6. Simulation: Event Generation and Propagation

Soon there will be virtual reality, and augmented reality. If you assume any rate of improvement at all, then games will become indistinguishable from reality . . ., it would seem to follow that the odds we are in base reality are one in billions. ∼ *Elon Musk*

To be able to search for new physics, one needs a good handle on the detector response to known physics processes. Depending on the analysis, some processes are more interesting than others. In general, the particle interactions of interest are referred to as *signal events*. Other interactions, which mimic or obscure the signal events, are typically called *background events*. These events are simulated using Monte Carlo* (MC) simulations, where one makes use of a model that describes the interactions and their probability to occur. A typical MC simulation consists of hundreds to millions of events that are constructed using these models with the use of random number generators. To determine the detector response to a particle interaction, one first has to start with the particle generation, which sets the conditions of the initial interaction. Afterwards, the propagation of the particle in the detector (medium) is simulated as best as possible. Below, an overview of the important background and signal simulations that are used in this analysis is given. A flowchart of the simulations steps is shown in Figure 6.1.

6.1 The software framework

IceTray is a modular framework written and used by the IceCube collaboration and mostly written in C++ for fast computation. A Python interface for most modules is provided for fast and easy implementation of the code. The framework is used in both online and offline processing and is stream-based with modules that act on events in the stream and essentially follows a flowchart of modules that is provided by the user.

To process the large amount of simulation that is required for the collaboration, a data processing and management framework called *IceProd* was developed. The setup is very light-weight, running as a Python application. It uses (complex) workflow DAGs (see below) across distributed

*While recovering from an illness in 1946, Stanislaw Ulam figured that the actual counting of successful attempts in playing a card game would yield him a much faster answer to the probability of success rather than doing the actual calculus. His work, shared with John von Neumann, needed to remain secret and adopted the code word “Monte Carlo”, referring to the gambling games in the Monte Carlo Casino in Monaco.

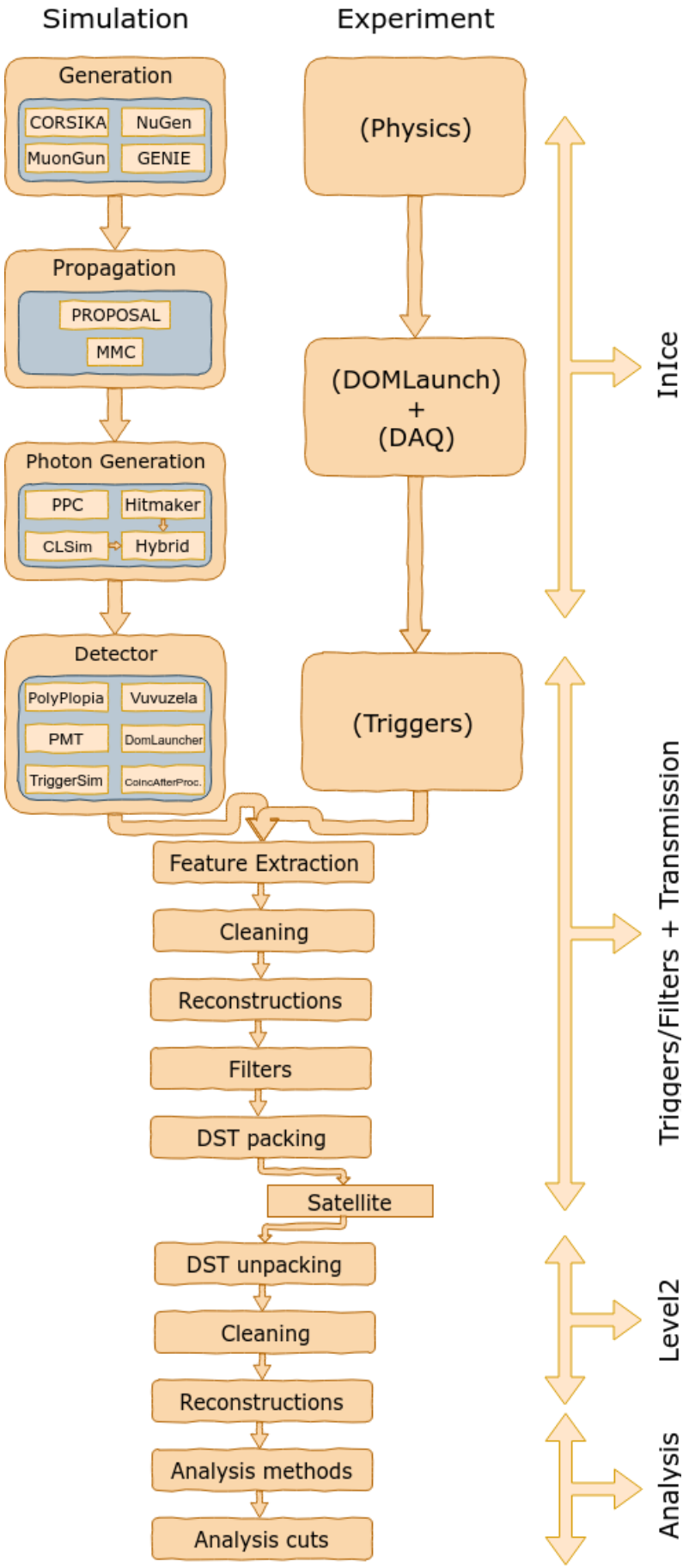

Figure 6.1: Flowchart of the simulation layout. On the left is shown how particles are injected and their interactions are simulated down to digitized waveforms. The right part shows real data processing. After triggering, both data and simulation go through the same processing chain to prepare for analysis.

computing grids in order to optimize usage of resources. A *dataset* is set up by running hundreds to thousands of jobs in parallel over multiple computing resources all over the world. Each dataset has specific input parameters that are fixed and every set is given a unique number. Distributions in physical parameters such as the direction, energy, position, etc. of the particle(s) are provided by random number generators [189].

HTCondor is an open source computing software that provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to HTCondor which places them into a queue. It chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

DAGMans (Directed Acyclic Graph Managers) are meta-schedulers for the execution of computations. A Directed Acyclic Graph (DAG) consists out of nodes in a graph. Each node represents a certain computation that needs a certain input or depends on the output or execution from another node. Once a DAG is set up, DAGMans submit the programs to HTCondor and processes the results. DAGMans are often used by analyzers for bulk computations on large amounts of data.

6.2 Generation

Simulations start with setting up the starting conditions of the physical processes one wants to simulate. For example, a shower event by itself is not well defined. The type of primary particle (H, He, Fe,...), the energy, the inclination and so on will all determine the properties of the full air shower that is produced. Or, do we want to simulate an SMP with charge 1/3 or 2/3?

Multiple different generators used in the IceCube collaboration serve different purposes; the ones that were used in this work are explained in more detail below.

6.2.1 Background simulation

In this work, we search for signal events that are not expected from the Standard Model. However, the IceCube neutrino observatory is not designed to look for particles with an anomalous charge. Therefore, it is necessary to properly account for the other physical processes that are not sought in this analysis but can mimic the signature of the signal. SMPs with a charge lower than the electron charge will produce less light compared to muons due to the squared charge dependency from the Cherenkov effect (see Eq. 4.7). Because the optical modules are located far away from each other, most of the light that is produced in tracks is absorbed in the ice. For this reason, there is a large uncertainty on the amount of Cherenkov photons that were produced when a track is seen in an event. Muons that originate from air shower events can produce dim tracks and are simulated with CORSIKA simulation. The NuGen program was used to simulate neutrino events. Both simulation programs are explained in more detail below. The datasets used in this analysis use SpiceLea as the ice model (see Section 5.6.1). An overview of the data samples and several simulation parameters is given in Table 6.2.

6.2.1.1 CORSIKA

A free, publicly available software framework that is widely used in the astrophysics community for the simulation of cosmic ray interactions is called CORSIKA (COsmic Ray SImulations for Kascade) [190]. It was originally developed for the KASCADE experiment and is now used by most people and collaborations to simulate air shower events. IceCube analyses, such as this one, use CORSIKA simulations to simulate the muonic component that is able to reach the in-ice detector.

The code is written in FORTRAN 77, but a C++ version is currently in the making [191].

The program initially injects a particle of specific type, energy, direction and position in the top of the atmosphere. The particles are tracked through the atmosphere until they undergo

Table 6.1: Best fit for parameters in Eq. 6.1. Numbers taken from Ref. [102].

j	R_c [V]	γ					$a_{i,j}$				
		p	He	CNO	Mg-Si	Fe	p	He	CNO	Mg-Si	Fe
1	$4 \cdot 10^{15}$	1.66	1.58	1.63	1.67	1.63	7860	3550	2200	1430	2120
2	$30 \cdot 10^{15}$	1.4					20		13.4		
3	$2 \cdot 10^{18}$	1.4					1.7		1.14		

reactions with the air nuclei or - in the case of unstable secondaries - decay. Multiple different hadronic interaction models exist to describe the interactions at high energies such as QGSJET [192], SIBYLL [193] and EPOS-LHC [194]. The production of new particles after decay or interaction creates showers of particles and these particles are saved and read out at a certain altitude. Because the flux of cosmic rays is exceedingly small at the highest energies, too many resources and too much time would be required to simulate an energy distribution as measured in experiments. Therefore, one often simulates a much harder spectrum and reweights the events accordingly later on (see Appendix B.4). CORSIKA datasets in IceCube are often subdivided into a low-energy and high-energy dataset. In this analysis, the former ranges from primary energies between 600 GeV to 100 TeV and uses a spectral index that is close to what is measured ($\gamma = 2.6$). The lower limit of the energy range at 600 GeV is due to the limited penetration depth of muons through the ice. The spectral index of the high-energy dataset is smaller, resulting in a harder spectrum ($\gamma = 2$) leading to more statistics for rare high-energy events. The primary energy ranges from 100 TeV to 100 EeV.

The spectrum used for this analysis, after reweighting, follows the following energy distribution:

$$\Phi_i\left(E_{\text{prim}}\right) = \sum_{j=1}^{3} a_{i,j} E^{-\gamma_{i,j}} \cdot \exp\left[-\frac{E}{Z_i R_{c,j}}\right]. \tag{6.1}$$

where we sum over three populations: particles accelerated from supernova remnants, a higher-energy galactic component of unknown origin and particles accelerated to ultra-high-energy from extra-galactic sources (more info in Section 3.1.2.1). γ is the spectral index, Z the particle atomic number and $a_{i,j}$ are the normalization constants for primary i in population j. R is the magnetic rigidity and R_c is the characteristic rigidity or cutoff above which a particular acceleration process reaches its limit. The 5 groups that are assumed to contribute significantly to the flux are: p, He, CNO, Mg-Si and Fe. This energy distribution follows the convention that is used in Ref. [102]. Table 6.1 summarizes the typical values for these parameters and shows the best fits for the normalization constants to describe the data.

Interactions

The atmosphere composition is always set at 78.1% N_2, 21% O_2, and 0.9% Ar, which is a good description of reality. However, the density of the air above the detector changes significantly during the year because of temperature differences in the Antarctic summer and winter. Most analyses, including this one, treat the muonic component as a background. They are not interested in the details of the showers and the changes during the year and therefore use an average of the atmospheric density. The atmospheric depth and densities were set to the averages of the month November for 5 years of data [195]. These monthly averages are shown in Figure 6.2.

The shower propagation and composition depends on the models that are used to simulate the high-energy interactions. The lowest energies are simulated with FLUKA (FLUktuierende KAskade) [196]. This model covers the energy range that can be compared with accelerator experiments. The SIBYLL model was used for the high-energy interactions. However, which model is the best for the highest energies is not known at the time of writing because there are no controlled laboratory measurements that are capable of reaching these energies. Several studies seem to indicate that the composition changes drastically at the highest energies, i.e.

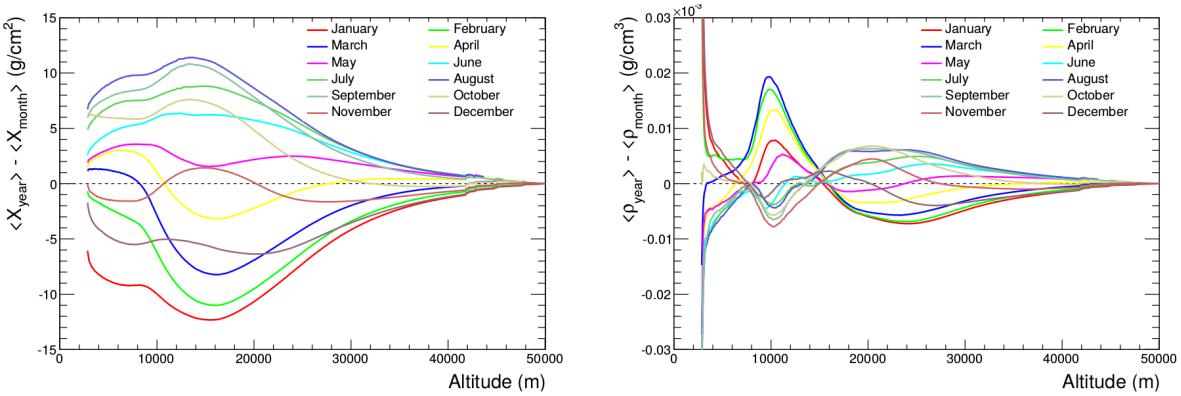

Figure 6.2: *Left:* Average atmospheric depth per month. *Right:* Average atmospheric density per month. Figures obtained with 5 years of data (2007-2011) [195].

IceCube [197], Pierre Auger Observatory [198], and Telescope Array [199]. However, the different experiments do not agree, presumably due to the different techniques used to determine the primary composition [195]. Recent work has shown that the most likely cause of discrepancy lies in non-perfect models [200]. Even though this discussion is far from resolved, it is fortunately of no importance for this analysis as these effects only become prominent at higher energies.

6.2.1.2 NuGen

The neutrino-generator (NuGen) is a neutrino event generator program that works with the IceTray framework. With this module, one can inject a primary neutrino on the surface of the Earth by specifying a few parameters in the steering file.

The physics implemented in this program is based on the ANIS-All Neutrino Interaction Generator [201]. However, the cross sections have been updated and the structure of the code has been changed significantly from ANIS to incorporate it in the IceTray framework.

The generator requires the first interaction to be near the detector and

- prepares a primary neutrino and injects it to the Earth,
- propagates the neutrino and works out interactions inside the Earth* (when they occur),
- makes a forced interaction inside the detection volume† (only if any neutrino reaches the detector site),
- stores injected neutrinos and all generated secondaries,
- stores interaction weight information.

The generator alternates between neutrino and antineutrino generation and assumes a neutrino-antineutrino ratio of (1:1), which is the expected ratio on Earth. The neutrino primary energy ranges from 100 GeV to 100 PeV.

The spectrum used for this analysis, after reweighting, follows the Honda2006 spectrum [135] for atmospheric neutrinos, SarcevicStd for the prompt component [138], and an astrophysical flux fit from Ref. [202] (see Section 3.3 for more information on these fluxes). The astrophysical flux measured by the IceCube collaboration follows an energy spectrum equal to

$$E^2\,(\Phi) = 1.5 \cdot 10^{-8} \left(\frac{E}{100\ \text{TeV}}\right)^{-0.3} \text{GeV cm}^{-2}\text{s}^{-1}\text{sr}^{-1}. \tag{6.2}$$

The distribution for these different components can be seen in Figure 6.3.

*Possible interactions are CC, NC, Glashow resonance for $\bar{\nu}_e$ and τ decay for $\overset{(-)}{\nu}_\tau$. CC interactions produce no new neutrinos and the simulation stops at the vertex point. The other interactions create new secondary neutrinos.

†In most cases, a neutrino will not interact within the medium, but for computational reasons at least one neutrino is forced to interact and the simulation is reweighted afterwards accordingly.

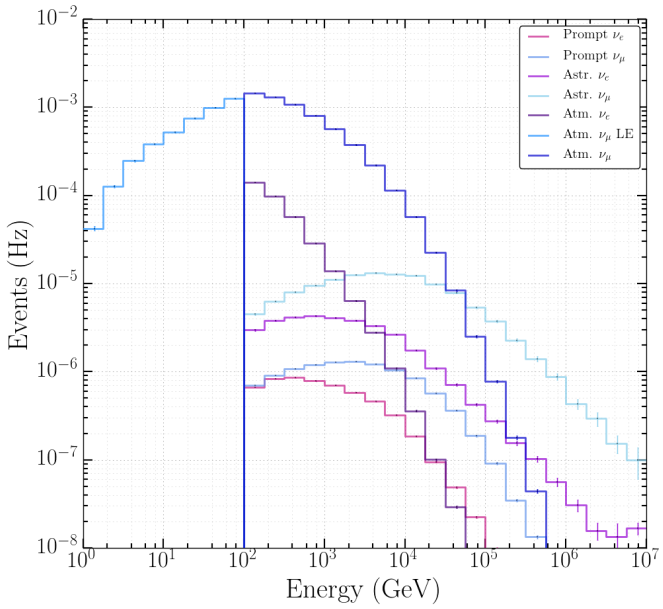

Figure 6.3: Distribution of weighted neutrino fluxes that were used for this analysis. The atmospheric ν_μ and ν_e fluxes were derived from Ref. [135], prompt from Ref. [138], and astrophysical from Ref. [202]. The energy refers to the energy of the primary neutrino.

6.2.1.3 GENIE

To include the lowest neutrino energies, which are not included in ANIS/NuGen, the GENIE (Generates Events for Neutrino Interaction Experiments) neutrino generator was implemented in IceTray. It is a well established generator, used by collaborations worldwide and written in C++ [203, 204].

The spectrum used for this analysis, after reweighting, follows the Honda2015 spectrum [205] for low-energy atmospheric neutrinos. The primary neutrino energy ranges from 0.5 to 100 GeV.

6.2.2 Signal simulation

As mentioned in Section 2.4, the signal flux is assumed to be isotropic close to the detector. The SMP starting points are randomly placed on a disk with a direction perpendicular to it as shown in Figure 6.4. The disk has a radius of 800 m and is located at a distance of 1000 m from the detector center. The disk itself is randomly rotated around the detector center to simulate an isotropic flux. The distribution of the azimuth, ϕ, and cosine of the zenith*, $\cos(\theta)$, is shown in Figure 6.5.

Because slow moving particles would require specialized treatment†, the minimal velocity of the particles is set as $\beta > 0.95$. The energy distribution is simulated with an E^{-1} spectrum and is later normalized to a flux of $10^{-14}\text{cm}^{-2}\ \text{s}^{-1}\ \text{sr}^{-1}$ with an E^{-2} spectrum (see Appendix B.2) where the absolute flux only plays a role for illustrative purposes (as we will see in Section 8.4.3).

Similar to the background, SpiceLea was used as the nominal ice model.

*See Appendix B.3 why we show the cosine of the zenith.

†Cherenkov photon production in Eq. 4.7 changes for $\beta < 1$ compared to relativistic particles ($\beta \approx 1$). Similarly, ionization effects as seen in Eq. 4.17 for slow moving particles would not be comparable to relativistic particles. Also, the IceCube detector is designed to trigger on relativistic particles and most of the cleaning tools (see Section 7.2) rely on relativistic speeds. Dedicated analyses on slow moving particles like monopoles are being done but need specialized simulation and analyses. This is beyond the scope of this work.

Table 6.2: Overview of the datasets used in this analysis. GaisserH3a from Ref. [102], Honda2015 from Ref. [205], Honda2006 from Ref. [135], Sarcevic from Ref. [138], and astrophysical from Ref. [202]. The dataset number is a unique number used in IceProd to distinguish data samples.

Generator	Type	Range [GeV]	Simulated γ	Weighted γ	Ice	Dataset
CORS.	5-comp.	$10^5 - 10^{11}$	2	GaisserH3a	SpiceLea	11937
CORS.	5-comp.	$600 - 10^5$	2.6	GaisserH3a	SpiceLea	11499
CORS.	5-comp.	$600 - 10^5$	2.6	GaisserH3a	SpiceLea	11808
CORS.	5-comp.	$600 - 10^5$	2.6	GaisserH3a	SpiceLea	11865
CORS.	5-comp.	$600 - 10^5$	2.6	GaisserH3a	SpiceLea	11905
CORS.	5-comp.	$600 - 10^5$	2.6	GaisserH3a	SpiceLea	11926
CORS.	5-comp.	$600 - 10^5$	2.6	GaisserH3a	SpiceLea	11943
CORS.	5-comp.	$600 - 10^5$	2.6	GaisserH3a	SpiceLea	12161
CORS.	5-comp.	$600 - 10^5$	2.6	GaisserH3a	SpiceLea	12268
GENIE	ν_μ	$0.5 - 100$	1	Honda2015	SpiceMie	12475
NuGen	ν_μ	$100 - 10^8$	2	atmos.: Honda2006 prompt: Sarcevic astro.: Astro.	SpiceLea	11029
NuGen	ν_μ	$100 - 10^8$	2	atmos.: Honda2006 prompt: Sarcevic astro.: Astro.	SpiceLea	12346
NuGen	ν_μ	$100 - 10^8$	2	atmos.: Honda2006 prompt: Sarcevic astro.: Astro.	SpiceLea	11883
NuGen	ν_e	$100 - 10^8$	2	atmos.: Honda2006 prompt: Sarcevic astro.: Astro.	SpiceLea	12034
NuGen	ν_e	$100 - 10^8$	2	atmos.: Honda2006 prompt: Sarcevic astro.: Astro.	SpiceLea	12646

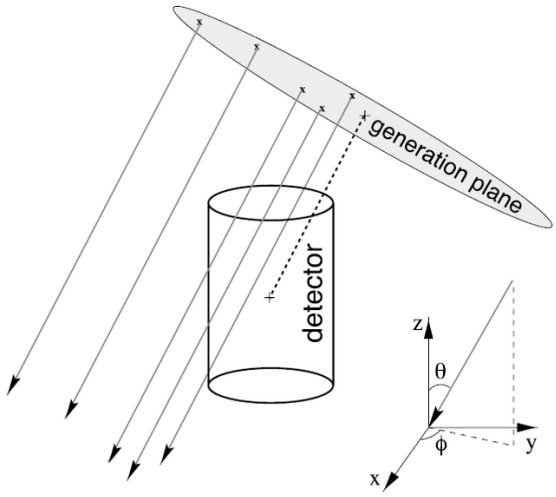

Figure 6.4: Illustration of how the particle injection works. The particle is first randomly positioned on a disk following a uniform distribution. The disk is then randomly rotated to simulate an isotropic flux.

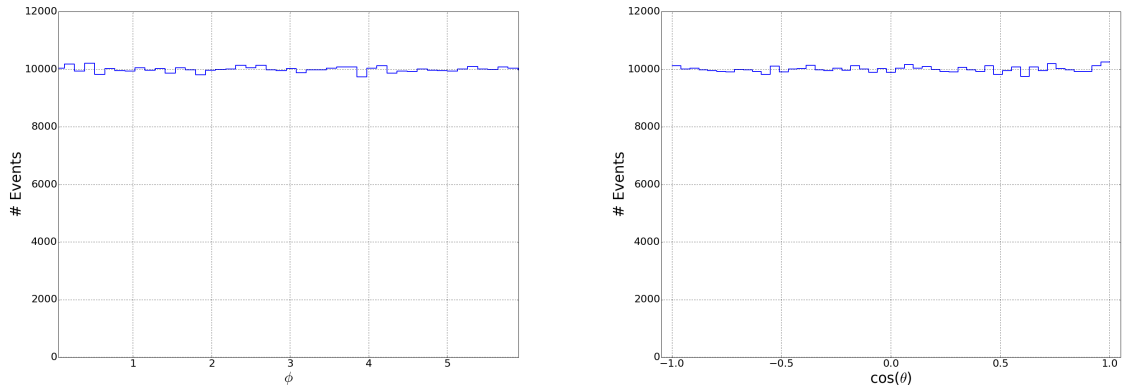

Figure 6.5: Illustration of uniform distributions of azimuth and cosine of the zenith for the particle injection in agreement with an isotropic flux (see Appendix B.1).

6.3 Propagation

After generation, the particles need to be propagated through the ice. The particles will interact, lose energy, produce new particles, and generate light. The particle interactions and light production are done in two different modules. The former module is called `PROPOSAL` and runs on normal CPUs, whereas the latter is called `ppc` and uses GPUs to simulate enormous amounts of photons that are propagated through the ice.

6.3.1 `PROPOSAL`

Using the cross sections of the important interactions, together with the properties of the traversed medium and the particles (mass, charge, spin, decay time, etc.), it is possible to simulate the energy losses, secondary production and the consequent interactions of these daughter particles. This is done in the software package `PROPOSAL` (the Propagator with Optimal Precision and Optimized Speed for All Leptons), fully written in C++. It was based on its predecessor `MMC` (Muon Monte Carlo), which was written in Java. In 2018, a substantially improved version of `PROPOSAL` was finalized. An illustration of the workings of the code is given in Figure 6.6 and an in-depth documentation is given in Ref. [206].

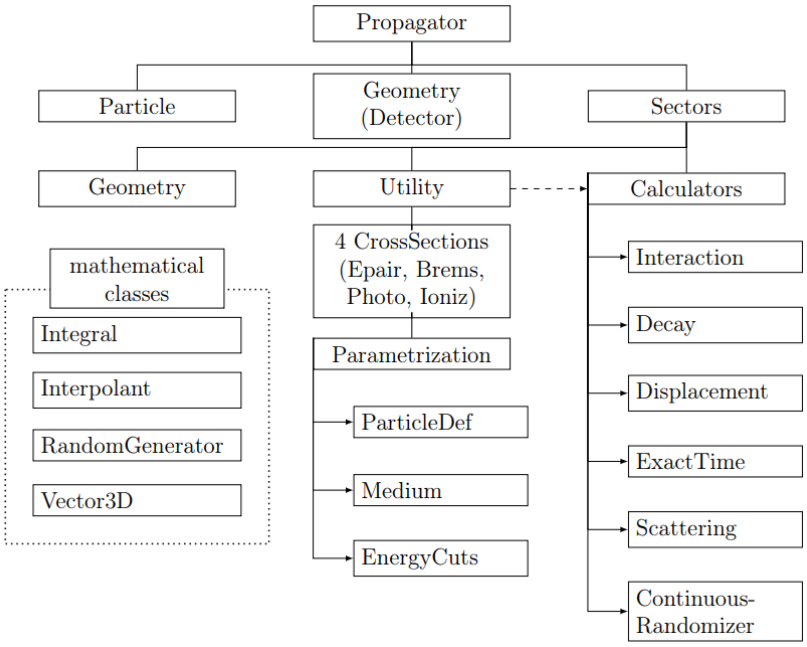

Figure 6.6: Overview of the class structure in PROPOSAL, from Ref. [206].

PROPOSAL for SMPs

Since we assume the SMPs to behave leptonically, it was chosen to use PROPOSAL for the signal propagation as well. The mass and charge of the particle are set in the input parameters and the cross section dependence on these parameters can be seen in Section 4.4. In general, there is a small dependence on the mass, but a squared dependence on the charge, except for bremsstrahlung, which has a quartic charge dependence.

The PROPOSAL module keeps track of all the particles that are produced during propagation and the accompanying energy losses in a tree-like structure (called an *I3MCTree*). This collection of particles and their interactions are forwarded to a light-production computation module.

6.3.2 Photoelectron generators

In Section 5.6.1 we already explained how the ice is simulated in the IceCube detector. The parameters $b_e(400)$ and $a_{dust}(400)$ define the photon propagation through the ice and determine if photons are absorbed in the ice or hit a DOM. To optimize computing time, the DOMs were scaled in radius (nominally with a factor of 5) to force more photon interactions. The number of photons emitted was then appropriately scaled down with the square of this scaling factor*. With the Frank-Tamm formula (Eq. 4.7) it is possible to calculate the expected number of photons produced per unit length in the wavelength interval of interest†

$$\frac{dN}{dx} = \int_{\lambda_1}^{\lambda_2} \frac{2\pi\alpha}{\lambda^2} \sin^2(\theta_c)\, d\lambda = 2\pi\alpha \sin^2(\theta_c) \left(\frac{1}{\lambda_1} - \frac{1}{\lambda_2}\right). \tag{6.3}$$

From this formula, we find that the expected rate of a Cherenkov emission profile is equal to ≈ 350 photons/cm. Together with the DOM acceptance curve, as shown in Figure 5.6, which has an overall average of around 7%, the expected *observed* number of photons is equal to 2450 m^{-1}.

PPC is a Photon Propagation Code, written in C++ that runs on graphic processing units (GPUs). This allows the code to run up to a hundred times faster than in a CPU-only environment. PPC employs both CUDA (NVIDIA GPU only) and OpenCL programming interface (both NVIDIA

*The surface of a sphere scales with the square of the radius.
†From Figure 5.6 it is clear that $\lambda_1 \approx 300$ nm and $\lambda_2 \approx 650$ nm.

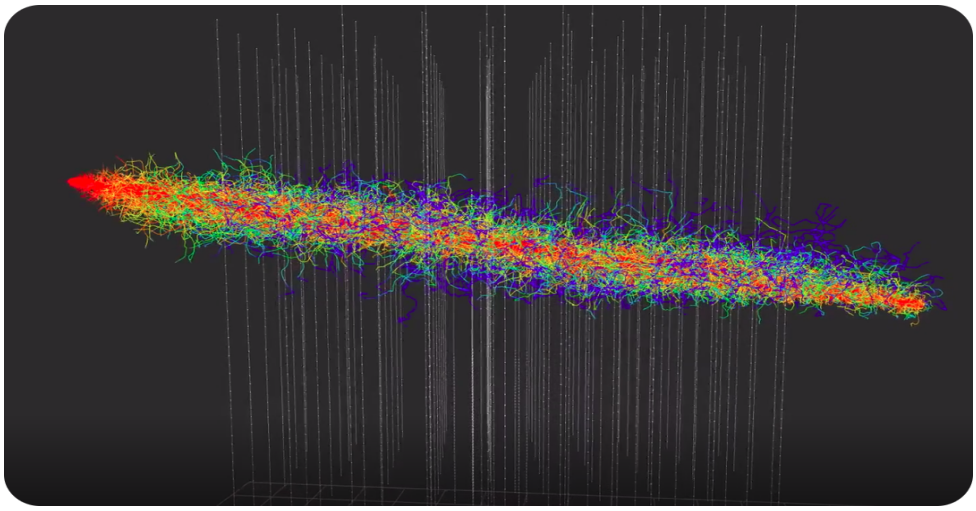

Figure 6.7: Simulation of a track event in IceCube. Each line represents a photon path and colors indicate how far they have traveled from their generation point.

and AMD GPUs) together with multiple CPU environments. GPU environments allow the tracking of thousands of photons simultaneously, vastly improving the computational speed. For more information, see Ref. [148].

Previous photon propagation codes, such as `Photonics` [207], produced 6-dimensional photon tables (3 spatial, 2 directional and 1 temporal). This meant that at least one set of tables had to be produced per particle type and per velocity and interpolation methods had to be used, with the accompanying inaccuracies. These tables also required significant disk space and the method was therefore replaced with the GPU-codes. Direct photon simulation also allows for other non-trivial implementations such as the tilting of ice layers.

Another photon propagation code is called `CLSim`, which uses GEANT4 to propagate particles. A hybrid version called `HybridCLsim` is often used. Muons are propagated using `PROPOSAL/MMC` and their stochastic losses (which are small showers) are simulated from tables whereas the "bare muons" (with their stochastics) are simulated using direct propagation. This avoids time loss for the rare but very computational high-energy cascade events.

An illustration of photon propagation in the IceCube detector for both a cascade and track simulation is shown in Figure 6.7.

6.4 Detector simulation

Further processing of the simulations involve:

- `Polyplopia`: a project dedicated to merge multiple events to account for coincident events (that are simulated independently). An estimated 15% of CORSIKA events result in coincident events and make up the bulk of bad reconstructions where downgoing muons are simulated as upgoing (example see Figure 7.7);
- `Vuvuzela`: the PMT noise is simulated as having an exponential component from thermal and radioactive decays, and a log-normal contribution for scintillation;
- `PMT`: the time from the first photon entering the PMT to the readout after passing along multiple dynodes has an uncertainty, referred to as "PMT jitter". The amplification of photoelectrons by the PMT is also not constant and is simulated in this module. Additionally, the module accounts for prepulses, late pulses, afterpulses and saturation of the PMT. More information can be found in Refs. [156, 208].
- `DOMLauncher`: the digitization of the PMT pulses and other behavior of the DOM mainboard (as explained in Section 5.2.1.2) is done in this module. The three main features of the

DOM that are simulated to generate launches are the discriminator, LC, and digitization.
- `trigger-sim`: simulation of the trigger behavior as explained in Sec. 5.4.1.

6.5 Processing

After a full particle propagation and detector response simulation, the sample is sent through the same PnF procedure as is done with the data (see Section 5.4.2). The different stages of processing are referred to as "Levels", where basic conversion from PnF formats to i3files (see Section 5.4.3) is called *Level0*. Reconstructions, calibrations and hit cleaning necessary for the filters are done at *Level1* while the filter processing is done at *Level2* (Section 5.4.2).

6.6 Simulation validation: burn sample

Getting the intricate details of physical events in non-trivial environments just right is not an easy task. In many steps of the way, simulations use fits and estimations. Some simulation datasets are reasonable to compare to the data, depending on the phase space one is looking at, while other datasets need other specifications. For example, analyses dedicated to measuring the cosmic ray interactions need much more fine-tuning in their models for the atmosphere, composition, interaction models, etc. than an analysis dedicated to search for muon tracks that first propagated through the Earth and have atmospheric muons as a background. Similarly, in this analysis the burn sample was used to compare data and MC to determine if the agreement between both made for a valid comparison. This is done throughout the analysis presented in Chapter 8.

It is for this reason that most analyses select a certain subset of the data they want to analyze to compare to the Monte Carlo simulations. For the present work, 10% of the total data, called the *burn sample*, was used to compare data to Monte Carlo. As indicated in Section 5.4.3, the data is saved in 8 hour runs and the burn sample consists of every run that ends on a '0'. The burn sample also allows to estimate the robustness of certain reconstructions and variables regarding differences in data and simulation.

After the discovery of an SPE offset in the DOM response in 2015, it was decided that multiple years of data was to be reprocessed in what was called *pass2 reprocessing* [160]. Aside from the SPE correction, the raw pulses were reprocessed with 2017 PnF, making the data more uniform in the course of the years for easier comparison*. This analysis makes use of data starting in the years 2011 to 2015 and are referred to as IC86-1 to IC86-5 where IC86 stands for the complete 86-string IceCube detector configuration and the last digit refers to the year of the season start. More recent years were not fully processed in time for this analysis. Only runs were considered that had

1. a positive tag from run coordinators (status == "good_i3"),
2. at least 5000 active optical modules,
3. all strings active during runtime.

The *livetime* is the total time that the detector was up and running and non-corrupted data was processed. Due to the increase in detector uptime over the years, this also means that the livetime of the different datasets has increased. The livetime for the different years is equal to

- 31.0 days of livetime for IC86-1,
- 32.2 days of livetime for IC86-2,
- 33.2 days of livetime for IC86-3,
- 36.6 days of livetime for IC86-4,
- 36.7 days of livetime for IC86-5,

resulting in a total burn sample livetime of around 170 days.

*Most filters did not undergo changes or only minor ones.

6.7 Event viewer

After a full simulation, it is possible to visualize the event in an event viewer called `Steamshovel`. Typical events in the IceCube detector are shown with this interface and are loaded from *i3files* that contain information about the detector geometry and the event (DOM positions and calibrations, detector hits, timestamps, trigger hits, etc.). Simulated events also contain the true values of the particles and can be compared with reconstructed variables. Event viewers allow for first guesses in how background events are able to be separated from signal, although both can have wide varieties in possible outputs.

The number of photons seen per DOM is indicated by the size of the spheres; the larger the sphere, the more PEs were seen. The color of the spheres indicates the time of the pulse registration. The color scale can be chosen, but usually a rainbow pattern is used where red indicates the earliest pulse hits and blue the last.

An example of a track event in the event viewer is given in Figure 6.7.

7. Reconstruction, Cleaning and Analysis Techniques

Shall I refuse my dinner because I do not fully understand the process of digestion? ~ Oliver Heaviside

As explained in Chapter 5, an IceCube event consists of a series of DOM hits. One hit contains charge, timing and positional information. This collection of multiple hits has to be translated (reconstructed) to certain properties of a particle. Some examples are the direction, position, track length, etc. Fast reconstructions of these properties, such as the ones necessary in online filtering, are usually done with simple algorithms. More sophisticated algorithms are performed offline on smaller data samples.

Because the in-ice IceCube detector is sparsely distributed, it is not straightforward to unambiguously reconstruct the particle (interactions). The scattering and absorption of photons, tilt of ice sheets, bubble column, etc. lead to uncertainties and make reconstruction challenging. Over the years, multiple different methods have been developed in the collaboration. Several reconstruction algorithms have been used in this analysis and are explained in more detail in this chapter. We start with the algorithms that reconstruct where the particle entered and in which direction it traversed the detector and give an estimate on the direction uncertainty.

A number of hits in an event can often be attributed to DOM noise. There are several techniques possible to remove these hits, some of which are used in this analysis. They are also discussed in more detail below.

Finally, we discuss the methods that were used to discriminate signal from background events with machine learning techniques.

7.1 Reconstruction

7.1.1 Likelihood

Reconstruction algorithms usually have no unique solutions to describe the set of measured values of an event. The likelihood $\mathcal{L}(\vec{x}|\vec{a})$ describes the probability of a set of parameters $\vec{a}$ to be expressed in a set of experimentally measured values $\vec{x}$. The parameters, $\vec{a}$, typically define the particle's characteristics (energy, direction, position, type, etc.) while the measured values $\vec{x}$ are determined from the detector response (number of PE, timing, position of hit DOMs, etc.). This likelihood is equal to the cumulative probability

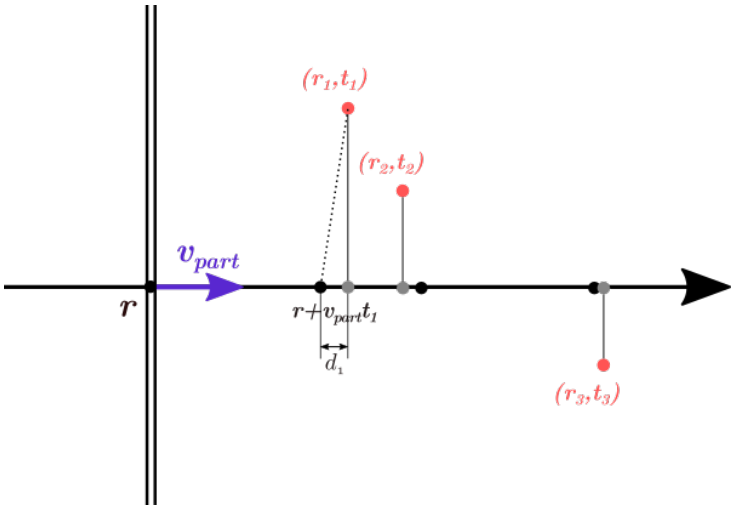

Figure 7.1: Figure illustrating how `LineFit` works. The position, $\vec{r}$, and velocity, $\vec{v}_{\text{part}}$ minimizing the distance of the DOMs to the track is calculated. The dotted line is one of the distances that is minimized in Eq. 7.4.

$$\mathcal{L}(\vec{x}|\vec{a}) = \prod_i p(x_i|\vec{a}), \tag{7.1}$$

where $p(x, \vec{a})$ is the probability that we measure a certain value x from a set of independent values $\vec{x}$, given an initial set of parameters $\vec{a}$. The maximum likelihood method is used to estimate the unknown parameters $\vec{a}$, which is done by maximizing $\mathcal{L}$. The reconstruction algorithms below rely on analyzing parameters that assume a single, long track

$$\vec{a} = (\vec{r_0}, t_0, \hat{\vec{p}}, E_0), \tag{7.2}$$

where $\vec{r_0}$ is the position vector of the particle at a time t_0 with a direction $\hat{\vec{p}}$ and initial energy E_0.

7.1.2 `Line-Fit`

One of the simplest approaches in constructing a parameter profile is by calculating the track that, overall, has the closest approach of all the hit optical modules. This is called `Line-Fit` (LF) [209]. If we assume that a particle starts at a position $\vec{r}$ at a time $t = 0$ and travels at a velocity of $\vec{v}_{\text{part}}$, then its position at any given time is

$$\vec{r'} = \vec{r} + \vec{v}_{\text{part}} t. \tag{7.3}$$

We want to calculate the best possible estimate of the velocity $\vec{v}_{\text{part}}$ and an initial position $\vec{r}$. Each DOM has a known location, $\vec{r}_i$, and measured time of a pulse, t_i. In this algorithm, one assumes that a wavefront perpendicular to the particle's direction is traveling along with the particle. If the velocity $\vec{v}_{\text{part}}$ is fixed, then the position of the particle at later times is known (black points in Figure 7.1). However, the Cherenkov wavefront should be set at an angle and because scattering, PMT jitter, noise, etc. are not taken into account, this will not agree with the DOM position projected along the particle path (grey dots). The unknown velocity $\vec{v}_{\text{part}}$ and position $\vec{r}$ are the analytical solutions after minimizing the distances d_i as shown in the figure*

*Minimizing $r_i - r'$ (dotted line in Figure 7.1) is the same as minimizing d.

$$
\begin{aligned}
S(\vec{r}, \vec{v}_{\text{part}}) &\equiv \sum_{i=1}^{N_{\text{hit}}} \rho^2(\vec{r}, \vec{v}_{\text{part}}, \vec{r}_i, t_i) \\
&= \sum_{i=1}^{N_{\text{hit}}} (\vec{r}_i - \vec{r} - \vec{v}_{\text{part}} t_i)^2 \,,
\end{aligned}
\tag{7.4}
$$

where $\rho(\vec{r}, \vec{v}_{\text{part}}, \vec{r}_i, t_i) = |\vec{r}_i - \vec{r} - \vec{v}_{\text{part}} t_i|$ and N_{hit} is the number of pulse hits. The analytical solution by minimizing this equation is equal to

$$
\vec{r} = \langle \vec{r}_i \rangle - \vec{v}_{\text{part}} \langle t_i \rangle \quad \text{and} \quad \vec{v}_{\text{part}} = \frac{\langle \vec{r}_i t_i \rangle - \langle \vec{r}_i \rangle \langle t_i \rangle}{\langle t_i^2 \rangle - \langle t_i \rangle^2},
\tag{7.5}
$$

where $\langle x \rangle$ denotes the average of a parameter x over all hits i.

Because this is an analytical equation, this algorithm is very fast and is therefore often used in online processing.

7.1.2.1 `Improved Line-Fit`

`Line-Fit` is usually implemented as a seed track for other, more computationally heavy algorithms (such as SPE, see Section 7.1.3). The simplifications that are used in LF can lead to angular deviations that converge to a local minimum instead of the global. The LF algorithm assumes that all hits will be near the track and hits far away from the track enter the least squares computation quadratically. Therefore, hits from far away often dominate the reconstruction even though the simple algorithm doesn't account for

1. The Cherenkov emission profile.
2. The scattering effects of the ice medium.
3. Noise hits that occur far from the track.

To reduce the effects of outliers, it was found that a basic filter could identify these scattered hits, and improve accuracy by almost a factor of two by removing them from the dataset. More formally, for each hit h_i (that consists of a charge, position and timing), the algorithm looks at all neighboring hits within a neighborhood of radius μ, and if there exists a neighboring hit h_j with a time stamp that is t earlier than h_i, then h_i is considered a scattered hit and is not used in the basic reconstruction algorithm. Optimal values of μ and t were found to be 156 m and 778 ns by tuning them on simulated muon data [210].

This "delay cleaning" is done by computing a Huberfit on the remaining data points and minimizing

$$
\sum_{i=1}^{N_{\text{hit}}} \phi(\rho(\vec{r}, \vec{v}_{\text{part}}, \vec{r}_i, t_i)),
\tag{7.6}
$$

where ρ is defined in Eq. 7.4 and the Huber penalty function ϕ is defined as

$$
\phi(\rho) \equiv \begin{cases} \rho^2 & \text{if } \rho < \mu \\ \mu\,(2\rho - \mu) & \text{if } \rho \geq \mu \end{cases} .
\tag{7.7}
$$

An example of the Huber penalty function is given in Figure 7.2. Because of the overall performance increase of this method, all LF computations were done with the improved version (although still often referred to as "Line-Fit").

7.1.3 SPE and MPE

A more intricate method of track reconstruction is done by taking the geometrical shape of the Cherenkov cone into account and relying on simulation fits where a seed track is implemented (usually from the fast `Line-Fit` algorithm) [209].

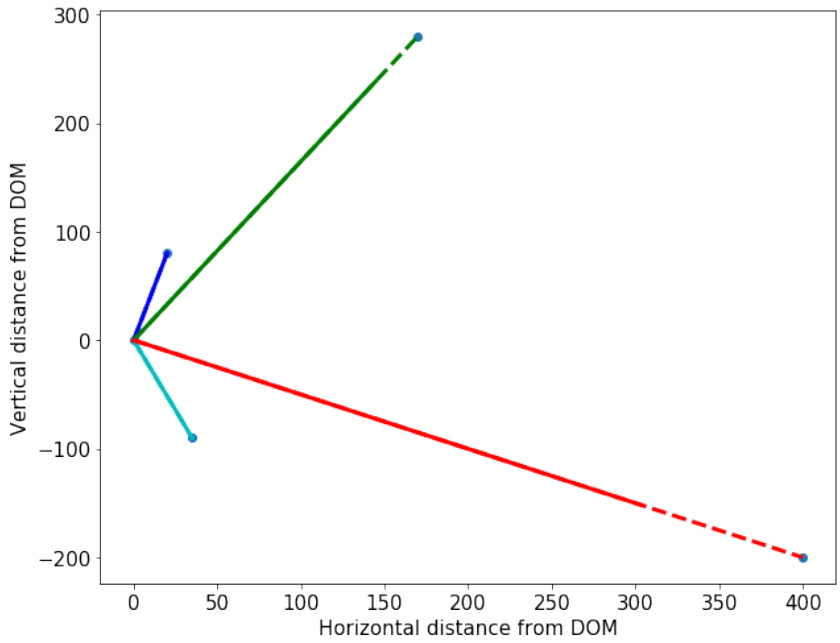

Figure 7.2: Illustration of Huber penalty for possible hit locations. The origin was set as the assumed particle position, $\vec{r} + \vec{v}_{\text{part}} t_i$. The solid line shows the result of $\phi(\rho)$ where the dotted lines show values for ρ.

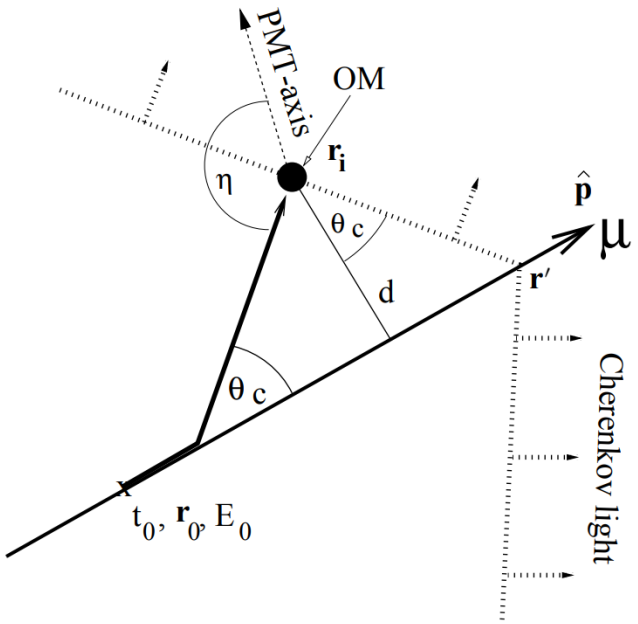

Figure 7.3: Figure illustrating a muon track passing close by an optical module and defining the parameters used in the reconstruction algorithms. Illustration from Ref. [209].

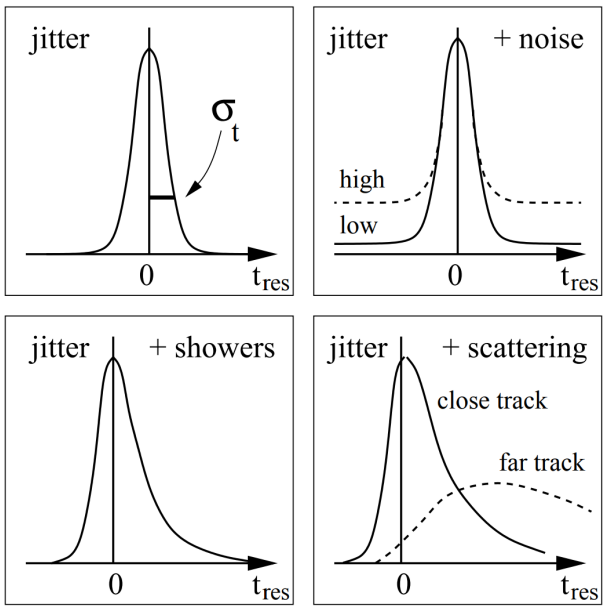

Figure 7.4: Schematic distributions of t_{res} where several effects are included. *Top left:* PMT jitter effects. *Top right:* jitter and random noise effects. *Bottom left:* adding indirect Cherenkov light from stochastic deposits along the track of a high-energy particle. *Bottom right:* adding scattering delay effects. Figure from Ref. [209].

Let us assume a particle is traveling close to a DOM with parameters as defined in Eq. 7.2 as illustrated in Figure 7.3. The minimal distance of the track to the DOM is equal to d and the PMT-axis (downwards relative to DOM) has an angle offset of η degrees of the Cherenkov wave direction. If Cherenkov photons would travel undelayed directly from the primary particle to an OM, the *time residual* (time between the observed hit time and the "expected" time) is a delta function centered around 0, where

$$t_{\text{res}} \equiv t_{\text{hit}} - t_{\text{geo}}, \tag{7.8}$$

with

$$t_{\text{geo}} = t_0 + \frac{\hat{\vec{p}} \cdot (\vec{r}_i - \vec{r}_0) + d \cdot \tan(\theta_c)}{c_{\text{vac}}}, \tag{7.9}$$

which is the time necessary for the particle to travel from the position $\vec{r}_0$ to $\vec{r}'$ as illustrated in the figure. The accompanying Cherenkov wavefront that sent out photons at a time t_0 from $\vec{r}_0$ will cross the DOM when the particle is at a position $\vec{r}'$. Due to noise effects, PMT jitter, light from secondary interactions, DOM orientation, etc. the time residual is smeared and shifted as can be seen in Figure 7.4. Stochastic energy deposits along the track of a high-energy particle (as described in Section 4.3.2.1) lead to an additional shift of the time residual since these photons arrive after the Cherenkov cone.

A time likelihood function $\mathcal{L}_{\text{time}}$ can be constructed from the p.d.f. of arrival times of single photons, p_1, at the locations of the hit DOMs. The p.d.f. of single photons was estimated with photon simulations in ice and fitted to a *Pandel function*. How this was done and for more information on this function, I refer the reader to Ref. [209].

$$\mathcal{L}_{\text{time}} = \sum_{i=1}^{N_{\text{hit}}} p_1(t_{\text{res}} | \vec{a} = d_i, \eta_i, ...). \tag{7.10}$$

An initial particle position and direction are found by maximizing the likelihood and iterated a couple of times to find the global maximum instead of a local. This fitting is called the *Single PhotoElectron (SPE) fit*.

Unfortunately, the description of single photons arriving at the optical modules is not fully correct since electrical and optical signal channels can only resolve multiple photons separated by a few 100 ns and $\approx$ 10 ns, respectively. Within this time window, only the arrival time of the first pulse is recorded.

In the *Multi-PhotoElectron (MPE) fit*, one accounts for the fact that the early photons in a DOM hit scattered less in the ice. The p.d.f. for the first photon out of a total of N to arrive with a time residual of t_{res} is

$$p_N^1(t_{\text{res}}) = N \cdot p_1(t_{\text{res}}) \cdot \left(\int_{t_{\text{res}}}^{\infty} p_1(t)dt \right)^{(N-1)} = N \cdot p_1(t_{\text{res}}) \cdot (1 - P_1(t_{\text{res}}))^{(N-1)}, \tag{7.11}$$

where P_1 is the cumulative distribution of the single photon p.d.f. The MPE likelihood computation is done by replacing p_1 in Eq. 7.10 with p_N^1.

Several variables that are obtained from these algorithms were used in this analysis. For example, the direction of the reconstructed particle was used to look for upgoing tracks, while the likelihood of the reconstruction was used to remove downgoing muon tracks that were wrongfully reconstructed as upgoing.

7.1.4 `Paraboloid`

In Sections 7.1.2 and 7.1.3, we discussed how a particle's direction could be estimated with likelihood estimations. The `Paraboloid` module tries to provide an estimate for the error on this direction. A highly energetic muon event with hundreds of hit DOMs will lead to a much better directional resolution than a dim track where only a handful of DOMs are hit. In general, the likelihood space around the estimated direction is scanned and compared to the likelihood of the initial track estimation. If the likelihood space around the minimum is relatively flat, there is a larger uncertainty in the directional estimation. With this method, the module also tries to parameterize if the maximum likelihood that is found from the reconstruction algorithms is far from, or close to the global maximum likelihood. The module constructs a grid of zenith and azimuth points near the minimum and for each point on the grid, it does a three-parameter minimization for the vertex holding the zenith and azimuth constant. The likelihood values for each point on the grid are then fit to a paraboloid (2D parabola) using a χ^2 minimization since the shape of the log-likelihood space near the minimum should have a paraboloid shape. Of importance are the parameters of the corresponding error, which are assumed to correspond to an ellipse for the 1σ contours.

The module computes the lengths of the semi-major and semi-minor axes of the 1σ error ellipse σ_1 and σ_2*. It was found that the quadratic mean of both uncertainties provides a good single-valued estimate for the angle uncertainty

$$\sigma_{\text{para}} = \sqrt{\frac{\sigma_1^2 + \sigma_2^2}{2}}. \tag{7.12}$$

Since σ_1 and σ_2 should follow univariate Gaussian distributions, σ_{para} will be the radius parameter in a bivariate distribution. Also, the great circle distance between the MC truth of the signal particle and the reconstruction direction, as calculated by the Vincenty formula, is a two-dimensional distribution. From the Rayleigh distribution, it follows that the mean of this distribution should be equal to $1.177\sigma_{\text{para}}$†. Therefore, if we divide the great circle distance with σ_{para}, we should obtain a variable that is distributed with a peak at 1.177 (mean). This variable is called the *paraboloid pull* and should be compared to an energy related variable (here

*The confidence intervals σ_θ and σ_ϕ can be found by rotating the minor and minor axes σ_1 and σ_2.

†The CDF of the Rayleigh distribution $1 - e^{-x^2/2\sigma^2}$ is equal to the containment for a bivariate normal distribution. Implementing $x = 1.177\sigma$ yields a factor of 0.5.

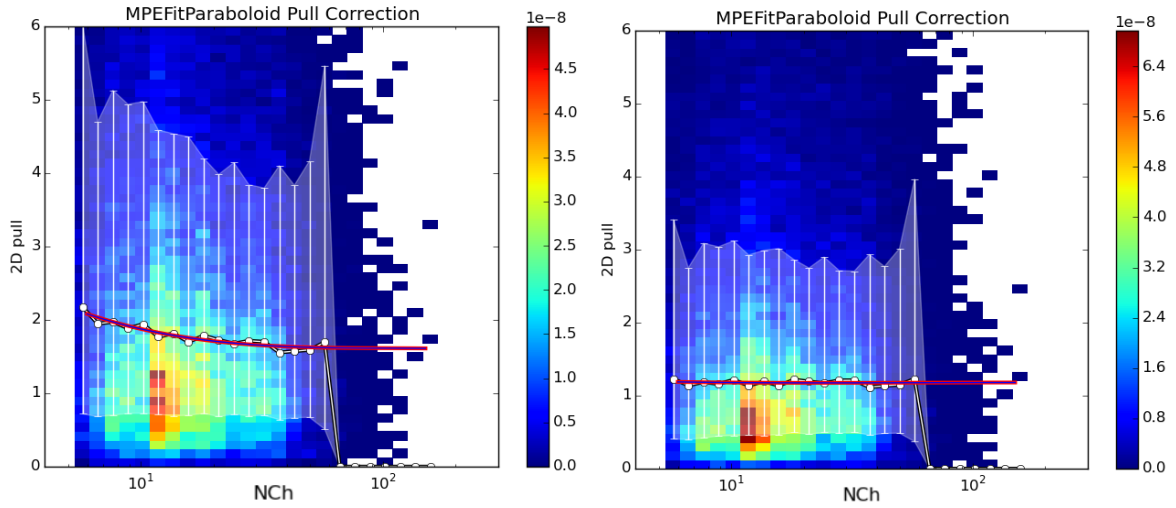

Figure 7.5: *Left:* Paraboloid pull in function of the number of hit DOMs shows that the global average (red/purple line) is not centered at 1.177. *Right:* Pull after correction.

the number of hit DOMs (NCh) is used). In Figure 7.5, it is clear that there is an offset that changes in function of the number of hit DOMs. This effect is seen in all analyses and can be explained by multiple factors in Monte Carlo simulations, but mainly stems from our incomplete knowledge and non-perfect simulations of the ice. The energy-dependent correction factor is computed from Monte Carlo events and applied to both data and MC. From the figure, it is clear that the pull is overestimated (σ_{para} is underestimated), meaning the correction will increase the directional uncertainty σ_{para}, which is by any means a conservative method. More information on this algorithm can be found in Ref. [211].

7.1.5 `Millipede`

To have a better handle on the particle energy and stochastic cascades along the track, the module `Millipede` was developed [212, 213]. The number of photons seen at each optical module depends on multiple factors that were mentioned throughout this text, such as the scattering and absorption in the ice, timing jitter, etc. In this module, the expected number of photons is assumed to depend on the energy that was deposited along a track and a *light yield factor*, Λ, that depends on the DOM position and the location of emission

$$\begin{aligned} N_{\text{exp},k} &= \rho_k + \sum_{i=1}^{n} \Lambda(\vec{r}_k, \vec{r}_i')E_i \\ &= \rho_k + \vec{\Lambda}(\vec{r}_k, \vec{r}_i') \cdot \vec{E}, \end{aligned} \tag{7.13}$$

where k refers to a certain DOM and i refers to a certain energy deposit such as illustrated in Figure 7.6. ρ is the average expected number of noise photons and is determined by the duration of the event. The light yield factor and energy are written as vectors to shorten the expressions.

The likelihood is assumed to follow a Poisson distribution with $N_{\text{seen},k}$ the number of hits that occurred and a mean value equal to the number of expected photons, $N_{\text{exp},k}$

$$\begin{aligned} \mathcal{L}_k &= \frac{(N_{\text{exp},k})^{N_{\text{seen},k}}}{N_{\text{seen}}, k!} e^{-N_{\text{exp},k}} \\ &= \frac{\left(\vec{\Lambda} \cdot \vec{E} + \rho_k\right)^{N_{\text{seen},k}}}{N_{\text{seen},k}!} e^{-\vec{\Lambda}\cdot\vec{E}-\rho_k}. \end{aligned} \tag{7.14}$$

For easier, faster and more accurate computation, the logarithm of the likelihood is used

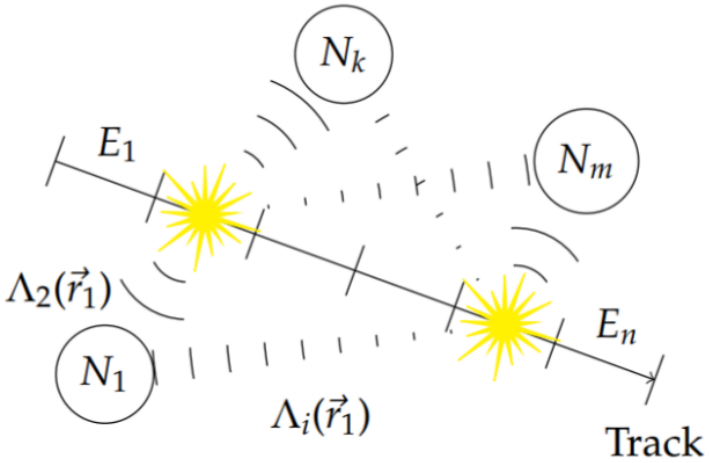

Figure 7.6: Illustration of the working principles of the `Millipede` toolkit. A track is subdivided into segments that each deposit a certain energy, E_i. Different segments can contribute to the total number photons seen per DOM, N_k.

$$\begin{aligned}\ln \mathcal{L}_k &= N_{\text{seen},k} \ln \left(\rho_k + \sum_{i=1}^{n} \Lambda(\vec{r}_k, \vec{r}_i') E_i \right) - \ln\left(N_{\text{seen},k}!\right) - \sum_{i=1}^{n} \Lambda(\vec{r}_k, \vec{r}_i') E_i - \rho_k \\ &= N_{\text{seen},k} \ln \left(\rho_k + \vec{\Lambda}(\vec{r}_k) \cdot \vec{E} \right) - \vec{\Lambda}(\vec{r}_k) \cdot \vec{E} - \rho_k - \ln\left(N_{\text{seen},k}!\right)\end{aligned} \tag{7.15}$$

Assuming a total of m DOMs that registered hits, we can maximize the total likelihood with respect to the energy, which gives

$$\nabla_{\vec{E}} \ln \mathcal{L} = \nabla_{\vec{E}} \sum_{k=1}^{m} \ln \mathcal{L}_k = \sum_{k=1}^{m} \left(\frac{N_{\text{seen},k} \vec{\Lambda}(\vec{r}_k)}{\vec{\Lambda}(\vec{r}_k) \cdot \vec{E} + \rho_k} - \vec{\Lambda}(\vec{r}_k) \right) = 0. \tag{7.16}$$

This equation holds if all terms in the sum vanish, i.e. if for all DOMs holds that

$$\begin{aligned} N_{\text{seen},k} &= \vec{\Lambda}(\vec{r}_k) \cdot \vec{E} + \rho_k \\ &\overset{\text{Eq.7.13}}{=} N_{\text{exp},k}. \end{aligned} \tag{7.17}$$

This can be written in a set of linear equations

$$\vec{N} - \vec{\rho} = \mathbf{\Lambda} \vec{E}, \tag{7.18}$$

where

$$\mathbf{\Lambda} = \begin{pmatrix} \Lambda(\vec{r}_1, \vec{r}_1') & \cdots & \Lambda(\vec{r}_1, \vec{r}_n') \\ \vdots & \ddots & \dots \\ \Lambda(\vec{r}_m, \vec{r}_1') & \cdots & \Lambda(\vec{r}_m, \vec{r}_n') \end{pmatrix}, \tag{7.19}$$

is the *response matrix*. This has to be inverted to find the energies in the vector $\vec{E}$. It describes the DOM response to light output from certain segments along a track. The entries in this matrix come from simulations that produce spline tables. Simplified sources, such as minimum ionizing muons and isotropically emitting point sources are simulated in Monte Carlo simulations at certain discrete points. Interpolation is done using spline functions. More information, such as how timing information can be implemented, can be found in Refs. [212, 214].

The `Millipede` algorithm was used to construct a handful of variables to discriminate the dim SMP tracks from muon tracks.

7.1.6 `FiniteReco`

Charged particles such as muons and electrons can be created from neutrino interactions. Electrons are stopped within a couple tens or hundreds of meters, but muons can travel up to kilometers and do not have a spherical light deposit. Because neutrinos do not produce Cherenkov light, this "sudden" light production from muons can look like a *starting track* in the IceCube detector. Charged particles that travel through matter also lose energy and can therefore be stopped, giving rise to *stopping tracks*. `FiniteReco` is a module that tries to reconstruct if particles are starting, stopping, contained or throughgoing. The hit DOMs around a seed track (usually the reconstructed track) are checked to have seen light and the first and last emission points along the track are used to check the possible hypotheses. Because SMPs are assumed to create long tracks, the module can be used to remove tracks that start or stop within the detector.

Because the edges of the detector are not well defined*, the likelihoods of individual DOMs to have seen a hit lead to a total likelihood that doesn't give a conclusive answer, but the starting and stopping probabilities can be compared to a throughgoing track hypothesis.

7.2 Pulse cleaning

As explained in Section 5.2.2, each DOM in IceCube has an intrinsic noise rate. This dark noise is observed in every triggered event and seen as random hits in the detector that are added to the hit pattern of tracks and cascades. These spurious hits are a large nuisance factor in event reconstructions, leading to misidentification and errors in the result. Noise cleaning should be done in early stages of event processing and analysis to reduce a large rate of bad reconstructed events that pass cut selections. One of the most conservative ways is to only look at HLC hits (*HLC cleaning*). However, this is too demanding for most low-energetic and/or dim events that will have multiple hits from isolated DOMs.

Another method, which tries to include these isolated hits, is the `seededRT` algorithm. This method relies on the "RT-cut", which was already implemented in the time of AMANDA operations. R is a designed radius and T refers to the time between multiple possible pulse times (e.g. the pulse of one DOM starts during the time window of a second DOM's pulse, or stops during the time window). The full description can be found in Ref. [215], but can be summarized as follows: DOMs are required to be in a temporal and spacial coincidence that is physically possible (e.g. the signal between DOMs cannot exceed the speed of light in vacuum). This method is, however, computationally expensive since all DOM pairs have to be looped over†. The `seededRT` algorithm takes a subset of seeds that are considered to be mostly signal related hits. These seeds can be provided by, for example, using HLC information. By adding all further hits found within the seed's RT-range to the list of seed hits and iterating until convergence, only those (SLC) hits are kept that cluster around the initial seed hits. Thus, the `seededRT` cleaning is applied in the opposite way as the previously used "RT-cut". While the old module looks at all hits and removes the ones not fulfilling the RT-condition, the `seededRT` initially keeps only a subset of the launches ("seeds") and adds those launches in the RT-range of the already kept ones. In this way, outlying RT-hit clusters from noise can be rejected and has a better noise hit rejection than the classic RT-cut. Therefore, the pulses used in this analysis are first cleaned using the `seededRT` algorithm.

7.3 `IceHive`

In Section 5.4.1, it was explained how multiple triggers were combined into one global trigger. In a first step, Q-frames (which are collections of hits in a global trigger and holds information on which triggers and filters were passed, also keeping the uncleaned pulses) are simply re-split

*Imagine a cascade 20 m below the lowest DOMs. It is still possible for light to reach the bottom modules of the detector.

†The number of pairs for n DOMs is equal to $\frac{1}{2}n(n-1)$ and scales with n^2.

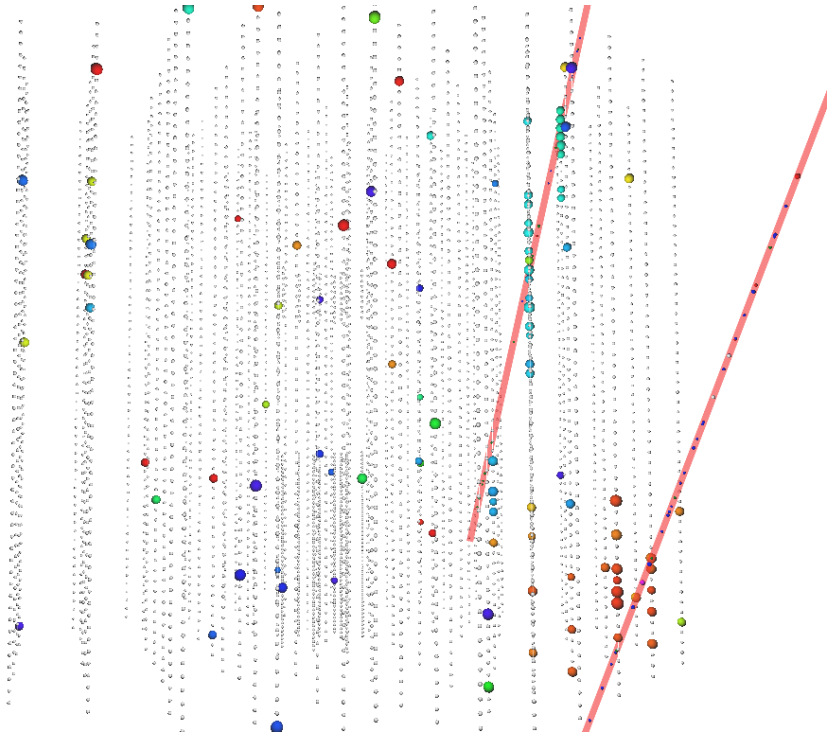

Figure 7.7: Event display of a simulated coincident event of two downgoing muons. The colors of the event range from red (early) to blue (late). The first muon hits the bottom of the detector, while the second traverses mainly the upper part. These events are reconstructed as a single upgoing event and therefore result in a large background contribution for analyses that focus on upgoing tracks such as this one. The scattered isolated hits are due to noise effects and mostly removed by pulse cleaning.

into the individual events that belong to the different subtriggers and are stored in separate P-frames (which hold all processing information and reconstructions, together with additional pulse cleaning). In about 15% of cases, the data read out in one of these P-frames contains more than one primary interaction. This pile-up effect is referred to as *coincident events*. It is a direct result of the traversal time of a couple of microseconds in the detector (the speed of light in vacuum is equal to ≈ 0.3 m/ns, meaning the particle travels around 100 m in 0.3 μs, without accounting for the delayed photon propagation necessary for detection), the large flux of low-energetic events and the trigger time windows of a couple of microseconds. This can be problematic for reconstructions, as can be seen in Figure 7.7 where two downgoing muons can be reconstructed as an upgoing track. These events can be split when looking at, for example, geometric separation. How this is done is explained in more detail below.

There are two modules that try to clean events more thoroughly than pulse cleaning alone. The first is `TopologicalSplitter` (TS), which starts from the Q-frames and loops over pulses and splits the event into clusters of pulses that contain at least a number of causally* connected pulses within a certain time window. Some extra cleaning, similar to `seededRT` cleaning, is done in addition and can split coincident events that have overlapping readout windows, but are geometrically separated.

The second module, also used in this analysis, is called `IceHive`. A full description can be found in the doctoral thesis of M. Zoll [216]. The module consists of two main parts: one that splits events and handles coincident events, `HiveSplitter`, and another that has a refined pulse cleaning, `HiveCleaning`.

*The time between two DOM hits cannot be less than the time that light would have taken.

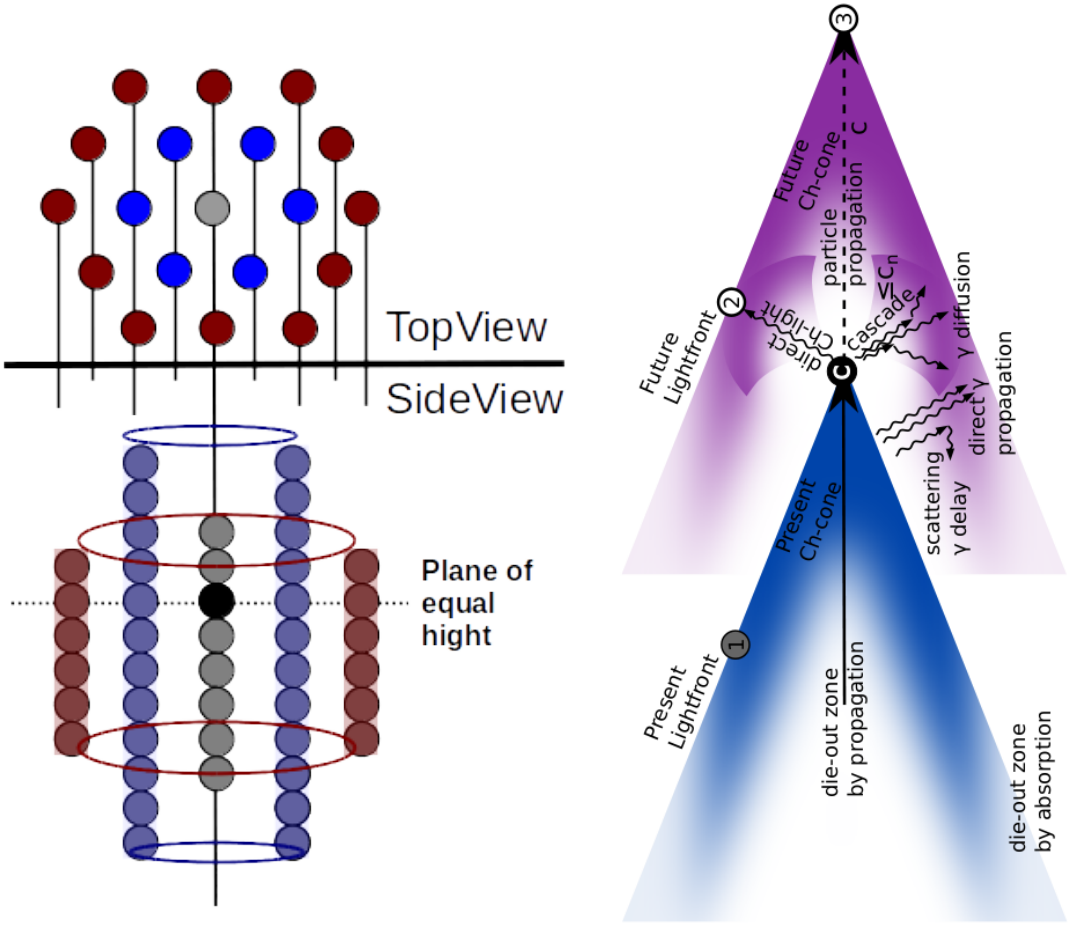

Figure 7.8: *Left:* The black circle illustrates a DOM that triggered a hit in the detector. The grey circles symbolize the DOMs along the string of the hit DOM. The number of DOMs that can be included in the active volume depends on the heights defined by the module. The blue/purple DOMs belong to the neighboring strings and the red/brown DOMs to the next-to-neighboring strings. The heights of both these sets of DOMs are also set by the module. This example shows $h_2 > h_1 > h_3$, where h_i is the height of the i-th string from the center. The heights are asymmetric in this example. *Right:* Illustration of Cherenkov emission profile of a traversing particle. Both figures from Ref. [216].

7.3.1 `HiveSplitter`

The module assumes that individual particles creates *clusters* of hits in the detector. A cluster can grow within a certain time window, but is separated from another cluster if it's not spatially connected. An initial cluster is formed if the multiplicity of hits exceeds a certain threshold (usually 3 or 4). The main difference in this module versus `TopologicalSplitter` is that it uses hexagons to describe the detector instead of assuming a spherical parameterization. It makes more sense to optimize the search volume, where hits are clustered together, with a shape that describes the detector well and uses a discrete spacing between larger volumes instead of a uniformly growing sphere. The hexagonal shape is set by defining three heights. The first height, h_1, is defined along the string of the hit DOM and is equal to the vertical distance along the string. The second height, h_2, is the vertical height along the neighboring strings. The third height, h_3, is the vertical height along the next-to-neighboring strings. An example is shown in Figure 7.8 (*left*).

When the active region is set, there is an additional check to see if DOMs can be "connected". `IceHive` assumes certain emission profiles (for both cascades and tracks) where light is produced. Three possible connections are assumed:

1. Hits occur at the same time, but at a spatial distance in agreement with the Cherenkov

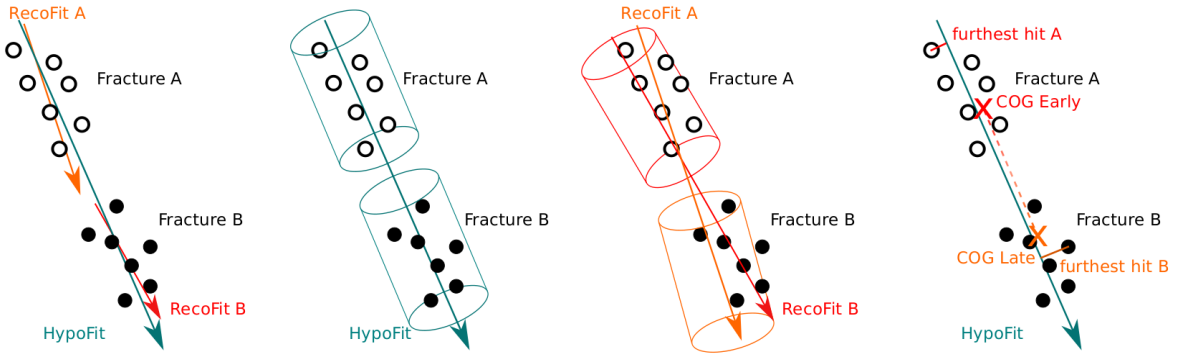

Figure 7.9: Schematic illustrations of possible recombination scenarios with the `CoincSuite` module. Two events are compared to each other or the combined event (HypoFit) in several different ways. Figure from Ref. [216].

emission profile (hits C&1 and 2&3 in Figure 7.8 (*right*)).
2. Hits occur at a different time and a different location, but in agreement with the Cherenkov emission profile (hits C&2 in Figure 7.8 (*right*)).
3. Hits on topologically identical sites of an emission pattern that has moved along with the propagation of the particle (hits C&3 and hits 1&2 on Figure 7.8 (*right*)).

These clusters are finally separated into different P-frames. Each P-frame is assumed to originate from a different primary particle.

7.3.2 `HiveCleaning`

Additionally, a similar cleaning as explained in Section 7.2 can be performed. Isolated hits that do not have neighboring hits occurring within a certain distance and time window, are removed. The main difference between this and `seededRT` cleaning is that the module again uses the hexagons as defined in the previous section.

7.3.3 Remark

The use of `IceHive` has a better performance in separating coincident events compared to `TopologicalSplitter` (see Ref. [216]), but often "over-performs" and splits clusters of hits that are originating from the same particle. This is predominantly the case for dim tracks that have large separations in between clusters (most of the triggered SMP events are of this type). To reduce this type of error, the module `CoincSuite` was designed, again by M. Zoll [216].

7.4 CoincSuite

Several testing algorithms allow to check if two or more split P-frames can originate from a single event. Five different scenarios were tested in this analysis, the first four are also chronologically shown in Figure 7.9:

1. Cluster alignment: the reconstructed direction of the individual clusters is compared to the direction of a reconstruction that uses the combined hits (HypoFit). The directions should be within a certain critical angle.
2. Cylinder cluster containment: the DOMs of the individual clusters should be able to be grouped together in a cylinder that has its center and direction along the HypoFit.
3. Cylinder cluster alignment: a cylinder around the reconstruction of each cluster is draw. The cylinders should overlap within a certain fraction.
4. COG* connection: the COG of the second quarter of hits of the the first cluster and the

*Similar to COM (Center Of Mass), the COG is a weighted average position of the hit DOMs. DOMs that register more light get a higher weight.

COG of the third quarter of hits of the second cluster are computed. These COGs should lie close enough and have to be in the vicinity of the HypoFit.

5. Velocity test: tests if the velocity of the HypoFit is close to the speed of light.

The combination of `IceHive` (which does a very good job in cleaning events, but often over-performs and splits events that shouldn't be) and `CoincSuite` (which recombines events that were wrongfully split), leads to a very powerful tool to clean events.

7.5 Event cleaning and reconstructions: summary

The previous sections described how events can be cleaned and how several reconstruction algorithms work. In Chapter 5, we described how such an event is constructed. Several triggers read out series of hits in the IceCube detector within a time window to an event. A hit consists of a DOM pulse, the DOM location and a timestamp. One DOM can have multiple hits in an event.

Reconstruction algorithms try to reconstruct certain particle properties such as the direction, location and track length. Cleaning algorithms remove noise hits and split clusters of hits that originate from coincident events to separate events.

Aside from cleaning and reconstruction, which are done on single events, several analysis techniques are also used in this work. These algorithms look at large samples and try to classify events as signal- or background-like. Therefore, Monte Carlo samples of both signal and background samples are used. The first algorithm that is discussed below is "minimal-Redundancy-Maximum-Relevance", which tries to filter out the most powerful variables from a large set of variables. Secondly, we discuss a machine learning technique called "Boosted Decision Trees". Such an algorithm is trained with Monte Carlo samples to "learn" how to discriminate signal from background events that outputs a single-valued parameterization. This can then be applied to real data.

7.6 Minimal-Redundancy-Maximum-Relevance

Having multiple variables that are able to discriminate signal from background events is a necessary tool that ensures the ability to conclude statements on a certain theory or exotic phenomenon. Single variables can show promising results, but when multiple variables are highly correlated, much of the discriminative power diminishes. When using BDTs, analyzers often try to include variables and remove them if they show to be highly correlated in a trial-and-error fashion.

In this analysis, I made use of a technique that was originally developed for analyses in biological sciences but can be used for most analyses that involve "data mining". Variables from a large sample set were selected with the condition of minimal-Redundancy-Maximal-Relevance (mRMR). To optimize the characterization of a certain class of events with a set of variables, these variables are selected with a *maximal relevance*. Here, "relevance" is characterized in terms of correlation of mutual information. Because combinations of individually good features do not necessarily lead to good classification performance, there is the additional requirement of *minimal redundancy*. The algorithm sorts variables according to their importance (defined as having a minimal redundancy and maximum relevance). More information about this algorithm can be found in Ref. [217].

In this analysis, mRMR was used to rank variables from a large set according to their importance and proved to lead to low correlated variables (as can be seen in the Appendix in Figure E.2).

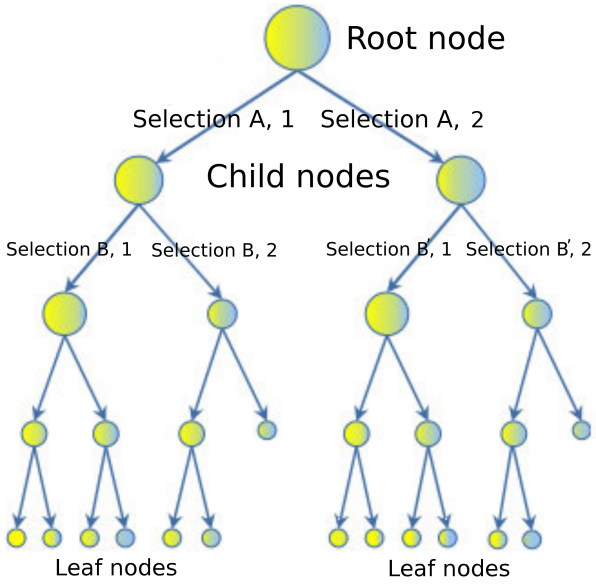

Figure 7.10: Illustration of decision tree scheme. Selection B and B' denote a selection on the same variable, but other requirement.

7.7 Boosted Decision Tree classifiers

Given a set of events that have a fixed set of variables constructed in an analysis, the question remains if the event is in fact a *signal* (dim track from an SMP) or *background* (muon from ν_μ or air showers) event. One can rely on Monte Carlo simulations to get a handle on the variable distribution in both sets. The most general, and still widely used, method is to use a cut-and-count approach where a cut is placed on a certain variable that discards events that fail the requirement. A Boosted Decision Tree (BDT), however, inspects a set of set variables by looking at large sample sets that are classified as either signal or background. The BDT is "trained" to classify an event to signal-like or background-like. This is done with a "BDT score" that ranges from -1 to 1. The higher the score, the more an event is regarded as signal-like. How this is done is given in more detail below. Boosted decision trees rely on multiple individual trees, collectively called a *forest*. Therefore, we will first explain how a single tree classification works.

7.7.1 Structure

The goal of a decision tree is to determine if an event is signal- or background-like. It uses a tree-like structure where certain selection criteria are used at different nodes as illustrated in Figure 7.10.

A decision tree is a binary tree that places an event into a certain node depending on the selection at a node. The depth of a decision tree can be arbitrarily long, but is determined by a set of criteria as defined in Section 7.7.2. An event consists of a certain set of variables $X = x_1, x_2, ..., x_n$ that are used in the classification. Before any selection criteria, the event is said to be represented by the *root node*. A binary selection then determines to which *child node* the event should be classified, for example:

$$\begin{aligned} \text{Selection A} &= x_1 > y_1 \quad \text{(option 1)} \\ &= x_1 \leq y_1 \quad \text{(option 2)}, \end{aligned} \tag{7.20}$$

where y_1 is the cut value for variable x_1. Similarly, the other selections determine where the event is eventually placed. The last nodes are referred to as *leaf nodes* and hold the probabilities of whether an event is more signal- or background-like. These probabilities are translated into a score ranging between -1 (background) and +1 (signal).

7.7.2 Training

To construct a decision tree, one first has to "train" the algorithm on MC. Given a certain "signal set" and "background set", all variables used in the BDT are histogrammed and at each bin for each variable the "best cut" is set at the first node selection. To determine the optimal cut, we first define the purity of a node, p, by

$$p = \frac{\sum_s w_s}{\sum_s w_s + \sum_b w_b}, \tag{7.21}$$

where w_s and w_b refer to the weights of the signal and background events*. The Gini index, g,

$$g(p) = p(1-p), \tag{7.22}$$

is used as a separation variable in this work†. Using the Gini index, the separation gain determines the effectiveness of the cut

$$\Delta S = g_p \cdot \sum w_p - \left(g_l \cdot \sum w_l + g_r \cdot \sum w_r \right), \tag{7.23}$$

where g_p and w_p denote the Gini index and weights of the parent nodes and similarly for the left and right child nodes. The cut that gives the highest separation gain is subsequently selected. The algorithm stops when one of the following criteria is met:

- a node only consists of signal or background events;
- a certain predefined maximal depth is reached;
- splitting would cause a child node to have less than a predefined minimal amount of events left;

and therefore determines the size of a tree. These selection criteria are necessary to avoid overtraining (see Section 7.7.4). An example of a very simple decision tree is given in Appendix C.

7.7.3 Boosting

As already implied in the text above, a BDT consists of a *forest* of decision trees. A user specified number of individual decision trees are trained sequentially, with a boosting process in between each training. Boosting consists of adjusting the weights of individual events according to whether the previously trained tree classifies them correctly. In this work, the AdaBoost‡ algorithm was used for boosting in which the score of an event is a weighted average of the scores the event receives from each tree in the forest [218].

A BDT stands for "boosted decision tree", but it must be understood that there are actually many trees (typically hundreds), and that boosting is a process that occurs between the training of consecutive trees. The approach makes use of the power of numbers: many weak single decision trees combined can be more powerful than one very good decision tree. In general, boosting follows the following steps:

1. Train a weak model on training data.
2. Compute the error of the model on each training example.
3. Give higher importance to examples on which the model made mistakes.
4. Re-train the model using "importance weighted" training examples.
5. Go back to step 2.

A simple example of a BDT is given in Appendix C.

We define a function to indicate whether an event is classified incorrectly: $I(s, y) = 0$ if $s = y$ and 1 otherwise. The error rate for a tree is then equal to

*These are the weights after scaling events to a certain spectrum.

†Other possible separation variables include the cross entropy $-p \cdot \ln(p) - (1-p) \cdot \ln(1-p)$ or the misclassification error $1 - \max(p, 1-p)$.

‡Short for Adaptive Boosting.

$$\epsilon = \frac{\sum_i w_i I(s,y)}{\sum_i w_i}. \tag{7.24}$$

The boosting factor for the tree is defined as

$$\alpha = \beta \cdot \ln\left(\frac{1-\epsilon}{\epsilon}\right), \tag{7.25}$$

with β a user defined *boosting beta* and changes the weight of the tree to

$$\begin{array}{ll} w' = w \cdot \exp(\alpha), & w' = w \cdot \exp(-\alpha) \\ \text{(correct classification)} & \text{(incorrect classification)}, \end{array} \tag{7.26}$$

after which the weights are renormalized so that $\sum w' = 1$. The process is repeated until the number of predefined trees is reached.

Due to its definition, the boost factor α will give good classifiers (which have low error rates) large boost values. Events that are misclassified are then given larger weights, making the algorithm more likely to classify them correctly in the subsequent tree classifier.

Once the entire BDT is trained, the events can be given a score based on the multiple tree classifiers. This is done by taking the weighted average of all the scores in the individual tree classifiers, using its boost factor α as the weight of the tree. The score of an event i is then given by

$$s_i = \frac{\sum_m \alpha_m \cdot s_{i,m}}{\sum \alpha_m}, \tag{7.27}$$

where we loop over the individual trees denoted with index m.

7.7.4 Overtraining

BDTs are very powerful tools, but if not used correctly, could lead to problems that are not easy to spot at first sight. Assume we train our BDT with a certain signal set and background set. If the BDT is trained up to the point of classifying statistical fluctuations, there is said to be *training sample overtraining*. An illustrative example is given in the Appendix in Figure E.1. Another example is data/MC overtraining. The former can be dealt with with the use of *pruning*, while the latter should show clear data/MC agreement, which is checked in the analysis.

7.7.4.1 Pruning

The problem with overtraining is essentially that there are certain splits in a classifier tree that are too specific and less important. In the method of *cost complexity pruning*, for each node the complexity is calculated as

$$\rho = \frac{\Delta S}{n_{\text{leaves}} - 1}, \tag{7.28}$$

with ΔS the separation gain as defined in Eq. 7.23 and n_{leaves} the number of leaves below the split node. The subtree of the node with the smallest value of ρ is removed and this is done repeatedly until a desired *pruning strength** is reached.

*A parameter on a scale from 0 to 100, which specifies the percentage of the pruning sequence to actually execute. 0 means no pruning is done and 100 signifies only a single root node remaining.

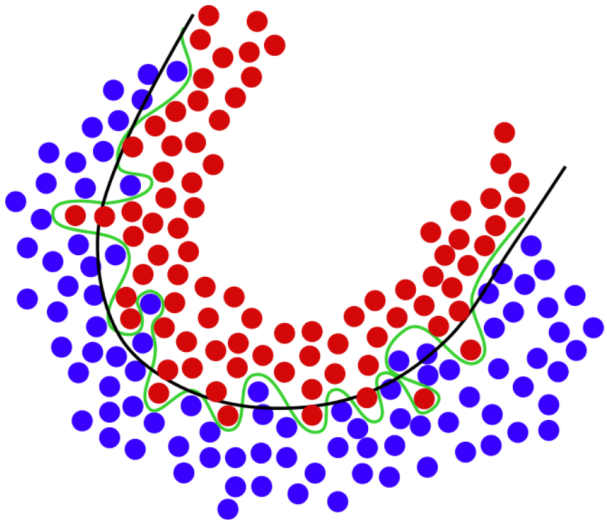

Figure 7.11: Example of overtraining. In black, the theoretical line between background (blue points) and signal (red points) is given. An overtrained BDT will have a (perfect) selection such as the green line. Illustration from [219].

7.7.4.2 Additional checks

BDTs are always trained and tested with different subsets of the same type (background or signal) of event. If the training set has significantly different scores than the testing set, the sample is most probably overtrained and one has to change the BDT input parameters. Additional to pruning, one typically uses Kolmogorov-Smirnov testing to indicate if the BDT score distributions from the training and testing sets could follow the same distribution pattern. Therefore, a p-value equal to the statistical probability that two samples are drawn from the same distribution is computed. Because training and testing samples will never be identical, there will always be some level of overtraining. As a rule of thumb, a $p_{KS} \lesssim 0.01$ indicates overtraining beyond the level of comfort. Appendix E shows the result of such checks.