Karlsruhe Institute of Technology

Title

Title

Master thesis
submitted by

Nils Braun

DATUM

Institut of Experimental Nuclear Physics (IEKP)

Advisor:	Prof. Dr. Michael Feindt
Coadvisor:	Prof. Dr. Ulrich Husemann

Editing time: November 2014 – November 2015

Karlsruhe Institute of Technology

Title

Title

Masterarbeit
eingereicht von

Nils Braun

DATUM

Institut für experimentelle Kernphysik (IEKP)

Referent:	Prof. Dr. Michael Feindt
Korreferent:	Prof. Dr. Ulrich Husemann

Bearbeitungszeit: November 2014 – November 2015

Contents

1. Track Finder Theory and Multivariate Classification

Before explaining the implemented changes to the track finder for the Belle II experiment in more detail, the principles of the Belle II software framework are explained briefly. More information can be found elsewere [1]. Afterwards common figures of merit for all track finders are explained and discussed and the working principles of the already implemented track finders are illustrated.

1.1 The Belle II Analysis Framework (`basf2`)

For simulation, data acquisition, data processing and analysis of the Belle II experiment the Belle Analysis Software Framework 2 (`basf2`) is used. Although - guided by its name - it seems to be build on top of the old software framework used for the Belle experiment it is a complete rewrite of the software using modern programming principles in the coding languages C++ [2] and Python 2.7 [3]. Together with external programming libraries like ROOT [4] or EvtGen [5] that are already on the market this framework builds the base for every software written for the experiment.

The software is divided into several packages - each serving a single purpose or summarizing code for a single detector. Examples for the packages are CDC, SVD or the tracking packaged which is described in more detail in later chapters.

Each usage of the Belle Analysis Software Framework - if it is either a simulation, a reconstruction or an analysis does not matter - consists of processing one or more so called *paths* build with *modules*. These modules perform a dedicated small task like simulating the hard scattering event (the so called `EvtGen` module), writing out data to a root file (the module is called `RootOutput`) or performing a track reconstruction (for example with the module `TrackFinderCDCAutomaton`). The presence, the order and the parameters of the modules are determined in *steering* files written with python. The modules itself can be written in C++ or python.

In these steering files a path is created, filled and passed to the framework which handles loading the corresponding C++ libraries and calling the modules for every event that should be processed. An example of a small steering file for track finding can be found in listing 1.1. Caused by this extremely modular structure not only parallel processing but also debugging of intermediate steps can be performed much easier.

Because many modules need the data produces by other modules before there is a need for intermodular communication. This communication is performed within the framework with the help of the data store. This class as a wrapper around a collection of named `TClonesArrays` from the ROOT library [6] which can store lists of instances of nearly arbitrary C++ classes. It is used widely in the framework to store all sorts of things like

missing

Figure 1.1: The visualization of the intermodular communication while processing the path implemented with the steering file described in listing 1.1.

More text

the hit information produced by the particles in the simulation or the found tracks after the track finding modules. The modules have read and write access to every so called store array in the data store. A visualization of the data flow between the modules created with the steering file in 1.1 can be found in figure 1.1. The data store can be written to or read from disk using ROOTs own serialization mechanism together with data member dictionaries for the C++ classes created by the C++ interpreter of ROOT called CINT [7].

missing

Listing 1.1: Python steering file to create a typical basf2 path. After loading the needed python libraries the path is created and filled with the modules. In the end this path is processed and for each event the modules are executed in the given order and with their given parameters. For more information on the used modules see their documentations.

1.2 Working Principle of the implemented CDC Track Finder in `basf2`

One part of this work was the improvement and further development of the track finder modules for the CDC tracking detector. Therefore the working principles of the two track finder for this detector are described here briefly. For more information on the first track finder - the legendre track finder - see [8]. More information on the second described track finder - the automaton track finder - can be found in [9].

The general purpose of a track finder algorithm is to partition all measured wire hits into exclusive sets of hits that may come from the same charged particle passing through the detector. It does so by using several assumptions on the charged particle producing the wire hits like the form of their trajectory and therefore the possible patterns of the hits. After fitting a mathematical model of a trajectory to these hits one can gain information on the momentum or the vertex position of this particle. There are several different approaches to find the correct sets of hits which are in the following called tracks.

The reason to have two track finders for the CDC is their different ansatz. The legendre track finder is a so called global track finder whereas the automaton track finder is a local one. A global algorithm uses the information of all wire hits simultaneously. The legendre track finder does this by applying a mathematical projection to the wire positions which should in principle project all hits belonging to the same track onto the same coordinates. A local algorithm however tries to use neighboring wire hits to construct clusters of hits. These clusters are then enlarged by using neighborhood relations until a full track can be found. In the following these two principles are described in more detail.

1.2.1 The Legendre Track Finder

The principle of using the legendre transformation for tracking algorithms in high energy particle experiments was first described by Alexopoulos [10]. It uses an extended version of the hough transformation introduced by Paul V.C. Hough in 1962 [11]. The algorithm uses

the fact that each trajectory in the r-ϕ-plane of the detector can be described by a circle - assuming no energy loss - because of the applied magnetic field. In a first approximation one can also assume that each particles comes from the interaction point which is valid for the bigger part of the decay products. Therefore the trajectory in the r-ϕ direction can be described by two parameters: the radius R of the circle and the angle θ between an arbitrary but fixed axis and the tangent to the circle at the interaction point.[1]

Simplified the idea is to calculate each trajectory that could have possibly created one of the axial hits and draw them all in a 2d histogram with the trajectory parameters R and θ as the coordinate axes. As there are only a small number of correct trajectory which are however responsible for a great number of hits there are is a small parameter set which appears very often in the histogram. These parameters can then be used to create tracks.

For applying this algorithm the x and y coordinate pair of every axial wire hit together with the drift length d is transformed by the function

$$x' = \frac{2x}{x^2 + y^2 - d^2} \qquad y' = \frac{2y}{x^2 + y^2 - d^2} \qquad d' = \frac{2R}{x^2 + y^2 - d^2}$$

$$R = x' \cos(\theta) + y' \sin(\theta) \pm d'$$

into the legendre space as it can be seen in figure 1.2. In the first step of the transformation, the wire hits are transformed to the inverted plane. With this transformation each circular trajectory through the interaction point is mapped onto a line. The two trajectory parameters R and θ are now functions of the slope and the axis interception of this line.

After that each drift circle is transformed into a pair of sinusoidal functions - also called sinograms. This function is constructed in this way to use the fact that each trajectory of a charged particle responsible for a wire hit must touch the drift circle tangentially. Each point on the constructed sinusoidal functions correspond to one possible trajectory of a particle which could have created such a hit. There are two sinusoidal functions because the osculation point can be on the far or the near side of the drift circle - the trajectory circle can circumscribe the drift circle or not.

With using the information of a single hit one ends up with an infinite number of trajectory hypothesis. But as a charged particle passed many drift cells until it leaves the CDC detector - in some cases up to 100 hits - several wire hits are created with the same trajectory parameters. As these same parameters correspond to the same point in the legendre space, the sinusoidal functions of the wire hits intersect in this point as it can also be seen in figure 1.2. The task of the legendre algorithm is now to do the transformation of the hit coordinates and find those intersections.

Imperfections due to energy loss and material effects make the sinusoidal function not interact in one single point but rather in smeared area. To copy with this problem but still find the intersections with a good performance a peak search in the legendre space is applied. The legendre space is divided into small bins. For each bin the number of sinusoidal functions passing this area is counted. The bin with the highest weight is assumed to be the bin with the highest number of sinusoidal intersections. From the wire hits contributing to this bin a new track is created and the search is repeated until a threshold in the bin entry is undercut. As the legendre space is mostly empty this procedure can be further improved in performance by refining the bin devision from very coarse bin sizes to finer ones only for those bins which have a certain amount of sinusoidal functions in them. Because these

[1] When dropping the last assumption of tracks coming from the origin one has to introduce another parameter - often called d_0 - which describes the minimal distance from the circle to the origin in the r-ϕ plane. It is easy to generalize the described algorithm to three dimensions. It has to be analyzed further if this third dimension can lead to an efficiency gain.

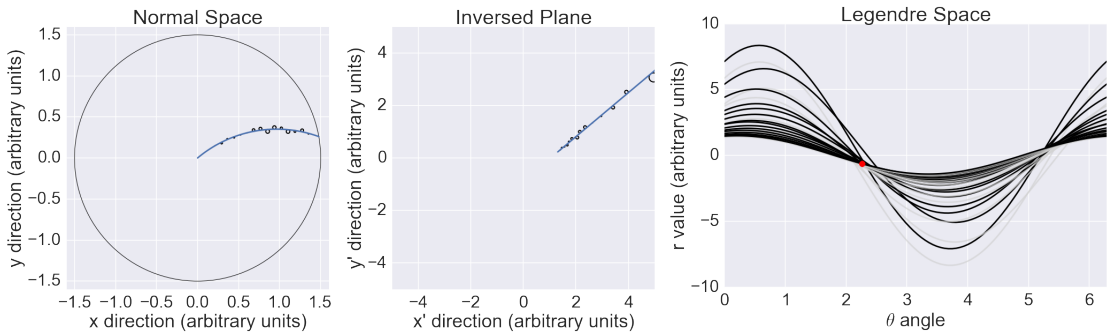


Figure 1.2: Transformation of some wire hits belonging to the same charge particle (left side) to the inverted plane (middle) and to the legendre space with the sinusoidal functions (right). As described in the text, the circular trajectory is first transformed into lines and then into intersecting sinograms. Each sinogram includes all possible trajectory parameters which would have touched the hit from which the function was created. The intersection corresponds to the parameters of the trajectory and are marked with a red circle. For better visibility only the wrong half of the sinograms is colored gray.

bins are divided into 4 subbins the concept is also called a quad tree search. The whole search is depicted in figure 1.3.

After finding the possible hit subsets for the track candidates a post processing procedure consisting of hit reassignment, hit deletion and track merging is applied to account for energy losses, trajectories not coming from the interaction point and finding inefficiencies. This post processing is described in more detail further down.

1.2.1.1 Stereo Hit Finding

The before mentioned legendre track finding algorithm does only only work for axial hits as a precise position in x and y is needed for calculating the position in the legendre space. Therefore it can not be applied for hits coming from stereo wires as they have a certain range of possible x-y-coordinates. But as soon as a trajectory with the axial hits is found, each stereo hit can be reconstructed to match the trajectory in such a way that its drift circle touches the trajectory circle of the candidate. As the stereo wire can be best approximated by a single line in 3d space the z position is now also fixed. As it is not possible to gain information whether the drift circle is includes in the trajectory circle or not - the same reason for the two sinograms before - each stereo hit leads to two slightly different reconstructed z positions.

An ideal trajectory in the z-r plane resembles a straight line analogous to the circle in the r-ϕ plane and can be described by two parameters also. This time these parameters are the slope $\tan\lambda$[2] and the distance z_0 on the z-axis to the interaction point. The plane spanned by these two parameters is analogous to the legendre space in the axial case. As before each point corresponds to a certain trajectory. By using the two reconstructed z information from each stereo hit a function in this plane can be drawn which includes every possible trajectory that would have passed the stereo hit. For axial hits this function was given by the two sinograms. For stereo hits it is a straight line in the $\tan\lambda$-z_0 plane:

$$z_0 = z_{\text{rec}} - \tan\lambda \cdot r_{\text{rec}}$$

with the reconstructed z position z_{rec} and the reconstructed radius r_{rec}. The whole process can be seen in figure 1.4. Again, a quad tree search is applied to search for the point

[2] It is described by the angle λ for consistency with the so called helix parameters

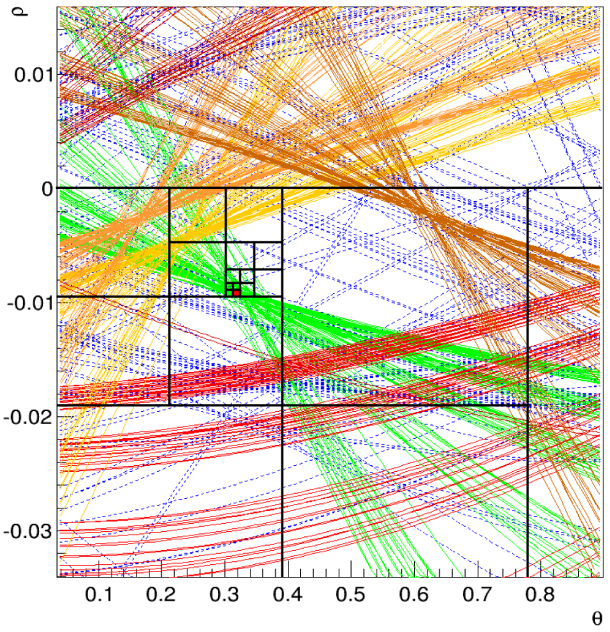


Figure 1.3: Depiction of a quad the search to find the bin with the most interceptions of sinograms shown in red here. Only the first round of the search is shown for better visibility. Each color shows sinograms that belong to the same charged particle. Taken from [12].

with the highest number of interceptions which corresponds to the trajectory parameters compatible with the largest number of hits. This time only the single highest trajectory is stored as one single trajectory in the r-ϕ plane can only have one single trajectory in the r-z plane. The whole algorithm is repeated for all other found axial-only trajectories.

As the energy loss has more or less no influence on the z motion because of the much higher momentum in this direction and a non-zero distance to the interaction point is already accounted for, a post processing is not needed here. See the chapter on the SegmentTrackCombiner for a description how to cope with the remaining inefficiencies.

1.2.2 The Automaton Track Finder

Clusterizer, Automaton-Principle

1.3 The used Figures Of Merit

For testing and developing and also for later usage in the experiment setup we need to compile numbers from the implemented track finder algorithms to show how well they work. There are three different classes of figures of merit to describe a track finder. All three classes listed here are described in more detail below. The three classes are:

- the efficiency (like the hit efficiency or the finding efficiency, also split up for different particle types, momentum regions or areas in the detector)
- the error rate (like the clone or fake rate)
- the computational performance (like timing and memory consumption)

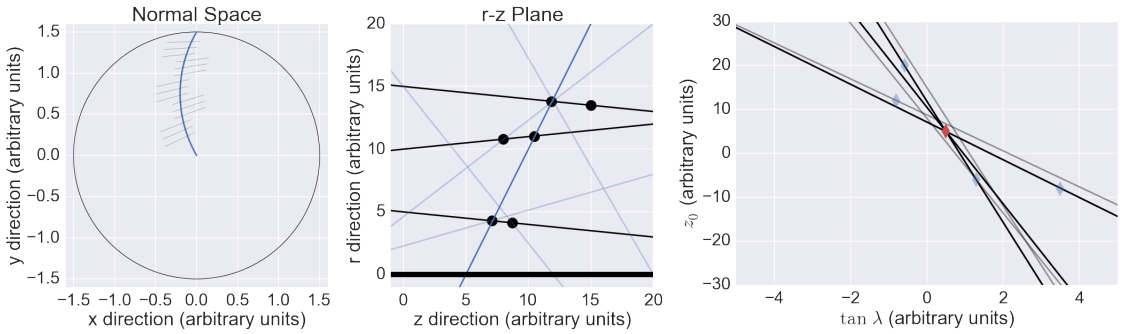


Figure 1.4: Stereo hit finder algorithm shown for a single track. The track trajectory in the r-ϕ plane leads to only two possible z positions for each stereo hit. On the left side the trajectory together with the stereo wires projected to the r-ϕ plane is show. In principle each stereo hit could have been created by an infinite number of tracks. Some of them are shown in color in the center picture. The dark blue track is the correct hypothesis. Each line correspond to a single point in the $\tan\lambda$-z_0 plane which is shown on the right side together with the correct trajectory parameters as a red diamond. The blue diamonds on the right side correspond to the blue lines in the center.

The last class should be quite clear and is measured by the basf2-own measuring algorithms. As the tracking is part of the online reconstruction and may be once adopted for the high level trigger, the timing performance is very important.

The other two classes can best be described with the algorithm how they get computed. It starts with a full Monte Carlo simulation of generic BB events with the full detector simulation afterwards. The created hits can then be parted into distinct sets - each describing a simulated tracks, called MCTrackCand (they are called track candidates to fit the naming convention of the track finder). Only those tracks are kept as a MCTrackCand, that have at least 3 hits in the CDC - otherwise they can not be fitted and even if there were a chance to find them via track finding, they could not be used for physics. After that, the simulated hits with the stripped MC information are used for track finding with the method to be evaluated. This can be done easily as the simulated hits are transformed in a format that is similar to the format that will be used later for the data coming from the experiment. After the track finder has produced a list of track candidates also, we can use the saved MC information to match tracks from the track finder to tracks from the mc algorithm (also called MC track finder) by counting the number of hits they share. The different cases are depicted and described in table 1.1.

The finding efficiency now describes the rate of MCTrackCands which are labeled matched by their total amount. Building this ratio can also be done for bins in various variables, like perpendicular momentum (p_T), angle in the curling plane (phi), number of tracks per event (multiplicity) and many more. A perfect track finder would have a finding efficiency of 100 %. In most of the cases, the finding efficiency drops for tracks in a certain region of these variables - like for low momentum tracks. The hit efficiency is the mean of the ratios between the number of hits in the MCTrackCands matched to a non-fake track candidate to the number of hits in total to this track. Here also a perfect track finder would have a hit efficiency of 100 %. As in most of the cases the track finder looses some outlaying hits the hit efficiency drops. The fake and clone rates are just the number of track candidates labeled as fake or clone by the matching algorithm divided by the number of found tracks in total. A perfect track finder would have both number set to 0 %. A high fake rate is caused by a track finder with too loose cuts when putting together single pieces of tracks to a big track or by one which picks up background hits often. A track finder with a high

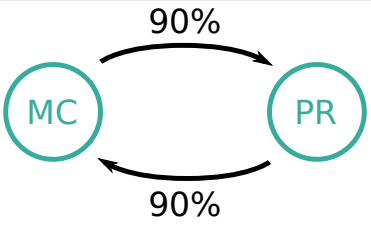
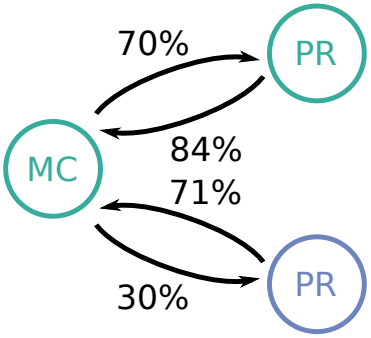
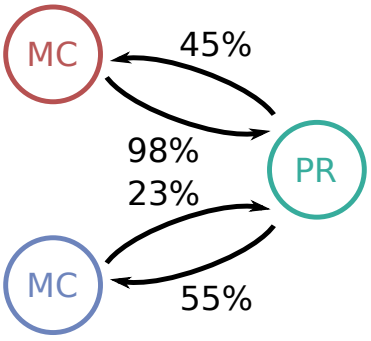
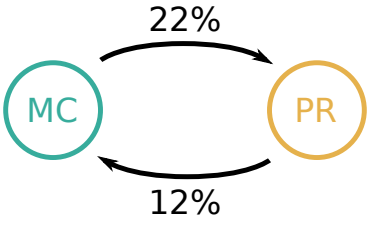
	There is a one to one connection between a MCTrackCand and a track from the track finder. The MCTrackCand is labeled found and the other track is labeled matched.
	The MCTrackCand is found twice. The track from the track finder with the higher percentage (the green one in this example) is labeled matched, the other one cloned. The MCTrackCand is nevertheless labeled found.
	The track from the track finder is created with hits from many different MCTrackCands. As none of the corresponding hit ratios exceeds 66 %, the track is called fake. There is no precise reason why the number 66% was chosen. The hit ratios of the MCTrackCands itself do not play any role here. TODO: Is found for MC possible?
	The found track does not describe any of the MCTrackCands well (or well enough) - but is made out of background hits. This track is also called a fake.

Table 1.1: This tabular shows the four different cases for the matching between tracks found by the track finder (on the left side of the pictures) and MCTrackCands (shown on the right side). The different colors differentiate between different tracks. The connection between tracks shows that these two tracks share hits. The two percentages on the arrows are the percentages of hits they share in respect to the total number of hits in the MCTrackCand/track candidate from the track finder.

missing, more text

Figure 1.5: Sketch with one step in the kalman fitter procedure.

clone rate on the other hand has to harsh cuts and splits up tracks into more than one piece.

1.4 Track Fitting

The track finding algorithms have the task to partition the measured hits into sets with each set forming a single track candidate. After that the purpose of a track fitting algorithm is to fit a model for the trajectory to the measured information of the hits to gain the particles properties like the momentum or the vertex position. The model used for the fit can be very easy without taken into account material effects or energy loss. The track fitting algorithms implemented in `basf2` however try take care of the interaction of the particles with material correctly by using the same algorithms in fitting that were used for simulating the detector geometry. As the simulation itself depends strongly on the track parameters this implies resimulating the particle with every fitting step. This procedure of stepping through the measurements iteratively fits perfectly well to the Kalman fitter algorithm used as the main procedure.

The Kalman fitter algorithm [13] is based on the idea of iteratively adding measurements (in this case the positions of the hits associated with this track) to the current state of the trajectory. Therefore the parameters change with every newly added hit and should in principle converge to the correct trajectory parameters. This is done by extrapolating a model of the trajectory with the current parameters to the position of the next hit measurement and comparing this extrapolated position with the real hit position. The deviation together with the errors of the measurement can be used to calculate new parameters for the trajectory. By transversing back and forth several times through the whole hit set the final parameter estimation is found. One step in this procedure with only a small hits set is depicted in figure 1.5.

When taking into account fake or background hits in a track, the kalman fitter algorithm may not lead to good results anymore. As each hit is used for estimating the parameters of the trajectory a wrongly attached hit can give a wrong bias on these parameters. Therefore a weighting scheme is applied to all hits after each kalman passage and this weight defines how strong a hit effects the final parameters. There are many different ways to do so including elastic tracking and nonlinear filters. The one currently used for the tracking in the Belle 2 software framework is the *deterministic annealing filter* (DAF) introduced by Frühwirth and Strandlie in [14]. The procedure is as follows:

(1) Set the weights of all hits to 1.

(2) Fit the track with a Kalman fitter taking into account the weights of each hit.

(3) Recalculate the weight of each hit with the distance of the hit to the current trajectory hypothesis.

(4) Dismiss hits with a weight below a certain threshold.

(5) Repeat with (2) until a defined number of iterations is reached or the fit is converged.

To overcome the difficulties of badly chosen starting parameters of the fit that would lead to dismissing a large number of hits in the first iteration an annealing schema is applied: each weight is transformed with a Maxwell-Boltzmann distribution depending on a "temperature" 'T which is decreased with every iteration. In the first few iterations with

a big temperature also hits with a small weight are kept while on later iterations these hits are dismissed.

The fitting procedures are implemented into `basf2` with the external library `genfit` [15] - a library for generic and experiment independent track fitting. Only a small interface for accessing the `genfit` procedures from the `basf2` code is implemented. See the chapter on the VXD momentum estimation for more information on this topic.

1.5 Multivariate Classification

When doing track finding - especially in post processing procedures - many decisions must me made: whether a hit belongs to a track, whether two tracks should be merged, whether a track should be dismissed as fake etc. In the context of inefficiencies and fakes due to background hits these decisions may depend on many input variables. Additionally there is the need to have a smooth transition between the two corner cases dismiss all and accept all to allow for subsequent optimization. This problem of deciding between two different possibilities is called a classification problem in statistics and there are many ways to deal with such problems (see for example [16] or [17]). For most of the classification tasks in the track finding package for the CDC detector a classification algorithm known as *boosted decision trees* (BDT) is used. In the following the main features are described. See [18] for more information on the implementation.

The task of a classification algorithm is to decide whether an input element described by a feature vector $\vec{x}$ belongs to one class of elements or the other - often these classes are the signal and the background class. The classifier maps the multidimensional feature vector $\vec{x}$ to a one dimensional output variable - often between 0 and 1 - which is designed in a way to separate signal and background by mapping signal like data to values near 1 and background like signals to values near 0. The output can therefore interpreted as the probability for the input data to look like a signal sample and a cut on the output can be used to distinguish between the two classes with a configurable purity.

A decision tree is one possible implementation of such a classifier. A decision tree is created by dividing the parameter space of the feature vectors $\vec{x}$ into small hypercubes. As this is done by applying a division in one variable at the time a tree structure like in figure 1.6 is formed. Each input data sample can then by mapped to the signal-fraction of the hypercube it belongs to. This signal-fraction is determined with a training sample of monte carlo generated events with a known classification information. The cut values and variables are chosen in a way to maximize a separate measure like the gini impurity that models the gain by this separation. It can also be calculated using the training sample.

Because the chosen cuts depend strongly on the training sample a single decision tree can easily be overtrained. A classifier is called overtrained when its separation power on the testing is much better than on an independent testing sample. To avoid overtraining an algorithm known as boosting is applied to the decision trees. This algorithm creates a whole set of boosted decision trees iteratively. Each decision tree is trained to separate the items the one before had classified wrongly. It does so by applying weights to the items. The final output of the classifier is a weighted sum of all single trees. By limiting the depth of a single decision tree a boosted decision avoids overtraining but because of the huge number of decision trees it still has a great separation power.

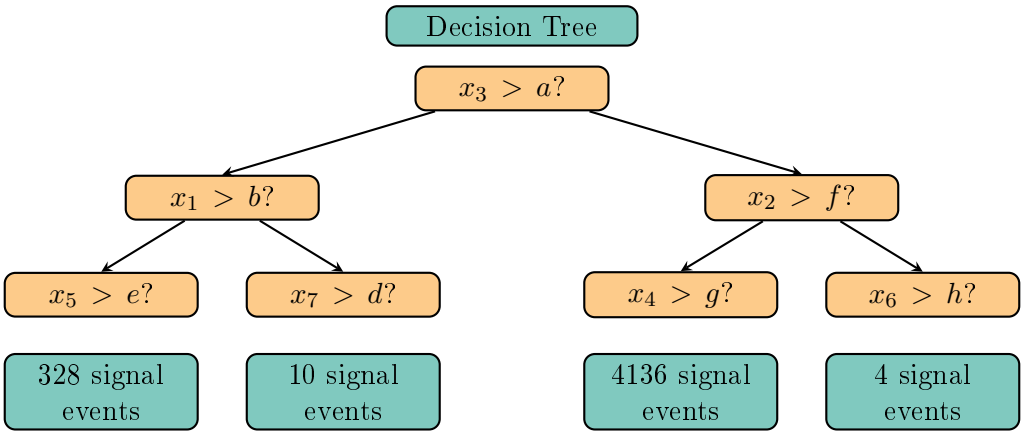


Figure 1.6: A fictitious decision tree dividing a 7-dimensional parameter space in 4 hypercubes with three layers of seven cuts. The last layer is used to generate the output of the decision tree by counting the number of signal events in this hypercube in the training sample.

Bibliography

[1] **Belle II Collaboration**, T. Abe *et al.*, "Belle II Technical Design Report," tech. rep., 2010. arXiv:1011.0352 [physics.ins-det].

[2] **International Organization for Standardization (ISO)**, "ISO International Standard ISO/IEC 14882:2014(E) – Programming Language C++," 2014. https://isocpp.org/std/the-standard.

[3] **Python Software Foundation**, "Python 2.7 pogramming language," 1991. https://www.python.org/.

[4] R. Brun and F. Rademakers, "ROOT – an object oriented data analysis framework," *Nucl. Instrum. Meth.* **A389** no. 1, (1997) 81–86.

[5] D. Lange, "The EvtGen particle decay simulation package," *Nucl. Instrum. Meth.* **A462** (2001) 152–155.

[6] "ROOT documentation for TClonesArray." https://root.cern.ch/root/html/TClonesArray.html.

[7] "ROOT documentation for CINT." https://root.cern.ch/drupal/content/cint.

[8] B. Kronenbitter, *Measurement of the branching fraction of $B^+ \to \tau^+\nu_\tau$ decays at the Belle experiment.* PhD thesis, KIT, 2014. https://ekp-invenio.physik.uni-karlsruhe.de/record/48604.

[9] O. Frost, "A Local Tracking Algorithm for the Central Drift Chamber of Belle II," Master's thesis, KIT, 2013. http://ekp-invenio.physik.uni-karlsruhe.de/record/48172.

[10] T. Alexopoulos, M. Bachtis, E. Gazis, and G. Tsipolitis, "Implementation of the Legendre Transform for track segment reconstruction in drift tube chambers," *Nucl. Instrum. Meth.* **A592** (2008) 456–462.

[11] H. C, "Method and means for recognizing complex patterns," Dec. 18, 1962. https://www.google.com/patents/US3069654. US Patent 3,069,654.

[12] V. Trusov, "Applying Legendre transformation method for Belle II tracking," *DPG spring meeting* (2014) .

[13] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME–Journal of Basic Engineering* **82** no. Series D, (1960) 35–45.

[14] R. Frühwirth and A. Strandlie, "Track fitting with ambiguities and noise: A study of elastic tracking and nonlinear filters," *Comput. Phys. Commun.* **120** no. 2-3, (1999) 197–214.

[15] C. Höppner, S. Neubert, B. Ketzer, and S. Paul, "A Novel Generic Framework for Track Fitting in Complex Detector Systems," *Nucl. Instrum. Meth.* **A620** (2010) 518–525, arXiv:0911.1008 [hep-ex].

[16] G. Cowan, *Statistical Data Analysis*. Oxford Science Publications. Clarendon Press, 1998.

[17] V. Blobel and E. Lohrmann, *Statistische und numerische Methoden der Datenanalyse*. Teubner, 1998.

[18] T. Keck, "The Full Event Interpretation for Belle II," Master's thesis, Karlsruher Institut für Technologie (KIT), 2014. https://ekp-invenio.physik.uni-karlsruhe.de/record/48602.

Todo list