifier trainings.

8.4. Results

The efficiencies and purities for a sample that includes continuum background and makes use of a more realistic selection are evaluated in the following. When continuum background in proportion to the number of $B\overline{B}$ pairs is added, the size of the data sample rises to 12.9 million events (with a total file size of 132 GB). As in [Section 7.3](#), a best-candidate selection using the classifier output as a ranking variable is performed on each of the four B particle lists.

[Figure 8.3](#) shows the distribution of the classifier output for hadronic and [Figure 8.4](#) for semileptonic B meson candidates. Similar to the plots on p. 106f., continuum background dominates everywhere except for the highest classifier outputs. Again, both the combinatorial background from the B^+B^- and $B^0\overline{B}^0$ Monte Carlo components as well as candidates created from continuum background are greatly suppressed for higher values. Candidates with a classifier output below 0.15 are not shown, but again contain large amounts of background, with hundreds of thousands of candidates, mostly from continuum background. Since the total number of candidates is much less than for the larger training in the previous chapter,

8. FEI on Converted Belle Data

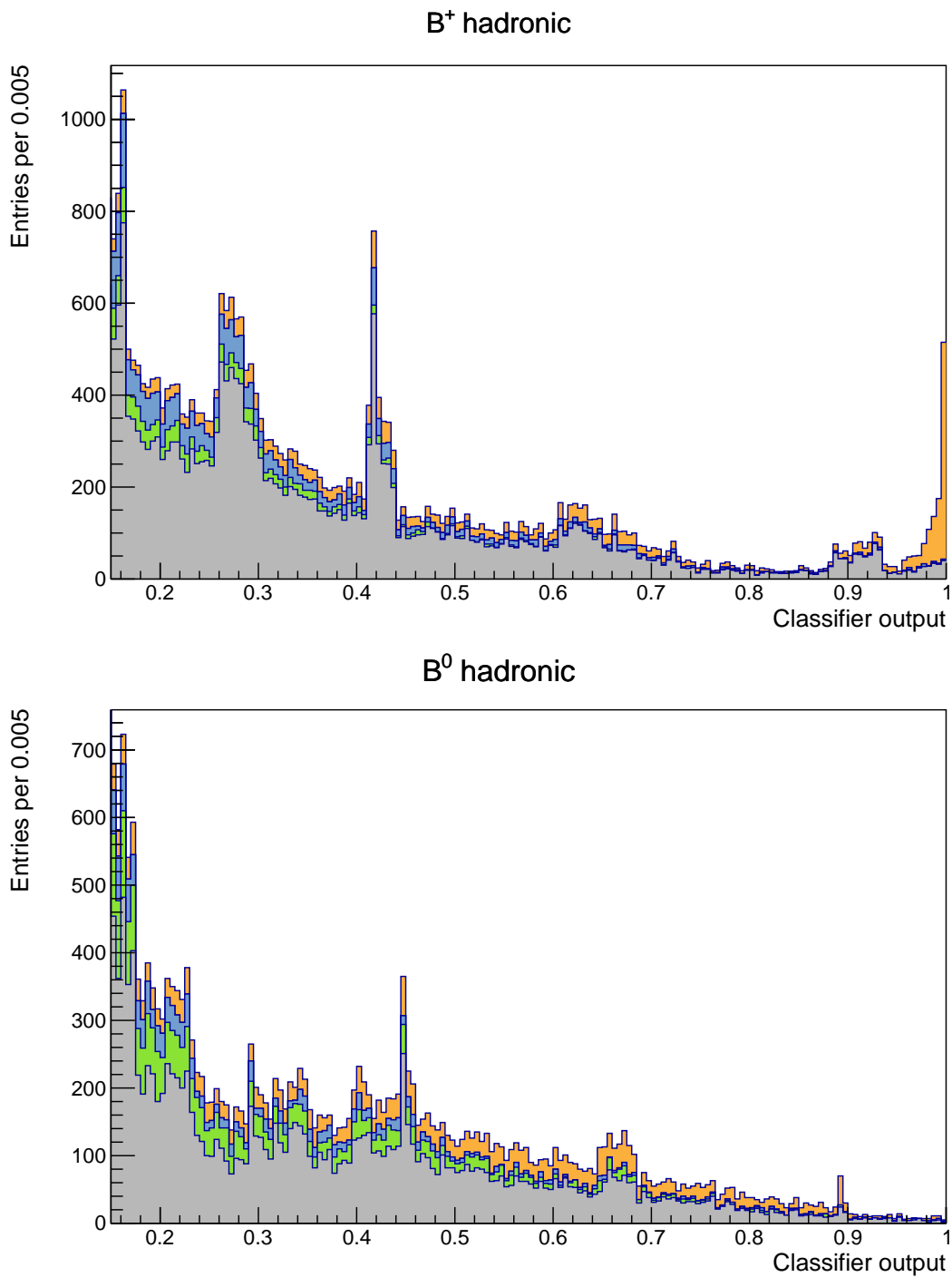


Figure 8.3.: Distributions of the multivariate classifier output for hadronic B decay channels. Correctly reconstructed candidates are shown in orange, background from $B^+ B^-$ events in blue, from $B^0 B^0$ events in green, and continuum background in gray. Large numbers of background candidates are found at values below 0.15; for clarity, these are omitted.

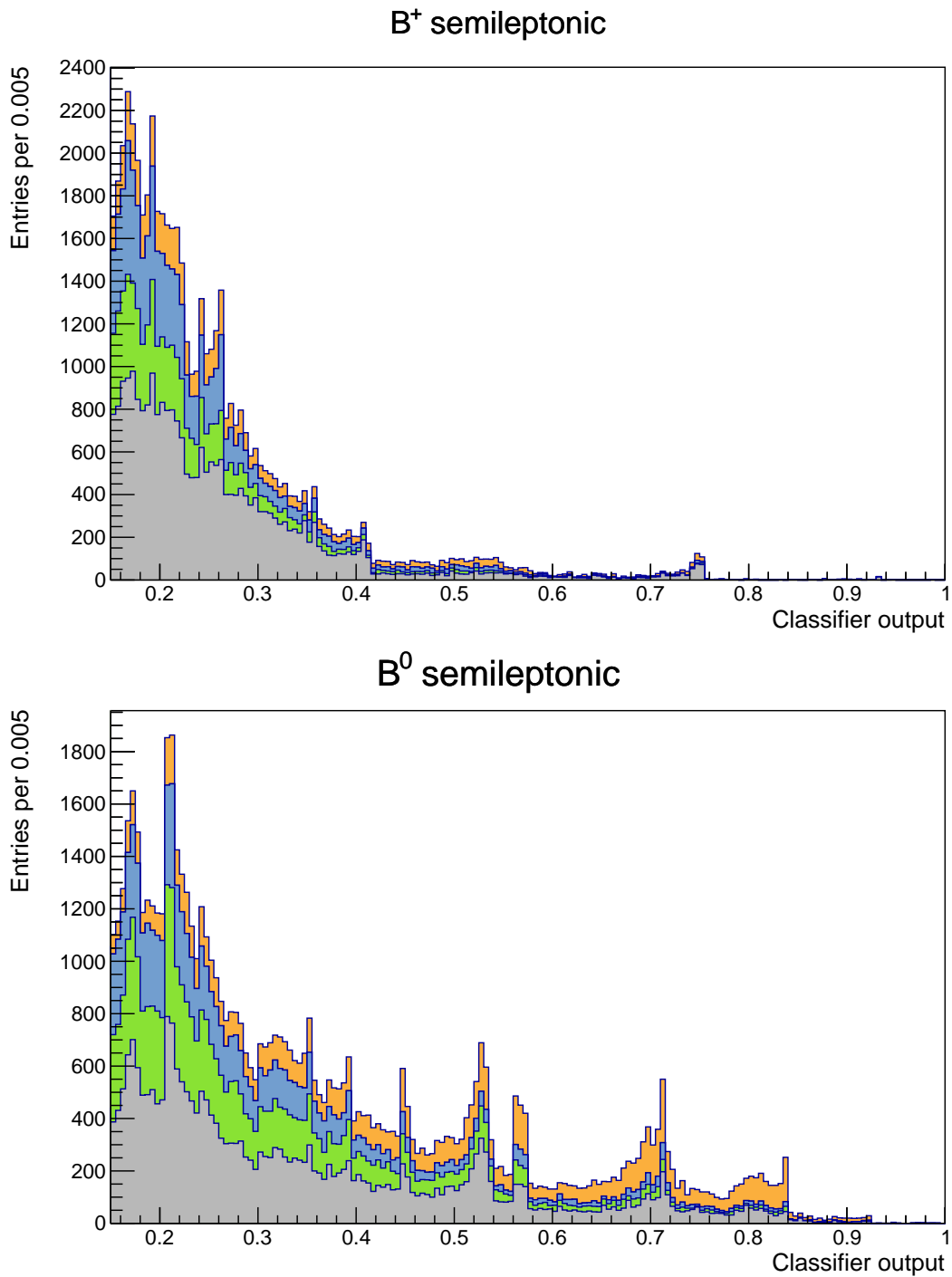


Figure 8.4.: Distributions of the multivariate classifier output for semileptonic B decay channels. Correctly reconstructed candidates are shown in orange, background from $B^+ B^-$ events in blue, from $B^0 \bar{B}^0$ events in green, and continuum background in gray. Large numbers of background candidates are found at values below 0.15; for clarity, these are omitted.

8. FEI on Converted Belle Data

Table 8.3.: Number of decay modes used compared to the number of modes present in the configuration for all types of combined particles in the training.

| Particle | Used decay channels |
|--------------------|---------------------|
| π^0 | 1 / 1 |
| K_S^0 | 1 / 1 |
| J/ψ | 2 / 2 |
| D^0 | 12 / 14 |
| D^+ | 10 / 10 |
| D^{+*} | 1 / 1 |
| D^{0*} | 2 / 2 |
| D_s^+ | 8 / 10 |
| D_s^{+*} | 1 / 2 |
| B_{had}^+ | 7 / 24 |
| B_{sl}^+ | 8 / 8 |
| B_{had}^0 | 7 / 22 |
| B_{sl}^0 | 6 / 8 |

shortcomings of the multivariate classifiers become more apparent. This results in prominent spikes in the classifier output, which again are the result of the transformation applied on the raw output to make it more probability-like and which needs to perform more work to correct the output of low-statistic trainings. Consequently, the size and position of these features also varies by decay mode, meaning they are less visible when more decay modes are included.

Figures 8.7 and 8.8 show the distributions in the control variables m_{bc} and $\cos\theta_{B,D^{(*)}l}$ for different cuts on the classifier output. As before, the plots show a great reduction of the background components when applying the cuts, leading to very pure samples with the hardest cuts. The plots can be compared to Figures 7.6 and 7.7 on p. 108f. For B meson candidates created in semileptonic decay modes the suppression of background appears to be somewhat less than in the Belle II training, which might be caused by the worse performance of the PID, in particular missing electron ID information from the electromagnetic calorimeter.

The efficiency and purity of candidates for different cut values is shown in Figure 8.7 for hadronic, and in Figure 8.8 for semileptonic B candidates. Only candidates in the signal region $m_{bc} > 5.27$ GeV, $|\Delta E| < 0.5$ GeV (for hadronic modes) and $|\cos\theta_{B,D^{(*)}l}| < 2$ (for semileptonic modes) are considered, and the stricter signal definition of ‘isSignal’ or ‘isSignalAcceptMissingNeutrino’ is used (see Section 7.5). When comparing these values with those shown in Figures 7.8 and 7.9 (p. 111), it is evident that much lower total efficiencies are reached in this training in all categories. For harder cuts on the classifier output, the reduced statistics produce significant fluctuations at higher purities. Compared with the post-cut efficiencies for B mesons listed in Table 8.2, the visible efficiencies are again reduced significantly because of the best-candidate selection and signal region cut.

The efficiencies shown here also remain below those achieved by the Full Reconstruction algorithm at Belle (see Figure 5.4), with different factors contributing to the decrease: For

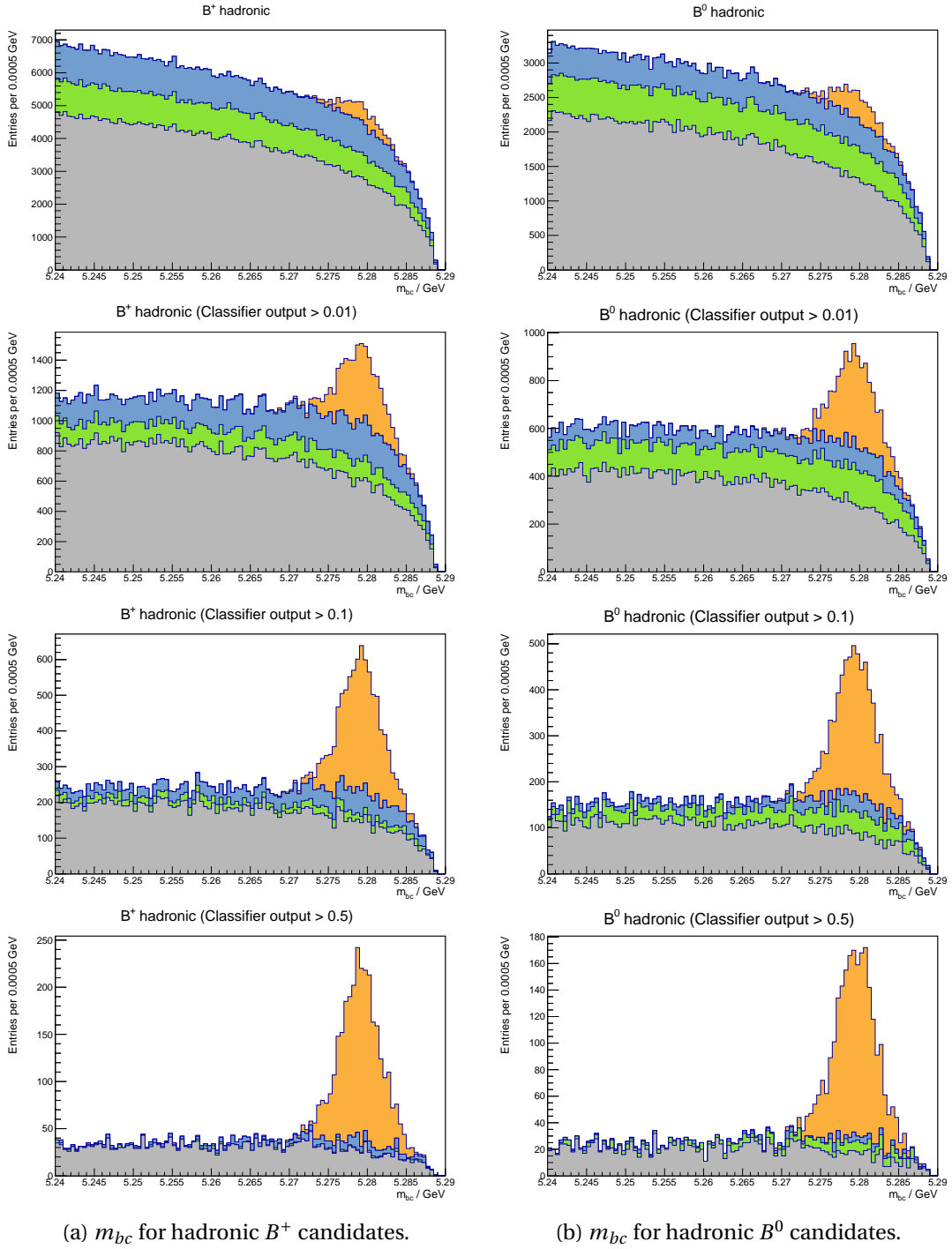
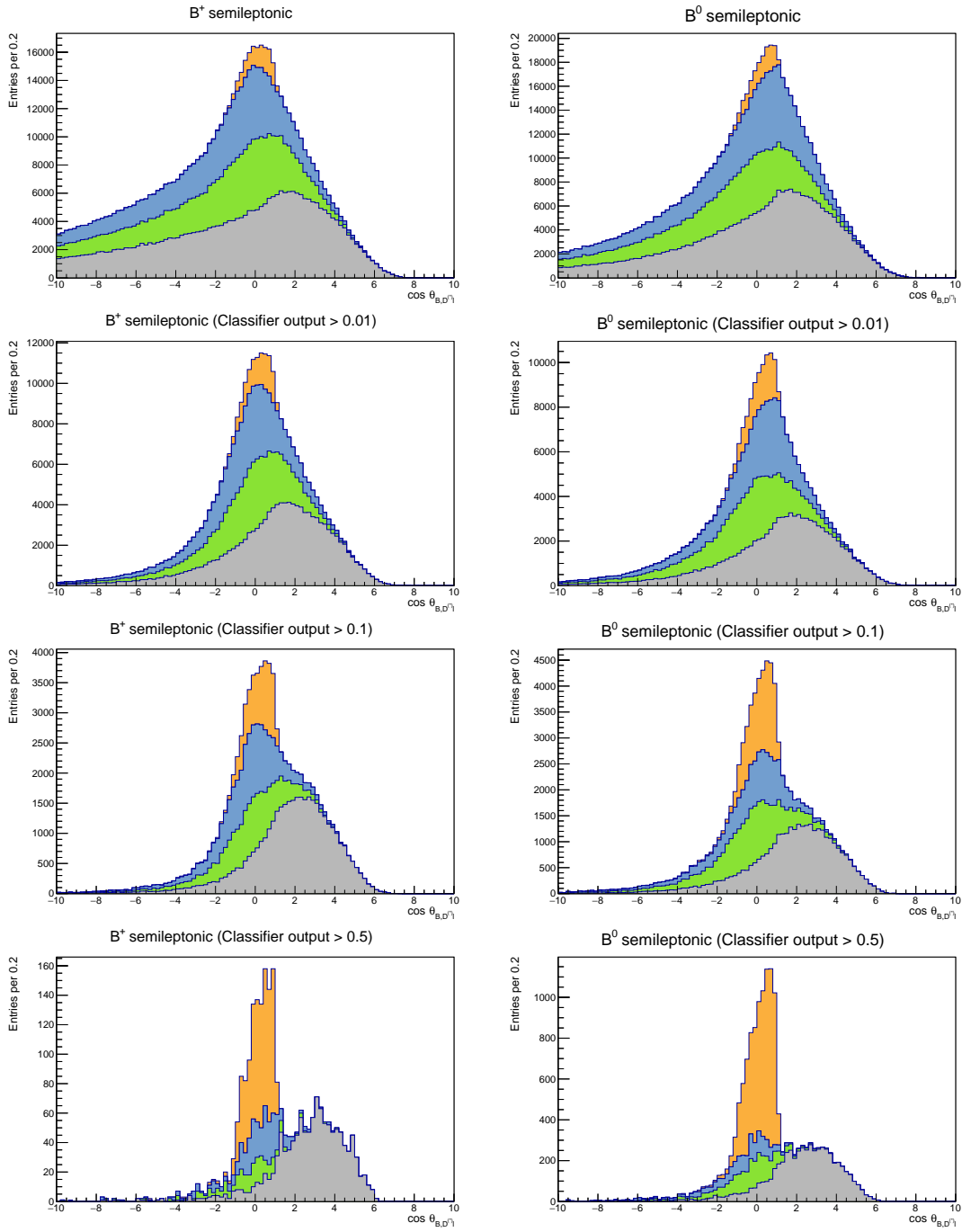


Figure 8.5.: Control plots for hadronic B^+ (left) and B^0 candidates (right), for (from top to bottom) no cut, $\sigma_{\text{MVA}} > 0.01$, 0.1 and 0.5. Correctly reconstructed candidates are shown in orange, background from $B^+ B^-$ events in blue, from $B^0 \bar{B}^0$ events in green, and continuum background in gray.

8. FEI on Converted Belle Data



(a) $\cos\theta_{B,D^{(*)}l}$ for semileptonic B^+ candidates.

(b) $\cos\theta_{B,D^{(*)}l}$ for semileptonic B^0 candidates.

Figure 8.6.: Control plots for semileptonic B^+ (left) and B^0 candidates (right), for (from top to bottom) no cut, $\sigma_{MVA} > 0.01$, 0.1 and 0.5. Correctly reconstructed candidates are shown in orange, background from B^+B^- events in blue, from $B^0\bar{B}^0$ events in green, and continuum background in gray.

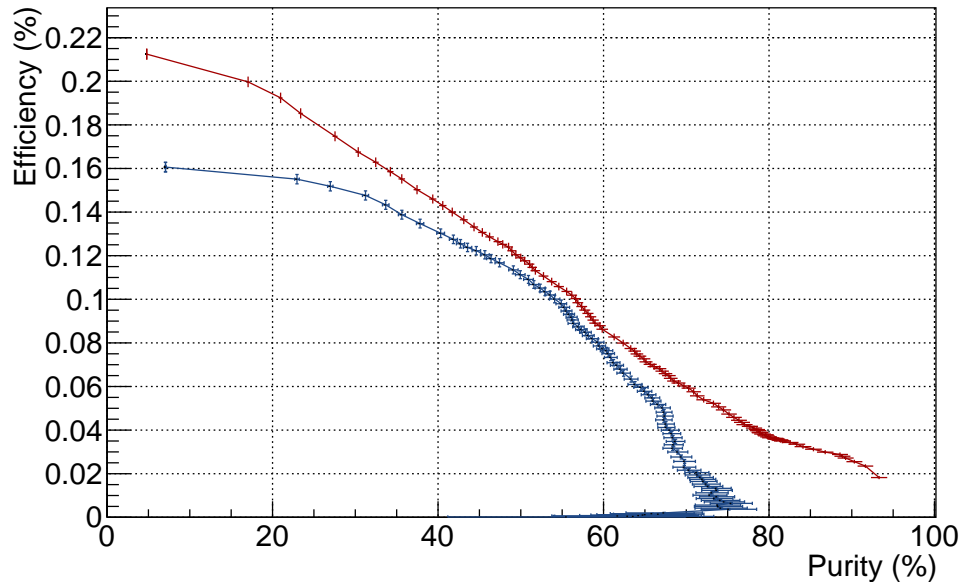


Figure 8.7.: Efficiency–purity curves for B_{tag} candidates reconstructed in hadronic channels. Values for purity and efficiency were determined in the signal region $m_{bc} > 5.27 \text{ GeV}$. B^+ candidates are shown in red, B^0 candidates in blue.

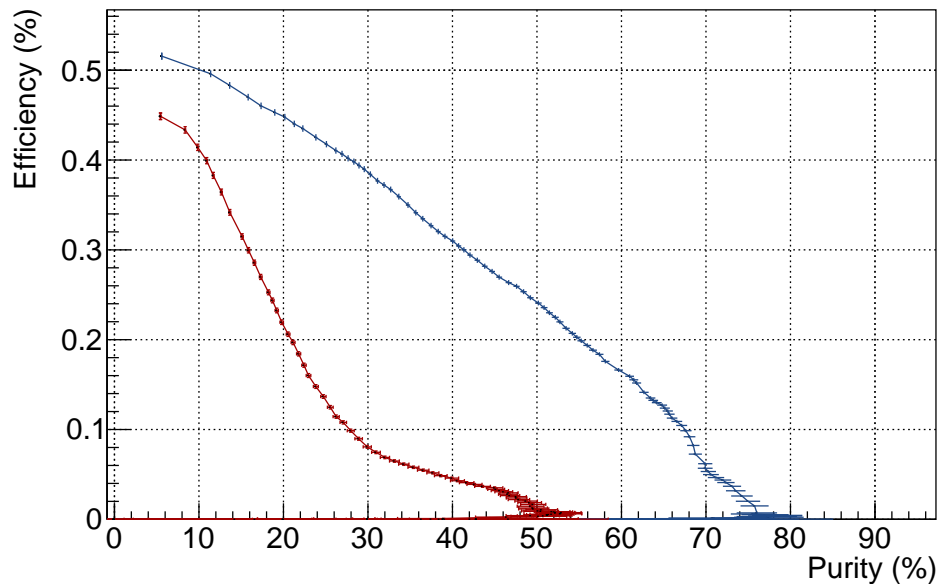


Figure 8.8.: Efficiency–purity curves for B_{tag} candidates reconstructed in semileptonic channels. Values for purity and efficiency were determined in the signal region $|\cos\theta_{B,D^{(*)}l}| < 2$. B^+ candidates are shown in red, B^0 candidates in blue.

8. FEI on Converted Belle Data

hadronic channels, this training used significantly fewer B decay modes than the Full Reconstruction, as can be seen by comparing [Table 6.3](#) and [Table 8.3](#). While this is not the case for semileptonic channels (where more decay modes are used), the semileptonic modes are more strongly affected by low efficiencies for electrons and muons. The relatively low statistics for B meson trainings do not influence the total efficiency, as no post-classifier cuts are applied at B level. They do, however, affect the shape of the efficiency–purity curves, and reduce the maximum possible purity.

8.5. Summary and Conclusions

In this chapter, the conversion of Belle mDST files into the Belle II format was introduced. It allows users performing an analysis on Belle data to use the more advanced analysis tools available in the Belle II software, which in turn also enables analysis software developers to receive feedback and fix shortcomings early. Within the scope of this thesis, the conversion of PID information and of beam parameters was added, as well as the robustness of reading Panther tables improved so as to be able to convert larger Monte Carlo samples.

Building on this conversion, a training of the Full Event Interpretation on a – compared to the sample used in the previous chapter – small set of Belle Monte Carlo was presented. This demonstrates both the possibility of using the FEI on Monte Carlo and data created by the predecessor experiment, but also that one can achieve quite reasonable performance with a training performed on a single 4-core machine. The reconstruction efficiency of this small-scale training is similar to that of a training of similar size on Belle II Monte Carlo, but remains below that presented in the previous chapter or that achieved by the Full Reconstruction algorithm.

Since the Belle Monte Carlo data sample contains beam background, it becomes possible to estimate the effect that adding this background would have for the training on Belle II Monte Carlo in the previous chapter. The expected impact is highest for photons, since each event typically contains a number of calorimeter clusters created through beam background. This is noticeable in the purity for reconstructed photons, but the post-cut efficiency and purity confirm that the multivariate classifiers are able to efficiently filter the clusters in question. The performance of particle identification at Belle is reduced compared to Belle II, which necessitates harder post-cuts for final-state particles to be able to efficiently reduce the number of candidates. A Belle-specific problem that became visible was the amount of duplicated tracks in the sample, which will need to be suppressed separately. These two issues and the post-classifier cuts currently necessary to mitigate them result in significant decreases in efficiency.

However, when comparing the resulting efficiencies with those achieved on a training on a similarly-sized Belle II Monte Carlo sample, efficiencies do not differ greatly. This can be seen in [Table 8.4](#), which summarises the number of channels and efficiencies for hadronic B^0 candidates for both FEI and Full Reconstruction trainings. Since for the small Belle II FEI training, decreased efficiencies can be attributed largely to the large number of excluded channels, it is likely that a larger input sample also results in similar gains in efficiency for Belle as seen on Belle II Monte Carlo.

As demonstrated in this chapter, the Full Event Interpretation thus can also be used with Belle mDST as input, with only minor changes to the configuration. With some optimisation

Table 8.4.: Comparison of the number of included channels and efficiencies for hadronic B^0 candidates for different trainings. Sample size refers to the number of $B^0\bar{B}^0$ events used for the training, post-cut efficiencies are taken from the automatic reporting. The exact sample size for the Full Reconstruction training is unknown, but for B mesons appears to have been at least one stream of exp. 65 [67, p. 71].

| | | sample size | channels | measured eff. | post-cut eff. |
|----------|-----|-------------|----------|---------------|---------------|
| Belle II | FEI | 1 million | 6 | | 0.30 % |
| | FEI | 40 million | 20 | 0.40 % | 0.51 % |
| Belle | FEI | 1.7 million | 7 | 0.16 % | 0.25 % |
| | FR | >20 million | 15 | 0.19 % | |

and larger input samples, trainings are likely to improve significantly and provide higher reconstruction efficiencies for hadronic and semileptonic candidates. Additionally, improvements inherent to the new FEI algorithm – like analysis-specific trainings – also become available to Belle analyses.

9. Conclusions

As stated in the introduction, a key point of the Belle II experiment is the expected factor 40 increase in instantaneous luminosity compared to the Belle experiment. Luminosity, however, is not the only factor influencing the competitiveness of physics analyses; the robustness of the underlying tools and how much time analysts can spend on optimising their selection procedure are important factors as well.

Within the scope of this thesis, the software framework BASF2, used at Belle II for applications ranging from the high-level trigger to physics analyses, was significantly improved. This includes a more consistent interface for exchanging data with other modules, the ability to create modules in Python and to utilise multiple cores in BASF2 to harness the features of modern CPUs while reducing memory requirements. Additionally, an interactive three-dimensional event display for the experiment was created, which serves both debugging and outreach applications.

A novel feature of BASF2 is the modularity of the analysis tools, which provide generic and tested implementations of steps commonly encountered during a physics analysis. As a consequence, users no longer need to spend time on repeatedly implementing standard tasks for each analysis, but can rely on the functionality of the provided tools. Using common tools also has the advantage of making analyses more reproducible, and making bugs in individual analysis steps much less likely.

Building on these tools and the advanced features added to BASF2, an automated framework for the hierarchical reconstruction of B mesons was developed. This framework, called Full Event Interpretation, uses a relatively simple configuration to define particles and their decay modes, creates appropriate reconstruction tasks and resolves dependencies between them. To ensure that this reconstruction can be performed in a limited time, background-reducing cuts are applied in multiple levels and without any user interaction. Crucially, a multivariate classifier is trained for each decay channel to increase the potency of these cuts, while channels with statistics insufficient for a viable training are discarded. Since it would be very inefficient to manually evaluate the results produced in these hundreds of steps, the FEI automatically produces a report listing the efficiencies and purities of the selections applied and contains control plots both for each decay channel and for the B meson candidates that are its final output, as well as other useful information. The algorithm is also easy to extend and maintain, and has a modular code base that is one-seventh the size of its predecessor at Belle. The FEI makes full use of the capabilities of both the analysis tools and the framework itself, and has been tremendously helpful in finding – and fixing – a number of issues within them.

The Full Event Interpretation is expected to have a large impact on physics analyses at Belle II, with a doubled reconstruction efficiency compared to Belle. The complete automation of the training procedure also allows users to create a training that is optimised for their specific signal-side selection, by using the event minus the signal-side as input. Since this

9. Conclusions

procedure discards a large number of background candidates, the classifier trainings become more attuned to the distributions of candidates in this data sample. Consequently, cuts on the output of these classifiers are less likely to discard correct candidates, resulting in further significant increases in efficiencies. With an analysis-specific training, the prior distributions of variables used in the training are also identical or very close to those in the final analysis. Thus, the classifier output for B meson candidates can be interpreted as a probability for the candidate to be correct. For a generic (i.e. not analysis-specific) training, this is not the case and tag-side candidates that cannot be correctly combined with a signal-side selection may have a higher classifier output. In addition to hadronic B decay channels, the Full Event Interpretation also includes semileptonic channels, for which reconstruction efficiencies are much higher. Contrary to the situation at Belle, both types of candidates are provided by the same configuration.

Recent additions to the software also allow the use of Belle data and Monte Carlo simulations within BASF2. While not all available information is converted yet, the Full Event Interpretation algorithm thus can also be used for the existing data, while only requiring minimal modifications to its configuration.

The refined tools presented in this thesis are expected to improve many analyses by providing a robust basis for physics analyses that allows users to concentrate on physics instead of programming, which in turn allows for better measurements. Similarly, the Full Event Interpretation delivers a reconstruction efficiency that is over a factor of two higher than that of its predecessor at Belle for a similar training. Through the conversion of Belle data into the BASF2 data format, these improvements – in particular the analysis-specific training mode of the FEI – are also available to Belle analyses. Accordingly, significant improvements can be expected in the reconstruction efficiency for certain analyses, half a decade after the Belle experiment stopped recording data, while at the same time allowing real-world tests of the Belle II software.

Appendix

A. Example FEI Report for a Small Training

This report¹ contains key performance indicators and control plots of the Full Event Interpretation algorithm, for a training on 4000000 events. The user-, pre-, and post-cuts as well as trained multivariate selection methods are described. Furthermore the resulting purities and efficiencies are stated. In 0.20 % of the events the Full Event Interpretation reconstructed a final particle (B0) correctly.

A.1. Summary

For each final-state particle a multivariate selection method is trained without any previous cuts on the candidates. Afterwards, a post cut is applied on the signal probability calculated by the method. This reduces combinatorics in the following stages of the Full Event Interpretation.

Table A.1.: Final-state particle efficiency and purity before and after the applied post-cut.

| Final-state particle | Efficiency in % | | Purity in % | |
|----------------------|-----------------|----------|-------------|----------|
| | recon. | post-cut | recon. | post-cut |
| π^+ | 78.66 | 78.20 | 66.894 | 83.504 |
| e^+ | 70.02 | 64.68 | 4.872 | 65.700 |
| K^+ | 79.50 | 77.37 | 12.220 | 71.442 |

For each decay channel of each intermediate particle a multivariate selection method is trained after applying a fast pre-cut on the candidates. Afterwards, a post-cut is applied on the signal probability calculated by the method. This reduces combinatorics in the following stages of the Full Event Interpretation. For some intermediate particles, in particular the final B mesons, an additional user-cut is applied before all other cuts.

¹ The following pages were automatically generated at the end of the training described. Some parts were edited for visual consistency.

A. Example FEI Report for a Small Training

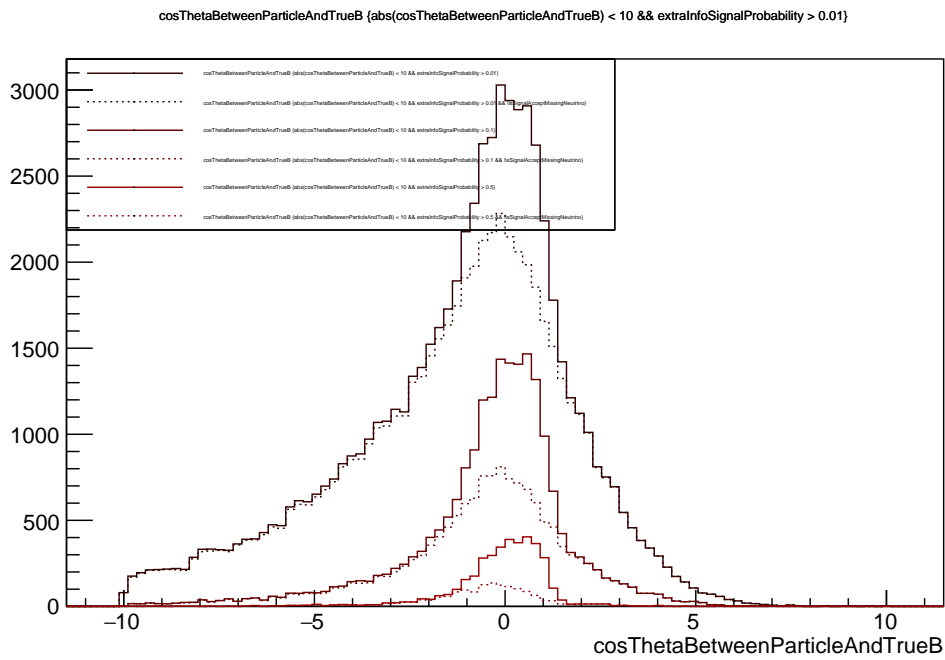


Figure A.1.: CosThetaBDL of all final B^0 candidates with different cuts on the signal probability. Dashed lines represents only background candidates. Solid lines represent all candidates.

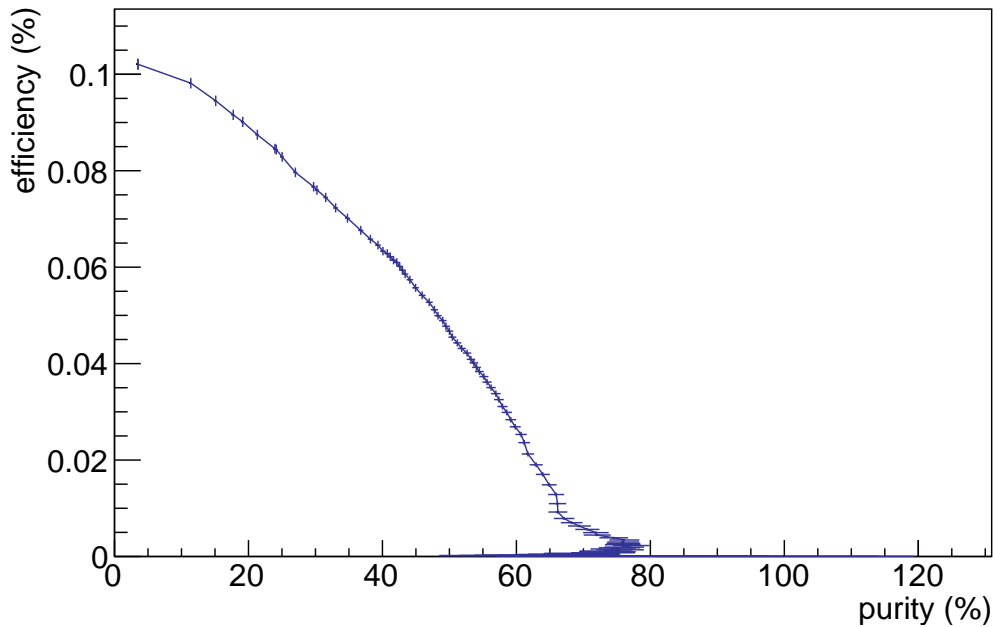






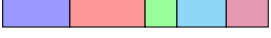
Figure A.2.: Purity over efficiency of all final B^0 candidates with different cuts on the signal probability.

Table A.2.: Per-particle efficiency and purity before and after the applied user-, pre- and post-cut.

| Particle | Efficiency in % | | | | Purity in % | | | |
|----------|-----------------|----------|---------|----------|-------------|----------|---------|----------|
| | recon. | user-cut | pre-cut | post-cut | recon. | user-cut | pre-cut | post-cut |
| D^+ | 5.48 | | 5.21 | 5.00 | 0.434 | | 4.983 | 13.600 |
| B^0 | 0.10 | | 0.10 | 0.10 | 2.059 | | 3.504 | 3.504 |

A.2. CPU time per channel

Table A.3.: Total CPU time spent in event() calls for each channel. Bars show **ParticleLoader**, **ParticleCombiner**, **ParticleVertexFitter**, **MCMatching**, **TMVAExpert**, **Other**, in this order. Does not include I/O, initialisation, training, post-cuts etc.

| Decay | CPU time | by module | per (true) candidate | Relative time |
|-----------------------------------|----------|---|---------------------------|---------------|
| π^+ | 29m50s |  | 207 μ s (207 μ s) | 25.48% |
| e^+ | 28m16s |  | 934 μ s (934 μ s) | 24.15% |
| K^+ | 27m40s |  | 365 μ s (365 μ s) | 23.64% |
| $D^+ \rightarrow K^- \pi^+ \pi^+$ | 26m59s |  | 452 μ s (—) | 23.05% |
| $B_{sl}^0 \rightarrow D^- e^+$ | 4m18s |  | 1ms (31ms) | 3.68% |
| Total | 1h57m | | | 100.00% |

A.3. Particle configuration

```

pi+:generic
  PostCutConfiguration: value=0.1
  MVAConfiguration: name=FastBDT, type=Plugin, config=!H:!V:NTrees=100:Shrinkage=0.10:
    RandRatio=0.5:NCutLevel=8:NTreeLayers=3, target=isPrimarySignal
  Variables: eid, eid_dEdx, eid_TOP, eid_ARICH, eid_ECL, Kid, Kid_dEdx, Kid_TOP,
    Kid_ARICH, prid, prid_dEdx, prid_TOP, prid_ARICH, muid, muid_dEdx,
    muid_TOP, muid_ARICH, muid_KLM, p, pt, pz, dr, dz, chiProb

e+:generic
  PostCutConfiguration: value=0.1
  MVAConfiguration: name=FastBDT, type=Plugin, config=!H:!V:NTrees=100:Shrinkage=0.10:
    RandRatio=0.5:NCutLevel=8:NTreeLayers=3, target=isPrimarySignal
  Variables: eid, eid_dEdx, eid_TOP, eid_ARICH, eid_ECL, Kid, Kid_dEdx, Kid_TOP,
    Kid_ARICH, prid, prid_dEdx, prid_TOP, prid_ARICH, muid, muid_dEdx,
    muid_TOP, muid_ARICH, muid_KLM, p, pt, pz, dr, dz, chiProb

K+:generic
  PostCutConfiguration: value=0.1
  MVAConfiguration: name=FastBDT, type=Plugin, config=!H:!V:NTrees=100:Shrinkage=0.10:
    RandRatio=0.5:NCutLevel=8:NTreeLayers=3, target=isPrimarySignal
  Variables: eid, eid_dEdx, eid_TOP, eid_ARICH, eid_ECL, Kid, Kid_dEdx, Kid_TOP,
    Kid_ARICH, prid, prid_dEdx, prid_TOP, prid_ARICH, muid, muid_dEdx,
    muid_TOP, muid_ARICH, muid_KLM, p, pt, pz, dr, dz, chiProb

D+:generic
  All channels use the same MVA configuration
  MVAConfiguration: name=FastBDT, type=Plugin, target=isSignal, config=!H:
    !V:NTrees=100:Shrinkage=0.10:RandRatio=0.5:NCutLevel=8:
    NTreeLayers=3
  Shared Variables: daughterProductOf(extraInfo(SignalProbability)), daughter(0,
    extraInfo(SignalProbability)), daughter(1,extraInfo(SignalProbability)
    )), daughter(2,extraInfo(SignalProbability)), chiProb,
    daughter(0, chiProb), daughter(1, chiProb), daughter(2,
    chiProb), useRestFrame(daughter(0, p)), useRestFrame(daughter(1,
    p)), useRestFrame(daughter(2, p)), useRestFrame(daughter(0,
    distance)), useRestFrame(daughter(1, distance)), useRestFrame(daught
    er(2, distance)), decayAngle(0), decayAngle(1), decayAngle(2),
    cosAngleBetweenMomentumAndVertexVector, daughterInvariantMass(0,
    1), daughterInvariantMass(0,2), daughterInvariantMass(1,
    2), daughterInvariantMass(0,1,2), Q
  PreCutConfiguration: variables=M, efficiency=0.95, purity=0.001
  PreCutConfiguration: binning=(500, 1.7, 1.95)
  PreCutConfiguration: userCut=
  PostCutConfiguration: value=0.01
D+:generic ==> K-:generic pi+:generic pi+:generic (decayModeID: 0)
  Individual Variables:

B0:semileptonic
  All channels use the same MVA configuration
  MVAConfiguration: name=FastBDT, type=Plugin, target=isSignalAcceptMissingNeutrino,
    config=!H:!V:NTrees=100:Shrinkage=0.10:RandRatio=0.5:
    NCutLevel=8:NTreeLayers=3
  Shared Variables: daughterProductOf(extraInfo(SignalProbability)), daughter(0,
    extraInfo(SignalProbability)), daughter(1,extraInfo(SignalProbability)
    )), chiProb, daughter(0, chiProb), daughter(1, chiProb),
    useRestFrame(daughter(0, p)), useRestFrame(daughter(1,

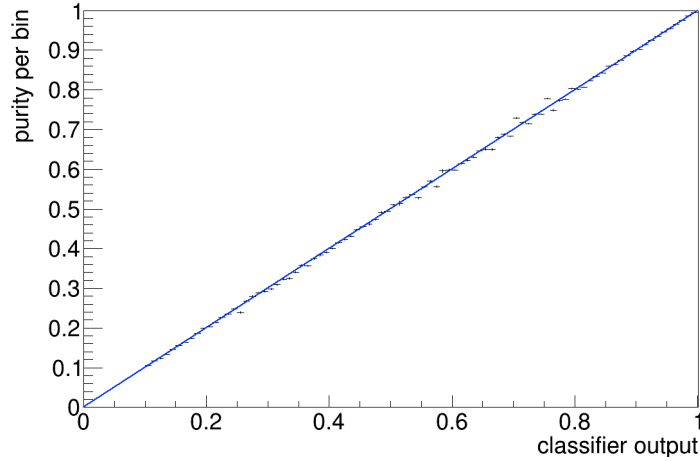
```

A.3. Particle configuration

```
p)), useRestFrame(daughter(0, distance)), useRestFrame(daughter(1,
distance)), decayAngle(0), decayAngle(1), cosAngleBetweenMomentumAnd
VertexVector, dr, dz, dx, dy, distance, significanceOfDistance,
deltaE
PreCutConfiguration: variables=daughterProductOf(extraInfo(SignalProbability)),
efficiency=0.95, purity=0.0001
PreCutConfiguration: binning=[0.0004510929897325762, 0.0006766394845988643,
0.0010149592268982965, 0.0015224388403474447, 0.00228365826052116
7, 0.0034254873907817508, 0.005138231086172626, 0.0077073466292589
396, 0.011561019943888409, 0.017341529915832612, 0.026012294873748
92, 0.03901844231062338, 0.05852766346593507, 0.0877914951989026,
0.13168724279835392, 0.19753086419753085, 0.2962962962962963,
0.4444444444444444, 0.6666666666666666, 1.0]
PreCutConfiguration: userCut=
PostCutConfiguration: None
B0:semileptonic ==> D-:generic e+:generic (decayModeID: 0)
Individual Variables:
```

A.4. Final state particles

A.4.1. π^+

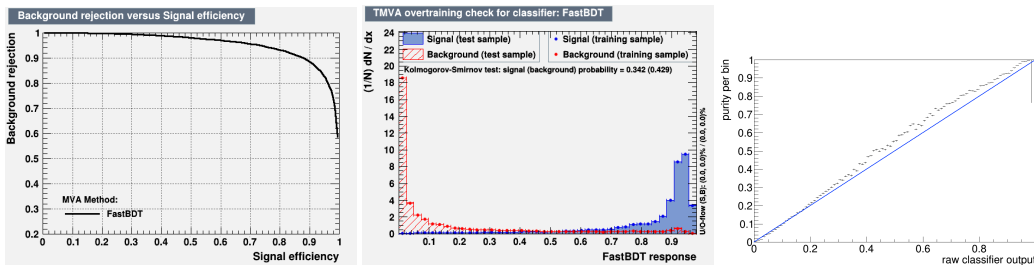


MVA

Table A.4.: List of variables used in the training.

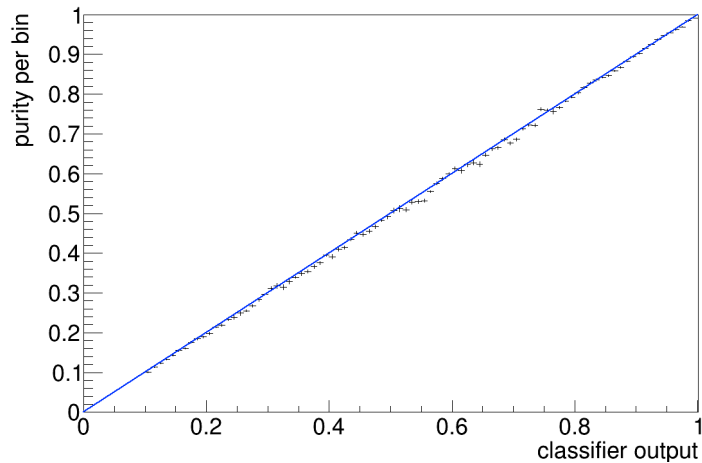
| Name | Description | Rank | Importance |
|------------|---|------|------------|
| muid_dEdx | muon identification probability from dEdx measurement | 1 | 247.90 |
| Kid_dEdx | kaon identification probability from dEdx measurement | 2 | 244.70 |
| Kid_TOP | kaon identification probability from TOP | 3 | 243.60 |
| eid_dEdx | electron identification probability from dEdx measurement | 4 | 135.10 |
| eid | electron identification probability | 5 | 119.40 |
| dr | transverse distance in respect to IP | 6 | 84.99 |
| p | momentum magnitude | 7 | 49.45 |
| Kid | kaon identification probability | 8 | 45.95 |
| chiProb | chi ² probability of the fit | 9 | 44.77 |
| muid | muon identification probability | 10 | 40.59 |
| dz | z in respect to IP | 11 | 40.14 |
| muid_TOP | muon identification probability from TOP | 12 | 11.76 |
| prid | proton identification probability | 13 | 9.52 |
| muid_KLM | muon identification probability from KLM | 14 | 9.09 |
| muid_ARICH | muon identification probability from ARICH | 15 | 4.28 |
| eid_ECL | electron identification probability from ECL | 16 | 3.50 |
| prid_dEdx | proton identification probability from dEdx measurement | 17 | 3.14 |
| pt | transverse momentum | 18 | 1.34 |

| | | | |
|------------|--|----|------|
| eid_TOP | electron identification probability from TOP | 19 | 0.17 |
| pz | momentum component z | 20 | 0.15 |
| eid_ARICH | electron identification probability from ARICH | 21 | 0.15 |
| Kid_ARICH | kaon identification probability from ARICH | | |
| prid_TOP | proton identification probability from TOP | | |
| prid_ARICH | proton identification probability from ARICH | | |



TMVA plots for Plugin/FastBDT using the configuration string `!H:!V:NTrees=100:Shrinkage=0.10:RandRatio=0.5:NCutLevel=8:NTreeLayers=3` with target variable `isPrimarySignal`. The training used 4421747 signal and 3282554 background samples. 4421747 signal and 3282554 background samples were used for testing.

A.4.2. e^+

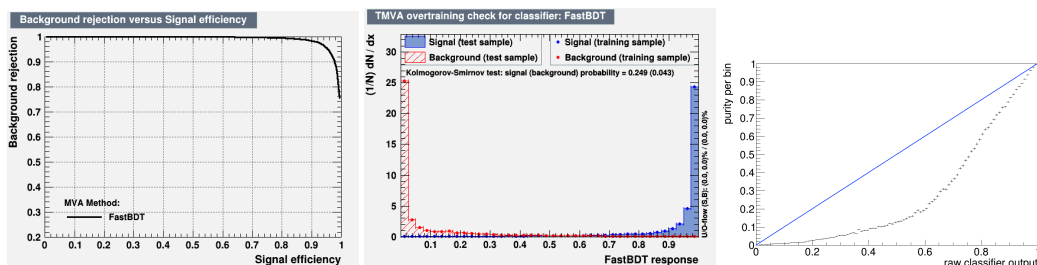


MVA

Table A.5.: List of variables used in the training.

A. Example FEI Report for a Small Training

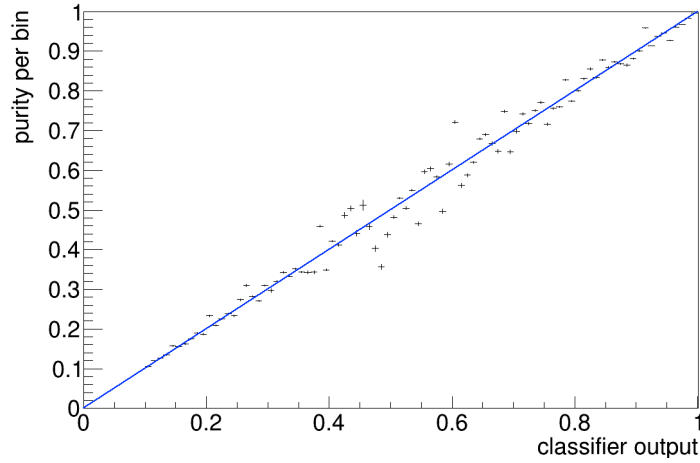
| Name | Description | Rank | Importance |
|------------|---|------|------------|
| prid_ARICH | proton identification probability from ARICH | 1 | 143.10 |
| pz | momentum component z | 2 | 21.83 |
| prid_TOP | proton identification probability from TOP | 3 | 21.46 |
| eid_dEdx | electron identification probability from dEdx measurement | 4 | 17.25 |
| Kid_TOP | kaon identification probability from TOP | 5 | 14.18 |
| dr | transverse distance in respect to IP | 6 | 13.64 |
| Kid_dEdx | kaon identification probability from dEdx measurement | 7 | 12.04 |
| eid_TOP | electron identification probability from TOP | 8 | 11.79 |
| prid | proton identification probability | 9 | 11.64 |
| pt | transverse momentum | 10 | 7.87 |
| dz | z in respect to IP | 11 | 5.03 |
| prid_dEdx | proton identification probability from dEdx measurement | 12 | 3.27 |
| muid_KLM | muon identification probability from KLM | 13 | 3.15 |
| eid_ECL | electron identification probability from ECL | 14 | 2.65 |
| eid_ARICH | electron identification probability from ARICH | 15 | 1.89 |
| eid | electron identification probability | 16 | 0.89 |
| chiProb | chi ² probability of the fit | 17 | 0.76 |
| muid_dEdx | muon identification probability from dEdx measurement | 18 | 0.48 |
| p | momentum magnitude | 19 | 0.05 |
| Kid | kaon identification probability | 20 | 0.04 |
| muid | muon identification probability | 21 | 0.03 |
| muid_TOP | muon identification probability from TOP | 22 | 0.02 |
| Kid_ARICH | kaon identification probability from ARICH | | |
| muid_ARICH | muon identification probability from ARICH | | |



TMVA plots for Plugin/FastBDT using the configuration string !H:!V:NTrees=100:Shrinkage=0.10:RandRatio=0.5:NCutLevel=8:NTreeLayers=3 with target variable *isPrimarySignal*. The

training used 966038 signal and 4716078 background samples. 966038 signal and 4716078 background samples where used for testing.

A.4.3. K^+



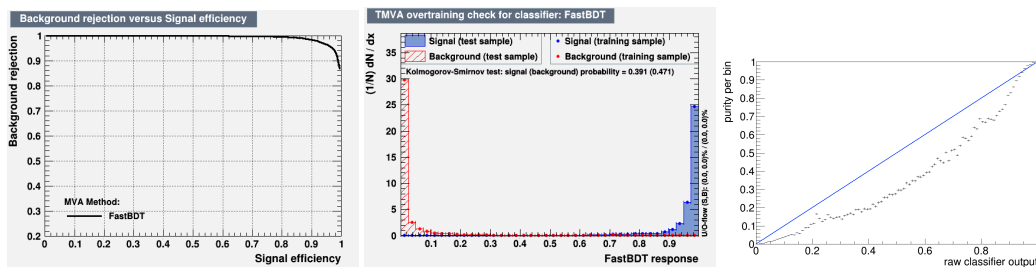
MVA

Table A.6.: List of variables used in the training.

| Name | Description | Rank | Importance |
|------------|---|------|------------|
| eid_TOP | electron identification probability from TOP | 1 | 215.40 |
| Kid_dEdx | kaon identification probability from dEdx measurement | 2 | 105.30 |
| pz | momentum component z | 3 | 64.12 |
| Kid_TOP | kaon identification probability from TOP | 4 | 62.68 |
| prid_TOP | proton identification probability from TOP | 5 | 50.01 |
| Kid | kaon identification probability | 6 | 49.79 |
| eid_dEdx | electron identification probability from dEdx measurement | 7 | 32.69 |
| muid_dEdx | muon identification probability from dEdx measurement | 8 | 25.85 |
| muid | muon identification probability | 9 | 25.82 |
| pt | transverse momentum | 10 | 21.31 |
| eid | electron identification probability | 11 | 20.75 |
| dz | z in respect to IP | 12 | 17.67 |
| prid_dEdx | proton identification probability from dEdx measurement | 13 | 16.81 |
| muid_ARICH | muon identification probability from ARICH | 14 | 15.45 |
| muid_KLM | muon identification probability from KLM | 15 | 6.04 |
| prid | proton identification probability | 16 | 5.05 |

A. Example FEI Report for a Small Training

| | | | |
|------------|--|----|------|
| eid_ARICH | electron identification probability from ARICH | 17 | 4.08 |
| dr | transverse distance in respect to IP | 18 | 3.72 |
| eid_ECL | electron identification probability from ECL | 19 | 2.73 |
| Kid_ARICH | kaon identification probability from ARICH | 20 | 0.12 |
| chiProb | χ^2 probability of the fit | 21 | 0.06 |
| prid_ARICH | proton identification probability from ARICH | 22 | 0.02 |
| p | momentum magnitude | 23 | 0.01 |
| muid_TOP | muon identification probability from TOP | | |



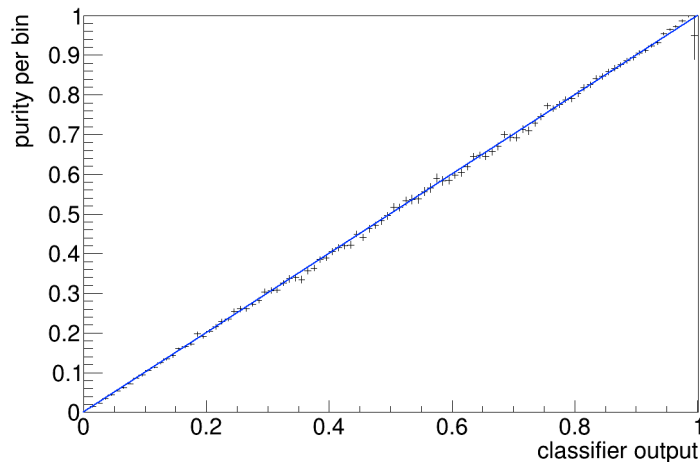
TMVA plots for Plugin/FastBDT using the configuration string `!H:!V:NTrees=100:Shrinkage=0.10:RandRatio=0.5:NCutLevel=8:NTreeLayers=3` with target variable `isPrimarySignal`. The training used 2423235 signal and 4351778 background samples. 2423235 signal and 4351778 background samples were used for testing.

A.5. Particle: D^+

In the reconstruction of D^+ 1 out of 1 possible channels were used.

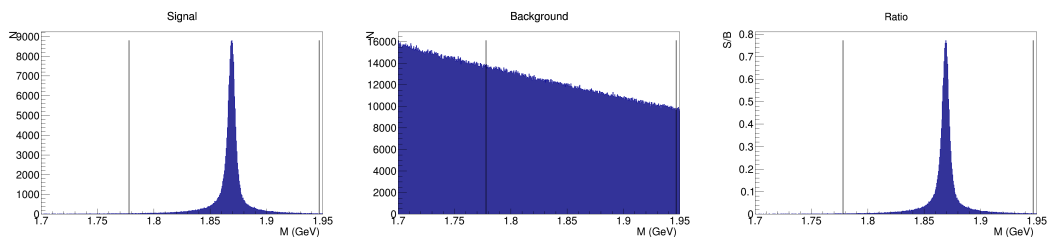
- $D^+ \rightarrow K^- \pi^+ \pi^+$

This amounts to 218327 signal events and 50088395 background events in total before cuts. Channel-specific pre-cuts and a particle-specific post cut on the signal probability of the particle were applied. After all cuts 199137 signal events and 1265127 background events remained.



A.5.1. Channel: $D^+ \rightarrow K^- \pi^+ \pi^+$

Pre-cut determination



Variable M for combinations of daughter candidates of this channel. The PreCut range (1.78, 1.95) for this channel is marked with vertical lines in the plots above. The PreCuts were determined on variable M with a desired total signal efficiency of 0.95000. For this channel only the efficiency is 0.95063 with a purity of 0.04983, corresponding to 207548.0 signal and 3957585.0 background events.

MVA

Table A.7.: List of variables used in the training.

A. Example FEI Report for a Small Training

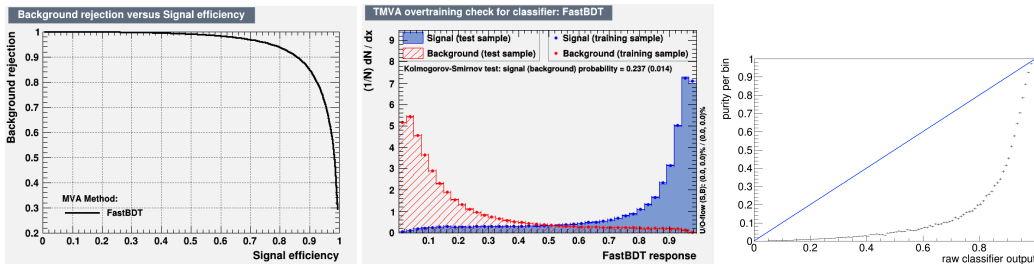
| Name | Description | Rank | Importance |
|---|---|------|------------|
| daughterInvariant-Mass(0,1,2) | Returns invariant mass of the given daughter particles. E.g. daughterInvariantMass(0, 1) returns the invariant mass of the first and second daughter. daughterInvariantMass(0, 1, 2) returns the invariant mass of the first, second and third daughter. Useful to identify intermediate resonances in a decay, which weren't reconstructed explicitly. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (\geq amount of daughters). | 1 | 2.43 |
| daughterInvariant-Mass(0,1) | Returns invariant mass of the given daughter particles. E.g. daughterInvariantMass(0, 1) returns the invariant mass of the first and second daughter. daughterInvariantMass(0, 1, 2) returns the invariant mass of the first, second and third daughter. Useful to identify intermediate resonances in a decay, which weren't reconstructed explicitly. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (\geq amount of daughters). | 2 | 1.40 |
| chiProb | χ^2 probability of the fit | 3 | 1.18 |
| daughter(0,extra-Info(SignalProbability)) | Returns value of variable for the i-th daughter. E.g. daughter(0, p) returns the total momentum of the first daughter. daughter(0, daughter(1, p) returns the total momentum of the second daughter of the first daughter. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (\geq amount of daughters). | 4 | 0.98 |
| daughter(1, chiProb) | Returns value of variable for the i-th daughter. E.g. daughter(0, p) returns the total momentum of the first daughter. daughter(0, daughter(1, p) returns the total momentum of the second daughter of the first daughter. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (\geq amount of daughters). | 5 | 0.97 |
| decayAngle(2) | cosine of the angle between the mother momentum vector and the direction of the i-th daughter in the mother's rest frame | 6 | 0.90 |

| | | | |
|--|--|----|------|
| useRestFrame(daughter(0, p)) | Returns the value of the variable using the rest frame of the given particle as current reference frame. E.g. useRestFrame(daughter(0, p)) returns the total momentum of the first daughter in its mother's rest-frame | 7 | 0.88 |
| daughter(2,extra- Info(SignalProba- bility)) | Returns value of variable for the i-th daughter.E.g. daughter(0, p) returns the total momentum of the first daughter. daughter(0, daughter(1, p) returns the total momentum of the second daughter of the first daughter. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (>= amount of daughters). | 8 | 0.85 |
| useRestFrame(daughter(0, dis- tance)) | Returns the value of the variable using the rest frame of the given particle as current reference frame. E.g. useRestFrame(daughter(0, p)) returns the total momentum of the first daughter in its mother's rest-frame | 9 | 0.77 |
| decayAngle(0) | cosine of the angle between the mother momentum vector and the direction of the i-th daughter in the mother's rest frame | 10 | 0.73 |
| daughter(2, chiProb) | Returns value of variable for the i-th daughter.E.g. daughter(0, p) returns the total momentum of the first daughter. daughter(0, daughter(1, p) returns the total momentum of the second daughter of the first daughter. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (>= amount of daughters). | 11 | 0.40 |
| useRestFrame(daughter(1, dis- tance)) | Returns the value of the variable using the rest frame of the given particle as current reference frame. E.g. useRestFrame(daughter(0, p)) returns the total momentum of the first daughter in its mother's rest-frame | 12 | 0.28 |
| Q | released energy in decay | 13 | 0.25 |
| useRestFrame(daughter(1, p)) | Returns the value of the variable using the rest frame of the given particle as current reference frame. E.g. useRestFrame(daughter(0, p)) returns the total momentum of the first daughter in its mother's rest-frame | 14 | 0.21 |

A. Example FEI Report for a Small Training

| | | | |
|---|---|----|------|
| daughter(1,extra-Info(SignalProbability)) | Returns value of variable for the i-th daughter. E.g. daughter(0, p) returns the total momentum of the first daughter. daughter(0, daughter(1, p) returns the total momentum of the second daughter of the first daughter. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (\geq amount of daughters). | 15 | 0.19 |
| decayAngle(1) | cosine of the angle between the mother momentum vector and the direction of the i-th daughter in the mother's rest frame | 16 | 0.18 |
| useRestFrame(daughter(2, distance)) | Returns the value of the variable using the rest frame of the given particle as current reference frame. E.g. useRestFrame(daughter(0, p)) returns the total momentum of the first daughter in its mother's rest-frame | 17 | 0.15 |
| daughterInvariantMass(1,2) | Returns invariant mass of the given daughter particles. E.g. daughterInvariantMass(0, 1) returns the invariant mass of the first and second daughter. daughterInvariantMass(0, 1, 2) returns the invariant mass of the first, second and third daughter. Useful to identify intermediate resonances in a decay, which weren't reconstructed explicitly. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (\geq amount of daughters). | 18 | 0.04 |
| daughterProductOf(extraInfo(SignalProbability)) | Returns product of a variable over all daughters. E.g. daughterProductOf(extraInfo(SignalProbability)) returns the product of the SignalProbabilities of all daughters. | 19 | 0.02 |
| daughterInvariantMass(0,2) | Returns invariant mass of the given daughter particles. E.g. daughterInvariantMass(0, 1) returns the invariant mass of the first and second daughter. daughterInvariantMass(0, 1, 2) returns the invariant mass of the first, second and third daughter. Useful to identify intermediate resonances in a decay, which weren't reconstructed explicitly. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (\geq amount of daughters). | 20 | 0.01 |

| | | | |
|--|--|----|------|
| daughter(0, chiProb) | Returns value of variable for the i-th daughter. E.g. daughter(0, p) returns the total momentum of the first daughter. daughter(0, daughter(1, p) returns the total momentum of the second daughter of the first daughter. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (\geq amount of daughters). | 21 | 0.01 |
| cosAngleBetween- MomentumAnd- VertexVector | cosine of angle between momentum and vertex vector (vector connecting ip and fitted vertex) of this particle | 22 | 0.01 |
| useRestFrame(daughter(2, p)) | Returns the value of the variable using the rest frame of the given particle as current reference frame. E.g. useRestFrame(daughter(0, p)) returns the total momentum of the first daughter in its mother's rest-frame | 23 | 0.00 |



TMVA plots for Plugin/FastBDT using the configuration string `!H:!V:NTrees=100:Shrinkage=0.10:RandRatio=0.5:NCutLevel=8:NTreeLayers=3` with target variable `isSignal`. The training used 103774 signal and 1978795 background samples. 103774 signal and 1978795 background samples where used for testing.

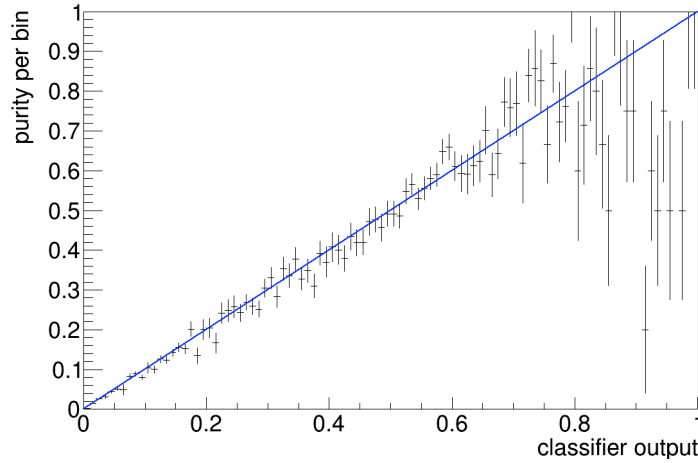
A. Example FEI Report for a Small Training

A.6. Particle: B^0

In the reconstruction of B^0 1 out of 1 possible channels were used.

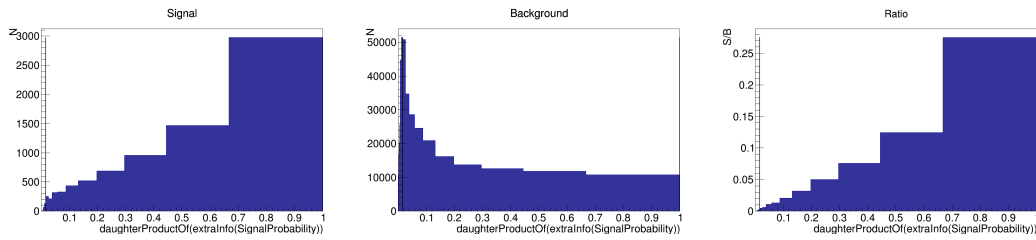
- B^0 :semileptonic $\rightarrow D^- e^+$

This amounts to 8391 signal events and 399171 background events in total before cuts. Channel-specific pre-cuts and a particle-specific post cut on the signal probability of the particle were applied. After all cuts 8166 signal events and 224862 background events remained.



A.6.1. Channel: B^0 :semileptonic $\rightarrow D^- e^+$

Pre-cut determination



Variable $daughterProductOf(extraInfo(SignalProbability))$ for combinations of daughter candidates of this channel. The PreCut range (0.02, 1.00) for this channel is marked with vertical lines in the plots above. The PreCuts were determined on variable $daughterProductOf(extraInfo(SignalProbability))$ with a desired total signal efficiency of 0.95000. For this channel only the efficiency is 0.97319 with a purity of 0.03504, corresponding to 8166.0 signal and 224862.0 background events.

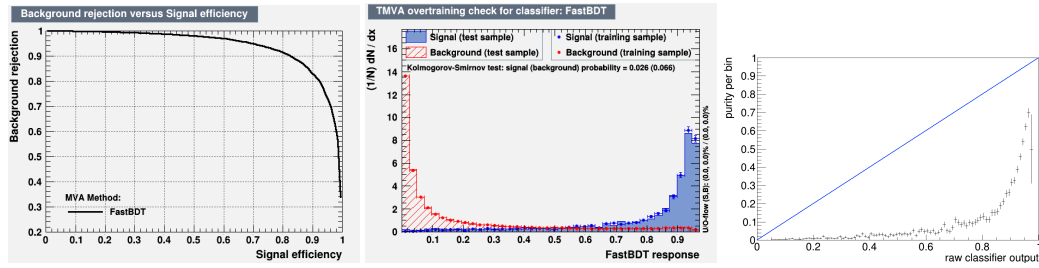
MVA

Table A.8.: List of variables used in the training.

| Name | Description | Rank | Importance |
|---|---|------|------------|
| dr | transverse distance in respect to IP | 1 | 0.70 |
| useRestFrame(daughter(0, p)) | Returns the value of the variable using the rest frame of the given particle as current reference frame. E.g. useRestFrame(daughter(0, p)) returns the total momentum of the first daughter in its mother's rest-frame | 2 | 0.57 |
| distance | 3D distance relative to interaction point | 3 | 0.30 |
| significanceOfDistance | significance of distance relative to interaction point (-1 in case of numerical problems) | 4 | 0.25 |
| decayAngle(1) | cosine of the angle between the mother momentum vector and the direction of the i-th daughter in the mother's rest frame | 5 | 0.25 |
| decayAngle(0) | cosine of the angle between the mother momentum vector and the direction of the i-th daughter in the mother's rest frame | 6 | 0.24 |
| daughter(0,extra-Info(SignalProbability)) | Returns value of variable for the i-th daughter.E.g. daughter(0, p) returns the total momentum of the first daughter. daughter(0, daughter(1, p) returns the total momentum of the second daughter of the first daughter. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (\geq amount of daughters). | 7 | 0.18 |
| daughter(1,extra-Info(SignalProbability)) | Returns value of variable for the i-th daughter.E.g. daughter(0, p) returns the total momentum of the first daughter. daughter(0, daughter(1, p) returns the total momentum of the second daughter of the first daughter. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (\geq amount of daughters). | 8 | 0.17 |
| chiProb | chi ² probability of the fit | 9 | 0.14 |
| daughter(1, chiProb) | Returns value of variable for the i-th daughter.E.g. daughter(0, p) returns the total momentum of the first daughter. daughter(0, daughter(1, p) returns the total momentum of the second daughter of the first daughter. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (\geq amount of daughters). | 10 | 0.11 |
| dz | z in respect to IP | 11 | 0.10 |

A. Example FEI Report for a Small Training

| | | | |
|---|---|----|------|
| cosAngleBetween-MomentumAnd-VertexVector | cosine of angle between momentum and vertex vector (vector connecting ip and fitted vertex) of this particle | 12 | 0.09 |
| useRestFrame(daughter(1, distance)) | Returns the value of the variable using the rest frame of the given particle as current reference frame. E.g. useRestFrame(daughter(0, p)) returns the total momentum of the first daughter in its mother's rest-frame | 13 | 0.09 |
| useRestFrame(daughter(1, p)) | Returns the value of the variable using the rest frame of the given particle as current reference frame. E.g. useRestFrame(daughter(0, p)) returns the total momentum of the first daughter in its mother's rest-frame | 14 | 0.06 |
| daughterProductOf(extraInfo(SignalProbability)) | Returns product of a variable over all daughters. E.g. daughterProductOf(extraInfo(SignalProbability)) returns the product of the SignalProbabilities of all daughters. | 15 | 0.06 |
| dx | x in respect to IP | 16 | 0.05 |
| deltaE | energy difference | 17 | 0.04 |
| dy | y in respect to IP | 18 | 0.02 |
| useRestFrame(daughter(0, distance)) | Returns the value of the variable using the rest frame of the given particle as current reference frame. E.g. useRestFrame(daughter(0, p)) returns the total momentum of the first daughter in its mother's rest-frame | 19 | 0.00 |
| daughter(0, chiProb) | Returns value of variable for the i-th daughter. E.g. daughter(0, p) returns the total momentum of the first daughter. daughter(0, daughter(1, p)) returns the total momentum of the second daughter of the first daughter. Returns -999 if particle is nullptr or if the given daughter-index is out of bound (\geq amount of daughters). | 20 | 0.00 |



TMVA plots for Plugin/FastBDT using the configuration string `!H:!V:NTrees=100:Shrinkage=0.10:RandRatio=0.5:NCutLevel=8:NTreeLayers=3` with target variable `isSignalAcceptMissing-Neutrino`. The training used 4083 signal and 112431 background samples. 4083 signal and 112431 background samples were used for testing.

Acknowledgements

Coffee: the finest organic suspension ever devised. It's got me through the worst of the last three years. I beat the Borg with it.

Kathryn Janeway in Star Trek: Voyager

Foremost I would like to thank Prof. Michael Feindt for kindling my interest in B meson physics and for the opportunity to work in his group. I also thank Prof. Günter Quast for agreeing to be my co-referee and for the great amount of assistance I received in the past weeks.

I thank Dr. Martin Heck and Prof. Thomas Kuhr for supervising my work and for their excellent advice. For many fruitful discussions I am grateful to Dr. Pablo Goldenzweig, Dr. Thomas Hauth, and Dr. Anže Zupanc.

As co-author of the Full Event Interpretation, Thomas Keck deserves my special thanks. Doing this without his enthusiasm and resourcefulness would have been a rather bleak endeavour.

Besides those mentioned above, I further thank Nils Braun and Markus Prim for their proofreading work. I also extend my thanks to everyone in the institute for the lovely atmosphere and many friendly discussions over coffee.

Bibliography

- [1] **Muon g-2**, G. W. Bennett *et al.*, “Final Report of the Muon E821 Anomalous Magnetic Moment Measurement at BNL,” *Phys. Rev.* **D73** (2006) 072003, [arXiv:hep-ex/0602035](https://arxiv.org/abs/hep-ex/0602035) [hep-ex].
- [2] **CMS**, S. Chatrchyan *et al.*, “Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC,” *Phys. Lett.* **B716** (2012) 30–61, [arXiv:1207.7235](https://arxiv.org/abs/1207.7235) [hep-ex].
- [3] **ATLAS**, G. Aad *et al.*, “Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC,” *Phys. Lett.* **B716** (2012) 1–29, [arXiv:1207.7214](https://arxiv.org/abs/1207.7214) [hep-ex].
- [4] V. C. Rubin and W. K. Ford, Jr., “Rotation of the Andromeda Nebula from a Spectroscopic Survey of Emission Regions,” *Astrophys. J.* **159** (1970) 379–403.
- [5] A. D. Sakharov, “Violation of CP Invariance, C Asymmetry, and Baryon Asymmetry of the Universe,” *Pisma Zh. Eksp. Teor. Fiz.* **5** (1967) 32–35. [Usp. Fiz. Nauk161,61(1991)].
- [6] A. Riotto and M. Trodden, “Recent progress in baryogenesis,” *Ann. Rev. Nucl. Part. Sci.* **49** (1999) 35–75, [arXiv:hep-ph/9901362](https://arxiv.org/abs/hep-ph/9901362) [hep-ph].
- [7] N. Cabibbo, “Unitary Symmetry and Leptonic Decays,” *Phys. Rev. Lett.* **10** (1963) 531–533.
- [8] M. Kobayashi and T. Maskawa, “CP Violation in the Renormalizable Theory of Weak Interaction,” *Prog. Theor. Phys.* **49** (1973) 652–657.
- [9] J. Christenson, J. Cronin, V. Fitch, and R. Turlay, “Evidence for the 2π decay of the K_2^0 meson,” *Phys. Rev. Lett.* **13** no. 4, (1964) 138.
- [10] I. I. Y. Bigi and A. I. Sanda, “On $B^0 - \bar{B}^0$ Mixing and Violations of CP Symmetry,” *Phys. Rev.* **D29** (1984) 1393.
- [11] **Belle**, **BaBar**, A. J. Bevan *et al.*, “The Physics of the B Factories,” *Eur. Phys. J.* **C74** no. 11, (2014) 3026.
- [12] **Belle**, K. Abe *et al.*, “An Improved measurement of mixing induced CP violation in the neutral B meson system,” *Phys. Rev.* **D66** (2002) 071102, [arXiv:hep-ex/0208025](https://arxiv.org/abs/hep-ex/0208025) [hep-ex].
- [13] The Royal Swedish Academy of Sciences, “Scientific background on the Nobel Prize in Physics 2008.” http://www.nobelprize.org/nobel_prizes/physics/laureates/2008/advanced-physicsprize2008.pdf. (accessed 28 September 2015).

Bibliography

- [14] **Belle**, J. Brodzicka *et al.*, “Physics Achievements from the Belle Experiment,” *PTEP* **2012** (2012) 04D001, [arXiv:1212.5342 \[hep-ex\]](#).
- [15] **Belle II**, T. Abe *et al.*, “Belle II Technical Design Report,” tech. rep., 2010. [arXiv:1011.0352 \[physics.ins-det\]](#).
- [16] J. Wiechczyński, “The Belle II experiment at the SuperKEKB collider.” (Presented at the European Physical Society conference on high energy physics), July, 2015.
- [17] C. D. Anderson, “The Positive Electron,” *Phys. Rev.* **43** (1933) 491–494.
- [18] G. D. Rochester and C. C. Butler, “Evidence for the Existence of New Unstable Elementary Particles,” *Nature* **160** (1947) 855–857.
- [19] **Gargamelle Neutrino**, F. J. Hasert *et al.*, “Observation of Neutrino Like Interactions Without Muon Or Electron in the Gargamelle Neutrino Experiment,” *Phys. Lett.* **B46** (1973) 138–140.
- [20] A. Abashian *et al.*, “The Belle Detector,” *Nucl. Instrum. Meth.* **A479** (2002) 117–232.
- [21] C. Pulvermacher, “dE/dx particle identification and pixel detector data reduction for the Belle II experiment,” Diploma thesis, KIT, 2012. <https://ekp-invenio.physik.uni-karlsruhe.de/record/48263>.
- [22] “Integrated luminosity of B factories.” http://belle.kek.jp/bdocs/lumi_belle.png. (accessed 30 September 2015).
- [23] T. Hara, T. Kuhr, and Y. Ushiroda, “Belle II Coordinate System and Guideline of Belle II Numbering Scheme.” Belle II internal note, 2011.
- [24] M. Yokoi, “BELLE検出器用CsI電磁カロリメータにおけるトリガーシステムの設計・製作 [Design and production of a trigger system for a CsI electromagnetic calorimeter for use at the Belle experiment],” Master’s thesis, Nara Women’s University, 1998. (Japanese).
- [25] D. R. Lide, *CRC Handbook of Chemistry and Physics, 88th Edition*. CRC Press, Taylor & Francis, Boca Raton, 2007–2008.
- [26] R. Frühwirth, R. Glattauer, J. Lettenbichler, W. Mitaroff, and M. Nadler, “Track finding in silicon trackers with a small number of layers,” *Nucl. Instrum. Meth.* **A732** (2013) 95–98.
- [27] V. Aulchenko *et al.*, “Electromagnetic calorimeter for Belle II,” *Journal of Physics: Conference Series* **587** no. 1, (2015) 012045.
- [28] L. Piilonen, “B-KLM Summary Talk.” 11th B2GM, Mar., 2012. <http://kds.kek.jp/contributionDisplay.py?sessionId=29&contribId=17&confId=8895>. Belle II internal.
- [29] T. Schlüter, “Vertexing and Tracking Software at Belle II,” *PoS Vertex2014* (2014) 039, [arXiv:1411.3485 \[physics.ins-det\]](#).

- [30] R. Itoh, “BASF - BELLE Analysis Framework,” *Comput.Phys.Commun.* **110** (1998) **CHEP 1997** (1997) pp.1–263. <https://www.ifh.de/CHEP97/paper/244.ps>.
- [31] I. Adachi, R. Itoh, N. Katayama, T. Tsukamoto, T. Hibino, M. Yokoyama, L. Hinz, and F. Ronga, “Computing system for the Belle experiment,” *eConfC0303241* (2003) MODT010, [arXiv:physics/0306120](https://arxiv.org/abs/physics/0306120) [physics].
- [32] S. Mineo, R. Itoh, N. Katayama, and S. Lee, “Distributed parallel processing analysis framework for Belle II and Hyper Suprime-Cam,” *PoS (ACAT2010)* **026** (2010) . https://inspirehep.net/record/924965/files/ACAT2010_026.pdf.
- [33] A. Moll, “The software framework of the Belle II experiment,” *Journal of Physics: Conference Series (CHEP 2010)* **331** (2011) 032024.
- [34] B. Stroustrup, “What is C++ 0x,” *CVu* **21** (2009) 21. <http://www.stroustrup.com/what-is-2009.pdf>. (accessed 2 August 2015).
- [35] “Boost C++ libraries.” <http://www.boost.org>.
- [36] R. Brun and F. Rademakers, “ROOT – an object oriented data analysis framework,” *Nucl. Instrum. Meth.* **A389** no. 1, (1997) 81–86.
- [37] S. Agostinelli *et al.*, “GEANT4: A Simulation toolkit,” *Nucl. Instrum. Meth.* **A506** (2003) 250–303.
- [38] “Google C++ testing framework.” <https://code.google.com/p/googletest/>.
- [39] “SCons – a software construction tool.” <http://www.scons.org/>.
- [40] “ROOT documentation for TTree.” <https://root.cern.ch/root/html/TTree.html>. (accessed 30 July 2015).
- [41] P. Canal, R. Brun, V. Fine, L. Janyst, J. Lauret, and P. Russo, “Support for significant evolutions of the user data model in ROOT files,” *Journal of Physics: Conference Series (CHEP 2009)* **219** no. 3, (2010) 032004.
- [42] “ROOT documentation for TTreeIndex.” <https://root.cern.ch/doc/master/classTTreeIndex.html>. (accessed 17 September 2015).
- [43] G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” *Proceedings of the April 18-20, 1967, spring joint computer conference (AFIPS)* (1967) 483–485.
- [44] G. E. Moore *et al.*, “Cramming more components onto integrated circuits,” *Electronics* **38** no. 8, (1965) 114–117. Reprinted in: *Proceedings of the IEEE*, **86**, no. 1, (1998) 82–85.
- [45] C. Moler, “Matrix computation on distributed memory multiprocessors,” *Hypercube Multiprocessors* (1986) 181–195. <https://books.google.com/books?id=QN8HNVwZEecC&oi=fnd&pg=PA181>.

Bibliography

- [46] S. Lee, R. Itoh, N. Katayama, H. Furusawa, H. Aihara, and S. Mineo, “A common real time framework for SuperKEKB and Hyper Suprime-Cam at Subaru telescope,” *Journal of Physics: Conference Series* **219** (2010) 022012.
- [47] R. Itoh, S. Lee, N. Katayama, S. Mineo, A. Moll, T. Kuhr, and M. Heck, “Implementation of parallel processing in the basf2 framework for Belle II,” *Journal of Physics: Conference Series (CHEP 2012)* **396** no. 2, (2012) 022026.
- [48] S. Casey, “How to determine the effectiveness of hyper-threading technology with an application,” 2011.
<https://software.intel.com/en-us/articles/how-to-determine-the-effectiveness-of-hyper-threading-technology-with-an-application/>. (accessed 30 July 2015).
- [49] “fork(2) Linux man page,” March, 2013.
- [50] M. Tadel, “Overview of EVE – the event visualization environment of ROOT,” *Journal of Physics: Conference Series (CHEP 2009)* **219** no. 4, (2010) 042055.
- [51] I. Hřivnáčová and B. Viren, “The virtual geometry model,” *Journal of Physics: Conference Series (CHEP 2007)* **119** no. 4, (2008) 042016.
- [52] T. Berners-Lee, R. Fielding, and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” 2005. <https://tools.ietf.org/html/rfc3986>. RFC 3986.
- [53] “DESY News – milestone for Belle II: Test beam measurements at the vertex detector successfully completed,” Feb., 2014.
<https://www.desy.de/news/@@news-view?id=7201&lang=eng>. (accessed 30 July 2015).
- [54] J. Tanaka, *Precise Measurements of Charm Meson Lifetimes and Search for $D^0 - \bar{D}^0$ Mixing*. PhD thesis, University of Tokyo, 2001.
<http://belle.kek.jp/belle/theses/doctor/2002/tanaka.pdf>.
- [55] W. Waltenberger, “RAVE: A detector-independent toolkit to reconstruct vertices,” *IEEE Trans. Nucl. Sci.* **58** (2011) 434–444.
- [56] D. Lange, “The EvtGen particle decay simulation package,” *Nucl. Instrum. Meth.* **A462** (2001) 152–155.
- [57] D. Zander, “Full Reconstruction and $Y(4140)$ Search at Belle,” Diploma thesis, KIT, 2009.
<https://ekp-invenio.physik.uni-karlsruhe.de/record/45463>.
- [58] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, and H. Voss, “TMVA: Toolkit for Multivariate Data Analysis,” *PoS ACAT* (2007) 040,
[arXiv:physics/0703039](https://arxiv.org/abs/physics/0703039).
- [59] T. Keck, “The Full Event Interpretation for Belle II,” Master’s thesis, KIT, 2014.
<https://ekp-invenio.physik.uni-karlsruhe.de/record/48602>.

- [60] J. H. Friedman, “Stochastic gradient boosting,” *Comput. Stat. Data Anal.* **38** no. 4, (Feb., 2002) 367–378.
- [61] M. Feindt and U. Kerzel, “The NeuroBayes neural network package,” *Nucl. Instrum. Meth.* **A559** no. 1, (2006) 190–194.
- [62] M. Pivk and F. R. Le Diberder, “SPlot: A Statistical tool to unfold data distributions,” *Nucl. Instrum. Meth.* **A555** (2005) 356–369, arXiv:physics/0402083 [physics.data-an].
- [63] B. Lipp, “sPlot-based Training of Multivariate Classifiers in the Belle II Analysis Software Framework,” Bachelor thesis, KIT, 2015.
<https://ekp-invenio.physik.uni-karlsruhe.de/record/48717>.
- [64] S. Neubauer, *Search for $B \rightarrow K^{(*)} \nu \bar{\nu}$ Decays Using a New Probabilistic Full Reconstruction Method*. PhD thesis, 2011.
<https://ekp-invenio.physik.uni-karlsruhe.de/record/48215>.
- [65] G. C. Fox and S. Wolfram, “Observables for the analysis of event shapes in e^+e^- annihilation and other processes,” *Phys. Rev. Lett.* **41** (Dec, 1978) 1581–1585.
- [66] M. Gelb, “Neutral B Meson Flavor Tagging for Belle II,” Master’s thesis, KIT, 2015.
<https://ekp-invenio.physik.uni-karlsruhe.de/record/48719>.
- [67] F. Keller, “Improvement of the Full Reconstruction of B Mesons at the Belle Experiment,” Diploma thesis, KIT, 2011.
<https://ekp-invenio.physik.uni-karlsruhe.de/record/45460>.
- [68] **BaBar**, B. Aubert *et al.*, “Measurement of the inclusive charmless semileptonic branching ratio of B mesons and determination of $|V_{ub}|$,” *Phys. Rev. Lett.* **92** (2004) 071802, arXiv:hep-ex/0307062 [hep-ex].
- [69] **BaBar**, B. Aubert *et al.*, “Observation of the semileptonic decays $B \rightarrow D^* \tau^- \bar{\nu}_\tau$ and evidence for $B \rightarrow D \tau^- \bar{\nu}_\tau$,” *Phys. Rev. Lett.* **100** (2008) 021801, arXiv:0709.1698 [hep-ex].
- [70] **Belle**, K. Hara *et al.*, “Evidence for $B^- \rightarrow \tau^- \bar{\nu}_\tau$ with a Hadronic Tagging Method Using the Full Data Sample of Belle,” *Phys. Rev. Lett.* **110** no. 13, (2013) 131801, arXiv:1208.4678 [hep-ex].
- [71] **Belle**, B. Kronenbitter *et al.*, “Measurement of the branching fraction of $B^+ \rightarrow \tau^+ \nu_\tau$ decays with the semileptonic tagging method,” *Phys. Rev.* **D92** no. 5, (2015) 051102, arXiv:1503.05613 [hep-ex].
- [72] **Belle**, O. Lutz *et al.*, “Search for $B \rightarrow h^{(*)} \nu \bar{\nu}$ with the full Belle $\Upsilon(4S)$ data sample,” *Phys. Rev.* **D87** no. 11, (2013) 111103, arXiv:1303.3719 [hep-ex].
- [73] **Belle**, A. Heller *et al.*, “Search for $B^+ \rightarrow l^+ \nu_l \gamma$ decays with hadronic tagging using the full Belle data sample,” *Phys. Rev.* **D91** no. 11, (2015) 112009, arXiv:1504.05831 [hep-ex].

Bibliography

- [74] **Belle**, M. Huschle *et al.*, “Measurement of the branching ratio of $\bar{B} \rightarrow D^{(*)} \tau^- \bar{\nu}_\tau$ relative to $\bar{B} \rightarrow D^{(*)} \ell^- \bar{\nu}_\ell$ decays with hadronic tagging at Belle,” [arXiv:1507.03233 \[hep-ex\]](#). Accepted for publication in Phys. Rev. D.
- [75] M. Feindt, F. Keller, M. Kreps, T. Kuhr, S. Neubauer, D. Zander, and A. Zupanc, “A hierarchical NeuroBayes-based algorithm for full reconstruction of B mesons at B factories,” *Nucl. Instrum. Meth.* **A654** no. 1, (2011) 432–440, [arXiv:1102.3876 \[hep-ex\]](#).
- [76] C. Pulvermacher, T. Keck, M. Feindt, M. Heck, and T. Kuhr, “An automated framework for hierarchical reconstruction of B mesons at the Belle II experiment,” *Journal of Physics: Conference Series (ACAT 2014)* **608** no. 1, (2015) 012048.
- [77] D. Zander, *Full Reconstruction And Search For Charged Higgs Effects In Semi-Tauonic B Decays*. PhD thesis, KIT, 2013. <https://exp-invenio.physik.uni-karlsruhe.de/record/48271>.
- [78] **Belle**, S. H. Lee *et al.*, “Evidence for $B^0 \rightarrow \pi^0 \pi^0$,” *Phys. Rev. Lett.* **91** (Dec, 2003) 261801, [arXiv:0308040 \[hep-ex\]](#).
- [79] **Belle**, A. Sibidanov *et al.*, “Study of Exclusive $B \rightarrow X_u \ell \nu$ Decays and Extraction of $|V_{ub}|$ using Full Reconstruction Tagging at the Belle Experiment,” *Phys. Rev.* **D88** no. 3, (2013) 032005, [arXiv:1306.2781 \[hep-ex\]](#).
- [80] B. Kronenbitter, *Measurement of the branching fraction of $B^+ \rightarrow \tau^+ \nu_\tau$ decays at the Belle experiment*. PhD thesis, KIT, 2014. <https://exp-invenio.physik.uni-karlsruhe.de/record/48604>.
- [81] K. Kirchgessner, “Semileptonische Markierungsseiten-Rekonstruktion,” Diploma thesis, KIT, 2012. <https://exp-invenio.physik.uni-karlsruhe.de/record/48181>.
- [82] S. Wehle, “Full Reconstruction of the $\Upsilon(5S)$ Resonance and Analysis of the Tetraquark State Z_B at Belle,” Diploma thesis, KIT, 2013. <https://exp-invenio.physik.uni-karlsruhe.de/record/48572>.
- [83] “Ohcount 3.0.0.” <https://github.com/blackducksw/ohcount>. (accessed 30 July 2015).
- [84] D. Eastlake and P. Jones, “US Secure Hash Algorithm 1 (SHA1),” 2001. <https://tools.ietf.org/html/rfc3174>. RFC 3174.
- [85] **Particle Data Group**, K. A. Olive *et al.*, “Review of Particle Physics,” *Chin. Phys.* **C38** (2014) 090001.
- [86] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *Communications of the ACM* **51** no. 1, (2008) 107–113.
- [87] “IBM Platform LSF.” <https://www-03.ibm.com/systems/platformcomputing/products/lsf/>. (accessed 20 September 2015).